Graph Neural Networks: A Powerful and Versatile Tool for Advancing Design, Reliability, and Security of ICs

Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu
{lma387,jk176,os22}@nyu.edu
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates (UAE)

ABSTRACT
Graph neural networks (GNNs) have pushed the state-of-the-art (SOTA) for performance in learning and predicting on large-scale data present in social networks, biology, etc. Since integrated circuits (ICs) can naturally be represented as graphs, there has been a tremendous surge in employing GNNs for machine learning (ML)-based methods for various aspects of IC design. Given this trajectory, there is a timely need to review and discuss some powerful and versatile GNN approaches for advancing IC design.

In this paper, we propose a generic pipeline for tailoring GNN models toward solving challenging problems for IC design. We outline promising options for each pipeline element, and we discuss selected and promising works, like leveraging GNNs to break SOTA logic obfuscation. Our comprehensive overview of GNNs frameworks covers (i) electronic design automation (EDA) and IC design in general, (ii) design of reliable ICs, and (iii) design as well as analysis of secure ICs. We provide our overview and related resources also in the GNN4IC hub at https://github.com/DfX-NYUAD/GNN4IC. Finally, we discuss interesting open problems for future research.

1 INTRODUCTION
The increasing complexity of IC design and manufacturing, coupled with the advancements in ML, have led researchers to employ various ML methodologies to advance different aspects of IC design. Accordingly, ML has beaten SOTA approaches in solving various EDA tasks, including hardware security and reliability. Among the different ML methodologies explored, GNNs have demonstrated great potential in transforming IC design workflows.

Circuits, whether in register-transfer level (RTL), gate-level or transistor-level netlist, or layout format, can be naturally represented as graphs. Thus, most EDA tasks can be modelled as graph problems. Given a rapid development of GNNs, the past three years have witnessed an exponential surge of research interest in using GNNs to address various circuit-related tasks. With many options available for circuit-to-graph conversion, extraction, GNN architectures – all for varying tasks – it is vital to summarize the achievements in the field and point out paths for future research.

There are few if any reviews on GNNs covering multiple domains of IC design. References [33, 36, 56, 66] present surveys on ML for advancing EDA in general, whereas References [47, 49, 51, 68] represent some up-to-date surveys on GNNs used for EDA. Wu et al. [82] survey the usage of GNNs for high-level synthesis (HLS) performance prediction. For hardware security, Liu et al. [46] and Sisejkoic et al. [73] review the use of ML in general, whereas Alrahis et al. [13] present a thorough review for using GNNs in particular. In this paper, we cover a wider range of important topics, presenting selected papers on GNNs for IC design, reliability, and security. The contributions of this paper are as follows.

1) We present a taxonomy and a generic pipeline of GNNs for advancing IC design, mainly (i) GNNs for EDA and IC design in general, (ii) GNNs for reliable IC design, and (iii) GNNs for design and analysis of secure ICs.

2) We provide a comprehensive survey. We review selected SOTA and provide detailed descriptions on the GNN tasks, architectures, graph types, circuit design level, features, etc.

3) We collect resources on GNNs for IC design, including the different applications, GNN models, benchmark datasets, and open-source codes, and summarize them in our GNN4IC hub [https://github.com/DfX-NYUAD/GNN4IC].

2 BACKGROUND
2.1 Electronic Design Automation
Sophisticated EDA tools are essential for IC designers to utilize advances in microelectronics while managing the ever-increasing complexities at the same time. In recent years, ML has become a key driver for next-generation EDA tools, as we also show in this paper. Still, the overall EDA pipeline remains largely the same.

System specification and architectural design. The objectives and high-level requirements for functionality, performance, and physical implementation are formulated. Modeling languages like SystemC can be used for a formalized description.

Behavioral and logic design. The specification and architecture are transformed into a behavioral RTL model which describes inputs, outputs, timing behavior, etc. for the whole system. Third-party components can be integrated at this stage as needed. Traditionally, this step is done manually, but nowadays it is also well supported by EDA tools offering HLS.

Logic synthesis. The behavioral model is transformed into a low-level circuit description, the gate-level netlist (GLN). This step requires the technology library for mapping from the generic model to the specific IC design that is to be implemented.

Physical design. The GLN is transformed into the actual physical layout of gates, macros, wires, memories, etc. Considering the high complexity of this stage, it is typically divided into the following tasks: partitioning and/or floorplanning, power and ground delivery, placement, clock delivery, routing, and timing closure.

Verification and signoff. The physical layout must be verified against various design and manufacturing rules, to ensure correct functionality and electrical behaviour. Once all rules are met, the design can be signed off and taped out.
2.2 Hardware Reliability

Technology scaling makes it challenging to ensure the reliability of ICs over the projected device lifetime. In the following, we provide an overview of important reliability concerns.

Process variation occurs due to imperfections in the IC manufacturing process. It is a time-independent source of variation that differs for each IC and even within the chip itself. Random dopant fluctuation, metal gate granularity, and line-edge roughness are among the typical sources of variation that are considered for SOTA FinFET devices [96]. An accurate estimation of process variation is prerequisite to ensure high yield at high performance.

Transistor aging is a time-dependent source of variation that depends on many parameters such as temperature, workload, and projected lifetime. Physical effects that lead to transistor aging include hot-carrier injection (HCI), and importantly, bias temperature instability (BTI) [17]. BTI is one of the dominant contributors to aging-induced degradations. Over an IC’s lifetime, electrical charges get trapped in the gate oxide of transistors, resulting in increased threshold voltages. In addition, interface traps can be generated at the Si-SiO2 layer resulting in more undesired charges and, hence, further increase in the threshold voltage. Consequently, transistor switching times and circuit path delays increase.

2.3 Hardware Security

The IC manufacturing process has become globalized, involving numerous entities across the globe. Such an outsourced supply chain leads to a plethora of security concerns, like implantation of malicious hardware Trojans (HTs), piracy of design intellectual property (IP), and reverse-engineering (RE) [65, 69], any of which can cause financial loss to IP owners and/or put end users at risk.

Next, we review selected threats and countermeasures relevant for this paper at hand. References [39–41] provide a wider coverage, also touching upon how to advance EDA for secure IC design.

2.3.1 Hardware Security Threats. HTs are malicious modifications to ICs that attackers implant to achieve a malignant outcome [69], such as leaking sensitive assets, e.g., crypto keys, or disrupt design functionality. HTs are typically stealthy (i.e., rarely activated) and can be integrated at different stages of the IC design flow, e.g., through adversarial third-party IP cores, “hacked” design tools, or by malign employees during chip fabrication [39].

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 [39].

RE is a process that aims to obtain the IC/IP design, technology, or functionality by analyzing the chip layer by layer [59]. RE can be leveraged either for various attacks or as defense approach, e.g., to detect IP infringement, verify IP implementation, and detect HTs.

2.3.2 Design-for-Trust Methods. Layout camouflaging alters the appearance of a chip to conceal the design IP, i.e., it obfuscates the design information either at the transistor-level [63], gate-level [64], or interconnect-level [62], protecting it from RE attacks.

Logic locking obfuscates the structure and functionality of a design by integrating key-controlled logic elements, referred to as key-gates. These key-gates bind the correct functionality of the design to a secret key that is only known to the legitimate IP owner.

The owner loads the secret key into an on-chip, tamper-proof memory after fabrication and testing. References [5, 12, 15, 16, 43, 72, 74] provide further details on logic locking.

2.4 Graph Neural Networks

Next, we provide a background on GNNs, define important graph-related concepts, and depict the notations used in this paper (Table 1). We begin by defining a graph as follows.

Definition 1. \( G = (V, E) \) denotes a graph with set \( V \) of nodes and set \( E \subseteq V \times V \) of edges. \( X \in \mathbb{R}^{n \times k} \) is a matrix of node features, where \( n \) is the number of nodes in \( G \). \( A \in \{0, 1\}^{n \times n} \) represents the adjacency matrix of \( G \) with \( A_{i,j} = 1 \) iff \((i, j) \in E\).

Definition 2. A directed graph is a graph in which the edges have a direction from one node to another. An undirected graph is a special case of directed graphs where there is a pair of edges with opposite directions for every two connected nodes. \( A \) of an undirected graph is symmetric. A hypergraph is a graph in which an edge can join any number of nodes.

Definition 3. Heterogeneous graphs have nodes and/or edges of different types, while homogeneous graphs have nodes and edges of same types.

Definition 4. GNNs learn on the structure and node attributes of \( G \) to generate a representation (i.e., embedding) \( Z \) that facilitates graph-based ML tasks. 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.

\[
Z^{(l)} = \text{Aggregate} \left( A, Z^{(l-1)}, \sigma \left( Z^{(l-1)} \right) \right)
\]

where \( Z^{(l)} \) is the node embeddings matrix at the \( l \)-th iteration and \( \sigma \) 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, the generated node embeddings \( Z \) can be used directly for different downstream tasks. That is, a GNN models a function \( f_G \) that generates \( z_v = f_G(G, v) \) for \( v \in G \), and the embedding is then passed to a downstream model \( g \) for classification, regression, or clustering. The predicted outcome for \( v \in G \) is denoted as \( \hat{y}_v \), where \( \hat{y}_v = g(z_v) \).

In graph-level tasks, pooling and readout functions generate a graph-level embedding, \( z_G \), which can be used for graph classification, regression or matching. Overall, a GNN models a function \( f_G \) that generates \( z_G = f_G(G) \). The embedding is then passed
to a downstream model \( g \) for classification, regression, or matching [37]. The predicted outcome for \( G \) is denoted as \( \hat{y}_G \), where \( \hat{y}_G = g(z_G) \). The node embeddings can further be used to solve edge-level tasks, such as link prediction and edge classification, which require the model to predict whether there is an edge between two target nodes (identified by set \( S \)) or to classify edge types.

In GNN-based link prediction, an enclosing subgraph around each target link is extracted [93]. Given \((S, G)\), the \((S, h)\) subgraph induced from \(G\) by \( V_{S,E} \{ u \mid d(u, v) \leq h \} \), where \( d(u, v) \) is the shortest path distance between nodes \( u \) and \( v \). The subgraphs hold information about the network surrounding the links. Therefore, by performing graph classification, the labels of the target links also become the labels of their corresponding subgraphs.

3 PROPOSED TAXONOMY

Figure 1 demonstrates a generic pipeline for using GNNs to solve circuit-related tasks. First, some circuit, digital or analog, given in RTL, GLN, or transistor-level format, is represented as a graph. Different tasks require different graph representations, such as directed, undirected, hypergraphs, heterogeneous, or homogeneous graphs. Various node and/or edge attributes can be extracted from the circuit, depending on the target task. Next, the graphs are passed to the GNN model, which solves some ML tasks, such as node classification or regression, graph matching or clustering, etc.

Researchers have utilized various GNN architectures like the graph convolutional network (GCN) [37], message passing neural network (MPNN) [29], GraphSAGE model [31], graph isomorphism network (GIN) [84], deep graph convolutional neural network (DGCNN) [94], graph attention network (GAT) [76], principal neighborhood aggregation (PNA) [26], etc. Researchers may also devise some custom GNN architecture as needed, specific to the circuits and tasks at hand. In general, the GNN model can be integrated with other ML algorithms, such as long-short-term memory (LSTM) models and reinforcement learning (RL).

In the following, we give a detailed survey for the circuit-related domains covered in our taxonomy: EDA, reliability, and security.

3.1 GNNs for EDA

GNNs have been proposed for various tasks throughout the EDA pipeline. We review selected works below and summarize in Table 2.

3.1.1 Behavioral and Logic Design. Wu et al. [80, 81] tackle design-space exploration (DSE) for HLS, considering user constraints and Pareto optimization. The framework consists of three modules: a GNN-based predictor for performance and resource utilization, an RL-based multi-objective DSE engine, and a code transformer to assist the other modules, by extracting data-flow graphs (DFGs) from high-level C/C++ code and generating RTL code.

D-SAGE [75] covers operation mapping in HLS for FPGAs. The authors find that structures around arithmetic operations impact the mapping between operations and FPGA resources and thus timing. Their framework then learns such mapping behaviour to perform mapping-aware timing characterization of arithmetic modules.

Lopera et al. [48] utilize GNNs for timing estimation at RTL. Their work applies GNNs for predicting delay and slew values of RTL components, by mapping design components to DAGs with multidimensional node and edge features. Two GNN architectures are used, MPNN and GCN, and their extracted node embeddings are passed through an MLP, which performs an edge-level regression task. With such edge-level prediction, however, this work does not predict critical paths of a design directly using GNN-based models of circuits, which we expect to be more powerful (Sec. 3.2).

3.1.2 Logic Synthesis. Kirby et al. [38] propose the use of GATs for prediction of routing congestion before placement. Graphs are simple undirected circuit representations with nodes representing gates and edges wires. Initial features are cell type, function, geometry. Their work is one of the first to use GNNs for EDA tasks.

Ghose et al. [28] predict routing congestion during synthesis. Their method works on netlist structures/graphs and can generalize across graphs. The authors find that popular embedding methods did not perform well for the task, but matrix factorization with clustering showed good promise. They also found that deep, wide GATs are not performing as well as shallow, wide versions of GraphSAGE.

Zhou et al. [98] propose a heterogeneous, directed GNN model to encode timing graphs. The model is trained using imitation learning, and aims to accelerate Lagrangian relaxation (LR)-based gate sizing. The model is shown to transfer well to larger unseen graphs.

Wu et al. [79] propose a hybrid framework to predict quality-of-result. The framework jointly considers circuit structures and temporal information, i.e., the order of synthesis steps. Thus, two hybrid GNN-based models that exploit spatio-temporal information are proposed; the first uses a GNN to characterize designs via “supernodes” encoding temporal information, the second uses a GNN for spatial learning and an LSTM for temporal learning.

3.1.3 Partitioning, Floorplanning. Lu et al. [50] tackle partitioning in 3D ICs, i.e., the assignment of modules to layers in the 3D stack. First, their work proposes a hierarchy-aware edge-contraction algorithm (to manage routing overheads observed with classical bin-based partitioning). Then, layer partitioning/assignment is mapped to a problem of weighted k-means clustering, which is solved using
an unsupervised GNN framework. Aside from the hierarchy, the feature vectors also describe timing parameters.

Mirhoseini et al. [60] tackle floorplanning in an industrial context. The authors approach this problem via RL and develop an edge-based GCN that describes the high-level netlist of modules.

3.1.4 Placement and Routing. Xie et al. [83] propose a GAT framework for estimating net lengths prior to placement. Features are derived from partitioning, e.g., cuts and cluster assignment, and are used as directional edge features to describe relations between nodes/nets. Hypergraphs with cell nodes are considered as well.

Lu et al. [54, 55] present a GraphSAGE-based framework that generates cell clusters to guide placement. The weighted k-means clustering is derived from learned node embeddings, which build on hierarchy information and memory affinity as initial features.

Agnesina et al. [2] propose an RL framework for tuning placement, with GraphSAGE used for processing the netlist graph considering initial features like area, fanout, gate type, etc. More features, describing the netlist from global perspective, are considered for the overall RL framework. The authors present a similar approach in [1] where they also touch upon partitioning in 3D ICs.

Guo et al. [30] tackle the prediction of arrival times and slacks, to support timing-driven placement without the need for static timing analysis (STA). Circuits are represented as heterogeneous graphs with edges for nets as well as cells and nodes for cell pins. Considered features are inspired by timing engines, e.g., pin capacitances and net distances are considered. The authors argue that “brute-force stacks” of (overly) deep GCN layers do not perform well, clear exploration of the GNN architecture is called for.

Back et al. [18] predict hotspots for design rule check (DRC) issues during placement, considering both pin access and routing congestion. To do so, the authors first devise a pin proximity graph, which models the spatial information on cell pins and pin-to-pin disturbances/conflicts. Then, they propose an ML model that combines GNN with U-Net; the latter is a fully convolutional network that is effective for semantic segmentation. The GNN and U-Net accommodate different features: the GNN uses the pin proximity graph, which captures local pin accessibility, whereas the U-net

| Platform | GNN Task | GNN | #Layers | Loss Function | Graph Type | Circuit Type | Features | Pooling | Readout |
|----------|----------|-----|---------|---------------|------------|--------------|---------|---------|---------|
| [48]     | Edge regression | GCN [37] | MSE | DAG | RTL node type, #inputs/Outputs Bitwidth, Depth level | NA | NA |
| D-SAGE [75] | Node/Edge classification | Custom GraphSAGE [31] | 2 | Cross-entropy | DFG | HLS | Operation type Bitwidth | NA | NA |
| [8]      | Imitation learning | HCON [34] | 3-6 | Weighted cross-entropy | Heterogeneous directed | GLN | Footprint, Size, Output pin (pin) Fanin, Area, Slack (pin), Gain (pin), Slow (pin) | NA | NA |
| CongestionNet [38] | Node regression | GAT [76] | 16 | MSE | Undirected | GLN | Cell type, Size, Pin count | NA | NA |
| [39]     | Node regression | GraphSAGE [31] | Cross | Squared error | Undirected | GLN | Pin number, Cell size | NA | NA |
| TP-GNN [50] | Node clustering | GraphSAGE [31] | 2 | Unsupervised clique | GLN | Hierarchy, Topology, Distance between | NA | NA |
| Edge-GNN [60] | Node clustering | GraphSAGE [31] | 2 | Unsupervised clique | GLN | Hierarchy and memory features | NA | NA |
| Net² [83] | Node regression | Custom GAT [83] | Directed | GLN | Driver’s area, Fanin, Fanout Cluster ID (edges) | NA | NA |
| GRANNITE [95] | Node regression | Sequential GCN [37] | Directed | GLN | Intrinsic state probabilities Inerting logic Pin state, output state core (edge) | NA | NA |
| [57]     | Node classification | GCN [37] | 3 | Cross-entropy | Directed | GLN | Controlability, Observability | NA | NA |
| PL-GNN [54, 55] | Node clustering | GraphSAGE [31] | 1 | Unsupervised | GLN | Hierarchy and memory features | NA | NA |
| [1, 2]   | GNN-RL | GraphSAGE [31] | Unsupervised | Directed | GLN | Unsupervised connection, Gate type Fanout, Area, Max. clique, & colorability | NA | NA |
| [30]     | Node classification | Node clustering | Custom [30] | Custom L2 | Heterogeneous | GLN | Fanin/Fanout, FE/PO Pin capacitance, Distance to die boundaries | NA | NA |
| DoomedRun [52] | Graph regression | GraphSAGE [31] | Directed | GLN | Worst slack/output slew | Mean pooling LSTM | NA | NA |
| DeepPR [24] | GNN-RL | GCN [37] | 2 | Directed | Hypergraph | GLN | Gate/poly length, α, β, γ Length of resistor, #Copies (capac.) Device type, Fanout | NA | NA |
| ParaGraph [45, 67] | Node regression | GraphSAGE [31] | 2 | Directed | Hypergraph | GLN | Pre-layout transistor netlist (TTL) | NA | NA |
| Circuit-CNN [92] | Graph regression | Custom [92] | Directed | Resonant | Transistor model parameters Node type, NMOS, PMOS, RC | NA | NA |
| Circuit Designer [77] | GCN [37] | 7 | Custom [77] | Transistor | Transistor model parameters Node type, NMOS, PMOS, R, C | NA | NA |
| IronMan [80, 81] | GNN-RL | GCN [37] | Directed | GLN | Node type, Data precision | Mean | NA |
considers a set of grid-based features to describe both local and global perspectives for routing congestion.

Cheng et al. [24] present a joint RL framework with gradient-based optimization to tackle both placement and routing. The authors utilize a multi-view embedding model to encode both global graph level and local node level information as well as distillation, all to improve exploration of the search space.

3.1.5 Timing Closure. Lu et al. [53] present an RL framework with GNN for gate sizing. Among others, the authors find that the GNN subgraph extraction is essential for predicting the total negative slack (TNS) impact of the related sub-circuits on the overall circuit.

Lu et al. [52] seek to identify design runs that would fail timing closure, by predicting the post-route TNS. Toward this end, the authors leverage GNNs to represent netlist graphs extract from prior stages, namely placement and clock delivery. They also utilize LSTM networks to realize sequential modeling of the design flow.

3.1.6 Verification, Testing. Zhang et al. [95] propose supervised learning of toggle rates for power estimation without the need for gate-level simulations. The authors represent netlists as graphs, and use register states and inputs from RTL simulation as features as well as combinational gates’ toggle rates as labels.

Ma et al. [57] propose a multi-stage GCN model to predict candidates for observation points in a netlist. Given a set of netlists with all nodes labeled as either difficult-to-observe or easy-to-observe, a classifier is trained to find a set of locations where the observation points should be inserted, which can maximize fault coverage and minimize observation points number and test pattern number.

3.1.7 Analog, Mixed Signal, and Transistor Design. Zhang et al. [92] use GNNs to tackle distributed circuit design, i.e., analog and mixed-signal (AMS) circuits operating at high frequencies with wavelengths comparable to or smaller than circuit components, as seen in 5G and 6G electronics. The authors show that their GNN model applies for both simulating such circuits as well as automating the otherwise complex and expert-driven design process. Simulation times are reduced significantly, and new structures/templates have been found during GNN-based design.

Ren et al. [67] predict layout parasitics and device parameters of AMS circuits, to render pre-layout simulations more efficient and accurate. The authors convert circuit schematics to graphs and leverage GraphSage, Relation GCN, and GAT networks. They argue that such ensemble modeling increases model accuracy over a large range of prediction values. Liu et al. [45] extend the work, by considering dropout as an efficient prediction of uncertainty for Bayesian optimization to automate transistor sizing. The authors argue that the inclusion of parasitic prediction in the optimization loop could guarantee satisfaction of all design constraints, while schematic-only optimization fail numerous constraints if verified with parasitic estimations.

Wang et al. [77] propose an RL framework with GCN for automatic transistor sizing. The authors show that their model enables transfer learning across five nodes and two circuit topologies with superior figure of merit over prior art.

Chen et al. [20] tackle symmetry matching in AMS circuits. The authors propose an unsupervised, inductive GNN framework that supports both system-level and device-level symmetry constraints extraction and is generalizable. A heterogeneous multi-graph representation has been proposed for interconnection modeling, and a circuit feature embedding algorithm has been shown to represent circuits with the most representative substructures.

3.2 GNNs for Hardware Reliability

Only recently, GNNs have been used to tackle reliability problems. We review selected works below and summarize in Table 3.

Alrahis et al. [10] present a GNN model to predict delay degradations in digital ICs, for any given timing path, due to process variations and device aging. The objective is to allow a designer to size guardbands such that correct design operation is ensured, without need for STA nor for access to sensitive transistor aging models during inference. The framework extracts subgraphs around timing paths of interest and performs a subgraph regression problem.

Chen et al. [21, 22] propose a GCN framework to predict aging-induced transistor degradation in analog ICs. Since analog circuits contain various different components (i.e., transistors, capacitors, resistors, etc.), the authors utilize a heterogeneous graph representation and GNN architecture to tackle this task.

3.3 GNNs for Hardware Security

GNNs have been used to address various hardware security concerns. We review selected works below and summarize in Table 4.

3.3.1 HT Detection. Yasaei et al. [87] propose GNN4TJ, a GNN platform for HT detection without prior knowledge of the design IP or the HT structure. GNN4TJ first converts a set of RTL designs into their corresponding DFGs, which are fed to a GNN to learn about the structure of the designs. Then, given a design to test, the GNN performs graph classification to predict the presence/absence of HTs across the entire design. Other works extend the concept of this framework using GLNs [86, 90].

Further GNN-based platforms [61, 85] importantly enable HT localization, i.e., HT detection at the node-/gate-level.

3.3.2 Piracy Detection. Yasaei et al. [88] propose a GNN-based method for detecting IP piracy. The framework, GNN4IP, assesses the similarity between the original and the questionable circuit to predict theft. GNN4IP can assess circuits given either as RTL or GLN. GNN4IP does not require the addition of any watermarks or fingerprints, reducing overheads and avoiding removal attacks [4, 27]. This is because the GNN generates an embedding for each circuit from its underlying structure, which is considered as “signature” of the design. Subsequently, the GNN optimizes the embeddings such that distances in the embedding space reflect the similarity between designs [31]. Finally, GNN4IP computes the cosine similarity score for embeddings to predict piracy across the two circuits.

Yu et al. [90] propose HW2VEC, which combines GNN4TJ [87] and GNN4IP [88] into a single framework. HW2VEC supports different graph types, such as DFG and abstract syntax tree (AST).

3.3.3 Functional Reverse Engineering. Alrahis et al. [8] propose GNN-RE, a GNN-based platform for functional RE. A given GLN is first transformed into an undirected graph, with gates represented as nodes that are initialized with a feature vector of gate type, neighboring gates, input degree, output degree, and connectivity to primary ports. Then, GNN-RE performs node classification to identify which gates belong to which sub-circuit/module.
In RelGNN [25], GLNs are represented as graphs which the GNN processes to discriminate between state and data registers. He et al. [32] propose an asynchronous bidirectional GNN (ABGNN) for sub-circuit classification in GLNs, focusing on arithmetic blocks and specifically adders. ABGNN represents GLNs as directed graphs, to maintain the natural data flow direction of circuits. ABGNN predicts boundaries of adder blocks by training two separate GNNs; one aggregates information from the respective fan-in cones and the other from the fan-out cones. The two generated embeddings are then combined as final embedding.

The above methods focus solely on structural circuit properties for prediction. In contrast, Wang et al. [78] propose a contrastive learning (CL)-based framework for netlist representation, to extract the logic functionality, which is universal and transferable across different circuits. The authors also propose a functionality-centric GNN that works in tandem with the netlist representation framework classification purposes. This GNN builds on ABGNN [32], but uses independent aggregators for different node types.

Zhao et al. [97] employ a GNN for functional RE under circuitrewriting. Authors consider the features used for GNN-RE [8] and a set of functionality-aware features, thus improving the prediction accuracy. Features are based on truth table information, signal probability, and negation-permutation-negation (NPN) class.

Bucher et al. [19] propose AppGNN, a platform that extends on GNN-RE [8] to handle approximate circuits. AppGNN employs a node-sampling algorithm while training the GNN on exact circuits, allowing the model to generalize to approximate computing.

Table 3: Summary of selected GNN frameworks for hardware reliability. NA means not applicable.

| Platform          | GNN Task      | GNN       | #Layers | Loss Function | Graph Type | Circuit Type       | Features                              | Pooling      | Readout |
|-------------------|---------------|-----------|---------|---------------|------------|-------------------|---------------------------------------|--------------|---------|
| GNN4IP [88]       | Node regression | H-GCN [21] | 2       | MSE           | Directed   | Post-layout TNL   | Design parameters                     | NA           | NA      |
| Deep H-GCN [22]   | Node regression | H-GCN [22] | 2-6     | MSE           | Directed   | Post-layout TNL   | Design parameters                     | NA           | NA      |
| GNN4AREL [19]     | Subgraph regression | PNA [26]  | 4       | MSE           | Directed   | GLN               | Gate type                              | NA           | Add     |

Table 4: Summary of selected GNN frameworks for hardware security. – means unspecified, and NA means not applicable.

| Platform          | GNN Task      | GNN       | #Layers | Loss Function | Graph Type | Circuit Type       | Features                              | Pooling      | Readout |
|-------------------|---------------|-----------|---------|---------------|------------|-------------------|---------------------------------------|--------------|---------|
| GNN4IP [88]       | Graph classification | GNN [7]   | 2       | Cross-entropy | Directed   | GLN               | Gate type                              | NA           | NA      |
| GNN4IP [88]       | Graph similarity | GNN [7]   | 2       | Cross-similarity | Directed   | GLN               | Gate type                              | NA           | NA      |
| GNN-RE [8]        | Node classification | GraphXANT [91] | 5       | Cross-entropy | Directed   | GLN               | Gate type                              | NA           | NA      |
| AppGNN [19]       | Node classification | GraphXANT [91] | 5       | Cross-entropy | Directed   | GLN               | Gate type                              | NA           | NA      |
| [97]              | Node classification | GNN [37]  | 3       | –             | DAG        | GLN               | Gate type                              | NA           | NA      |
| NCL [78]          | Node classification | FGNN [78] | –       | Negative log-likelihood | Directed | GLN               | Gate type                              | NA           | NA      |
| ReIGNN [25]       | Node classification | GraphSAGE [31] | 3       | Negative log-likelihood | –       | GLN               | Gate type                              | NA           | NA      |
| ICNet [23]        | Graph regression | GAT [76]  | 2       | Mean squared error | MSE | Undirected | GLN               | Gate type                              | Attention    | Attention |
| GNNUnlock [7]     | Node classification | GraphXANT [91] | 2       | Cross-entropy | Undirected | GLN               | Gate type                              | NA           | NA      |
| GNNUnlock+ [6]    | Node classification | GraphXANT [91] | 2       | Cross-entropy | Undirected | GLN               | Gate type                              | NA           | NA      |
| OMLA [9]          | Subgraph classification | GIN [84]  | 5-6     | Cross-entropy | Undirected | GLN               | Gate type                              | Summing node features from the same layer | Sum        |
| MuxLink [14]      | Link prediction | DGCNN [94] | 4       | Cross-entropy | Undirected | GLN               | Gate type                              | DGNN sortpooling [94] | Max       |
| Titan [58]        | Subgraph classification | GIN [84]  | 6       | Cross-entropy | Undirected | GLN               | Gate type                              | Summing node features from the same layer | Sum        |

Kunal et al. [42] propose GANA, a GCN-based platform for RE of analog circuits.

3.3.4 Attacks on Design-for-Trust Methods. Various GNN-based algorithms have demonstrated strong results for breaking logic locking and layout camouflaging, either by learning on and predicting the structures of the IC netlists despite the presence of these design-for-trust measures, or by directly learning, predicting, and bypassing the structures of these measures themselves [6, 9, 11, 14, 73]. The related GNN attacks work predominantly in an oracle-less setting, which is more powerful than seminal attacks like the Boolean Satisfiability (SAT)-based attack [74]. These SOTA GNN works exposed the following shortcomings in the design-for-trust measures.

**Key leakage.** OMLA is a GNN-based attack on X(N)OR locking [9]. OMLA represents the locked GLN as an undirected graph and extracts an h-hop enclosing subgraph around each key-gate. The subgraphs include information on the key-gates themselves but also on their surrounding circuitry. Thus, OMLA deciphers the key-bit values by performing subgraph classification.

**Link formation.** MuxLink [14] and UNTANGLE [11] exposed a new vulnerability – link formation – to break MUX-based logic locking [3, 35, 71] which was previously considered ML-resilient. The intuition behind MuxLink and UNTANGLE is that modern ICs contain a considerable degree of repetition and IP reuse [70]. Thus, MuxLink and UNTANGLE both employ a GNN to learn the remaining/unlocked parts of a locked design and then perform predictions regarding the obfuscated interconnects. More specifically,
MuxLink and UNTANGLE convert the problem of deciphering the inputs of a MUX key-gate to a link-prediction problem and solves it using a GNN. Titan [58] is a GNN-based attack on large-scale obfuscation that combines the concepts of OMLA, MuxLink, and UNTANGLE into a holistic attack platform that can handle both gate and interconnect obfuscation. To achieve this, Titan expands the GNN model to more classes and considers more target nodes.

**Structural leakage.** GNNUnlock [6, 7] targets on provably-secure logic locking (PSLL). It use a GNN to identify and isolate the protection logic in PSLL, which is generally embedded in the design. Identifying this protection logic facilitates its removal, enabling the recovery of the original design [44, 89].

**Attack scalability.** As indicated, the SAT-based attack is a seminal attack on logic locking [74]. Although this attack succeeds on a wide range of locking techniques, its run-time can vary from seconds to days, depending on the circuit and the locking technique. The ICNet [23] platform represents the locked netlist as graph and encodes each node/gate with a key-gate mask, indicating if the node represents a key-gate or not, and the gate type. The employed GCN then performs a regression task, predicting the run-time of the SAT-based attack on the given circuit.

**REFERENCES**

[1] Anthony Agnesina et al. 2020. A General Framework For VLSI Tool Parameter Optimization with Deep Reinforcement Learning. In NeurIPS.

[2] Anthony Agnesina, Kyungwook Chang, and Sung Ryu Lim. 2020. VLSI Placement Parameter Optimization using Deep Reinforcement Learning. In ICCAD.

[3] Abdurahman Alaql, Mo Mohsin Rahman, and Swaprag Bhuina. 2021. SCOPE: Synthesis-Based Constant Propagation Attack on Logic Locking. TCAD.

[4] Souha Alkabani et al. 2007. Remote activation of ICs for piracy prevention and digital right management. In ICCAD. 674–677.

[5] Lilas Alrahis et al. 2019. Functional Reverse Engineering on SAT-Attack Resilient Logic Locking. In ISCAS 1–5.

[6] Lilas Alrahis et al. 2021. GNNUnlock+: A Systematic Methodology for Designing Graph Neural Networks-based Oracle-less Unlocking Schemes for Provably Secure Logic Locking. TETF (2021).

[7] Lilas Alrahis et al. 2022. GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking. In DATE. 780–785.

[8] Lilas Alrahis et al. 2022. GNN-RE: Graph Neural Networks for Reverse Engineering of Gate-Level Netlists. TCAD. 41, 8 (2022), 2435–2448.

[9] Lilas Alrahis et al. 2022. OMLA: An Oracle-Less Machine-Learning-Based Attack on Logic Locking. TCAS-II. 69, 3 (2022), 1602–1606.

[10] Lilas Alrahis, Johann Knechtel, Florian Klemme, Humam Amrouch, and Ozgur Sinanoglu. 2022. GNN4REL: Graph Neural Networks for Predicting Circuit Reliability Degradation. TCAD. 41, 11 (2022), 5836–5857.

[11] Lilas Alrahis, Satwik Patnaik, Abdullah Hand, Muhammad Chaﬁque, and Ozgur Sinanoglu. 2021. UNTANGLE: Unlocking Routing and Logic Obfuscation Using Graph Neural-Networks-based Link Prediction. In ICCAD. 1–9.

[12] Lilas Alrahis, Satwik Patnaik, Johann Knechtel, Han Saleh, Baker Mohamed, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. 2021. UNSAIL: Thwarting Oracle-Less Machine Learning Attacks on Logic Locking. TIFS. 16 (2021), 2580–2523.

[13] Lilas Alrahis, Satwik Patnaik, Muhammad Chaﬁque, and Ozgur Sinanoglu. 2022. Embracing Graph Neural Networks for Hardware Security. In ICCAD.

[14] Lilas Alrahis, Satwik Patnaik, Muhammad Chaﬁque, and Ozgur Sinanoglu. 2022. MuxLink: Circumventing Learning-Resilient MUX-Locking Using Graph Neural-Networks-based Link Prediction. In DATE. 694–699.

[15] Lilas Alrahis, Muhammad Yasin, Nimisha Limaye, Han Saleh, Baker Mohamed, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. 2019. ScanSAT: Unlocking State and Dynamic Scan Obfuscation. TETF (2019), 1–9.

[16] Lilas Alrahis, Muhammad Yasin, Han Saleh, Baker Mohamed, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. 2019. ScanSAT: Unlocking Obfuscated Scan Resources. In ASP-DAC. 352–357.

[17] Humam Amrouch, Behnam Khaleghi, Andreas Gerstlauer, and Jorg Henkel. 2016. Reliability-aware design to suppress aging. In DAC. 1–6.

[18] Kyeonghyeon Baek, Hyunbom Park, Suwan Kim, Kyumyoung Cho, and Tae-whan Kim. 2022. Pin Accessibility and Routing Congestion Aware DRC Hotspot Prediction using Graph Neural Network and U-Net. In ICCAD.

[19] Tim Bucher, Lilas Alrahis, Guillaume Paum, Sergio Bampi, Ozgur Sinanoglu, and Hussam Amrouch. 2022. AppGNN: Approximation-Aware Functional Reverse Engineering using Graph Neural Networks. ICCAD (2022). 1–6.

[20] Hao Chen, Keren Zhi, Mingjie Liu, Xiuyan Tang, Nan Sun, and David Z. Pan. 2021. Universal Symmetry Constraint Extraction for Analog and Mixed-Signal Circuits with Graph Neural Networks. In DAC. 1243–1248.

[21] Tinghnan Chen et al. 2021. Analog IC Aging-Induced Degradation Estimation via Heterogeneous Graph Convolutional Networks. In ACP-DAC. 898–903.

[22] Tinghnan Chen et al. 2022. Deep H-GCN: Fast Analog IC Aging-Induced Degradation Estimation. TCAD. 41, 7 (2022), 1998–2009.

[23] Zhiqian Chen et al. 2020. Estimating the circuit de-obfuscation runtime based on graph deep learning. In DATE. 358–363.

[24] Ruoyu Cheng and Junchi Yan. 2021. On Joint Learning for Solving Placement and Routing in Chip Design. NearPS. 34 (2021), 16508–16519.

[25] Subhajit Dutta Chowdhury et al. 2021. ReIGNN: State Register Identification Using Graph Neural Networks for Circuit Reverse Engineering. In ICCAD. 1–9.

[26] Gabriele Corso et al. 2020. Principal neighbourhood aggregation for graph nets. NeurIPS 33 (2020), 13269–13281.

[27] Aijiao Cui, Gang Qu, and Yan Zhang. 2015. Ultra-low overhead dynamic watermarking on scan design for hard IP protection. TIFS. 10, 11 (2015), 2298–2313.

[28] Anur Ghose et al. 2021. Generalized Cross-Graph Embedding for GNN-Based Congestion Prediction. In ICCAD.

[29] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In NIPS. 1263–1272.

[30] Zicheng Guo et al. 2021. A Timing Engine Inspired Graph Neural Network Model for Pre-Routing Slack Prediction. In DAC. 1207–1212.

[31] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In NIPS. 1025–1035.

[32] Zhuxuan He, Ziyi Wang, Chen Bai, Haoyu Yang, and Bei Yu. 2021. Graph Learning-Based Arithmetic Block Identification. In ICCAD. IEEE. 1–8.

[33] Gaynne Huang et al. 2021. Machine Learning for Electronic Design Automation: A Survey. TODAES. 26, 5, Article 40 (2021).
