IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 23 May 2021; revised 14 August 2021; accepted 25 August 2021.
Date of publication 6 September 2021; date of current version 6 September 2022.

*Digital Object Identifier 10.1109/TETC.2021.3108487*

# GNNUnlock+: A Systematic Methodology for Designing Graph Neural Networks-Based Oracle-Less Unlocking Schemes for Provably Secure Logic Locking

**LILAS ALRAHIS, (Member, IEEE), SATWIK PATNAIK, (Member, IEEE), MUHAMMAD ABDULLAH HANIF, (Graduate Student Member, IEEE), HANI SALEH, (Senior Member, IEEE), MUHAMMAD SHAFIQUE, (Senior Member, IEEE), AND OZGUR SINANOGLU, (Senior Member, IEEE)**

Lilas Alrahis, Muhammad Shafique, and Ozgur Sinanoglu are with the Division of Engineering, New York University Abu Dhabi, Abu Dhabi 129188, UAE
Satwik Patnaik is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA
Muhammad Abdullah Hanif is with the Institute of Computer Engineering, Vienna University of Technology, 1040 Wien, Austria
Hani Saleh is with the System on Chip Center (SoCC), Khalifa University, Abu Dhabi 127788, UAE

CORRESPONDING AUTHORS: LILAS ALRAHIS (lma387@nyu.edu) and SATWIK PATNAIK (satwik.patnaik@tamu.edu)

**ABSTRACT** Leading-edge design houses outsource the fabrication process to pure-play foundries eliminating the expenses of owning and maintaining a fab. The intellectual property (IP) of an outsourced design is now subject to IP piracy, which drives the need for a protection mechanism. Logic locking is a technique that aims to thwart IP piracy throughout the supply chain. However, state-of-the-art, provably secure logic locking (PSLL) techniques are vulnerable to functional and structural analysis-based attacks. Few removal attack protection mechanisms have been developed, such as diversified tree logic and wire entanglement, to protect PSLL against structural attacks. In this work, we significantly enhance GNNUnlock (**GNNUnlock+**) and demonstrate how the most advanced PSLL techniques armed with removal attack protection have no impact on its effectiveness. Our evaluation demonstrates that GNNUnlock+ is $89.66\% - 100\%$ successful in breaking benchmarks locked using 9 different PSLL techniques—Stripped functionality logic locking, tenacious and traceless logic locking, Anti-SAT, SAT attack resistant logic locking (SARLock), Anti-SAT with diversified tree logic (Anti-SAT-DTL), Anti-SAT with wire entanglement, SARLock-DTL, corrupt and correct (CAC) and CAC-DTL. GNNUnlock+ can break the considered techniques under different parameters, synthesis settings, and technology nodes. Moreover, GNNUnlock+ successfully breaks corner cases where even the most advanced state-of-the-art attacks fail.

**INDEX TERMS** Logic locking, IP protection, hardware security, oracle-less attacks, machine learning, graph neural networks

## I. INTRODUCTION

DESIGN companies are outsourcing fabrication to third-party, off-shore foundries to reduce the economic costs of integrated circuit (IC) manufacturing. Outsourcing the fabrication exposes the design intellectual property (IP) to theft and piracy. Consequently, there has been a growing demand for developing design-for-trust approaches to regain trust in the supply chain [2]. Researchers proposed numerous countermeasures, such as logic locking [3], split manufacturing [4], and layout camouflaging [5], to protect the design IP from reverse engineering, theft, and piracy. Nevertheless, most techniques ensure protection during specific stages of the design flow. On the other hand, logic locking protects the design IP throughout the supply chain [6].

Logic locking conceals the functionality of a design by embedding key-controlled logic into the original design [3]. A correct key unlocks the locked design and ensures the correct functionality. Researchers evaluated the resilience of logic locking techniques by formulating attacks that aim at recovering the secret key [7], [8]. These attacks are referred to as "oracle-guided" as they require a functional chip with the correct key loaded onto its memory to act as an

**TABLE 1. Capabilities offered by different oracle-less attacks on provably secure logic locking techniques.**

| Attack \ Capability | Different Design Formats | Different Locking Techniques | Different Parameter Settings | Targeted Removal Attack Protection |
|---|---|---|---|---|
| SPS [17] | ✗ | ✗ | ✓ | ✗ |
| RE-based [11] | ✗ | ✗ | ✓ | ✗ |
| FALL [13] | ✗ | ✗ | ✗ | ✗ |
| SFLL-HD-Unlocked [12] | ✗ | ✗ | ✗ | ✗ |
| GNNUnlock [1] | ✓ | ✓ | ✓ | ✗ |
| GNNUnlock+ | ✓ | ✓ | ✓ | ✓ |

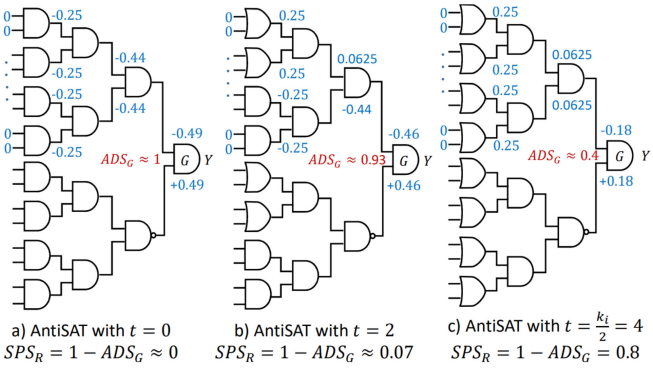
"oracle" [7]. The Boolean satisfiability (SAT)-based attack is one such oracle-guided attack that broke all primitive logic locking techniques. The SAT-based attack iteratively rules out incorrect keys using discriminating input patterns (DIP) determined by a SAT solver. Researchers developed provably secure logic locking (PSLL)[1] techniques to specifically thwart the SAT-based attack by ensuring that *each DIP can rule out exactly one incorrect key, enforcing an exponential number of attack iterations.* In principle, a PSLL technique implements a corrupt and restore strategy. A *perturb* unit introduces controlled errors into the design for some protected input patterns. These errors are corrected using a *restore* unit when the correct key is applied. We denote the perturb and restore units as the *protection logic.*

Although PSLL techniques mitigate the SAT-based attack, recent works have demonstrated that "oracle-less" attacks are feasible. The oracle-less attacks are considered more potent for evaluating the security of logic locking techniques due to the following reasons. (i) Once an adversary gains access to the physical chip (with the embedded key), logic locking becomes vulnerable to physical and tampering attacks, such as probing and fault injection [9], (ii) a hardware Trojan can be designed to leak the secret key upon the activation of the chip [10]. In contrast, the oracle-less attacks rely only on the structure of the locked design, eliminating the need for an oracle, which poses a more potent threat [11]–[13]. The goal of the oracle-less attacks is to analyze the structure of the locked design and locate the protection logic. An attacker can either (i) remove the protection logic to recover the original design or (ii) solve the secret key from the hints embedded in the structure of the protected logic. Although the attacks share the same objective (i.e., challenge the security guarantees of PSLL techniques), *each attack aims to circumvent a specific locking technique under restricted parameter settings and design formats*, resulting in a large set of attack strategies. Next, we discuss the drawbacks of the current oracle-less attacks on PSLL techniques (see Table 1).

### A. STATE-OF-THE-ART AND THEIR LIMITATIONS

*Targeted Removal Attack Protection.* PSLL techniques in their basic form are vulnerable to partitioning and removal attacks [14]. Researchers followed different approaches to address the vulnerabilities as outlined next. For example, when the protection block of the Anti-SAT [14] is incorporated with the original design using *wire entanglement* [15],

[1]The schemes are referred to as "provably secure" because they are mathematically proven secure against oracle-guided attacks.




a) AntiSAT with $t = 0$ $SPS_R = 1 - ADS_G \approx 0$

b) AntiSAT with $t = 2$ $SPS_R = 1 - ADS_G \approx 0.07$

c) AntiSAT with $t = \frac{k_l}{2} = 4$ $SPS_R = 1 - ADS_G = 0.8$

**FIGURE 1. Resilience of Anti-SAT [14], [16] against the SPS attack [17].**

it becomes resilient to structural and partitioning-based attacks. Similarly, the work in [16] demonstrated how replacing the protection logic with a diversified tree logic (DTL), having a controlled number ($t$) of perturbations within the protection logic's AND-tree structure, can withstand removal attacks such as the signal probability skew (SPS) attack [17]. Hence, although removal attacks are powerful against basic PSLL techniques, they fail to account for the advanced and recent implementations.

The SPS attack [17] identifies the output signal $Y$ of the Anti-SAT block as shown in Figure 1(a). The authors in [17] show that the absolute difference of the probability skew (ADS) of the inputs of a gate is maximum for gate $G$ ($ADS_G \approx 1$) that generates the output of the Anti-SAT block. An attacker computes the signal skew for all the gates in the locked design and looks for the highest ADS gate, identifying $G$. Hence, the resilience of Anti-SAT against the SPS attack ($SPS_R$) is determined as follows.

$$SPS_R = 1 - ADS_G. \tag{1}$$

The naïve implementation of Anti-SAT is vulnerable to the SPS attack [17]. By replacing some of the AND gates with OR gates in the tree structure, the resilience against SPS increases as shown in Figures 1(b) and 1(c). *Hence, a simple gate replacement can thwart the SPS attack.*

*Locking Technique-Specific Attacks.* A PSLL technique can be implemented using different perturb and restore blocks, and various approaches exist for attacking them. However, the attacks are technique-specific and are not scalable to other variants, as highlighted in Table 2. For example, the functional analysis (FALL) attacks can break certain implementations of stripped functionality logic locking (SFLL-HD) [6] and tenacious and traceless logic locking (TTLock) [6] but cannot break other PSLL techniques [18].

*Restricted Parameter Settings.*

SFLL-HD$^{hd}$ [6] corrupts the output of the locked design for all input patterns from a Hamming distance (HD) $hd$ from the secret key $K^*$. The FALL attacks [13] on SFLL-HD$^{hd}$ retrieve the secret key *without* requiring an oracle [13].

**TABLE 2. Comparison of the state-of-the-art oracle-less attacks on PSLL techniques.**

| Attack \ Defense | SARLock [18] | SARLock-DTL [16] | Anti-SAT [14] | Entangled Anti-SAT [19] | Anti-SAT-DTL [16] | TTLock [6] | SFLL-HD [6] | CAC [16] | CAC-DTL [16] |
|---|---|---|---|---|---|---|---|---|---|
| SPS [17] | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| RE-based [11] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| SFLL-HD-Unlocked [12] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| FALL [13] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| GNNUnlock [1] | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| GNNUnlock+ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Sirone and Subramanyan *et al.* [13] developed three attack algorithms to deal with specific ranges of *hd* values. The algorithms are based on specific functional properties of the protection logic that appear under the specific range of *hd*. The *SlidingWindow* algorithm does not have any constraints on the *hd* value, but it does not fare well since it requires computationally hard SAT calls. The *SFLL-HD-Unlocked* attack is another unlocking scheme for SFLL-HD that relies on Gaussian elimination to recover the secret key *without* requiring an oracle [12]. This attack is not applicable when $hd \leq 4$ due to the composition of singular matrices that cannot be solved using Gaussian elimination. We summarize the applicable ranges of *hd* for the different attacks on SFLL-HD in Table 3.

To demonstrate the limitations of current attacks on SFLL-HD, we lock a set of ISCAS-85 and ITC-99 benchmarks using SFLL-HD with $K/hd = 2$, where $K$ is the key-size, generating 192 locked designs. The locking ratio of $K/hd = 2$ is critical as it achieves the highest resilience to removal attacks, as indicated in [6]. We launched the FALL attacks [13] and the SFLL-HD-Unlocked attack [12] on these locked benchmarks and observed that the attacks report 0 keys, indicating that they cannot circumvent the security guarantees of these designs. *This experimental analysis highlights the need for a global attack strategy that can break different PSLL techniques under all possible scenarios*.

*Restricted Design Formats.* Finally, the state-of-the-art attacks on PSLL techniques do not apply to industrial formats (Verilog files), as indicated in Table 3.

### B. ASSOCIATED RESEARCH CHALLENGES

The above discussion (Section I.A) raises an important question: *How does an attacker identify the protection logic of the PSLL technique without focusing on a specific structure?* Formulating a global attack strategy on PSLL techniques is an open research problem that poses the following research challenges:

**TABLE 3. Restrictions of the oracle-less attacks on SFLL-HD.**

| Attacks on SFLL-HD | | Applicable Range of $hd$ | Type of Design |
|---|---|---|---|
| RE-based [11] | | Any value | 2-input gates |
| FALL Attacks | SlidingWindow | Any value | Topologically sorted AND-OR-INVERT gates |
| | AnalyzeUnateness | $hd = 0$ | |
| | Hamming2D | $hd \leq K/4$ | |
| SFLL-HD-Unlocked [12] | | $hd > 4$ | BENCH format |
| **GNNUnlock [1] / GNNUnlock+** | | **Any value** | **Any type** |

1) *Recognizing all implementation variants:* The protection logic can have a non-trivial internal topology, which depends on (i) the secret key-value, (ii) the *HD* value (e.g., SFLL-HD), (iii) the choice of the logic blocks forming the protection logic (e.g., Anti-SAT), (iv) the choice of $t$ in DTL-based locking, and (v) the key-size. Hence, different settings will result in different topologies. Thus, utilizing an exact matching algorithm would be a naïve approach.
2) *Handling different synthesis settings and design formats:* The structure of the protection logic and its integration with the original design depends on the synthesis procedures and the technology library. The attack model should be able to understand and process different design formats.
3) *Bypassing targeted removal attack protection:* State-of-the-art PSLL techniques hinder removal attacks. Developing an attack that tackles such advanced protection implementation is required.
4) *An Insight into graph-based learning methods:* Machine learning (ML) approaches are promising candidates for the detection of protection logic. In particular, graph-based learning approaches, such as graph neural networks (GNNs), are interesting as they directly operate on the graph structure of the locked design. However, GNNs have not been used before in the context of attacking PSLL techniques. More insight into the different GNN models, corresponding aggregation functions, and various design options for identifying the protection logic, is required.

### C. OUR NOVEL CONCEPT AND CONTRIBUTIONS

To address the challenges mentioned above (Section I.B), we propose GNNUnlock+, an oracle-less framework based on graph learning, for attacking PSLL techniques. *We formulate the problem of locating the protection logic in the locked design as a node classification problem and solve it using a GNN, as illustrated in Figure 2.* We represent the circuit/design as a graph, where the nodes represent the logic gates and the edges represent the wires. The GNN propagates messages between the nodes in such a way that each node becomes aware of its neighborhood and the global graph structure. Finally, we perform node classification to determine whether a node in the locked design is a part of the protection logic or not. *Hence, attacking a PSLL technique*

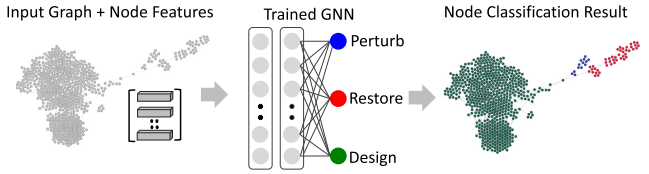



**FIGURE 2. Breaking PSLL using GNNs.**

*becomes a classification problem for all the gates in the locked design.*

The initial version of GNNUnlock+ (a.k.a GNNUnlock) [1] was able to handle different variants of PSLL techniques. However, the post-processing method proposed in [1] was limited to Anti-SAT, TTLock, and SFLL-HD. Providing more insight into the development of GNNUnlock is essential to ensure that the community can find the best fit model for unlocking any desired PSLL technique. In this extension, we explain all the pieces that come together to highlight the performance of GNNUnlock+ in circumventing additional PSLL techniques. Figure 3 shows an overview of our methodology and contributions, which are also enumerated below.

1) We develop a framework for the *netlist-to-graph transformation (Section IV.B)* which efficiently extracts features to capture the functionality of each gate and connectivity in the gate-level netlist (in any format). We present a detailed comparative study of different features for the detection of the protection logic.
2) We facilitate *GNN learning on locked designs (Section IV.C)*, which helps in identifying the structural features of the nodes in the targeted protection logic. Additionally, we present a detailed analysis of the steps followed to optimize the GNN architecture.
3) We propose a *post-processing rectification procedure (Section IV.D)* guided by the properties of the protection logic, connectivity of the design, and the predictions of the GNN (on the design-under-attack) to limit any misclassifications.
4) Finally, in order to not restrict the application of GNNUnlock+ to current PSLL implementations, we explain the required steps to optimize GNNUnlock+ to future PSLL techniques where our post-processing might be outdated/obsolete (Section IV.C). With the continuous evolution of attacks on PSLL techniques, many defense strategies are being developed, to which our GNNUnlock+ model must adapt. Rather than deploying GNNUnlock+ merely on current PSLL techniques, we give insight into retraining GNNUnlock+ on new datasets that reflect new PSLL techniques.

We showcase the effectiveness of GNNUnlock+ by breaking 1,224 benchmarks locked using 9 different PSLL techniques—SFLL-HD [6], TTLock [6], Anti-SAT [14], SAT attack resistant logic locking (SARLock) [18], Anti-SAT with diversified tree logic (Anti-SAT-DTL) [16], Anti-SAT with wire entanglement [15], [19], SARLock-DTL [16], corrupt and correct (CAC) [16] and *CAC-DTL* [16]. *To the best of our knowledge, no oracle-less attack has been successful against DTL-based PSLL techniques other than GNNUnlock+.We make our attack available online [20].*

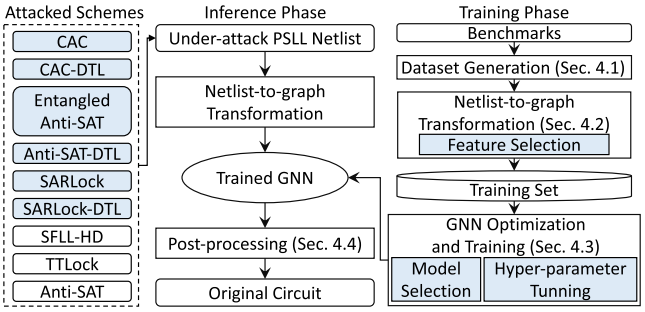



**FIGURE 3. Overview of our novel contributions. The highlighted box represents the contributions presented in this version of the work.**

The remainder of the paper is organized as follows. Section 2 explains the threat model while Section III presents the background information and a detailed discussion on the most relevant work. In Section IV, the concept and implementation details of our proposed GNNUnlock+ attack are presented. Section V details the experimental setup and discusses the results. Finally, Section VI presents concluding remarks.

## II. ATTACKER MODEL

We assume that the attacker is present in the untrusted foundry with access to the GDSII mask information and reverse-engineering tools that facilitate the gate-level netlist extraction. Besides, he/she is aware of the locking algorithm and can distinguish between the regular primary inputs (PIs) and key-inputs (KIs). He/she also knows the usage of the technology library and the usage of different synthesis settings. Note that we do not assume the attacker to have access to an unlocked chip (i.e., an *oracle-less* setting). For SFLL-HD$^{hd}$ [6], the attacker knows the *hd* value, while the type of logic function is known for the case of Anti-SAT [14].

## III. BACKGROUND AND RELATED WORKS

In this section, we present the necessary background information about PSLL techniques and GNNs. The common notations and abbreviations used are highlighted in Table 4.

### A. PROVABLY SECURE LOGIC LOCKING (PSLL)

In Anti-SAT [14], the outputs of two complementary logic functions ($gl1$ and $\overline{gl2}$), are fed into an AND gate, as shown in Figure 4(a). When the correct key is in place, the output of the AND gate $Y$ is 0. For an incorrect key, the output signal $Y$ could be 0 or 1 depending on the input $X$. The output signal $Y$ is XORed with an internal net from the original design. Having the $Y$ signal highly skewed to 0 ensures resilience against the SAT-based attack [7]. Nevertheless, Anti-SAT is vulnerable to removal attacks [17].

To mitigate removal attacks, authors in [19] obfuscate the structural characteristics of the Anti-SAT block using wire entanglement. They use a MUX-based interconnect network

**TABLE 4. Commonly used abbreviations and notations.**

| Term | Definition | Term | Definition |
|---|---|---|---|
| ADS | Absolute difference of probability skew | $\sigma(.)$ | A non-linear activation function |
| CAC | Corrupt and correct | $n$ | # of nodes in the netlist |
| DIP | Distinguishing input pattern | $\boldsymbol{X}$ | Set of feature vectors |
| DTL | Diversified tree logic | $h$ | Hidden dimension |
| FALL | Functional analysis attacks | $\mathcal{G}$ | A graph |
| GCN | Graph convolutional network | $X$ | Protected inputs |
| GNN | Graph neural network | $\boldsymbol{\nu}$ | Set of nodes in $\mathcal{G}$ |
| $hd$ | Hamming distance | $\boldsymbol{\varepsilon}$ | Set of edges |
| KIs | Key-inputs | $m$ | Length of a feature vector |
| MLP | Multi-layer perceptron | $\boldsymbol{Z}$ | Set of final embeddings |
| PIs | Primary inputs | $s$ | Length of an embedding |
| POs | Primary outputs | $\boldsymbol{z}_u \in \mathbb{R}^s$ | Final embedding of node $u$ |
| PSLL | Provably secure logic locking | $L$ | # GNN aggregation layers |
| SARLock | SAT attack-resilient logic locking | $\boldsymbol{x}_u \in \mathbb{R}^m$ | Feature vector of node $u$ |
| SAT | Boolean satisfiability | $Adj$ | Adjacency matrix of $\mathcal{G}$ |
| SFLL | Stripped functionality logic locking | $N(u)$ | The neighbors of a node $u$ |
| SPS | Signal probability skew | $\boldsymbol{h}_u^l$ | Embedding of $u$ in the $l^{th}$ layer |
| TTLock | Tenacious and traceless logic locking | $\boldsymbol{W}^l$ | Layer-wise trainable weight matrix |
| $K$ | Key-size | $t$ | # of AND gates replaced in DTL |
| $SPS_R$ | Resilience against the SPS attack | $K^*$ | The secret key |
| IN | In-degree | OUT | Out-degree |

to increase the connectivity between the Anti-SAT block and the original design. An example of wire entanglement using 2-input MUXes is shown in Figure 4(b). Each MUX takes an input wire from the original design and an input wire from the Anti-SAT block, obfuscating the interconnect structure.

In SARLock [18], a comparator unit and a set of XOR gates (*mask*) are utilized to have at most one incorrect key-value corrupting the output for an input combination, as shown in Figure 4(c). As a result, the SAT-based attack can prune out a single key-value with each attack iteration, requiring an exponential number of attack iterations.

TTLock and SFLL-HD [6] were developed to achieve resilience against removal attacks (Figure 4(d)). These techniques protect particular input patterns specified by the secret key. The output of the design is corrupted by a *perturb* unit for the protected input patterns. A key-controlled *restore* unit restores the value of the corrupted net when the correct key is applied. TTLock protects a single input pattern that is the same as the locking key. Tracing the KIs helps identify the *restore* unit of TTLock, which is a comparator. However, the structure of the *perturb* unit depends on the selected secret key, and it is not controlled by the external KIs, as demonstrated in Figure 4(e). SFLL-HD$^{hd}$ protects all input patterns that are an *hd* away from the secret key. TTLock is equivalent to SFLL-HD$^0$. For SFLL-HD$^{hd>0}$, the *restore* and *perturb* units are HD checkers (*HDC*), as illustrated in Figure 4 (f). The structure of the *perturb* unit depends on the hard-coded key, making it difficult to be traced.

*Diversified Tree Logic (DTL) [16].* The protection logic added in Anti-SAT, SARLock, and SFLL-HD can be implemented using a layer of XOR gates feeding an AND-tree. The AND-tree achieves SAT resilience as it has an onset size of 1. In [16], DTL is used instead of the typical AND-tree, replacing a number $t$ of AND gates with OR/NAND/XOR gates which provides a tunable resilience between removal attacks and SAT-based attacks. Figure 4 shows how DTL can be integrated with PSLL techniques such as Anti-SAT (Figure 4(g)) and SARLock (Figure 4(h)). DTL can be integrated with CAC locking technique, as shown in Figure 4(i). A DTL block with the hard-coded secret key $K^*$ corrupts the functionality of the original design (*perturb*). Having the *perturb* block implemented using the diversified structure gives it a higher chance of blending in with the original design [16]. The output is then corrected using the *restore* logic implemented using two DTL blocks and an OR gate. Replacing some of the gates in the DTL structure hinders removal attacks (see Figure 1). Without loss of generality, we consider the replacement of AND gates with OR gates.

### B. GRAPH NEURAL NETWORKS (GNNS)

GNNs [21] are a class of deep learning models used to operate on graph-structured data. GNNs learn how to aggregate information across the graph to compute node embeddings so that similar nodes in the context of the learning problem are close to each other in the embedding space, facilitating subsequent classification. The embeddings for a node captures its input features, neighborhood, and graph structure. GNN variants use different aggregators to collect information from each node's neighborhood and different updaters to update each node's hidden states. In inductive learning settings, the same aggregation parameters are shared for all nodes. Once the model is trained, those parameters can be used to generate embeddings on entirely unseen graphs. We

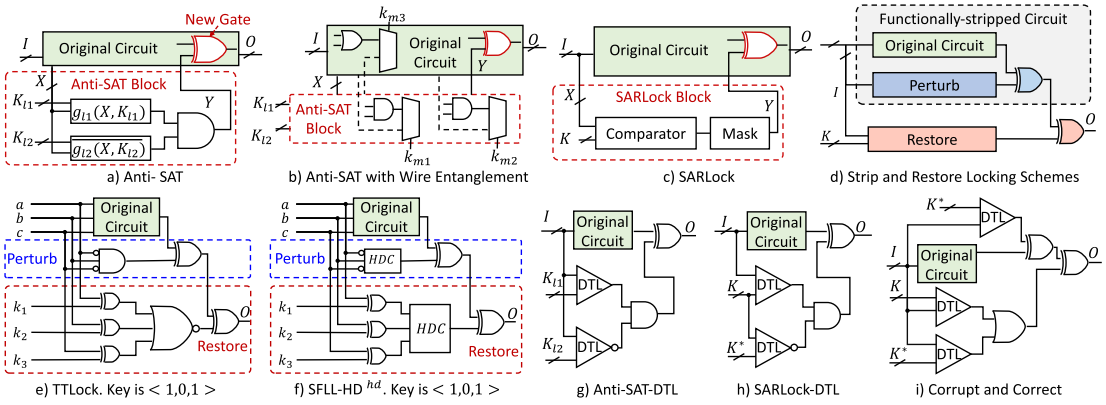



**FIGURE 4. Different PSLL techniques. Anti-SAT, entangled Anti-SAT [14], [19], SARLock [18], TTLock [6], SFLL-HD [6], and DTL [16].**

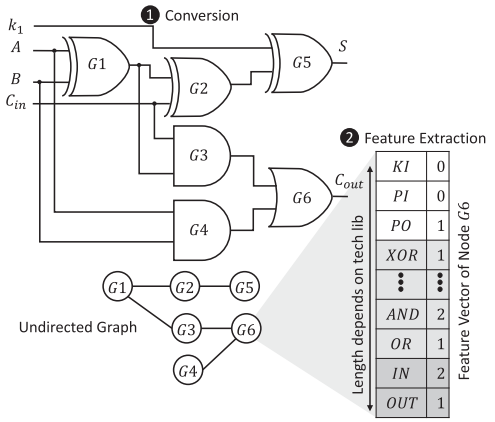



**FIGURE 5. Proposed netlist-to-graph transformation and feature extraction.**

follow such an inductive approach in this work since we train on a set of locked benchmarks and then *test the model on unseen benchmarks locked with unknown key-values.* We provide further details about the GNN topology in Section IV.C.

## IV. PROPOSED GNNUNLOCK+ FRAMEWORK

Figure 3 shows an overview of the main steps of GNNUnlock+ and we discuss each of them in detail below.

### A. DATASET GENERATION

By knowing the underlying locking technique, we generate a set of locked benchmarks for training and validation. Each benchmark is locked several times with randomly generated key-values and different key-sizes $K$, which results in an extensive training dataset. The training process is not aware of the correct key-values. However, different key-values cause the generation of varying *perturb* and *restore* structures, and hence variants are accounted for during training. GNNUnlock+ transforms locked benchmarks into graphs (see Section IV.B) and labels the nodes. For the case of SFLL-HD$^{hd}$, TTLock, and CAC, a node belongs to the *perturb*, the *restore*, or the *design* logic. For the case of Anti-SAT and SARLock, a node either belongs to the Anti-SAT/SARLock block or the *design* logic.

GNNUnlock+ attacks each design independently by excluding its corresponding graphs from training/validation. For example, when attacking `b17_C` from ITC-99, only the graphs of `b14_C`, `b15_C`, `b20_C`, and `b21_C` are used in training, while the graphs of `b22_C` are used for validation to evaluate the model on unseen data and avoid biasing.

### B. NETLIST-TO-GRAPH TRANSFORMATION

We model the connectivity between the gates in a design using an undirected graph $\mathcal{G}(\boldsymbol{\nu}, \varepsilon, \boldsymbol{X})$. Set $\boldsymbol{\nu}$ represents the logic gates (nodes), while set $\varepsilon$ represents the wires (edges). We illustrate an example of a logic locked netlist and the corresponding graph in Figure 5. The set $\boldsymbol{\nu}$ does not contain PIs, KIs, or primary outputs (POs) of the design. Each node is

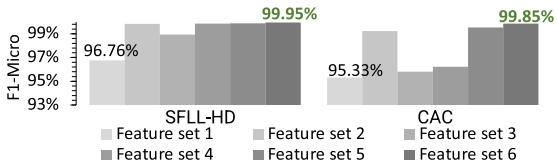



**FIGURE 6. Classification performance using different feature combinations.**

**TABLE 5. The feature set taxonomy.**

| Feature Set | Root Gate | 1-hop Gates | 2-hop Gates | PO-C | PI-C | KI-C | IN | OUT | Length $m$ |
|---|---|---|---|---|---|---|---|---|---|
| Feature set 1 | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | $q$ |
| Feature set 2 | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | $q+3$ |
| Feature set 3 | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | $q+3+2$ |
| Feature set 4 | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | $q$ |
| Feature set 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | $q+3$ |
| Feature set 6 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | $q+3+2$ |

*✓(×) means the feature vector includes (does not include) the attribute. PO-C, PI-C, KI-C, IN, and OUT stands for primary output connectivity, primary input connectivity, key-input connectivity, in-degree, and out-degree, respectively. $q$ represents the number of unique logic gates in the technology library*

associated with a feature vector that gets incorporated into the learning process. $\boldsymbol{X}$ is the set of feature vectors $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}, \boldsymbol{x}_u \in \mathbb{R}^m$, where $n$ indicates the total number of nodes in the netlist and $m$ represents the length of the feature vector $|\boldsymbol{x}_u| = m$. Each locked design is represented by a graph instance with a corresponding adjacency matrix *Adj*. To feed multiple graphs (of different sizes) to the GNN, a block-diagonal matrix is created for each dataset. Each block represents the *Adj* of one locked design.

We investigate and explore different feature vectors and their effect on classification. The extracted features must aid in identifying the protection logic and be known to the attacker. For example, the type of the Boolean gate of a node is a known feature and will complete the graph's representation of the corresponding netlist. The direct connectivity of a gate to a port can also help in identifying the protection logic. Finally, the in-degree (IN) and out-degree (OUT) of the nodes can help in capturing the connectivity of the neighborhood. We study different combinations of these features to select the best features for classification. We show the classification results (using F1-Micro)[2] on the ISCAS-85 benchmark `c2670` locked using the SFLL-HD (with $hd = 2$) and using the CAC technique (with $t = 0$) for different feature sets in Figure 6.[3]

Table 5 summarizes the feature set taxonomy. The feature set 1 captures the type and number of gates appearing in the one-hop neighborhood of the node. The length of the vector $|\boldsymbol{x}| = m$ depends on the number of Boolean gates in the target library ($q$). If a node is connected to two XOR gates, then 2

[2]The F1-score is the harmonic mean of the precision and recall. Hence, the F1-score considers the false positives and the false negatives, which is helpful to assess the performance of a classifier for uneven class distribution. The F1-Macro is the arithmetic mean of the per-class F1-scores. The F1-Micro considers all samples at once and measures the proportion of correctly classified samples out of all the samples.
[3]Details on the experimental setup can be found in Section V.

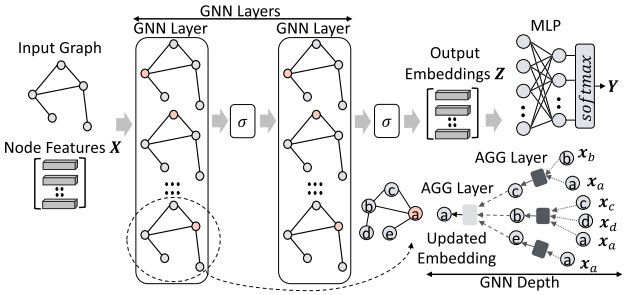



**FIGURE 7. Illustration of a GNN [21]. The goal is to obtain an embedding for each node capturing its features and the topological information of the graph. Neighborhood aggregation is performed in each GNN layer. In each aggregation layer, the nodes aggregate information from their neighbors and update their states with aggregation, update, and activation functions. The number of aggregation layers depends on the chosen neighborhood size (depth). Here, the depth is 2.**

will be placed in the vector's entry corresponding to XOR. This feature set achieves the lowest score as indicated in Figure 6. The feature set 2 additionally includes information on whether a node is connected to a PI, a PO, or a KI. The total length of feature set 2 is $m = q + 3$. If a gate is connected to PI, then 1 will be placed in the entry corresponding to PI. Including the *connectivity-to-ports* information eases the identification of the protection logic as it is typically connected to KIs and PIs. Furthermore, the feature set 3 adds the structural information for IN and OUT. The length of feature set 3 is $m = q + 3 + 2$. If a gate drives two other gates in the netlist, then 2 will be placed in the entry corresponding to OUT. Expanding the search space further, we test using feature sets 4, 5, and 6, which are similar to sets 1, 2, and 3 but consider all gates appearing in the two-hops neighborhood of a node instead of the one-hop neighborhood. As can be observed from Figure 6, feature set 6 results in the best classification performance and thus is used in all our experiments. An example of such a feature vector can be seen in Figure 5. The feature vector extracted for gate *G*6 shows that the gate is connected to a PO with $IN = 2$ and $OUT = 1$. It also shows that 1 XOR gate, 2 AND gates, and 1 OR gate exist in the node's 2-hop neighborhood. The feature vectors are standardized by removing the mean and scaling to unit variance.

### C. GNN TOPOLOGY

Throughout this paper, we use bold lowercase characters to denote vectors and bold uppercase characters to denote matrices. The commonly used notations are listed in Table 4. In GNNs, nodes aggregate information from their neighbors using neural networks (see Figure 7). Each node $u \in \nu$ gathers information from its neighbors $N(u)$, where each one of these neighbors collects information from their neighbors. The aggregator functions in *GraphSAGE* [21] learn to aggregate feature information to obtain the set $\boldsymbol{Z}$ of embeddings $\{z_1, ..., z_n\}, z_u \in \mathbb{R}^s$. The embeddings of length $s$ are then used to perform the desired task, such as node classification. $L$ aggregation layers are constructed for each node to gather information from the $L$-hop neighborhood (depth).[4] The aggregation is performed as follows.

$$\boldsymbol{h}_u^0 = \boldsymbol{x}_u \tag{2}$$

$$\boldsymbol{h}_u^l = \sigma(\boldsymbol{W}^l \cdot \| \, (\boldsymbol{h}_u^{l-1}, AGG(\{\boldsymbol{h}_v^{l-1}, \forall v \in N(u)\}))) \tag{3}$$

$$AGG = \sum_{v \in N(u)} \frac{\boldsymbol{h}_v^{l-1}}{|N(u)|}, \tag{4}$$

where $\boldsymbol{h}_u^l$ represents the embedding of node $u$ in the $l^{th}$ aggregation layer. The initial embedding $\boldsymbol{h}_u^0$ is equal to the node's features $\boldsymbol{x}_u$. The previous state of each node $\boldsymbol{h}_u^{l-1}$ is concatenated ($\|$ is the concatenation operation) with its neighbors' current state to update its embedding $\boldsymbol{h}_u^l$. $\sigma(.)$ denotes a non-linear activation function such as rectified linear unit (`ReLU`) and $\boldsymbol{W}^l$ is a layer-wise trainable weight matrix, which is shared by all the nodes. In our work, we use the mean aggregator function as described in Equation (3). After $L$ layers of aggregation, we obtain the final embedding of the node as follows: $z_u = \boldsymbol{h}_u^L$.

In GNNs, dropout is applied before the non-linear activation of the aggregation function. The final embeddings are then passed on to a simple multi-layer perceptron (MLP) to classify the nodes into $C$ classes. The `softmax` activation function is used in the final layer to get probabilities of a node being in a particular class (classification).

$$\boldsymbol{X}^{MLP} = \sigma(\boldsymbol{Z} \cdot \boldsymbol{W}^{MLP}) \tag{5}$$

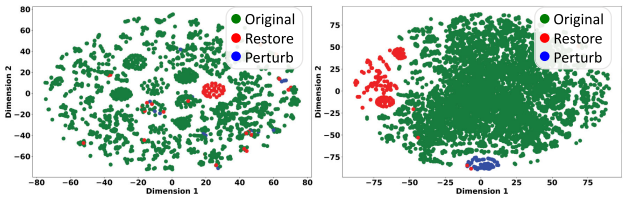
$$\boldsymbol{Y}^{MLP} = softmax(\boldsymbol{X}^{MLP}) \tag{6}$$

$$softmax(\boldsymbol{y})_c = \frac{e^{y_c}}{\sum_{d=1}^{c} e^{y_d}}, \tag{7}$$

$y_c$ refers to each element in the vector of predictions made by the model $\boldsymbol{y}$. The class with the highest probability value is considered the output of the prediction.

Due to neighborhood aggregation, the deeper the GNN model is, the more multi-hop neighbors get incorporated for the root node's computation. Hence, training time can grow exponentially with GNN depth [22]. Several methods, including GraphSAGE, perform layer sampling to ensure a limited set of neighbors is considered for a node in the next layer. Such sampling speeds up training but suffers from scalability issues. We leverage the graph sampling method *GraphSAINT* [22] to construct mini-batches for training. Rather than building a GNN on a full training graph and then performing layer sampling, GraphSAINT samples the training graph $\mathcal{G}$ and then builds a GNN on the sampled graph $\mathcal{G}_s$ for each mini-batch, where $|\nu_s| \ll |\nu|$, ensuring scalability

[4]The $L$-hop neighborhood of a node $u \in \nu$ is the set of nodes at a distance less than or equal to $L$ from $u$.




a) Normalized input features for ISCAS-85 benchmark circuits

b) GNN output embeddings for ISCAS-85 benchmark circuits

**FIGURE 8. GNNUnlock+ embeddings for SFLL-HD locked benchmarks.**

concerning graph size and GNN depth. We use their random walk-based sampler in our experiments.[5] More details on the sampler can be found in [22].

### 1) WHY GRAPH NEURAL NETWORKS

The GNN maps the nodes to a new embedding space in which similarity between the nodes indicates similarity between the gates in the design. Identifying the nodes belonging to the protection logic in a design is challenging for ML algorithms that assume samples are independent of each other. To visualize the superiority of GNNs in identifying the protection logic in PSLL techniques, we lock several ISCAS-85 benchmarks using SFLL-HD having $hd = 2$. The initial (normalized) node features and the final GNN node embeddings for the locked benchmarks are projected non-linearly to 2D using t-distributed stochastic neighbor embedding (`t-SNE`). It can be observed from Figure 8 that performing the classification on the initial feature vectors is a challenging task due to the high overlap between the classes (mainly the *perturb* and the *design* nodes). However, after the GNN aggregation, the classes become highly separable and the classification task becomes easier.

### 2) GNN OPTIMIZATION

Here, we discuss the design steps required to find the topology and hyperparameter settings for which the GNN can optimally identify the protection logic. Primarily, in GNNUnlock+, for the first time, we present the possibility of having a flexible attack platform that can be tailored to any desired PSLL technique, giving the attacker more control and freedom. We show the optimization steps followed and the corresponding prediction outputs on the ISCAS-85 benchmark `c7552` locked using the CAC technique ($t = 0$) as a representative example.

The first step is to initialize a single-layer GNN having a depth of $L = 1$. This means that each node will update its embedding based on the information gathered from its direct neighborhood. We initially select the GraphSAGE architecture [21], with the concatenation aggregation and the `ReLU` activation function. We use GraphSAINT [22] to perform graph sampling during training. We use the random walk sampler having 3000 walkers with a walk length of 2. The training runs for 2000 epochs and we use the model with the best performance on the validation set to evaluate the test set accuracy. We utilize Adam optimizer and perform a grid search on the hyperparameter space defined by a hidden dimension: $\{128, 256, 512\}$, dropout: $\{0.0, 0.1, 0.2, 0.3\}$ and initial learning rate: $\{0.1, 0.01, 0.001, 0.0001\}$, as in [22]. We present the classification accuracy results as heat-maps in Figure 9. We observe that the best performance is obtained when the hidden dimension is set to 128 with a dropout of 0.1 and learning rate of 0.01, reaching a classification accuracy of 99.96%.

Next, we investigate the effect of varying the number of GNN layers and the corresponding depth $L$. We vary the number of layers from 1 to 3 and the corresponding depth from 1 to 4 using a step size of 1. We illustrate the classification results in terms of accuracy, F1-Micro, and F1-Macro scores as heat-maps in Figure 10. We also present the corresponding training time in Figure 11. We observe that the performance of the GNN degrades with the increase in depth and the number of layers. This phenomenon is commonly referred to as the *over-smoothing* problem [23]. Increasing the GNN depth and the number of layers causes all the node embeddings to converge to the same value *reducing the separability between the classes.* Next, we explain the over-smoothing problem via the notion of the *receptive field*. We define the receptive field for a node in a GNN as the set of nodes determining its embedding. In GNNs, each node has a receptive field of $L$-neighborhood. Increasing the GNN depth will cause a growing overlap between the receptive fields of different nodes. As a result, nodes with highly overlapped receptive fields will have highly similar embeddings. Moreover, increasing the depth and the number of layers increases the training time, as shown in Figure 11. Therefore, the depth and the number of layers should not be large. A 2-layer GNN having a depth of $L = 1$ is selected as optimal for the dataset under-evaluation.

After evaluating different hyperparameter settings on the GraphSAGE baseline, we study the effect of different aggregator, updater, and activation functions on the performance of the GNN. For the updater, we switch between concatenation and averaging when combining the embedding of the root node with the embeddings of its neighborhood. Converting the concatenation operation in GraphSAGE's aggregation function to an averaging operation results in what is called a graph convolutional network (GCN) [24]. Such aggregation is referred to as convolution as it is a linear approximation of a localized spectral convolution. For the activation function, we switch between the `ReLU`, leaky rectified linear unit (`LeakyReLU`), and the identity activation function ($I$). As for the type of aggregation used, we switch between the following backbone architectures: *GraphSAGE* [21], graph attention network (*GAT*) [25], and gated attention networks (*GAAN*) [26]. *GAT* [25] utilizes a neural network to compute the importance of the edges for

[5]The experimental section in [22] shows that the random edge and random walk-based samplers achieve the highest accuracy.

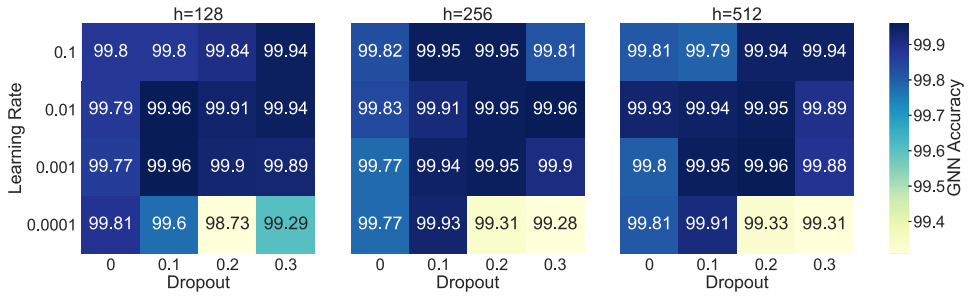



**FIGURE 9.** **Tuning hyper-parameters.** $h$ **represents the hidden dimension.**

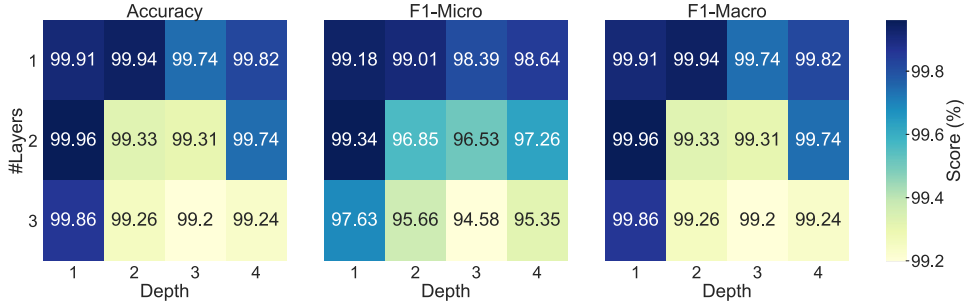



**FIGURE 10.** **Identifying the optimal number of layers and corresponding depth.**

aggregation. GraphSAINT implements multi-head attention of $K$ as follows:

$$\boldsymbol{h}_u^l = \Big\|_{k=1}^{K} \sigma\left(\sum_{v \in N(u)} \alpha_{v,u}^k \boldsymbol{W}^k \boldsymbol{h}_u^{l-1}\right) \tag{8}$$

$$\alpha_{v,u}^k = \boldsymbol{LeakyReLU}((\boldsymbol{a}^k)^\intercal [\boldsymbol{W}^k \boldsymbol{h}_v \parallel \boldsymbol{W}^k \boldsymbol{h}_u]), \tag{9}$$

where $\alpha_{v,u}$ holds the weighting factor (i.e., importance) of node $v$'s state for the root node $u$. Note that $\alpha_{v,u}$ is computed as a byproduct of an attentional mechanism $\boldsymbol{a}$. In *GAAN*, additional gating mechanisms are followed to ensure different values for different heads' calculations. We visualize the results from the third optimization step in Figure 12. In all of the evaluated cases, GraphSAINT sampling combined with GraphSAGE concatenation-based aggregation achieves the best results. For this dataset under-evaluation, the `LeakyReLU` activation function achieves the best results for the chosen GNN configuration. We summarize the GNN configurations for all the datasets in Table 6.

### D. POST-PROCESSING

Although our trained GNN achieves high node classification accuracy on its own (with an average of 99.99%, 99.99%, 99.95%, 99.92% and 99.82%, for the case of Anti-SAT, SARLock, TTLock, SFLL-HD$^2$, and CAC, respectively), we propose a post-processing algorithm to rectify any potential misclassifications to enhance the accuracy further. The algorithm considers predictions made by the GNN, connectivity of the circuit, and known properties of the protection logic, explained further in the following subsections.

#### 1) ANTI-SAT AND SARLOCK

Every node in the Anti-SAT/SARLock block must have at least one KI in its fan-in cone. If a node without KIs in its fan-in cone is predicted as an Anti-SAT/SARLock node, then the prediction will be dropped. For each *predicted design* node, we check its fan-in cone. If there are only other *predicted Anti-SAT/SARLock* nodes in the fan-in cone, then the node initially classified as a *design* node is considered part of the Anti-SAT/SARLock block. The flow of our post-

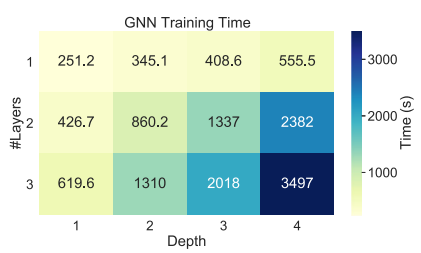



**FIGURE 11.** **The effect of the number of layers and depth on training time.**

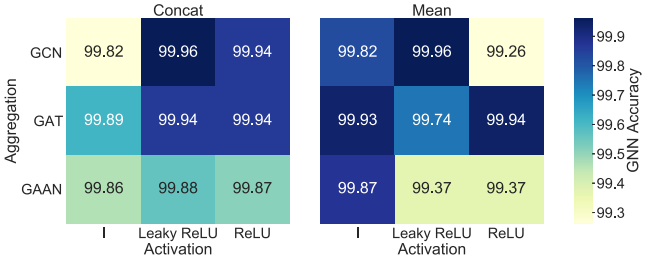



**FIGURE 12.** **Evaluating different aggregate, update, and activation functions.**

**TABLE 6. GNN configuration and sampling details.**

| Dataset | Arch. | | | | | | Training | | GraphSAINT Random Walk Sampling [22] | | | Classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Hidden Dimension | GNN Layers | GNN Depth | Updater | Aggregation | Activation | Learning Rate | Dropout | Epochs | Roots | Walk Length | |
| Anti-SAT Datasets | 512 | 2 | 2 | Concat. | GraphSAGE Mean [21] | ReLU | 0.01 | 0.1 | 500 | 3000 | 2 | Softmax |
| SARLock Datasets | 512 | 2 | 2 | | | ReLU | 0.01 | 0.1 | | | | |
| SFLL-HD Datasets | 512 | 2 | 2 | | | ReLU | 0.01 | 0.1 | | | | |
| CAC ISCAS Datasets | 128 | 2 | 1 | | | LeakyReLU | 0.01 | 0.1 | | | | |
| CAC ITC Datasets | 256 | 2 | 2 | | | ReLU | 0.1 | 0 | | | | |

processing algorithm for Anti-SAT and SARLock is shown in Figure 13(a).

#### 2) TTLOCK, SFLL-HD$^{hd}$ AND CAC

We depict our post-processing algorithm for the case of SFLL-HD, TTLock, and CAC techniques in Figure 13(b). The *predicted restore* nodes are visited to identify possible protected inputs (set $X$) and the scheme's protected output. The following properties then guide the post-processing procedure. (i) All *restore* nodes have KIs in their fan-in cone. (ii) All *perturb* nodes have connections with the *restore* nodes in the netlist and are controlled solely by protected inputs. If a *predicted restore* node has KIs in its fan-in cone, then the prediction is confirmed. Otherwise, the algorithm checks if the node is a *perturb* node. A *predicted perturb* node is verified if it has a connection to other *predicted restore* nodes in the netlist and if the node has $X$ (and no other PIs) in its fan-in cone. In order to not misclassify *perturb* nodes as *design* nodes, the algorithm checks the fan-in cone of the predicted *design* nodes. If a predicted *design* node has only the $X$ and other *predicted perturb* nodes in its fan-in, it is a *perturb* node.

## V. EXPERIMENTAL INVESTIGATIONS

### A. EVALUATION SETUP AND TOOL FLOW

We evaluate GNNUnlock+ on selected benchmarks from the ISCAS-85 and ITC-99 suite. We implement the Netlist-to-graph transformation in Perl/Python3. We use Tensorflow with Python3 implementation of GraphSAINT for GNN training [22]. We perform training on a single node with 24 cores (2x Intel Xeon CPUs E5-2695 v2@2.4 GHz), 256 GB RAM, and one NVIDIA Tesla K20m GPU (2,496 CUDA cores and 5 GB of GDDR5 memory). Moreover, we list all the characteristics of the datasets in Table 7. We summarize the experimental setup in Figure 14.

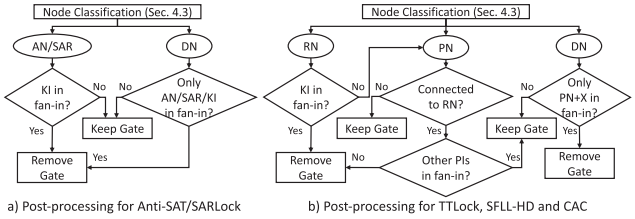



**FIGURE 13. Post-processing flow. Anti-SAT, SARLock, restore, perturb, and design nodes are indicated by AN, SAR, RN, PN, and DN, respectively.**

#### 1) DATASET GENERATION FOR ANTI-SAT AND SARLOCK

The benchmarks (in *BENCH* format) were locked using Anti-SAT and SARLock. We implement Anti-SAT and SARLock using Perl, as described in [14], [18]. Each ISCAS-85 benchmark is locked twice, using each technique, with $K : \{8, 16, 32, 64\}$.[6] In total, we generate 30 Anti-SAT and 30 SARLock ISCAS-85 benchmarks. $K = 8$ is used to evaluate the effectiveness of GNNUnlock+ in isolating the protection block when its size is minimal compared to that of the original design. We lock each ITC-99 benchmark twice, using each technique, with $K : \{32, 64, 128\}$ resulting in a total of 36 Anti-SAT and 36 SARLock ITC-99 benchmarks. These locked benchmarks are converted to graphs using the approach explained in Section IV.B. Using the corresponding graphs, we create two labeled datasets for Anti-SAT/SARLock block identification, one for ISCAS-85 benchmarks and one for ITC-99 benchmarks. A feature vector $|\boldsymbol{x_u}| = 13$ is associated with each node $u$. The Anti-SAT and SARLock locking scripts accept designs in *BENCH* format, which includes a restricted set of logic gates (8 gates).[7] Hence, only 8 out of the 13 features are required to represent the neighborhood of each node. The rest of the features represent IN, OUT, and the connectivity to PIs, KIs, or POs.

#### 2) DATASET GENERATION FOR ANTISAT-DTL

We implemented the DTL technique in Perl and applied it to the Anti-SAT technique as described in [16]. We lock the designs in BENCH format. The $t$ parameter, representing the diversity, is the number of AND gates in the original AND-tree structure replaced by OR gates. For the Anti-SAT-DTL datasets, only the first layer of AND-tree logic was diversified with OR gates *since it produces the highest output corruptibility*. Each ISCAS-85 benchmark is locked twice with $K : \{8, 16, 32, 64\}$ for $t : \{4, K_l/2\}$.[6] In total, we generate 30 locked ISCAS-85 benchmarks, for each $t$. Each ITC-99 benchmark is also locked twice with $K : \{32, 64, 128\}$ for $t : \{4, K_l/2\}$, resulting in a total of 36 locked ITC-99 benchmarks for each setting of $t$. Note that the higher the value of the $t$, the higher the resilience against removal attacks such as SPS [17] (see Figure 1).

*Adapting to Randomization.* We perform the replacement of AND gates in the initial datasets in a deterministic manner. We replace the AND gates only in the first AND-tree layer

[6] $K = 64$ is not considered for c3540 due to the limited #PIs.
[7] $\{XOR, XNOR, AND, NAND, OR, NOR, BUF, NOT\}$.

**TABLE 7. Summary of generated datasets.**

| Dataset | Benchmarks | Circuit Format | #Classes | $\|x\|$ | #Circuits |
|---|---|---|---|---|---|
| Anti-SAT | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| Anti-SAT-DTL $t = 4$ | ITC-99 | BENCH | 2 | 13 | 36 |
| | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| Anti-SAT-DTL $t = K_l/2$ | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| Anti-SAT-RND-DTL $t = 4$ | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| Anti-SAT-DTL TR/VA $t = 0$ TE $t = 4$ | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| Entangled Anti-SAT | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| SARLock | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| SARLock-DTL $t = K/2$ | ISCAS-85 | BENCH | 2 | 13 | 30 |
| | ITC-99 | BENCH | 2 | 13 | 36 |
| TTLock | ISCAS-85 | Verilog netlist $65nm$ | 3 | 34 | 45 |
| | ITC-99 | Verilog netlist $65nm$ | 3 | 34 | 54 |
| SFLL-HD$^2$ | ISCAS-85 | Verilog netlist $65nm$ | 3 | 34 | 45 |
| | ITC-99 | Verilog netlist $65nm$ | 3 | 34 | 54 |
| | | Verilog netlist $45nm$ | 3 | 18 | 54 |
| SFLL-HD$^4$ | ITC-99 | Verilog netlist $65nm$ | 3 | 34 | 54 |
| SFLL-HD$^{16}$ | ISCAS-85 | Verilog netlist $65nm$ | 3 | 34 | 48 |
| SFLL-HD$^{32}$ | ITC-99 | Verilog netlist $65nm$ | 3 | 34 | 72 |
| SFLL-HD$^{64}$ | | Verilog netlist $65nm$ | 3 | 34 | 72 |
| CAC | ISCAS-85 | Verilog netlist $45nm$ | 3 | 18 | 45 |
| | ITC-99 | Verilog netlist $45nm$ | 3 | 18 | 54 |
| CAC $t = K/2$ | ISCAS-85 | Verilog netlist $45nm$ | 3 | 18 | 45 |
| | ITC-99 | Verilog netlist $45nm$ | 3 | 18 | 54 |

(closest to the XOR layer as shown in Figure 1). We evaluate the performance of GNNUnlock+ in breaking Anti-SAT-DTL when the AND gates (to be replaced) are chosen randomly from the entire AND-tree structure. Therefore, each ISCAS-85 benchmark is locked twice with $K$: $\{8, 16, 32, 64\}$[6] for $t$: $\{4\}$, where the $t$ AND gates are selected at random. In total, we generate 30 locked ISCAS-85 benchmarks. Each ITC-99 benchmark is also locked twice with $K$: $\{32, 64, 128\}$ for $t$: $\{4\}$, where the selection of the $t$ AND gates is randomized, resulting in a total of 36 locked ITC-99 benchmarks. Note that modifications in the DTL play a role in SAT resilience. The choice of $t$ and selection of $t$ AND gates are based on the trade-off between SAT-attack resilience and SPS attack resilience [16]. We evaluate GNNUnlock+ on these datasets to support our claim that GNNUnlock+ can handle randomness.

*Adapting to Unseen Structures.* To support our claim that GNNUnlock+ can adapt to unseen structures, we also train and validate GNNUnlock+ framework on regular Anti-SAT locked benchmarks ($t = 0$) and then test on Anti-SAT-DTL with perturbations within the AND-tree structure ($t = 4$). Each ISCAS-85 benchmark is locked twice with $K$: $\{8, 16, 32, 64\}$[6] for $t = 0|4$, depending on whether it is used for training/validation or testing. In total, we generate 30 locked ISCAS-85 benchmarks. Each ITC-99 benchmark is locked twice with $K$: $\{32, 64, 128\}$ for $t = 0|4$, depending on whether it is used for training/validation or testing, resulting in a total of 36 locked ITC-99 benchmarks.

### 3) DATASET GENERATION FOR ENTANGLED ANTI-SAT

Anti-SAT with MUX-based wire entanglement is implemented in Perl as described in [15], [19]. Additional $K_m$ key-inputs are required to control the newly added 2-input MUXes. The MUXes are divided between the Anti-SAT and design blocks and the true and false wires are selected randomly. Each ISCAS-85 benchmark is locked twice with $K$: $\{8, 16, 32, 64\}$.[6] $K_m$ is fixed to $K/2$ for all of the cases.

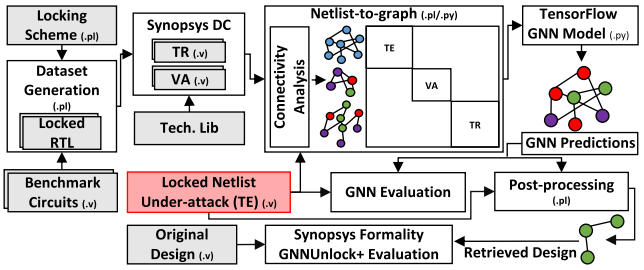



**FIGURE 14. Evaluation setup showing different components and platforms. TE, TR, and VA, stands for testing, training, and validation, respectively.**

Hence, we lock each benchmark with a total key-size of $K + K_m$. In total, we generated 30 entangled Anti-SAT locked ISCAS-85 benchmarks. Each ITC-99 benchmark is locked twice with $K$: $\{32, 64, 128\}$, fixing $K_m = K/2$, resulting in a total of 36 locked ITC-99 benchmarks.

To visualize the effect of wire entanglement on partitioning-based attacks, we lock the ISCAS-85 benchmark `c3540` with Anti-SAT having $K = 32$ and with entangled Anti-SAT having $K = 32$ and $K_m = 32/2 = 16$. The corresponding graphs are shown in Figure 15. *We observe from Figures 15 (a) and 15(b) that the structure of the Anti-SAT block remains similar even when MUXes are in place.* However, the interconnections between the Anti-SAT and the original design are now obfuscated, as observed from Figures 15(c) and 15(d), which thwarts partitioning-based attacks.

### 4) DATASET GENERATION FOR SARLOCK DTL

We integrate the DTL technique with SARLock as described in [16]. Each ISCAS-85 benchmark is locked twice with $K$: $\{8, 16, 32, 64\}$ for $t = K/2$.[6] In total, we generate 30 locked ISCAS-85 benchmarks. Each ITC-99 benchmark is locked twice with $K$: $\{32, 64, 128\}$ for $t = K/2$, resulting in a total of 36 locked ITC-99 benchmarks. Note that only the first AND-tree layer is diversified using OR gates.

### 5) DATASET GENERATION FOR TTLOCK AND SFLL-HD$^{hd}$

We implement SFLL-HD$^{hd}$ in Perl as described in [6] and perform locking at RTL—TTLock is the particular case of SFLL-HD$^{hd}$ when $hd = 0$. We lock each ISCAS-85 benchmark thrice with $K$: $\{8, 16, 32, 64\}$ for $hd$: $\{0, 2\}$.[6] In total, we generate 45 locked ISCAS-85 benchmarks, for each $hd$. Additionally, we also lock each ITC-99 benchmark thrice with $K$: $\{32, 64, 128\}$ for $hd$: $\{0, 2, 4\}$, resulting in a total of 54 locked ITC-99 benchmarks for each setting of $hd$ . We then synthesize these locked RTL files using the standard ASIC design flow for the $65nm$ LPe technology using Synopsys Design Compiler. Since we leverage the complete standard cell library for synthesis, a larger feature vector $|\boldsymbol{x}|$ (than the case of BENCH) is required to capture all the possible logic gates. Thereby, in this case, $|\boldsymbol{x}| = 34$.

We also conduct experiments to verify that GNNUnlock+ can handle different circuit formats and technology libraries. To that end, we use the Nangate $45nm$ open cell library for

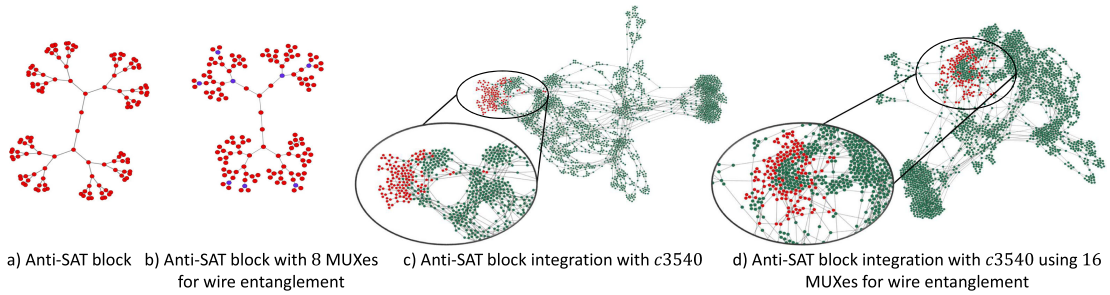



**FIGURE 15. Anti-SAT with wire entanglement [15], [19]. The green, red, and blue nodes indicate the design, restore, and MUX gates, respectively.**

synthesis, when $hd = 2$ for SFLL-HD$^{hd}$. Varying the circuits' format affects $|\boldsymbol{x}|$ only. In this case, $|\boldsymbol{x}| = 18$ is used. We also consider other corner cases for when $hd$ : $\{16, 32, 64\}$ with corresponding $K : \{32, 64, 128\}$ (see Section V.I).

#### 6) DATASET GENERATION FOR CAC AND CAC-DTL

We implement CAC [16] in Perl and lock the benchmarks at RTL. We lock each ISCAS-85 benchmark thrice with $K : \{8, 16, 32, 64\}$ for $t : \{0, K/2\}$.[6] In total, we generate 45 locked ISCAS-85 benchmarks for each $t$. Additionally, we lock each ITC-99 benchmark thrice with $K$ : $\{32, 64, 128\}$ for $t : \{0, K/2\}$, resulting in a total of 54 locked ITC-99 benchmarks for each setting of $t$. We synthesize CAC locked RTL files using Synopsys Design Compiler utilizing the Nangate 45*nm* open cell library, for which $|\boldsymbol{x}| = 18$.

#### 7) EVALUATION METHODS AND METRICS

GNNUnlock+ attacks each design independently by excluding its corresponding graphs from training and validation. For example, when attacking the the ITC-99 benchmark `b17_C` (locked using TTLock), only the graphs of `b14_C`, `b15_C`, `b20_C`, and `b21_C` are included in the training set, resulting in 161,942 training nodes (36 graphs), while the graphs of `b22_C` are included only in the validation set (to evaluate the framework on unseen data), resulting in 74,753 validation nodes (9 graphs). We test GNNUnlock+ on `b17_C`, which results in 110,685 testing nodes (9 graphs).

We compare the predictions made by the GNN with the true labels of the nodes to evaluate the performance of the attack. We report the accuracy and the non-averaged precision, recall, and F1-score for each classifier. We further evaluate GNNUnlock+ by removing the predicted protection logic to retrieve the unlocked design without accessing the true labels (see Section IV.D). Further, we also compare the recovered design with the original design for functional equivalency using Synopsys Formality.

### B. BREAKING ANTI-SAT WITH GNNUNLOCK+

For the ISCAS-85 dataset, the GNN gave 100% node classification accuracy for 25 graphs and an average of 99.98% accuracy across all 30 graphs. For the remaining 5 graphs, only 6 nodes were misclassified and then rectified during post-processing. For example, two *design* nodes feeding the XOR gate connecting the Anti-SAT block with the rest of the circuitry got misclassified as part of the Anti-SAT block. *However, having no KIs in their fan-in, the prediction is ignored.* For the ITC-99 dataset, the GNN gave 100% node classification accuracy for 35 out of 36 tested graphs. Only one node was misclassified in one of the `b21_C` graphs locked with $K = 128$, which was rectified during post-processing. We report detailed results in Table 8.

Having 100% node classification accuracy for a design locked with different key-sizes (such as the case for `c7552`, `b14_C`, etc.) shows that *the ratio of sizes between Anti-SAT block and design does not affect the performance of GNNUnlock+ and overall recovery of the original design*. Obtaining 100% node classification accuracy for Anti-SAT is validated since the Anti-SAT block has a specific structure controlled by external KIs, *making it easy to be learned*.

### C. BREAKING ANTI-SAT-DTL WITH GNNUNLOCK+

#### 1) EFFECT OF $t$ ON GNNUNLOCK+

*Since GNNUnlock+ considers the structure of the graph in addition to gate features, we expect simple gate replacement using DTL to have little to no impact on its effectiveness.* Indeed, for the case of Anti-SAT-DTL, having $t = 4$, the GNN achieved an average accuracy of 99.99% across all 30 ISCAS-85 graphs, where only 6 nodes were misclassified and rectified during post-processing. For the ITC-99 dataset, the GNN gave 100% node classification accuracy for 29 out of 30 tested graphs. We report detailed results in Tables 9 and 10, for $t = 4$ and $t = K_l/2$, respectively (resp). *Although changing the t parameter affects the performance of removal*

**TABLE 8. Results of GNNUnlock+ on Anti-SAT.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AN | DN | AN | DN | AN | DN | AN | DN | | |
| c2670 | 8 | 99.79 | 100 | 100 | 99.98 | 99.90 | 99.99 | – | 2 | 99.98 | 100 |
| c3540 | 6 | 99.55 | 99.99 | 99.78 | 99.98 | 99.67 | 99.99 | 1 | 2 | 99.98 | 100 |
| c5315 | 8 | 99.90 | 100 | 100 | 99.99 | 99.95 | 99.99 | – | 1 | 99.99 | 100 |
| c7552 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b14_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b15_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b20_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b21_C | 6 | 100 | 99.99 | 99.92 | 100 | 99.96 | 99.99 | 1 | – | 99.99 | 100 |
| b17_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |

*A* design *node and an Anti-SAT node are indicated by DN and AN, respectively.*

**TABLE 9. Results of GNNUnlock+ on Anti-SAT-DTL, having $t = 4$.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AN | DN | AN | DN | AN | DN | AN | DN | | |
| c2670 | 8 | 99.69 | 100 | 100 | 99.97 | 99.84 | 99.98 | – | 3 | 99.97 | 100 |
| c3540 | 6 | 99.78 | 99.99 | 99.78 | 99.99 | 99.78 | 99.99 | 1 | 1 | 99.98 | 100 |
| c5315 | 8 | 99.90 | 100 | 100 | 99.99 | 99.95 | 99.99 | – | 1 | 99.99 | 100 |
| c7552 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b14_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b15_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b20_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b21_C | 6 | 100 | 99.99 | 99.89 | 100 | 99.94 | 99.99 | 2 | – | 99.99 | 100 |
| b17_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |

*A* design *node and an Anti-SAT node are indicated by DN and AN, respectively.*

**TABLE 10. Results of GNNUnlock+ on Anti-SAT-DTL, having $t = K_l/2$.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AN | DN | AN | DN | AN | DN | AN | DN | | |
| c2670 | 8 | 99.79 | 100 | 100 | 99.98 | 99.90 | 99.99 | – | 2 | 99.98 | 100 |
| c3540 | 6 | 99.33 | 99.99 | 99.78 | 99.97 | 99.55 | 99.98 | 3 | 1 | 99.96 | 100 |
| c5315 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| c7552 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b14_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b15_C | 6 | 100 | 99.99 | 99.89 | 100 | 99.94 | 99.99 | 2 | – | 99.99 | 100 |
| b20_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b21_C | 6 | 100 | 99.99 | 99.67 | 100 | 99.83 | 99.99 | 6 | – | 99.99 | 100 |
| b17_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |

*A* design *node and an Anti-SAT node are indicated by DN and AN, respectively.*

**TABLE 11. Results of GNNUnlock+ on Anti-SAT-RND-DTL, having $t = 4$.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AN | DN | AN | DN | AN | DN | AN | DN | | |
| c2670 | 8 | 99.79 | 100 | 100 | 99.98 | 99.90 | 99.99 | – | 2 | 99.98 | 100 |
| c3540 | 6 | 98.89 | 99.99 | 99.78 | 99.95 | 99.33 | 99.97 | 1 | 5 | 99.94 | 100 |
| c5315 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| c7552 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b14_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b15_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b20_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b21_C | 6 | 100 | 99.99 | 99.94 | 100 | 99.97 | 100 | 1 | – | 99.99 | 100 |
| b17_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |

*A* design *node and an Anti-SAT node are indicated by DN and AN, respectively.*

*attacks such as SPS (see Figure 1), GNNUnlock+ handles such irregularities and is capable of achieving the optimal performance for all variations of t.*

#### 2) EFFECT OF RANDOMIZATION ON GNNUNLOCK+

As explained in Section V.A, we replaced $t = 4$ AND gates in the DTL structures with OR gates in Table 11. We observe that GNNUnlock+ is robust to such random modifications, achieving an average GNN accuracy of 99.98% and 99.99% for the ISCAS-85 and ITC-99 benchmarks, respectively.

#### 3) EVALUATING GNNUNLOCK+ ON UNSEEN DTL STRUCTURES

As shown in Table 12, the GNNUnlock+ platform was trained on benchmarks locked using basic Anti-SAT ($t = 0$) and then tested on benchmarks locked using Anti-SAT-DTL having $t = 4$. We achieve an average classification accuracy of 99.93% and 99.96% on the ISCAS-85 and ITC-99 benchmarks, respectively. The obtained results confirm that our framework can predict completely unseen data equally well. Replacing several AND gates by OR gates in the DTL affects the functionality of the block and SAT resilience. However, the tree structure of the Anti-SAT block remains, allowing GNNUnlock+ to understand and identify DTL variants.

**TABLE 12. Results of GNNUnlock+ on Anti-SAT-DTL for trained on $t = 0$ and tested on $t = 4$.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AN | DN | AN | DN | AN | DN | AN | DN | | |
| c2670 | 8 | 99.69 | 99.94 | 99.38 | 99.97 | 99.53 | 99.95 | 6 | 3 | 99.91 | 100 |
| c3540 | 6 | 98.86 | 99.84 | 96.43 | 99.95 | 97.63 | 99.9 | 16 | 5 | 99.80 | 100 |
| c5315 | 8 | 99.90 | 100 | 100 | 99.99 | 99.95 | 99.99 | – | 1 | 99.99 | 100 |
| c7552 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b14_C | 6 | 100 | 99.90 | 98.77 | 100 | 99.38 | 99.95 | 22 | – | 99.91 | 100 |
| b15_C | 6 | 100 | 99.91 | 97.94 | 100 | 98.96 | 99.96 | 37 | – | 99.92 | 100 |
| b20_C | 6 | 100 | 99.98 | 99.33 | 100 | 99.66 | 99.99 | 12 | – | 99.98 | 100 |
| b21_C | 6 | 100 | 99.99 | 99.99 | 100 | 99.97 | 100 | 1 | – | 99.99 | 100 |
| b17_C | 6 | 100 | 99.99 | 99.33 | 100 | 99.66 | 99.99 | 12 | – | 99.99 | 100 |

*A* design *node and an Anti-SAT node are indicated by DN and AN, respectively.*

**TABLE 13. Results of GNNUnlock+ on entangled AntiSAT.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AN | DN | AN | DN | AN | DN | AN | DN | | |
| c2670 | 8 | 98.37 | 100 | 100 | 99.82 | 99.18 | 99.91 | 17 | – | 99.84 | 100 |
| c3540 | 6 | 99.79 | 99.98 | 99.58 | 99.99 | 99.68 | 99.99 | 2 | 1 | 99.97 | 100 |
| c5315 | 8 | 100 | 99.98 | 99.71 | 100 | 99.85 | 99.99 | 3 | – | 99.98 | 100 |
| c7552 | 8 | 100 | 99.99 | 99.9 | 100 | 99.95 | 99.99 | 1 | – | 99.99 | 100 |
| b14_C | 6 | 100 | 99.99 | 99.84 | 100 | 99.92 | 99.99 | 3 | – | 99.99 | 100 |
| b15_C | 6 | 100 | 99.99 | 99.95 | 100 | 99.97 | 99.99 | 1 | – | 99.99 | 100 |
| b20_C | 6 | 100 | 99.99 | 99.95 | 100 | 99.97 | 99.99 | 1 | – | 99.99 | 100 |
| b21_C | 6 | 100 | 99.99 | 99.95 | 100 | 99.97 | 99.99 | 1 | – | 99.99 | 100 |
| b17_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |

*A design node and an Anti-SAT node are indicated by DN and AN, respectively.*

**TABLE 14. Results of GNNUnlock+ on SARLock.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SAR | DN | SAR | DN | SAR | DN | SAR | DN | | |
| c2670 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| c3540 | 6 | 99.77 | 100 | 100 | 99.99 | 99.89 | 99.99 | – | 1 | 99.99 | 100 |
| c5315 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| c7552 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b14_C | 6 | 99.94 | 100 | 100 | 99.99 | 99.97 | 99.99 | – | 1 | 99.99 | 100 |
| b15_C | 6 | 99.98 | 100 | 100 | 99.99 | 99.94 | 99.99 | – | 2 | 99.89 | 100 |
| b20_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b21_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b22_C | 6 | 100 | 99.99 | 99.94 | 100 | 99.97 | 99.99 | 1 | – | 99.99 | 100 |
| b17_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |

*A design node and a SARLock node are indicated by DN and SAR, respectively.*

### D. BREAKING ENTANGLED ANTI-SAT USING GNNUNLOCK+

Anti-SAT integration with wire entanglement increases the connections between the Anti-SAT block and the original design. We expect that such integration will affect GNNUnlock+ as its classification stage is based on neighborhood aggregation. The intuition of GNNUnlock+ is that gates in the netlist are naturally defined by their neighbors and connections. For the ISCAS-85 dataset, the GNN gave an average node classification accuracy of 99.95% across all 30 graphs. For the case of ITC-99 benchmarks, the GNN reports an average node classification accuracy of 99.99% across all 36 graphs. Comparing the results presented in Table 13 with the results on basic Anti-SAT in Table 8, we notice more misclassifications for the case of entangled Anti-SAT.

### E. BREAKING SARLOCK WITH GNNUNLOCK+

For the ISCAS-85 dataset, the GNN gave 100% node classification accuracy for 29 of the graphs and an average of 99.99% accuracy across all 30 graphs. For the remaining graph, only one design node was misclassified and then rectified during post-processing. For the ITC-99 dataset, the GNN gave 100% node classification accuracy for 34 out of 36

**TABLE 15. Results of GNNUnlock+ on SARLock-DTL, having $t = K/2$.**

| Test | #Test Graphs | Prec. (%) | | Rec. (%) | | F1-Score (%) | | #Mislassified Nodes | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SAR | DN | SAR | DN | SAR | DN | SAR | DN | | |
| c2670 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| c3540 | 6 | 99.77 | 100 | 100 | 99.99 | 99.89 | 99.99 | – | 1 | 99.99 | 100 |
| c5315 | 8 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| c7552 | 8 | 99.79 | 100 | 100 | 99.99 | 99.90 | 99.99 | – | 2 | 99.99 | 100 |
| b14_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b15_C | 6 | 99.94 | 100 | 100 | 99.99 | 99.97 | 99.99 | – | 1 | 99.99 | 100 |
| b20_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b21_C | 6 | 100 | 99.99 | 99.83 | 100 | 99.92 | 99.99 | 3 | – | 99.99 | 100 |
| b22_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |
| b17_C | 6 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | 100 | 100 |

*Design node and SARLock node are indicated by DN and SAR, respectively.*

tested graphs. *The SARLock block has a specific out-of-cone structure making it vulnerable to identification and removal.* We present detailed results in Table 14. Replacing $t = K/2$ AND gates by OR gates in SARLock-DTL does not help in protecting from GNNUnlock+ (see Table 15).

### F. BREAKING SFLL-HD$^{hd}$ AND TTLOCK WITH GNNUNLOCK+

For the SFLL-HD$^2$ ISCAS-85 dataset, the GNN gave a node classification accuracy of 100% for 20 tested graphs and an average accuracy of 99.83% across all 45 ISCAS-85 graphs. In total, 38 nodes were misclassified which were rectified using post-processing. For example, 27 *design* nodes from 25 graphs were misclassified as *perturb* nodes. *The misclassified nodes belonged to NOR-tree-like structures in the original designs.This misclassification did not affect the removal of the protection logic because non-protected input patterns controlled these nodes and hence such predictions were dropped.* For SFLL-HD$^2$ ITC-99 dataset, the GNN gave a node classification accuracy of 100% for 29 of the graphs and an average accuracy of 99.97% across all 54 graphs (see Table 16).

For the TTLock ISCAS-85 dataset, the GNN gave a node classification accuracy of 100% for 29 of the graphs and an average accuracy of 99.94% across all 45 graphs. For the remaining 16 graphs, only 19 nodes were misclassified. The results show that *GNNUnlock+ can accurately detect all three types of nodes* with precision and recall values ranging from 93.17%to 100%. For the TTLock ITC-99 dataset, the GNN gave a node classification accuracy of 100% for 16 of the graphs and an average accuracy of 99.95% across all 54 graphs. Comparing accuracy results on ISCAS-85 Vs. ITC-99 (for both SFLL-HD$^{hd}$ and TTLock) confirms that *GNNUnlock+ has a consistent performance for predicting the protection logic regardless of the benchmark size*. Table 17 represents all the averaged results for the TTLock datasets. It is observed that the *restore* predictor is 100% successful (with 100% precision and recall) in all ITC-99 test cases. This also attests that the separation between the *perturb* and *design* nodes is challenging.

**TABLE 17. Examining the effect of $hd$ value and technology node on the performance of GNNUnlock+.**

| Dataset | Benchmarks | Tech. Node (nm) | Acc. (%) | Avg. Prec. (%) | Avg. Rec. (%) | Avg. F1-Score (%) | Removal Success (%) | Avg. TR Time (s) |
|---|---|---|---|---|---|---|---|---|
| TTLock | ISCAS-85 | 45 | 99.94 | 99.20 | 99.53 | 99.33 | 100 | 1,910 |
| TTLock | ITC-99 | 45 | 99.95 | 97.48 | 99.36 | 98.33 | 100 | 22,777 |
| SFLL-HD$^2$ | ITC-99 | 45 | 99.94 | 98.73 | 99.77 | 99.21 | 100 | 24,123 |
| SFLL-HD$^2$ | ITC-99 | 65 | 99.97 | 99.56 | 99.85 | 99.70 | 100 | 22,052 |
| SFLL-HD$^4$ | ITC-99 | 65 | 99.90 | 98.40 | 99.60 | 98.95 | 100 | 233,017 |
| SFLL-HD$^{16}$ | ISCAS-85 | 65 | 99.24 | 97.68 | 97.30 | 97.40 | 100 | 1,907 |
| SFLL-HD$^{32}$ | ITC-99 | 65 | 99.83 | 97.56 | 98.45 | 97.96 | 100 | 28,049 |
| SFLL-HD$^{64}$ | ITC-99 | 65 | 99.69 | 97.60 | 97.96 | 97.74 | 100 | 32,494 |

*To study the effect of hd on the performance of GNNUnlock+*, we launch the attack on the SFLL-HD$^4$ ITC-99 dataset. We report the averaged metrics in Table 17. Here, we also report the results of GNNUnlock+ on the SFLL-HD$^2$ ITC-99 dataset with a technology node of 45*nm* to evaluate the effect of the target library on the performance of the attack. The trend is the same in all cases, while there is a slight difference in the values. GNNUnlock+ detects all three types of nodes accurately with average accuracy values of 99.94%, 99.97%, and 99.90%, for SFLL-HD$^2$ in 45*nm*, SFLL-HD$^2$ in 65*nm*, and SFLL-HD$^4$ in 65*nm* datasets, respectively. All nodes were classified correctly (100%) after post-processing verifying that *GNNUnlock+ can handle different technology nodes and various parameter settings.*

### G. BREAKING CAC WITH GNNUNLOCK+

For the ISCAS-85 dataset locked using CAC, the GNN achieved a node classification accuracy of 100% for 28

**TABLE 16. Results of GNNUnlock+ on SFLL-HD$^2$.**

| Test | #Test Graphs | Prec. (%) | | | Rec. (%) | | | F1-Score (%) | | | #Misclassified Nodes | | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | RN | PN | DN | RN | PN | DN | RN | PN | DN | RN | PN | DN | | |
| c2670 | 12 | 99.55 | 96.36 | 100 | 100 | 99.37 | 99.46 | 99.77 | 97.84 | 99.73 | – | 4 as RN | 24 as PN | 99.53 | 100 |
| c3540 | 9 | 98.31 | 98.97 | 100 | 100 | 98.30 | 99.88 | 99.15 | 98.63 | 99.94 | – | 5 as RN | 3 as PN<br>2 as RN | 99.79 | 100 |
| c5315 | 12 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | – | 100 | 100 |
| c7552 | 12 | 100 | 100 | 99.99 | 100 | 99.85 | 100 | 100 | 99.93 | 99.99 | – | 1 as DN | – | 99.99 | 100 |
| b14_C | 9 | 99.88 | 100 | 99.98 | 100 | 99.43 | 100 | 99.94 | 99.72 | 99.99 | – | 2 as RN<br>5 as DN | – | 99.97 | 100 |
| b15_C | 9 | 99.94 | 100 | 99.99 | 100 | 99.68 | 100 | 99.97 | 99.84 | 99.99 | – | 3 as DN<br>1 as RN | – | 99.99 | 100 |
| b20_C | 9 | 99.46 | 100 | 99.99 | 100 | 99.04 | 100 | 99.73 | 99.52 | 99.99 | – | 9 as RN<br>2 as DN | – | 99.98 | 100 |
| b21_C | 9 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | – | 100 | 100 |
| b22_C | 9 | 99.94 | 97.83 | 99.99 | 100 | 99.67 | 99.96 | 99.97 | 98.74 | 99.98 | – | 1 as RN<br>3 as DN | 27 as PN | 99.96 | 100 |
| b17_C | 9 | 99.46 | 95.60 | 100 | 100 | 99.52 | 99.95 | 99.73 | 97.52 | 99.98 | – | 6PN as RN | 3 as RN<br>57 as PN | 99.94 | 100 |

*The* design, perturb, *and* restore *nodes are indicated by DN, PN, and RN, respectively.*

**TABLE 18. Results of GNNUnlock+ on CAC, having $t = 0$.**

| Test | #Test Graphs | Prec. (%) | | | Rec. (%) | | | F1-Score (%) | | | #Misclassified Nodes | | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | RN | PN | DN | RN | PN | DN | RN | PN | DN | RN | PN | DN | | |
| c2670 | 12 | 89.66 | 100 | 100 | 100 | 100 | 98.62 | 94.55 | 100 | 99.30 | – | – | 63 as RN | 98.80 | 100 |
| c3540 | 9 | 100 | 92.54 | 99.98 | 97.68 | 100 | 100 | 98.83 | 96.12 | 99.99 | 5 as PN<br>1 as DN | – | – | 99.87 | 100 |
| c5315 | 12 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | – | – | – | 100 | 100 |
| c7552 | 12 | 98.55 | 100 | 100 | 100 | 96.06 | 99.97 | 99.27 | 97.99 | 99.98 | – | 5 as RN | 3 as RN | 99.92 | 100 |
| b14_C | 9 | 99.80 | 97.86 | 99.98 | 99.51 | 97.45 | 100 | 99.66 | 97.65 | 99.99 | 5 as PN | 2 as RN<br>4 as DN | – | 99.96 | 100 |
| b15_C | 9 | 100 | 85.19 | 99.99 | 95.87 | 100 | 100 | 97.89 | 92.00 | 100 | 40 as PN<br>2 as DN | – | – | 99.89 | 100 |
| b20_C | 9 | 100 | 99.11 | 99.98 | 99.51 | 96.10 | 100 | 99.75 | 97.58 | 99.99 | 5 as DN | 9 as DN | 2 as PN | 99.97 | 100 |
| b21_C | 9 | 99.80 | 97.86 | 100 | 99.51 | 98.71 | 100 | 99.66 | 98.28 | 100 | 5 as PN | 2 as RN<br>1 as DN | – | 99.99 | 100 |
| b22_C | 9 | 99.80 | 99.13 | 100 | 99.80 | 99.13 | 100 | 99.80 | 99.13 | 100 | 2 as PN | 2 as DN | – | 100 | 100 |
| b17_C | 9 | 99.82 | 91.51 | 99.99 | 97.80 | 98.48 | 100 | 98.80 | 94.87 | 99.99 | 36 as PN<br>2 as DN | 3 as RN<br>3 as DN | – | 99.96 | 100 |

*The design, perturb and restore nodes are indicated by DN, PN, and RN, respectively.*

**TABLE 19. Results of GNNUnlock+ on CAC, having $t = K/2$.**

| Test | #Test Graphs | Prec. (%) | | | Rec. (%) | | | F1-Score (%) | | | #Misclassified Nodes | | | GNN Acc. (%) | Removal Success (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | RN | PN | DN | RN | PN | DN | RN | PN | DN | RN | PN | DN | | |
| c2670 | 12 | 97.50 | 45.45 | 100 | 98.35 | 99.55 | 94.07 | 97.92 | 62.41 | 96.94 | 15 as PN | 1 as RN | 249 as PN<br>22 as DN | 94.96 | 100 |
| c3540 | 9 | 93.51 | 97.78 | 99.93 | 98.82 | 87.13 | 99.63 | 96.09 | 92.15 | 99.78 | 5 as PN | 13 as RN | 16 as PN | 99.30 | 100 |
| c5315 | 12 | 99.55 | 96.15 | 99.79 | 97.48 | 97.83 | 99.98 | 98.50 | 96.98 | 99.88 | 9 as PN<br>14 as DN | 2 as RN<br>3 as DN | 2 as RN | 99.68 | 100 |
| c7552 | 12 | 99.78 | 96.04 | 100 | 99.02 | 99.09 | 100 | 99.40 | 97.54 | 100 | 9 as PN | 2 as RN<br>3 as DN | – | 99.90 | 100 |
| b14_C | 9 | 99.82 | 96.10 | 99.96 | 98.77 | 97.77 | 100 | 99.29 | 96.92 | 99.98 | 16 as PN<br>5 as DN | 3 as RN<br>6 as DN | – | 99.89 | 100 |
| b15_C | 9 | 100 | 95.80 | 100 | 98.96 | 100 | 100 | 99.48 | 97.86 | 100 | 18 as PN | – | – | 99.96 | 100 |
| b20_C | 9 | 99.71 | 96.05 | 99.99 | 99.06 | 97.49 | 100 | 99.38 | 96.77 | 100 | 16 as PN | 5 as RN<br>5 as DN | – | 99.96 | 100 |
| b21_C | 9 | 98.38 | 98.16 | 99.99 | 99.59 | 91.89 | 100 | 98.98 | 94.92 | 99.99 | 7 as PN | 27 as RN<br>6 as DN | 1 as RN | 99.93 | 100 |
| b22_C | 9 | 100 | 99.75 | 99.99 | 99.94 | 99.75 | 100 | 99.97 | 99.75 | 99.99 | 1 as PN | 1 as DN | – | 99.99 | 100 |
| b17_C | 9 | 99.82 | 91.51 | 99.99 | 97.80 | 98.48 | 100 | 98.80 | 94.87 | 99.99 | 36 as PN<br>2 as DN | 3 as RN<br>3 as DN | – | 99.96 | 100 |

*The design, perturb and restore nodes are indicated by DN, PN, and RN, respectively.*

tested graphs and an average accuracy of 99.65% across all 45 graphs. In total, 72 nodes were misclassified and then consequently rectified using post-processing. For the ITC-99 dataset, the GNN yielded a node classification accuracy of 100% for 36 graphs and an average accuracy of 99.96% across all 54 graphs (see Table 18).

Modifying the DTL using $t = K/2$ slightly affects the accuracy of the framework. For the case of the ISCAS-85 benchmarks, the accuracy drops from an average of 99.65% to an average of 98.46%. For the case of the ITC-99 benchmarks, the node classification accuracy drops to 99.95% (see Table 19). *The outcomes indicate that CAC-DTL is indeed more challenging to break than SFLL-HD. Nevertheless, relying on the GNN predictions, the post-processing step can rectify all misclassifications.* To visualize such misclassifications, we show the graph corresponding to the ISCAS-85 benchmark *c*2670 locked using CAC with a key-size of 32 and $t = 16$, in Figure 16. The GNN misclassified 22 nodes as indicated in the figure. However, with the help of the predictions, such misclassifications are rectified.

### H. PROTECTION LOGIC REMOVAL

Once the protection logic is identified by GNNUnlock+, the tip of this block (flip/perturb) must be set as a constant value to obtain the original design. For example, the Anti-SAT block outputs a 0 for all the input patterns once the correct key is in place. Thus, fixing the flip signal as a constant 0 is equivalent to removing the protection logic from the circuit. Throughout our experimental evaluation of GNNUnlock+ on the different PSLL techniques, we observed constant flip and perturb signals of 0s for the evaluated techniques. Revising GNNUnlock+ to account for different flip signal values is reserved for future work.

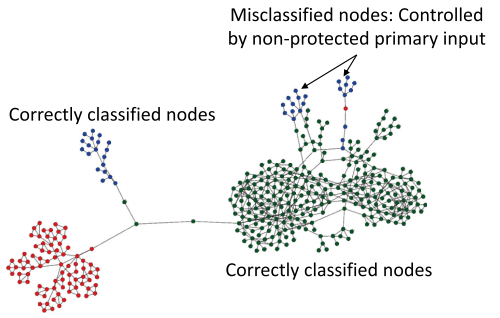



**FIGURE 16. Post-processing for correcting misclassifications in the case of CAC having $t = K/2$. 22 nodes were misclassified and rectified. The green, red, and blue nodes indicate the design, restore, and perturb gates, respectively. The coloring is based on the GNN's predictions.**

### I. COMPARISON WITH STATE-OF-THE-ART ATTACKS

To demonstrate the superiority of GNNUnlock+, a set of ITC-99 benchmarks are locked using SFLL-HD with $K/hd = 2$, generating two datasets for $K = 128$, $hd = 64$ and one for $K = 64$, $hd = 32$. The ISCAS-85 benchmarks are locked using $K = 32$, $hd = 16$. We list the characteristics in Table 7. When the FALL attacks [13] were launched on these benchmarks, they reported 0 keys. Next, we launch the SFLL-HD-Unlocked [12] attack on the same benchmarks, which failed to identify the perturb signals and did not recover the key. However, GNNUnlock+ was successful in retrieving the unlocked designs in all tested cases (see Table 17).

### J. ADAPTING TO FUTURE PSLL TECHNIQUES

We release our optimal GNNUnlock+ models to the community to generate accurate predictions on future unseen data [20]. However, we do not restrict the applicability of GNNUnlock+ to current PSLL techniques only; we explain all the required steps to optimize GNNUnlock+ to future PSLL techniques where our post-processing might be outdated (see Section IV.C). With the continuous evolution of attacks on PSLL techniques, many defense techniques are being developed, which necessitates the adaptability of the GNNUnlock+ framework. We provide insights on how to retrain GNNUnlock+ on new datasets that reflect new PSLL implementations. Next, we discuss how some of the recent PSLL techniques can be handled by GNNUnlock+.

The error-controllable encryption [27] technique is a particular case of DTL, where a designer can replace a few AND gates in the Anti-SAT tree structure by XOR/OR/NAND gates. This gate replacement helps in controlling the corruptibility of the Anti-SAT block, which in turn helps thwart the SPS attack. Since GNNUnlock+ does not consider the signal skew as a hint for detecting the Anti-SAT block, its performance will not be affected by such gate-level replacements, as experimentally demonstrated in Section V.III. Strong Anti-SAT [28] selects critical minterms to corrupt primary outputs upon the application of an incorrect key. CAS-Lock [29] replaces the AND-tree in Anti-SAT with a cascade of AND and OR gates. The aforementioned techniques utilize protection blocks that have specific structures and functionalities, making them vulnerable to GNNUnlock+ and other removal attacks. G-Anti-SAT [30] is another technique that aims to thwart approximate attacks by increasing corruption while maintaining SAT resilience. G-Anti-SAT utilizes K-maps to design various complementary and non-complementary logic functions for the Anti-SAT block, replacing the AND-tree structures. However, the protection block will always have AND functions with a large number of inputs, as explained in [31]. Thus, the overall protection logic has a distinct structure that can be captured by GNNUnlock+. The noise-based logic locking technique [32] modifies the Anti-SAT block and connects the protection logic to the original circuit using two XOR gates instead of one. Having this two-point integration thwarts the SPS attack. Nevertheless, since GNNUnlock+ looks for the learned structure of the protection logic instead of searching for the Anti-SAT output, this technique is still vulnerable.

## VI. CONCLUSION

In this paper, we propose **GNNUnlock+** framework for unlocking provably secure logic locking (PSLL) techniques. GNNUnlock+ leverages a graph neural network (GNN) and learns the structural features of the added protection logic. The GNN does not learn a syntactic implementation but absorbs the trend of the protection logic, allowing it to handle variants naturally. We also develop a post-processing algorithm to rectify any misclassifications by the GNN, depending solely on the nodes' connectivity, the GNN predictions, and the known properties of the protection logic. We presented the required design steps to tailor GNNUnlock+ for unlocking any desired PSLL technique. We also demonstrated the superiority of GNNUnlock+ by comparing it to 3 state-of-the-art attacks in unlocking 9 PSLL techniques. GNNUnlock+ can break all locked benchmarks while the state-of-the-art attacks struggled with some corner cases. Moreover, we also show that GNNUnlock+ can handle targeted removal attack protection, such as using diversified structures in the protection logic or wire entanglement. We believe GNNUnlock+ opens up new frontiers in advancing the state-of-the-art defenses and attacks in PSLL techniques.

## ACKNOWLEDGMENTS

This work was supported by the Center for Cyber Security (CCS), New York University Abu Dhabi (NYUAD). A preliminary version of this work was presented at DATE 2021 [1].

## REFERENCES

[1] L. Alrahis *et al.*, "GNNUnlock: Graph neural networks-based Oracle-less unlocking scheme for provably secure logic locking," in *Proc. Des., Automat. Test Eur. Conf.*, 2021, pp. 780–785.
[2] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.
[3] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.
[4] R. Jarvis and M. McIntyre, "Split manufacturing method for advanced semiconductor circuits," U.S. Patent 7 195 931, 27 Mar., 2007.
[5] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *Proc. 51st Annu. Des. Automat. Conf.*, 2014, pp. 1–5.
[6] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1601–1618.
[7] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. Int. Symp. Hardware Oriented Secur. Trust*, 2015, pp. 137–143.
[8] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. Int. Symp. Hardware Oriented Secur. Trust*, 2017, pp. 95–100.
[9] M. T. Rahman, S. Tajik, M. S. Rahman, M. Tehranipoor, and N. Asadizanjani, "The key is left under the mat: On the inappropriate security assumption of logic locking schemes," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust.*, 2020, pp. 262–272.
[10] A. Jain, Z. Zhou, and U. Guin, "TAAL: Tampering attack on any key-based logic locked circuits," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 26, no. 4, pp. 1–22, 2021.
[11] L. Alrahis, M. Yasin, H. Saleh, B. Mohammad, and M. Al-Qutayri, "Functional reverse engineering on SAT-attack resilient logic locking," in *Proc. Int. Symp. Circuits Syst.*, 2019, pp. 1–5.
[12] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (SFLL-hd)–unlocked," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 10, pp. 2778–2786, Oct. 2019.
[13] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 2514–2527, Mar. 2019.
[14] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Proc. Int. Conf. Cryptograph. Hardware Embedded Syst.*, 2016, pp. 127–146.
[15] S. Khaleghi, K. D. Zhao, and W. Rao, "IC piracy prevention via design withholding and entanglement," in *Proc. 20th Asia South Pacific Des. Automat. Conf.*, 2015, pp. 821–826.
[16] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and IC camouflaging schemes," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 2, pp. 347–359, Feb. 2019.
[17] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of Anti-SAT," in *Proc. 22nd Asia South Pacific Des. Automat. Conf*, 2017, pp. 342–347.
[18] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proc. Int. Symp. Hardware Oriented Secur. Trust*, 2016, pp. 236–241.
[19] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 2, pp. 199–207, Feb. 2019.
[20] L. Alrahis, "GNNUnlock source code," 2021. [Online]. Available: https://github.com/DfX-NYUAD/GNNUnlock
[21] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.
[22] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graph-SAINT: Graph sampling based inductive learning method," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–19.
[23] 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 *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 3438–3445.
[24] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.
[26] 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," 2018, *arXiv:1803.07294*.
[27] Y. Shen, A. Rezaei, and H. Zhou, "A comparative investigation of approximate attacks on logic encryptions," in *Proc. 23rd Asia South Pacific Des. Automat. Conf.*, 2018, pp. 271–276.
[28] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong anti-SAT: Secure and effective logic locking," *Proc. 21st Int. Symp. Qual. Electron. Des.*, 2020, pp. 199–205.
[29] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "CAS-Lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Trans. Cryptograph. Hardware Embedded Syst.*, vol. 1, pp. 175–202, 2020.
[30] J. Zhou and X. Zhang, "Generalized SAT-attack-resistant logic locking," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 2581–2592, Feb. 2021.
[31] J. Zhou and X. Zhang, "A new logic-locking scheme resilient to gate removal attack," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2020, pp. 1–5.
[32] A. Rezaei and A. Mahani, "Noise-based logic locking scheme against signal probability skew analysis," *IET Comput. Digit. Techn.*, vol. 15, no. 4, pp. 279–295, 2021.

**LILAS ALRAHIS** (Member, IEEE) received the MSc and PhD degrees in electrical and computer engineering from Khalifa University, UAE, in 2016 and 2020, respectively. She is currently a postdoctoral associate with New York University Abu Dhabi. Her research interests include hardware security, design for trust, logic locking, and applied machine learning. She was the recipient of 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.

**SATWIK PATNAIK** (Member, IEEE) received the PhD degree in electrical engineering from the Tandon School of Engineering, New York University, 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. He was the recipient of Bronze Medal in Graduate Category at ACM/SIGDA Student Research Competition held at ICCAD 2018, and the Best Paper Award at Applied Research Competition held in conjunction with Cyber Security Awareness Week, in 2017.

**MUHAMMAD ABDULLAH HANIF** (Graduate Student Member, IEEE) is currently working toward the PhD degree with the Faculty of Informatics, Technische Universität Wien, Vienna, Austria. In the past, he has worked as a research associate in Vision Processing Lab, Information Technology University , Pakistan, and as a lab engineer with Ghulam Ishaq Khan Institute of Engineering Sciences and Technology , Pakistan.

**HANI SALEH** (Senior Member, IEEE) is currently an associate professor of electronic engineering with Khalifa University since 2012. He is a co-founder/active researcher with the Khalifa University Research Center and the System on Chip Research Center. Prior to academia, he worked for many leading chip design companies including Apple, Intel, AMD, Qualcomm, Synopsys, Fujitsu, and Motorola with the University of Texas at Austin.

**MUHAMMAD SHAFIQUE** (Senior Member, IEEE) received the PhD degree in computer science from the Karlsruhe Institute of Technology, Germany, in 2011. In 2016, he joined the Faculty of Informatics, Institute of Computer Engineering, Technische Universität Wien, Austria, as a professor of computer architecture and robust, energy-efficient technologies. Since 2020, he is with New York University Abu Dhabi, UAE. His research interests include brain-inspired computing, AI, & ML hardware and system-level design, energy-efficient systems, robust computing, hardware security, emerging technologies, FPGAs, MPSoCs, and embedded systems. He holds one U.S. patent, has coauthored six Books, more than ten book chapters, and more than 200 conference and journal papers. He was the recipient of 2015 ACM/SIGDA Outstanding New Faculty Award, AI 2000 Chip Technology Most Influential Scholar Award in 2020, six gold medals, and several best paper awards.

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