# *NetDeTox*: Adversarial and Efficient Evasion of Hardware-Security GNNs via RL-LLM Orchestration

Zeng Wang[†§], Minghao Shao[†‡§], Akashdeep Saha[‡], Ramesh Karri[†], Johann Knechtel[‡], Muhammad Shafique[‡], Ozgur Sinanoglu[‡]

[†]NYU Tandon School of Engineering, New York, USA

[‡]NYU Abu Dhabi, Abu Dhabi, UAE

{zw3464, shao.minghao, as19360, rkarri, johann, muhammad.shafique, ozgursin}@nyu.edu

## Abstract

*Graph neural networks* (*GNNs*) have shown promise in hardware security by learning structural motifs from netlist graphs. However, this reliance on motifs makes GNNs vulnerable to adversarial netlist rewrites; even small-scale edits can mislead GNN predictions. Existing adversarial approaches, ranging from synthesis-recipe perturbations to gate transformations, come with high design overheads. We present ***NetDeTox***, an automated end-to-end framework that orchestrates *large language models* (*LLMs*) with *reinforcement learning* (*RL*) in a systematic manner, enabling focused local rewriting. The RL agent identifies netlist components critical for GNN-based reasoning, while the LLM devises rewriting plans to diversify motifs that preserve functionality. Iterative feedback between the RL and LLM stages refines adversarial rewritings to limit overheads. Compared to the SOTA work AttackGNN, *NetDeTox* successfully degrades the effectiveness of all security schemes with fewer rewrites and substantially lower area overheads (reductions of 54.50% for GNN-RE, 25.44% for GNN4IP, and 41.04% for OMLA, respectively). For GNN4IP, ours can even optimize/reduce the original benchmarks' area, in particular for larger circuits, demonstrating the practicality and scalability of *NetDeTox*.

## 1 Introduction

Graph neural networks (GNNs) have been widely adopted in hardware security, including IP piracy detection [31], reverse engineering [3], logic locking attacks [2], and hardware Trojan detection [16]. However, recent work shows that GNN-based methods are vulnerable to adversarial netlist transformations using reinforcement learning (RL) [12] to rewrite whole designs or large language models (LLMs) [11] to introduce more various localized transformations. While these attacks can evade GNN analysis, they incur significant design overheads, creating an undesirable trade-off.

While LLMs excel at hardware design tasks such as Verilog generation [20, 22, 23], assertion generation [15, 29], and testbench synthesis [5, 19], and have been applied to hardware security analysis [24–26], their ability to manipulate gate-level netlists remains limited. The complex semantics and structural features of netlists pose significant challenges for LLM-based transformation [10, 32]. Recent planning-based code generation approaches [4, 13] show that chain-of-edits methods can effectively handle large codebases through iterative, context-aware transformations.

Drawing on this, we approach evasion through iterative netlist rewriting, generating a sequence of targeted transformations that adversarially alter GNN-detected graph structures while optimizing for area and preserving functionality. This approach requires addressing a key challenge: how to effectively reason about netlist transformations. Pure LLM-based methods struggle with gate-level representations [9, 11], while RL excels at structural netlist processing [7, 12]. We bridge this gap through a hybrid architecture where RL identifies critical transformation locations while LLMs provide context-aware reasoning for hardware security applications.

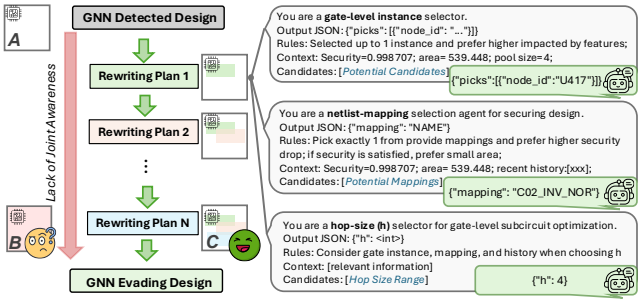


**Figure 1: Demonstration of *NetDeTox* for netlist rewriting. Direct rewriting of a GNN detected design *A*, without joint security–area awareness, produces a suboptimal design *B*. In contrast, *NetDeTox* employs context-aware iterative planning with coordinated gate selection, subnetlist mapping, and hop-size determination, all to achieve an efficient design *C*.**

We propose *NetDeTox*, a novel framework that combines RL-based gate selection with LLM-driven rewriting to progressively evade GNN-based security tools. Rather than processing entire designs through LLMs, *NetDeTox* focuses on targeted gate pools identified by RL. As shown in Fig. 1, our framework employs an LLM-in-the-loop process that iteratively generates rewriting plans by coordinating gate selection, subnetlist mapping, and hop-size determination. This coordinated planning enables hardware-aware transformations that jointly optimize for security evasion and area efficiency, achieving evasion with minimal overheads.

Our main contributions are:

- We introduce *NetDeTox*, an RL-assisted, LLM-guided framework for subnetlist rewriting that evades GNN-based analyses while respecting area and implementation constraints.
- We demonstrate consistent shortcomings of robust GNN tools (OMLA, GNN4IP, GNN-RE) across six diverse LLM backends, indicating robustness beyond a single model family.

[§]Authors contributed equally to this research.

- We present a detailed evaluation, including comparative case studies and ablation studies (e.g., for the impact of planning order and LLM strategy selection on evasion efficacy and cost).

## 2 Background And Related Works

### 2.1 GNNs for Hardware Security

GNNs learn graph representations by aggregating information from node neighborhoods. Circuit netlists naturally map to graphs with gates as nodes and nets as edges, making GNNs well-suited for hardware security analysis [8].

GNN-based methods have been applied to both defensive and offensive hardware security tasks. On the defensive side, GNN4TJ [30] targets hardware Trojan (HT) detection in third-party IP by flagging HT-related structures, while GNN4IP [31] addresses IP piracy by embedding circuit pairs and measuring their similarity (e.g., cosine distance) to detect unauthorized copies. On the offensive side, OMLA [2] attacks synthesized logic-locked designs by exploiting structural context around key-controlled gates to predict key bits with high accuracy, and GNNRE [3] performs reverse engineering through node-level classification, assigning gates to their original functional modules with approximately 98.8% accuracy on standard benchmarks. These applications span diverse GNN architectures and settings, including node classification, graph classification, and graph similarity learning.

### 2.2 Adversarial Netlist Generation

Recent work has explored generating adversarial netlists to evade GNN-based hardware security tools. AttackGNN [12] employs RL to discover synthesis recipes that systematically rewrite entire designs, successfully undermining GNN4IP, GNNRE, OMLA, and other GNN-based tools. However, it couples rewriting with per-cycle RL reward updates, increasing computational complexity and runtime. LLM-Pirate [11] leverages LLMs for localized gate transformations to evade GNN4IP, but treats the LLM as a template-driven substitution engine with limited circuit-context awareness. Both approaches prioritize evasion effectiveness over design quality, resulting in significant area and timing overheads. While recent work [33] addresses area reduction during circuit transformation, it lacks evaluation against real GNN-based security tools. This motivates our work, achieving effective GNN evasion while maintaining design quality through context-aware, targeted transformations.

## 3 Threat Model

We consider a standard black-box adversary [11, 12] who seeks to evade GNN-based security tools, namely OMLA, GNN4IP, and GNNRE. Each tool returns a real-valued security score: GNN4IP provides a similarity ranging from −1 to 1, while OMLA and GNNRE return classification confidence scores ranging from 0 to 1. The adversary operates post-deployment with access only to the detector's input-output interface, receiving scores for any netlist provided. The adversary has no knowledge of internal parameters, training data, or model architecture. The adversary cannot alter or retrain these GNN-based security tools.

Our framework requires no knowledge of specific GNN vulnerabilities—it operates purely through black-box queries. Netlist transformations must preserve functional equivalence and respect standard design rules. Our evaluation demonstrates transferability across multiple GNN architectures and LLM backends. The adversary's goal is generating functionally equivalent netlists that evade detection while maintaining design quality.

**Table 1: Predefined Subnetlist Mapping Options.**

| Map. | Gate Composition | Map. | Gate Composition |
|---|---|---|---|
| C01 | INV, NAND, BUF | C11 | INV, NOR, XOR, BUF |
| C02 | INV, NOR, BUF | C12 | INV, NOR, XNOR, BUF |
| C03 | INV, NAND, LOGIC, BUF | C13 | INV, AND, OR, BUF |
| C04 | INV, NAND, AND, BUF | C14 | INV, AND, OR, LOGIC, BUF |
| C05 | INV, NAND, OR, BUF | C15 | INV, AND, OR, XOR, BUF |
| C06 | INV, NAND, XOR, BUF | C16 | INV, AND, OR, XNOR, BUF |
| C07 | INV, NAND, XNOR, BUF | C17 | INV, NAND, LOGIC, XOR, BUF |
| C08 | INV, NOR, LOGIC, BUF | C18 | INV, NAND, LOGIC, XNOR, BUF |
| C09 | INV, NOR, AND, BUF | C19 | INV, NOR, LOGIC, XOR, BUF |
| C10 | INV, NOR, OR, BUF | C20 | INV, NOR, LOGIC, XNOR, BUF |

## 4 Our Framework: *NetDeTox*

Prior approaches [11, 12] operate at whole-netlist scale, possibly inflating silicon area. Moreover, presenting entire netlists to LLMs poses practical challenges: (i) large inputs incur substantial token costs, and (ii) raw gate-level abstractions offer limited semantic clarity for effective reasoning. To address these challenges, we propose *NetDeTox* (Fig. 2), which employs an RL-guided policy to assemble gate pools and an LLM to generate context-aware rewriting plans that localize edits while preserving functional and implementation constraints. This targeted pipeline achieves comparable GNN evasion with significantly lower area overheads.

### 4.1 RL-Guided Gate Pooling

We represent the netlist as a directed graph, with gates as nodes and wires as edges. Each node is annotated with features like gate type, fan-in, fan-out, and logic level, which are used to bin nodes and steer diversity. An RL policy samples from these bins to form candidate gate pools, directing selection toward high-impact regions rather than repeatedly visiting similar structures. More specifically, after each rewrite, the policy updates its bin preferences using a lightweight reward $R = \alpha\,\Delta\text{Security} - \beta\,\Delta\text{Area}$. Each step consists of (1) constructing bins, (2) selecting a bin via the RL policy, (3) sampling gates from that bin, and (4) updating both the policy and the bins. This produces a compact, non-redundant set of RL-selected targets for subsequent LLM-driven rewriting and is critical to downstream evasion performance.

### 4.2 LLM Planning for Subnetlist Rewriting

Given the RL-selected pools, the LLM orchestrates subnetlist rewrites by setting the strategy and neighbourhood as follows.

**Gate Selection:** At each iteration, the LLM receives a pool sampled from the updated bin for that round, which reflects prior edits, security/area feedback, allowed gate types, and structural rules. The LLM selects $N$ target gates from the pool.

**Subnetlist Mapping:** The LLM chooses from pre-defined mapping options (Tab. 1) to replace logic with distinct primitive compositions, preserving functionality and increasing structural diversity. A prioritized histogram of historical area and security values guides the choices, so that different regions adopt tailored motifs with limited overlaps with GNN-learned motifs.

**Hop-Size Selection:** During subnetlist construction, the LLM selects a hop size $h$ of logic levels around the chosen gates to include.

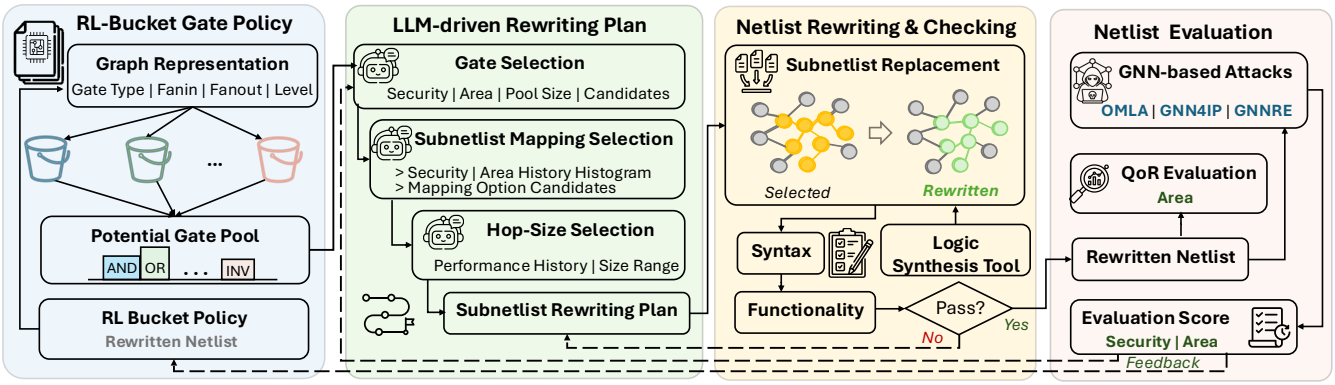


**Figure 2: *NetDeTox* framework overview.**

Larger $h$ enables extensive structural rewriting while smaller $h$ localizes rewriting. The choice is based on some pre-defined range and informed by prior outcomes.

**Transformation Planning Order:** We schedule decisions in a logical, policy-driven progression: gate selection, mapping choice, then $h$ determination, enabling the LLM to adapt its reasoning to the evolving circuit context (see Sec. 5.7.3).

## 4.3 Subnetlist Rewriting and Validation

**Rewriting:** Using the LLM-guided transformation plan, the framework extracts and rewrites the subnetlist defined by the selected gates and hop size $h$. The subnetlist is isolated for local rewriting with LLM-chosen mappings, and the rewritten subnetlist replaces the original one in the design.

**Syntax/Functional Validation:** The plan is rejected if the subnetlist introduces syntax errors, connectivity issues, or functional mismatches (Fig. 2). LLM revises its plan, ensuring only valid, functional designs advance.

## 4.4 Putting It All Together

Shown in Fig. 2, *NetDeTox* operates in an iterative loop: node features drive bin segregation, RL samples candidate gates, LLM selects targets, chooses mappings, and sets $h$. The resulting subnetlist is extracted, rewritten, and inserted back. The rewritings are validated. Next, the modified netlist is fed to the target GNN-based tools to evaluate its security. Finally, the updated security and area metrics are fed back into the next LLM prompt and the RL bin policy, closing the loop and progressively improving evasion.

# 5 Experimental Investigation

## 5.1 Setup

**LLM Planning.** The LLM generates rewriting plans sequentially: first, it selects $N = 5$ target gates, then it chooses one of 20 pre-defined mapping options (Table 1), and finally it sets a hop size $h \in [1, 20]$ for the rewriting of the neighbourhood. All selected gates share the same mapping and $h$ to reduce prompt overhead. Prompts incorporate prior *security* and *area* metrics for planning.

**RL-Guided Binning.** The proposed framework partitions nodes/-gates into up to 24 bins. In each round, 20 candidates are sampled via a softmax over bin scores, with per-bin quotas to maintain structural diversity. RL updates use *reinforce* policy gradients [27], with reward defined as a weighted sum of security score reduction ($\alpha = 1.5$) and area-overhead reduction ($\beta = 0.5$). The weights are empirically determined to adjust bin preferences across iterations.

**Netlist Evaluation.** Subnetlist rewriting is performed in ABC [18] using {`rewrite -z`, `refactor -z`, `resub -z`} as specified by LLM-selected mappings. Security is assessed by querying pre-trained models (OMLA, GNN4IP, GNN-RE) for detection/classification scores. Area is evaluated via Yosys [28] synthesis for NanGate45 [1]. Following AttackGNN and LLMPirate, for comparison, we consider only area overheads.

**LLM Selection** We benchmarked six LLM backends to avoid bias: `LlaMA-4-Maverick-17B`, `GPT-4o-mini`, `GPT-5`, `Qwen-3-235B`, `DeepSeek-V3`, and `Gemini-2.5-Flash`. All models used a 2,048 token budget (reasoning included) with temperature = 0.8. Prompts enforce JSON outputs for reproducibility.

**Scope of Experiments** We evaluated *NetDeTox* on SOTA GNN-based tools: OMLA, GNN4IP, and GNN-RE, all on the same benchmarks synthesis flows, and backend LLMs as in [12]. We also evaluated *NetDeTox* on GNN4IP, the tool used in [11]. Our analysis shows not only strong evasion performance but also consistent cost savings and scalability. Finally, we conduct various ablation studies, confirming the practical relevance of our proposed framework.

## 5.2 Case Study 1: OMLA and AttackGNN

**Security Evaluation:** For OMLA, successful evasion is indicated by key accuracy around 50% (random guessing), meaning the attack cannot recover the secret key. *NetDeTox* achieves this threshold in 23/24 benchmark-backend pairs (Fig. 3), with only Qwen-3 on `c3540` showing marginal underperformance. Compared to AttackGNN, which pushes OMLA scores between 45%and 55%, *NetDeTox* drives them more consistently toward the 50% region.

**Overhead Analysis:** From Tab. 2, *NetDeTox* achieves more favorable area-security trade-offs than AttackGNN. For example, on `c1908`, it achieves as little as 24.79% area overheads, whereas AttackGNN incurs 80.28% overheads. Reduced overheads scale well with larger designs, which offer more subnetlist optimization opportunities. Critically, targeted subnetlist rewriting avoids the potentially exponential area growth of global rewriting.

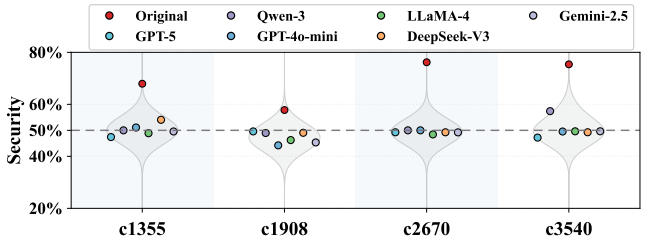


**Figure 3: Security of *NetDeTox* netlists against OMLA. Original refers to the baseline design. Scores ≈ 50% indicate successful evasion; the original's score > 50% shows high baseline vulnerability to OMLA.**

**Table 2: Area overhead [%]: AttackGNN vs *NetDeTox* on OMLA. Abbreviations: AG = AttackGNN, G4oM = GPT-4o-mini, L4 = LLaMA-4 Maverick 17B, DS-V3 = Deepseek-V3, Q3 = Qwen-3 235B, G2.5 = Gemini-2.5 Flash, G5 = GPT-5.**

| Case | AG | G4oM | L4 | DS-V3 | Q3 | G2.5 | G5 |
|---|---|---|---|---|---|---|---|
| c1355 | 121.93 | 138.88 | 116.64 | 144.48 | 106.61 | 111.59 | **72.55** |
| c1908 | 80.28 | 44.51 | **24.79** | 43.54 | 42.01 | 32.50 | 59.51 |
| c2670 | 77.23 | 39.69 | **9.24** | 11.12 | 75.89 | 69.87 | 10.15 |
| c3540 | 50.57 | **17.57** | 28.96 | 34.44 | 34.04 | 35.00 | 23.61 |

*Overhead Status:* baseline worse better best

**Table 3: Area overhead: AttackGNN vs *NetDeTox* on GNN4IP.**

| Case | AG | G4oM | L4 | DS-V3 | Q3 | G2.5 | G5 |
|---|---|---|---|---|---|---|---|
| c432-BE280 | 48.55 | 32.51 | **16.23** | 27.25 | 33.64 | 21.70 | 27.35 |
| c432-CS320 | 99.69 | 76.55 | **25.31** | 46.74 | 129.04 | 96.43 | 228.88 |
| c432-CS640 | 114.04 | **16.02** | 75.44 | 53.22 | 113.45 | 115.20 | 130.76 |
| c432-CS1280 | 121.80 | **8.29** | 88.31 | 78.12 | 115.72 | 149.61 | 154.90 |
| c432-RN320 | 71.43 | 72.17 | 25.60 | **20.68** | 97.47 | 56.40 | 169.20 |
| c432-RN640 | 105.76 | 116.08 | 155.43 | 108.31 | **90.91** | 95.01 | 121.62 |
| c432-RN1280 | 115.87 | 137.00 | **8.86** | 141.37 | 131.97 | 115.64 | 108.09 |
| c432-SL320 | 96.79 | 114.89 | 40.29 | **35.91** | 155.77 | 79.71 | 73.43 |
| c432-SL640 | 125.89 | 199.64 | 87.95 | **22.43** | 184.61 | 183.77 | 126.61 |
| c432-SL1280 | 127.95 | 184.01 | **92.00** | 97.25 | 127.46 | 195.07 | 176.66 |
| c499-CS320 | 81.56 | **54.87** | 81.03 | 92.28 | 85.38 | 87.03 | 122.41 |
| c499-CS640 | 87.36 | 87.55 | 102.94 | **84.36** | 126.82 | 101.92 | 87.23 |
| c499-CS1280 | 93.49 | 92.55 | 93.10 | **27.71** | 107.54 | 72.49 | 110.95 |
| c499-RN320 | 78.49 | 82.86 | **75.37** | 84.20 | 98.22 | 90.65 | 128.56 |
| c499-RN640 | 84.70 | 92.61 | **78.92** | 82.71 | 100.71 | 87.34 | 163.95 |
| c499-RN1280 | 92.50 | 90.70 | 76.95 | **-8.35** | 102.05 | 84.05 | 85.05 |
| c499-SL320 | 77.31 | 78.96 | 85.30 | 89.25 | 89.93 | 94.40 | 99.33 |
| c499-SL640 | 83.57 | 101.86 | 110.01 | **78.75** | 99.29 | 84.34 | 118.16 |
| c499-SL1280 | 92.86 | 77.13 | 68.75 | **56.75** | 52.63 | 85.81 | 91.32 |
| c880-CS320 | 12.75 | 0.74 | -1.16 | **-2.02** | -1.35 | 0.67 | -0.61 |
| c880-CS640 | 11.35 | -0.76 | 1.38 | **-3.15** | 2.29 | -0.19 | -0.48 |
| c880-CS1280 | 11.08 | 0.89 | -4.60 | **-13.77** | -1.36 | 2.22 | 1.23 |
| c880-CS2560 | 10.58 | -2.35 | 3.99 | -0.52 | **-3.27** | -1.08 | 1.23 |
| c880-RN320 | 12.37 | **-2.51** | -1.82 | -1.32 | **-2.51** | -0.13 | -0.75 |
| c880-RN640 | 10.04 | -0.39 | -0.24 | **-5.74** | -0.63 | -0.63 | -2.75 |
| c880-RN1280 | 10.13 | **-6.89** | 0.68 | 0.81 | 4.22 | 0.54 | 0.00 |
| c880-RN2560 | 9.31 | 2.40 | -12.51 | **-15.21** | -1.82 | 1.29 | 6.11 |
| c880-SL320 | 13.39 | -2.58 | 1.11 | **-3.07** | -0.86 | 0.31 | -2.40 |
| c880-SL640 | 14.15 | 0.53 | 3.42 | 3.66 | **-0.63** | 2.55 | 0.43 |
| c880-SL1280 | 11.20 | -1.41 | -3.13 | -4.58 | -2.02 | **-5.99** | -3.77 |
| c880-SL2560 | 9.93 | -5.34 | **-26.80** | -26.69 | -1.14 | 3.67 | 6.11 |

*Overhead Status:* baseline worse better best

## 5.3 Case Study 2: GNN4IP and AttackGNN

**Security Evaluation:** For GNN4IP, successful evasion requires similarity scores ≤ 0, indicating designs appear unrelated. *NetDeTox* achieves this threshold in 90.3% of cases (168/186 model-benchmark pairs, Fig. 4), with LLaMA-4 and DeepSeek-V3 consistently reaching ≤ 0 across synthesis flows. Interestingly, some benchmarks like c499 achieve near-zero scores across all backends, demonstrating successful evasion even on familiar training circuits. In comparison, AttackGNN leaves scores between 0 and −1, while *NetDeTox* pushes almost all designs toward −1.

**Overhead Analysis:** Notably, 41.9% of all cases show area reductions (Tab. 3). For instance, `c880-RN320` achieves a security score of −0.991 with -2.51% area overhead. This occurs because localized subnetlist rewrites enable better area control during structural changes, allowing us to discover more efficient implementations while evading detection. Even when area increases, our results remain substantially better than AttackGNN's overhead. This demonstrates that security and efficiency are not necessarily conflicting objectives: *NetDeTox* can harden designs while reducing area.

## 5.4 Case Study 3: GNN-RE and AttackGNN

**Security Evaluation:** For GNN-RE, successful evasion requires classification accuracy ≤25%. *NetDeTox* reduces GNN-RE's accuracy from 100% to as low as 4.9%, which is a 95.1% drop (Fig. 5). Notably, DeepSeek-V3 and Gemini-2.5 consistently achieve ≤25% across 141/144 cases spanning 4 to 32-bit benchmarks, indicating robust evasion independent of circuit scale. AttackGNN also pushes accuracy into the 0–25% range, meaning that *NetDeTox* matches its proven evasion effectiveness.

**Overhead Analysis:** Tab. 4 shows *NetDeTox* consistently outperforms AttackGNN overheads (54-113%) across GNN-RE benchmarks. Despite GNN-RE's larger circuit sizes, where area overhead becomes more critical, *NetDeTox* maintains overhead mostly below 40%, with few cases achieving area reduction. LLM selection significantly impacts overhead, with GPT-4o-mini, DeepSeek-V3 and Qwen-3 consistently achieving lower costs, enabling flexible area-security trade-offs.

**Table 4: Area overhead: AttackGNN vs *NetDeTox* on GNN-RE.**

| Case | AG | G4oM | L4 | DS-V3 | Q3 | G2.5 | G5 |
|---|---|---|---|---|---|---|---|
| add_mul_4bit | 65.06 | 33.42 | 60.00 | 36.96 | 35.44 | 42.28 | **27.59** |
| add_mul_8bit | 75.54 | 15.49 | 28.86 | 49.77 | 14.17 | 17.89 | **13.26** |
| add_mul_16bit | 62.76 | **3.51** | 6.35 | 28.14 | 12.44 | 4.97 | 93.05 |
| add_mul_32bit | 72.83 | 3.35 | **1.72** | 10.34 | 4.27 | 12.46 | 9.92 |
| add_mul_comb_4bit | 71.85 | **31.09** | 44.28 | 59.53 | 40.47 | 36.95 | 36.36 |
| add_mul_comb_8bit | 78.95 | 23.66 | 27.80 | 22.75 | 18.67 | 15.57 | **15.33** |
| add_mul_comb_16bit | 54.19 | 8.08 | **-3.19** | 32.14 | 11.07 | 3.38 | 3.78 |
| add_mul_comb_32bit | 73.68 | 6.02 | **0.44** | 4.36 | 6.21 | 18.27 | 4.92 |
| add_mul_comp_4bit | 61.45 | 32.65 | 14.51 | 37.64 | 28.80 | 27.89 | **32.43** |
| add_mul_comp_8bit | 73.09 | 9.31 | **5.92** | 14.75 | 6.62 | 84.18 | 19.70 |
| add_mul_comp_16bit | 61.44 | 6.49 | 10.27 | 12.59 | **3.25** | 92.43 | 3.45 |
| add_mul_comp_32bit | 71.87 | 24.54 | 13.14 | **4.76** | 4.48 | 24.16 | 16.01 |
| add_mul_comp_sub_4bit | 60.26 | 19.34 | 17.24 | 25.00 | **13.03** | 17.37 | 18.82 |
| add_mul_comp_sub_8bit | 70.65 | 3.79 | **3.51** | 6.30 | 8.41 | 6.94 | 5.14 |
| add_mul_comp_sub_16bit | 62.45 | 9.47 | 2.73 | **1.21** | 3.59 | 31.51 | 7.36 |
| add_mul_comp_sub_32bit | 70.65 | 8.90 | 11.14 | 10.20 | **4.55** | 25.96 | 5.15 |
| add_mul_mix_4bit | 75.24 | 38.59 | 80.83 | 41.75 | 44.90 | 54.61 | **33.50** |
| add_mul_mix_8bit | 81.32 | **13.00** | 34.88 | 14.60 | 29.15 | 91.18 | 14.60 |
| add_mul_mix_16bit | 77.00 | 2.64 | 4.43 | 12.84 | 4.24 | 55.89 | **2.10** |
| add_mul_mix_32bit | 113.02 | 6.99 | 9.05 | 7.30 | **6.90** | 22.86 | 11.71 |
| add_mul_sub_4bit | 61.90 | 25.40 | 23.23 | 18.18 | 21.36 | **17.17** | 26.26 |
| add_mul_sub_8bit | 71.95 | 33.16 | **1.48** | 6.72 | 9.84 | 6.46 | 10.94 |
| add_mul_sub_16bit | 63.63 | 16.89 | 3.35 | 10.93 | 4.29 | 9.10 | **-0.38** |
| add_mul_sub_32bit | 71.69 | 9.45 | **1.71** | 5.67 | 4.22 | 65.51 | 2.63 |

*Overhead Status:* baseline worse better best

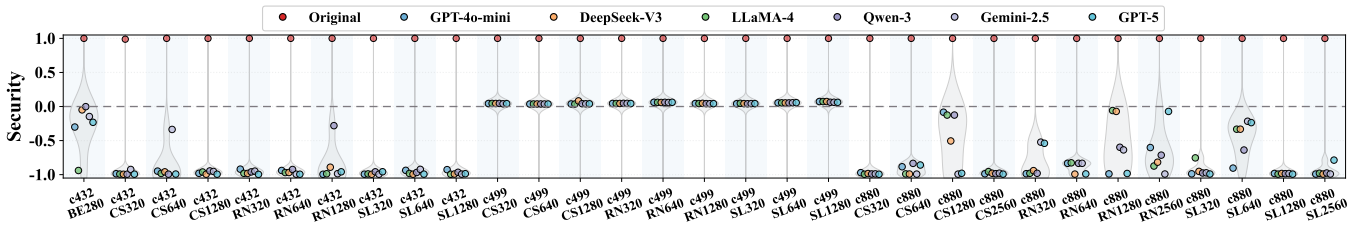


**Figure 4: Security of *NetDeTox* netlists against GNN4IP. Original refers to the baseline design. Scores $< 0$ indicate successful evasion; the original's score $\approx 1$ shows high baseline vulnerability to GNN4IP.**

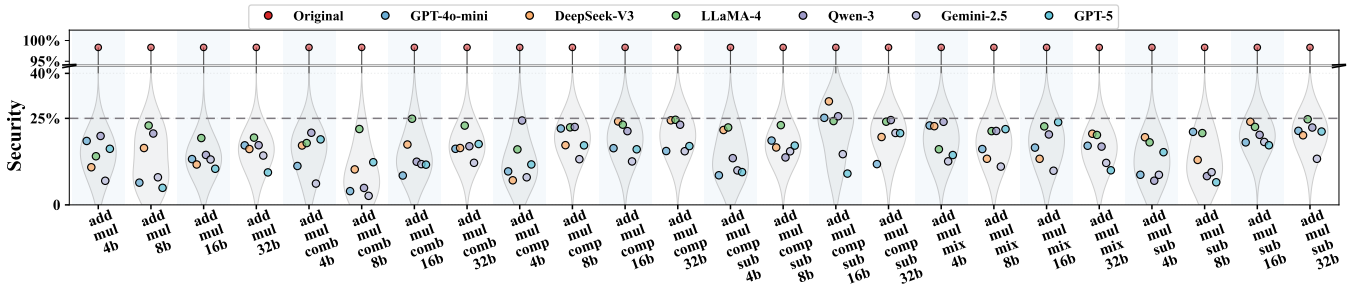


**Figure 5: Security of *NetDeTox* netlists against GNN-RE. Original refers to the baseline design. Scores $\leq 25\%$ indicate successful evasion; the original's score $\approx 100\%$ shows high baseline vulnerability to GNN-RE.**

## 5.5 Summary for Comparison with AttackGNN

Across all three tasks, *NetDeTox* consistently breaks OMLA (23/24 pairs at $\sim$ 50%), GNN4IP (168/186 at $\leq 0$), and GNN-RE (141/144 at $\leq$ 25%) across six LLM backends. Beyond robustness, we frequently observe negative overhead, i.e., *NetDeTox* produces more area-efficient circuits than the originals, thanks to efficient exploration of local structures, with scalability consistent across circuit sizes. Backend-agnostic effectiveness, localized rewriting, and predictable scaling redefine the security–cost trade-off.

## 5.6 Case Study 4: GNN4IP and LLMPirate

**Security Evaluation:** Recall the previous discussion for *NetDeTox* netlists against GNN4IP (Sec. 5.3). For LLMPirate, most LLMs (CoPilot, GPT-3.5, GPT-4, Claude, and CodeLlama-13B) evade GNN4IP on most netlists, whereas CodeLlama-7B and Llama3-8B succeed on only 10 and 11 out of 32, respectively, indicating limited flexibility.
**Overhead Analysis:** Our method significantly improves over LLMPirate [11]. LLMPirate incurs $>$ 200% gate-count overheads, whereas *NetDeTox* achieves comparable security with substantially lower overheads (Tab. 5). Notably, we observe negative overheads, -9.91% (DeepSeek-V3) and -27.38% (LLaMA-4), indicating that ours finds much more efficient adversarial structures. This difference comes from *NetDeTox*'s LLM reasoning about area overhead during planning, selecting gate combinations that minimize cost while altering structures. In contrast, LLMPirate applies local transformations without area awareness, accumulating uncontrolled overheads.
**Takeaway:** Context-aware planning enables *NetDeTox* to achieve negative overheads, avoiding LLMPirate's $>$ 200% gate inflation through area-conscious transformation selection, all while achieving similar evasion performance.

**Table 5: Gate-Count Overheads for GNN4IP.**

| Model | Overhead Dist. | Avg | Min | Std |
|---|---|---|---|---|
| GPT-4o-mini | | **103.49%** | 3.70% | 1.03 |
| DeepSeek-V3 | | **82.93%** | -9.91% | 0.74 |
| LLaMA-4 | | **81.08%** | -27.38% | 0.82 |
| Qwen-3 235B | | **124.91%** | 3.35% | 1.04 |
| Gemini-2.5 Flash | | **124.73%** | 3.35% | 1.13 |
| GPT-5 | | **138.48%** | 4.23% | 1.16 |

## 5.7 Ablation Studies

*5.7.1 RL and LLM Integration .* Here we study the benefits of the proposed hybrid LLM+RL integration within *NetDeTox*. For example, for OMLA on `c3540` (Fig. 6), we see closely matching attack strength for LLM+RL and LLM-only. However, with about half the area overhead (20.73% vs 42.26%) and converging significantly faster (18 vs 38 iterations), LLM+RL is superior. RL-only stalls at 56.10% security strength after 49 iterations. This reveals complementary roles: RL identifies *where* to transform (high-impact locations), while LLM determines *how* to transform (context-aware rewriting). Neither component alone achieves both effective evasion and area.
**Takeaway:** RL+LLM achieves 50% less overheads and 50% faster convergence than LLM-only, while RL-only underperforms for evasion, clearly demonstrating that both components are necessary.

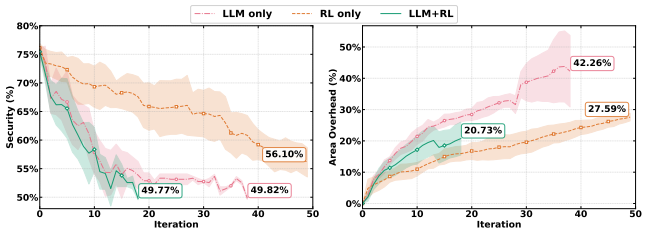


**Figure 6: Ablation case study on OMLA for `c3540`, averaged over five runs. Shaded areas indicate variance.**

*5.7.2 RL Queries.* Tab. 6 shows that *NetDeTox* reduces RL queries significantly over the RL-only approach in AttackGNN, providing up to 272× speedup. This finding aligns with faster convergence noted above. The benefits accrue in all three case studies, i.e., query-efficiency benefits generalize to diverse attack settings.
**Takeaway:** Beyond security-efficiency trade-offs, *NetDeTox* delivers runtime gains of over 100×, enabling large-scale deployment.

**Table 6: Average RL Queries for AttackGNN vs *NetDeTox*.**

| Dataset | G4oM | L4 | DS-V3 | Q3 | G2.5 | G5 | ↓ Queries |
|---|---|---|---|---|---|---|---|
| OMLA | 21.75 | **11.00** | 17.75 | 23.25 | 13.50 | 16.00 | 227.27× |
| GNN4IP | 14.90 | 9.77 | **9.16** | 19.06 | 22.68 | 23.52 | 272.89× |
| GNN-RE | 39.24 | 57.08 | **16.88** | 57.08 | 121.25 | 86.25 | 148.15× |

*5.7.3 LLM Planning Order.* Here we evaluate whether planning order influences security and area. Without loss of generality, for `c880-CS2560` on GNN4IP, all possible planning orders are executed five times (Fig. 7), and the resulting efficiencies are analyzed. MLH (Mapping–Location–Hop size) and LMH (Location–Mapping–Hop size) outperform other schedules, achieving area reductions efficiently while maintaining the target security. Considering both final area and security, LMH offers the best balance and is adopted as the default for plan generation.
**Takeaway:** LMH ordering balances security and area most effectively, showing that planning sequence directly shapes evasion–efficiency trade-offs.

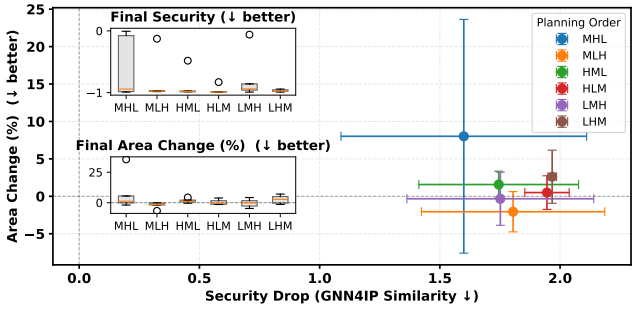


**Figure 7: Efficiency of different planning orders on security-area trade-offs. M denotes subnetlist mapping selection, L denotes gate selection, and H denotes hop size selection.**

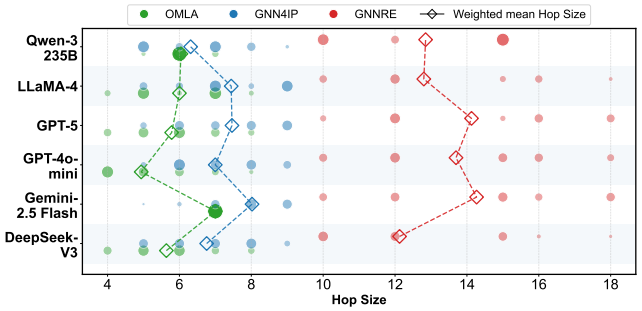


**Figure 8: LLM-preferred hop-size strategies for OMLA, GNN4IP, and GNN-RE. Bubble size indicates relative helpfulness for evasion, while diamonds show weighted mean hop sizes for top-5 choices.**

*5.7.4 LLM Planning Tuning.* Figs. 8 and 9 reveal how hop size and subnetlist mapping affect evasion effectiveness across tasks and LLM backends. Hop-size sensitivity varies by task: OMLA achieves best results at smaller hops ($h$ = 4-8) as it targets localized key-gate structures and was trained on subgraphs with $h \leq 3$, while GNN-RE benefits from larger contexts ($h$ = 12-16) due to its module-level classification. GNN4IP remains relatively stable across hop sizes, as its graph-level similarity detection is less sensitive to localized transformations. Mapping selection correlates with hop sizes: smaller hops ($h$ = 4-8) predominantly use mappings C01-C04, while larger hops ($h \geq 12$) favor mappings C08-C09, and other mappings (C06, C07) see limited usage due to their heavy reliance on XOR/XNOR gates (which introduce area overhead despite effectively altering structure). No single mapping dominates—optimal choices depend on the target GNN and backend. DeepSeek-V3 and Gemini-2.5 show stable performance across configurations, while GPT-5 exhibits higher variance, suggesting different LLMs have varying sensitivity to transformation granularity.
**Takeaway:** Optimal hop sizes and mappings vary across attacks, and LLM-driven tuning ensures security with low overhead.

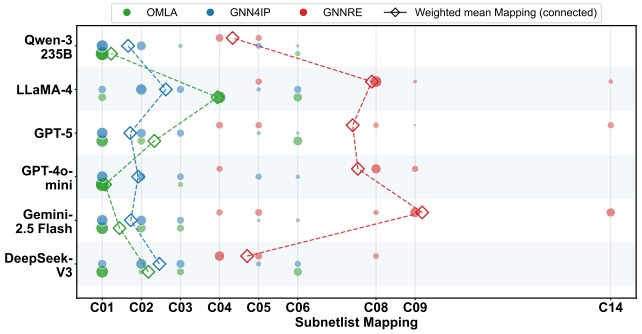


**Figure 9: LLM-preferred subnetlist mapping strategies for OMLA, GNN4IP, and GNN-RE.**

## 6 Conclusion

We presented *NetDeTox*, a novel framework that combines LLM planning with RL targeting to evade GNN-based security tools through subnetlist rewriting. Focusing on compact subnetlists rather than full circuits or isolated gates, *NetDeTox* enables efficient, function-preserving edits evading detection. Compared to prior SOTA, AttackGNN, *NetDeTox* reduces area overheads significantly and achieves comparable (sometimes even better) effectiveness while evading OMLA, GNN4IP, and GNN-RE, all with consistent effectiveness over six diverse LLM backends and circuit scales. Ablation studies show RL and LLM supplement each other well, as only their joint interaction achieves lower overheads and faster convergence than LLM-only and RL-only approaches. Future work should explore LLM-based synthesis transformations [17], further evaluation metrics [21], and other prospects for evasion in other security tasks like side-channel [6] or fault-injection [14] attacks.

## References


[1] [n. d.]. NanGate FreePDK45 Generic Open Cell Library. https://si2.org/open-cell-and-free-pdk-libraries/.

[2] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. 2021. OMLA: An oracle-less machine learning-based attack on logic locking. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 3 (2021), 1602–1606.

[3] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. 2021. GNN-RE: Graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 8 (2021), 2435–2448.

[4] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D C, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, Balasubramanyan Ashok, and Shashank Shet. 2024. Codeplan: Repository-level coding using llms and planning. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 675–698.

[5] Jitendra Bhandari, Johann Knechtel, Ramesh Narayanaswamy, Siddharth Garg, and Ramesh Karri. 2024. Llm-aided testbench generation and bug detection for finite-state machines. *arXiv preprint arXiv:2406.17132* (2024).

[6] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*. Springer, 16–29.

[7] Animesh Basak Chowdhury, Marco Romanelli, Benjamin Tan, Ramesh Karri, and Siddharth Garg. 2024. Retrieval-guided reinforcement learning for boolean circuit minimization. *arXiv preprint arXiv:2401.12205* (2024).

[8] Ziad El Sayed, Zeng Wang, Hana Selmani, Johann Knechtel, Ozgur Sinanoglu, and Lilas Alrahis. 2024. Graph Neural Networks for Integrated Circuit Design, Reliability, and Security: Survey and Tool. *Comput. Surveys* (2024).

[9] Wenji Fang, Wenkai Li, Shang Liu, Yao Lu, Hongce Zhang, and Zhiyao Xie. 2025. NetTAG: A Multimodal RTL-and-Layout-Aligned Netlist Foundation Model via Text-Attributed Graph. *arXiv preprint arXiv:2504.09260* (2025).

[10] Wenji Fang, Jing Wang, Yao Lu, Shang Liu, and Zhiyao Xie. 2025. Geneda: Unleashing generative reasoning on netlist via multimodal encoder-decoder aligned foundation model. *arXiv preprint arXiv:2504.09485* (2025).

[11] Vasudev Gohil, Matthew DeLorenzo, Veera Vishwa Achuta Sai Venkat Nallam, Joey See, and Jeyavijayan Rajendran. 2024. Llmpirate: Llms for black-box hardware ip piracy. *arXiv preprint arXiv:2411.16111* (2024).

[12] Vasudev Gohil, Satwik Patnaik, Dileep Kalathil, and Jeyavijayan Rajendran. 2024. {AttackGNN}:{Red-Teaming}{GNNs} in Hardware Security Using Reinforcement Learning. In *33rd USENIX Security Symposium (USENIX Security 24)*. 73–90.

[13] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716* (2024).

[14] Ayush Jain, M Tanjidur Rahman, and Ujjwal Guin. 2020. Atpg-guided fault injection attacks on logic locking. In *2020 IEEE Physical Assurance and Inspection of Electronics (PAINE)*. IEEE, 1–6.

[15] Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. 2023. Llm-assisted generation of hardware assertions. *arXiv e-prints* (2023), arXiv–2306.

[16] Hazem Lashen, Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu. 2023. Trojansaint: Gate-level netlist sampling-based inductive learning for hardware trojan detection. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[17] Xihan Li, Xing Li, Lei Chen, Xing Zhang, Mingxuan Yuan, and Jun Wang. 2024. Circuit Transformer: A Transformer That Preserves Logical Equivalence. *arXiv preprint arXiv:2403.13838* (2024).

[18] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. 2007. *ABC: A System for Sequential Synthesis and Verification*. Technical Report. University of California, Berkeley.

[19] Ruidi Qiu, Grace Li Zhang, Rolf Drechsler, Ulf Schlichtmann, and Bing Li. 2024. Autobench: Automatic testbench generation and evaluation using llms for hdl design. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*. 1–10.

[20] Prithwish Basu Roy, Akashdeep Saha, Manaar Alam, Johann Knechtel, Michail Maniatakos, Ozgur Sinanoglu, and Ramesh Karri. 2025. Veritas: Deterministic Verilog Code Synthesis from LLM-Generated Conjunctive Normal Form. *arXiv preprint arXiv:2506.00005* (2025).

[21] Minghao Shao, Abdul Basit, Ramesh Karri, and Muhammad Shafique. 2024. Survey of Different Large Language Model Architectures: Trends, Benchmarks, and Challenges. *IEEE Access* 12 (2024), 188664–188706. doi:10.1109/ACCESS.2024.3482107

[22] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. 2024. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems* 29, 3 (2024), 1–31.

[23] Zeng Wang, Lilas Alrahis, Likhitha Mankali, Johann Knechtel, and Ozgur Sinanoglu. 2024. Llms and the future of chip design: Unveiling security risks and building trust. In *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 385–390.

[24] Zeng Wang, Minghao Shao, Jitendra Bhandari, Likhitha Mankali, Ramesh Karri, Ozgur Sinanoglu, Muhammad Shafique, and Johann Knechtel. 2025. VeriContaminated: Assessing LLM-driven verilog coding for data contamination. *arXiv preprint arXiv:2503.13572* (2025).

[25] Zeng Wang, Minghao Shao, Rupesh Karn, Jitendra Bhandari, Likhitha Mankali, Ramesh Karri, Ozgur Sinanoglu, Muhammad Shafique, and Johann Knechtel. 2025. SALAD: Systematic Assessment of Machine Unlearning on LLM-Aided Hardware Design. *arXiv preprint arXiv:2506.02089* (2025).

[26] Zeng Wang, Minghao Shao, Mohammed Nabeel, Prithwish Basu Roy, Likhitha Mankali, Jitendra Bhandari, Ramesh Karri, Ozgur Sinanoglu, Muhammad Shafique, and Johann Knechtel. 2025. Verileaky: Navigating ip protection vs utility in fine-tuning for llm-driven verilog coding. *arXiv preprint arXiv:2503.13116* (2025).

[27] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[28] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys-a free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, Vol. 97.

[29] Zhiyuan Yan, Wenji Fang, Mengming Li, Min Li, Shang Liu, Zhiyao Xie, and Hongce Zhang. 2025. Assertllm: Generating hardware verification assertions from design specifications via multi-llms. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*. 614–621.

[30] Rozhin Yasaei, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. 2021. Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1504–1509.

[31] Rozhin Yasaei, Shih-Yuan Yu, Emad Kasaeyan Naeini, and Mohammad Abdullah Al Faruque. 2021. GNN4IP: Graph neural network for hardware intellectual property piracy detection. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 217–222.

[32] Lizi Zhang, Azadeh Davoodi, and Rasit Onur Topaloglu. 2025. ReBERT: LLM for Gate-Level to Word-Level Reverse Engineering. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–7.

[33] Guangwei Zhao and Kaveh Shamsi. 2024. Adversarial Circuit Rewriting against Graph Neural Network-based Operator Detection. *ACM Transactions on Design Automation of Electronic Systems* 30, 1 (2024), 1–19.