

# Automated Hardware Trojan Insertion in Industrial-Scale Designs

Yaroslav Popryho<sup>✉</sup>, Debjit Pal<sup>✉</sup>, and Inna Partin-Vaisband<sup>✉</sup>

*Department of Electrical and Computer Engineering*

*University of Illinois Chicago, Chicago, IL, USA*

{ypopry2, dpal2, vaisband}@uic.edu

**Abstract**—Industrial Systems-on-Chips (SoCs) often comprise hundreds of thousands to millions of nets and millions to tens of millions of connectivity edges, making empirical evaluation of hardware-Trojan (HT) detectors on realistic designs both necessary and difficult. Public benchmarks remain significantly smaller and hand-crafted, while releasing truly malicious RTL raises ethical and operational risks. This work presents an *automated and scalable* methodology for generating HT-like patterns in industry-scale netlists whose purpose is to stress-test detection tools without altering user-visible functionality. The pipeline (i) parses large gate-level designs into connectivity graphs, (ii) explores *rare* regions using SCOAP testability metrics, and (iii) applies parameterized, function-preserving graph transformations to synthesize trigger-payload pairs that mimic the statistical footprint of stealthy HTs. When evaluated on the benchmarks generated in this work, representative state-of-the-art graph-learning models fail to detect Trojans. The framework closes the evaluation gap between academic circuits and modern SoCs by providing reproducible challenge instances that advance security research without sharing step-by-step attack instructions.

## I. INTRODUCTION

Modern Systems-on-Chip (SoCs) integrate tens to hundreds of millions of *logic* gates at the gate level, (standard-cell instances on the order of  $10^7$ – $10^8$ +) with overall gate-equivalents commonly reach into the billions, while total transistor counts are in the tens of billions. At this scale, the practical risk around hardware Trojans (HTs) is not only *insertion* but the difficulty of *finding* stealthy, low-activity malicious reconvergent logic buried deep in the design netlist. This detection challenge intensifies as designs scale out and heterogeneity increases, where weakly exercised logic and long reconvergences are commonplace.

The community relies heavily on TrustHub [1] and circuits derived from ISCAS’85/’89 [2]. These are invaluable for taxonomy and reproducibility, but their scale is orders of magnitude smaller than contemporary SoCs: ISCAS’85/’89 designs typically contain *thousands* of gates, and TrustHub instances frequently wrap or modify these baselines, with the maximum of tens of thousands of gates. In contrast, production *gate-level* netlists routinely comprise *tens to hundreds of millions of standard-cell instances*, and—counting large

This work was supported in part by the National Science Foundation under Grant No. 2238976, titled CAREER: Unified Reference-Free Early Detection of Hardware Trojans via Knowledge Graph Embeddings.

SRAM macros in NAND2-equivalents—often reach the multi-billion-gate scale [3]. This produces a persistent simulation-to-silicon gap: detectors that excel on academic circuits often fail to generalize under industrial class imbalance. Closing this gap requires benchmarks and procedures that are faithful to industrial scale and data skew.

Moreover, hand-crafted HTs bias benchmarks toward human-designed motifs and cannot cover the long tail of “rare, hard-to-drive, hard-to-observe” patterns. Existing automated insertion efforts either target narrow settings or still evaluate primarily on small designs [4], [5]. What is missing is an *automated and scalable* methodology that (i) preserves I/O functionality by construction, (ii) emulates stealthy statistical signatures (rare triggers; long reconvergences; benign payload equivalents), and (iii) produces precise per-net and per-cone labels and metadata. These requirements motivate the methodology introduced next and define how artifacts should look to meaningfully stress-test detectors.

This paper introduces an *automated and scalable* methodology for generating *Trojan-like, function-preserving patterns* in industry-size designs for the sole purpose of evaluating and stress-test HT detectors. Rather than prescribing attack guidelines, the framework focuses on **how to stress-test detectors** using automatically synthesized structural anomalies that mimic the statistical footprint of stealthy triggers and payloads. The pipeline is designed to operate over millions of edges, to integrate with commodity EDA tooling, so that artifacts remain useful for research yet safe to share.

Given an industrial-scale gate-level netlist and a set of graph-based HT detectors, the objective is to automatically produce *Trojanized netlists*, i.e., netlist mutations that: (i) preserve original I/O functionality and timing constraints, (ii) embed *rare-triggered* nodes (low controllability/observability; see Section II-B) within *cones of influence* (COIs), and (iii) are labeled at the net and cone level to support supervised and semi-supervised evaluation. By constraining I/O behavior while varying the difficult-to-control internal structure, functional correctness can be decoupled from detection difficulty, and sensitivity can be quantified under controlled conditions.

This creates the missing benchmark dataset on which existing graph/machine learning (ML)-based frameworks [6]–[9] can be *empirically* evaluated for both effectiveness and

scalability on industrial-scale designs, *under the extreme class imbalance inherent to HT detection (i.e., Trojan-labeled nets/COIs are significantly rare—typically  $\ll 0.1\%$  of all nodes/re-gions; see Sec. II-B)*, enabling head-to-head comparisons. Specifically, beyond the methodology itself, the primary contributions of this paper are as follows:

- a *rare-node mining* strategy grounded in SCOAP testability and structural constraints that yields diverse, stealthy instances stressing HT detectors,
- a *reproducible evaluation framework* reporting detector ability to detect and localize stealthy/unseen Trojans, resource use, and ablations across designs with millions of nodes and edges, and
- a *benchmark dataset* with per-cone labels, metrics, and splits for fair comparison at scale.

The rest of this paper is organized as follows. Section II defines HT triggers/payloads; explores detector families and their scalability; details SCOAP for fast rarity mining and quantified scale gaps when transferring from small benchmarks to modern SoCs. Section III presents the methodology, including graph construction, rare-node mining, and instrumentation guardrails. Section IV reports large-scale experiments and sensitivity analyses and discusses their limitations. Section V concludes and outlines directions for future research.

## II. BACKGROUND

Hardware Trojans (HTs) are malicious modifications that lie dormant under normal operation and activate under rare conditions. A typical HT comprises a *trigger* (see Fig. 1)—often a low-probability internal event such as a deep state pattern, a long counter value, or an uncommon handshake—and a *payload* (see Fig. 2) that once activated perturbs behavior including data leakage, control override, denial of service, or subtle performance degradation. Effective HTs minimize switching activity and structural footprint, favoring deep, reconvergent regions with naturally low controllability and observability. This threat model creates two coupled challenges at system scale: identifying statistically rare internal events under extreme class imbalance and doing so on designs with millions of nodes and edges.

```

1 // Triggers on a symmetric, rare data pattern.
2 assign trigger_signal = (par_dataH[7:4] == {
   par_dataH[0], par_dataH[1], par_dataH[2],
   par_dataH[3]});

```

Figure 1. Example of hardware Trojan trigger (Palindrome Data).

```

1 // Causes state corruption across multiple
   transactions.
2 // Original line: else if (rstCountH) recd_bitCntH
   <= 0;
3 else if (rstCountH && !trigger_signal) recd_bitCntH
   <= 0;

```

Figure 2. Example of hardware Trojan payload (Reset Disable).

### A. Detector families and their scalability

**Formal verification.** Property checking, equivalence checking, and information-flow verification (e.g., GLIFT/IFS) offer strong guarantees when specifications are precise and complete [10]–[12]. The most visible downsides are (i) *scalability*:

state-space growth and deep sequential logic make proofs expensive on industrial blocks; (ii) *specification burden*: non-trivial effort is required to write and maintain comprehensive properties or security policies; (iii) *reference dependence*: many flows assume a golden model, trusted micro-architecture intent, or policy baselines that may be unavailable; and (iv) *resolution*: passing a property check may not localize suspicious gates/nets, leaving engineers to manually triage large COIs once violations occur. These factors limit coverage for unseen Trojans that fall outside the encoded properties.

**Side-channel analyses.** Power, timing, electromagnetic, or thermal measurements can *detect* HTs without full design visibility and are applicable *pre-silicon* (simulation/emulation) and *post-silicon* on fabricated chips; they do not imply inserting Trojans after fabrication [13], [14]. The most obvious drawbacks are (i) *noise and variability*: process, voltage, temperature, and measurement noise reduce sensitivity and reproducibility; (ii) *localization*: signatures are spatially coarse, especially in multi-domain SoCs, making it hard to pinpoint malicious nodes; (iii) *reference or calibration needs*: many techniques rely on golden chips, golden traces, or extensive calibration runs; and (iv) *limited pre-silicon utility*: modeling side-channels accurately before tape-out is challenging, which delays actionable feedback.

**Graph- and ML-based** detectors learn over netlists to classify nodes or logic COIs (e.g., GNN4TJ at RTL, gate-level GNNs, HW2VEC) [6], [9], [15], [16]. They often need no golden reference, but face practical limits: large feature/edge tensors, multi-hop growth on high-fan-out/reconvergent logic, and distribution shift from small benchmarks to SoC-scale designs. Sampling can drop the long paths that link rare triggers to payloads; partitioning improves memory but may sever those paths or split COIs near rare nodes, undermining detection exactly where it matters.

### B. SCOAP testability: metrics and rarity

Testability metrics quantify how difficult it is to control or observe internal signals. SCOAP (*Sandia Controllability/Observability Analysis Program*) [17] assigns three nonnegative integers to each net  $n$ : combinational controllability for logic 0,  $CC0(n)$  (effort to force logic 0), combinational controllability for logic 1,  $CC1(n)$  (effort to force logic 1), and combinational observability,  $CO(n)$  (effort to propagate  $n$  to a primary output). By convention, primary inputs have  $CC0 = CC1 = 1$ , and primary outputs have  $CO = 0$ . For a combinational network in topological order, common recurrences include:

$$\text{INV: } CC0(y) = CC1(x) + 1, \quad CC1(y) = CC0(x) + 1, \\ CO(x) = CO(y) + 1,$$

$$\text{AND: } CC1(y) = \sum_i CC1(x_i) + 1, \\ CC0(y) = \min_i CC0(x_i) + 1, \\ CO(x_i) = CO(y) + \sum_{j \neq i} CC1(x_j) + 1,$$

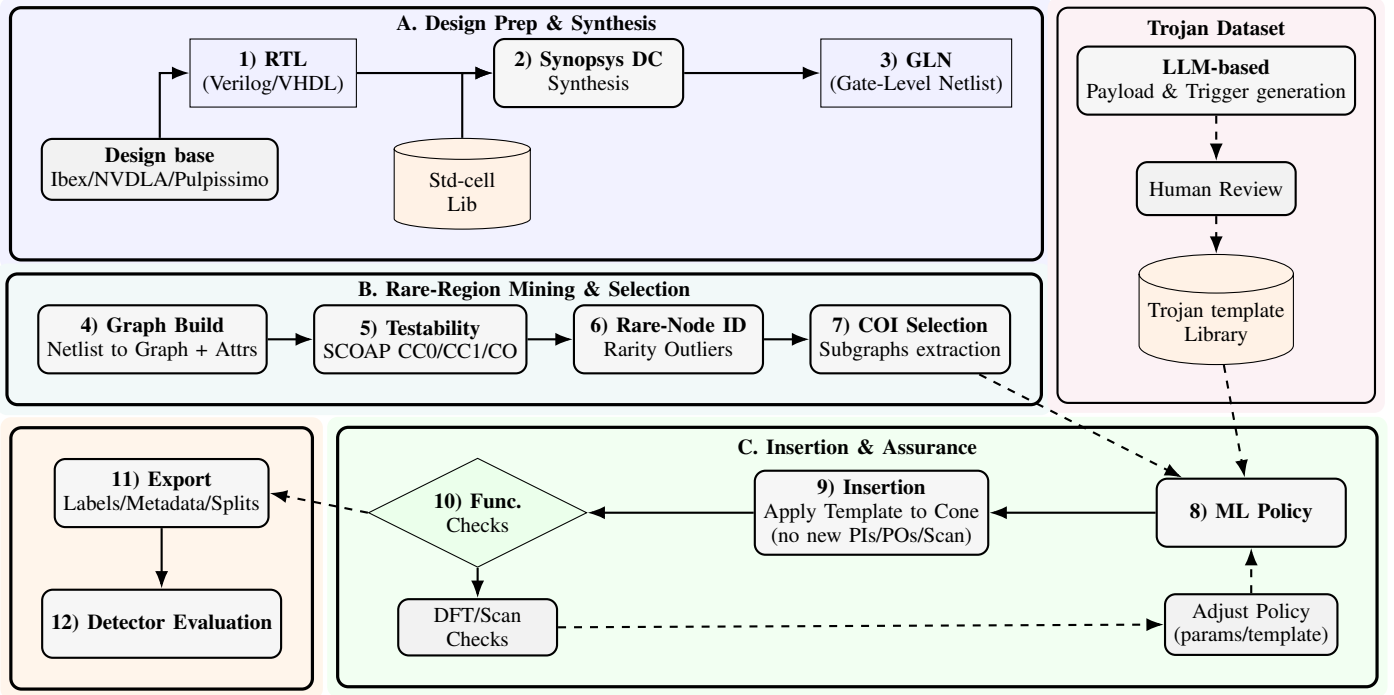


Figure 3. End-to-end pipeline. RTL and constraints are synthesized with **Synopsys DC** to a gate-level netlist (GLN). The GLN is converted to a graph with attributes; **SCOAP** yields testability scores used to identify *rare* nodes and realistic COIs. A function-preserving *template library* (populated via LLM-assisted trigger & payload generation under human review) is combined with an **ML-based policy** to select placements and parameters. Inserted patterns must preserve functionality, **DFT/scan** checks, and **STA/area** budgets; failures trigger parameter readjustment. Successful instances are exported with labels/metadata/splits for reproducible detector evaluation. *No new PIs/POs/scan behaviors are introduced at any stage.*

$$\begin{aligned} \text{OR: } CC0(y) &= \sum_i CC0(x_i) + 1, \\ CC1(y) &= \min_i CC1(x_i) + 1, \\ CO(x_i) &= CO(y) + \sum_{j \neq i} CC0(x_j) + 1. \end{aligned}$$

For two-input XOR/XNOR, controllabilities follow the usual pairwise minima of mixed sums; observabilities are analogous and depend on the easier of the two input values. Sequential circuits are handled by cutting at registers (treating flip-flop outputs as pseudo-inputs and inputs as pseudo-outputs) and evaluating per cycle.

In practice, these integers are combined into a rarity score, for example:

$$R(n) = \max\{CC0(n), CC1(n)\} + \alpha CO(n), \quad \alpha \in [0.5, 1], \quad (1)$$

and nets are ranked by percentiles to isolate outliers. A *higher* score indicates a net that is difficult to activate and/or difficult to propagate to an output (rarely activated under normal conditions), whereas a *lower* score indicates a net that is comparatively easy to drive and observe (frequently activated). In this paper, the ranking guides the placement of observe/control points, the focus of specification-driven formal checks, and the emphasis of COIs during ML training.

### C. Need for unseen Trojans and diverse benchmarks

Robust detection requires evaluation on *unseen* trigger-payload combinations that differ statistically from training

data and that reflect realistic class imbalance. A suitable benchmark dataset should preserve I/O functionality by construction for safe, comparable results; span a range of trigger sparsities and reconvergence depths; include benign payload equivalents that mimic footprint without harm; and provide precise per-net and per-cone labels for HT detectors. With such artifacts ML-based detectors can be trained under realistic conditions rather than overfitting to a narrow family of Trojans.

## III. METHODOLOGY

This section details the end-to-end flow, as illustrated in Fig. 3. For each, the unmodified gate-level netlist (GLN) synthesized with *Synopsys DC* serves as the golden baseline. Synthesized netlists and constraints are produced with standard combinational optimizations, preserving test structures. The resulting gate level netlists span from  $\sim 6.5 \times 10^5$  to  $\sim 1.0 \times 10^7$  edges across designs (see Table I), and the overall generation time—parsing through export—ranges from a median  $\sim 7$  min at  $\approx 735$  k edges to  $\sim 64.8$  min at  $\approx 9.7$  M edges, with memory rising near the 140 GB. Runtime and memory distribution across stages are shown in Table II.

Following *Graph Build* (Fig. 3B-(4)), each GLN is converted into a directed connectivity graph whose nodes represent nets (optionally gates) and whose edges represent driver  $\rightarrow$  load relations. Lightweight structural attributes are attached to every node, including gate class, fan-in/fan-out, topological depth, local reconvergence signatures derived from

Table I  
SCALING BEHAVIOR OF CREATED DESIGNS.

Design	# of Nodes	# of Edges	Parse Time [min]	AST Peak RAM [GB]
D1	30K	184K	1.8	2.3
D2	212K	426K	4.1	10.4
D3	120K	735K	7.1	29.1
D4	367K	2.24M	22.2	65.9
D5	883K	4.95M	42.1	120.5
D6	1.24M	7.42M	53.2	131.7
D7	1.78M	9.68M	64.7	140.5

Table II  
RUNTIME AND PEAK RAM BY STAGE.

Stage	D3 Time [min]	D4 Time [min]	Peak RAM [GB]
Parsing/Graph Build	5.8	19.8	14.5
SCOAP (forward/backward)	4.5	16.2	11.2
Candidate Mining	2.2	7.0	6.8
Template Insertion	1.9	6.2	4.1
Functionality checks	1.3	2.8	3.6
Labeling/Export	0.8	2.7	2.2
<b>Total (median)</b>	16.5	54.7	15.2

input-cone overlap, and distances to design interfaces and sequential cut points. The flow then executes *Testability/SCOAP* (Fig. 3B-(5)) to compute CC0, CC1, and CO in linear time; rarity is ranked using the score  $R(n)$  as defined in (1). *Rare-Node ID* (Fig. 3B-(6)) selects outlier nets by  $R(\cdot)$  percentile and filters them by structural context typical of stealthy regions—such as reconvergent fan-in and branching into disjoint neighborhoods—so that candidates exhibit both low activity and realistic connectivity (see the stylized COI in Fig. 4). *COI Selection* (Fig. 3B-(7)) then extracts compact subgraphs around these rare nets, assembling/compiling per-subgraph metadata (size, depth to interfaces and registers, reconvergence descriptors) for the downstream policy.

Trigger-payload candidates are produced with an *Large-Language Model* (LLM). The model is used *offline* to build a fixed dataset of stealthy trigger mechanisms and pass-through payload behaviors that serve as reusable templates (the “Trojan Dataset” in Fig. 3). The LLM prompt is designed to encourage diversity in both activation mechanisms and payload effect. On the trigger side, this includes patterns such as sequence detectors, counter windows, parity or Hamming checks, and inactivity- or timing-based anomalies. On the payload side, representative behaviors includes guarded multiplexers, shadow paths, inert togglers, and small offsets or bit flips. These representative families of trigger and payload

mechanisms are summarized in Table III. The initial LLM-generated candidates are normalized into typed descriptors—trigger family, payload family, and parameter ranges such as tap count, local depth, permissible fan-out growth, and reconvergence tolerance. Each descriptor is then *compiled* into a local graph rewrite—a small, function-preserving replacement in the gate level *connectivity graph*. Concretely, within a bounded neighborhood an untriggered path is substituted with a trigger-payload template, while all primary inputs/outputs are left untouched. Each rewrite is validated with equivalence checking (following the flow [18]): the “golden” and “trojanized” netlists are read, verification is restricted to the edited COI, compare points are mapped on the COI boundary, the trigger is constrained to its inactive value, and `verify cmd` is run to either prove equivalence or return a counterexample. Logic outside the COI is black-boxed to accelerate the check.

The *ML Policy* is implemented as a compact multilayer perceptron (MLP) with a two-input design that decides *where* to place a given template of triggers and payloads. For every candidate, the pipeline constructs (i) a feature vector that summarizes the *subgraph of rare nets*, including its rarity percentiles, reconvergence indicators, and distances between design elements, and (ii) a feature vector that encodes the *template choice and parameters*, including trigger/payload pair, number of taps, and the intended local insertion pattern. These two vectors are concatenated and passed through the MLP, which outputs two interpretable scores: an *acceptance score* estimating whether the proposed candidate will pass the required checks, and a *stealthy score* estimating how statistically challenging the instance will be for detectors. A simple weighted combination ranks all candidates. The MLP is trained

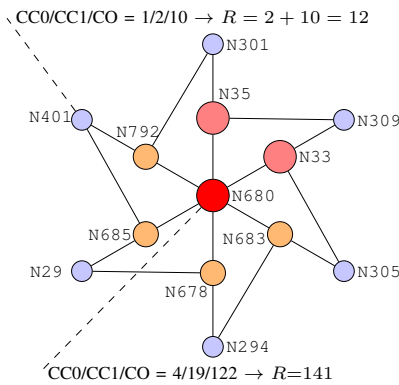


Figure 4. An example of rarity-annotated subgraph. Connections indicate edges that reconverge toward a central high- $R$  node. Nodes are labeled by net name; color encodes rarity  $R$  (blue  $\rightarrow$  orange  $\rightarrow$  red for low  $\rightarrow$  medium  $\rightarrow$  high), Example SCOAP values (calculated based on (1) w/  $\alpha = 1$ ) are shown for N680, and N401.

Table III  
SAMPLE OF HARDWARE TROJAN TRIGGERS AND PAYLOADS

Name	Mechanism
<b>Triggers</b>	
3-State Sequence	Activates only when the FSM follows a specific, rare path.
Hamming Distance	Fires when input data is very close to a secret value.
Watchdog Timer	Activates from the prolonged absence of a specific event.
Glitch Detector	Triggers on a transient, single-cycle pulse on a signal line.
Hash-like Combo	Fires on a non-obvious combination of data and internal state.
<b>Payloads</b>	
Timing Leak	Delays a critical output signal, leaking data via timing.
Off-by-One Error	Subtly corrupts data by adding one to an output value.
FSM Deadlock	Forces the state machine into a valid state but prevents it from leaving.
Counter Drift	Causes a sampling counter to miscount, slowly degrading operation.
Reset Disable	Prevents a key register from resetting, corrupting future operations.

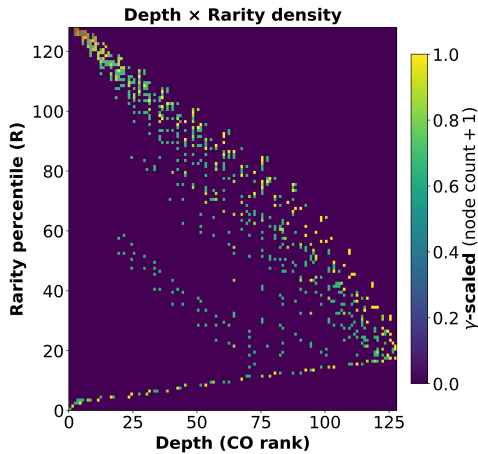


Figure 5. Nodes rarity heatmap for a SoC with many shallow logic COIs (the *Ibex RISC-V CPU* core). As expected, set of rare nodes shifts to the *right* because deeper nets (higher CO) usually raise *R*. Yet, extremely rare nets can be identified at modest depth (upper-left) when controllability is very poor.

in a self-supervised manner using the pipeline’s own history: candidates that passed all checks become positive examples; candidates that failed equivalence or DFT/scan checks become structured negatives. Over time, the policy learns to prefer subgraphs and parameterizations that are both likely to be accepted and measurably hard for detectors, without requiring a golden reference model.

*Insertion* (Fig. 3C-(9)) applies the selected template as a local graph rewrite (deterministic, template-driven splice) inside the chosen subgraph of rare nets. The tool extracts a small cone of influence (bounded by flip-flops) around a rare net and inserts a parameterized trigger–payload template whose *inactive branch is a pass-through* (i.e., it leaves the signal unchanged, equivalent to a wire). Every insertion is subjected to checks to ensure *functionally unperturbed* modification (forcing trigger to be inactive; setting *scan\_en*, *test\_mode* to 0) (Fig. 3C-(10)); any mismatch triggers automatic rollback and negative feedback to the policy. In parallel, *DFT/Scan Checks* verify that scan chains, test enables, and observability/controllability of scan cells are unchanged; candidates that alter test behavior are rejected and recorded as policy negatives.

Ideally, Trojan triggers should be inserted deep inside the logic path and must be activated rarely. However, there are examples of SoCs, where the logic is broken into many short, simple stages separated by registers (*Ibex RISC-V CPU* core). Figure 5 shows that scenario: nodes rarity grows with depth because combinational observability, CO, contributes directly to  $R(n) = \max\{CC0(n), CC1(n)\} + \alpha CO(n)$ , ( $\alpha = 1$ ), yet the upper-left cluster reveals nets that are extremely rare even at modest depth when controllability is very poor. This reveals an insertion trade-off. While targeting the rarest trigger might lead to a logic COI of minimal depth that is controlled by a long, weakly observable path, pursuing greater logic depth increases activation rarity via CO at the risk of incurring significant timing and routing overhead.

Finally, the *Export* stage assembles/compiles each success-

ful instance into a reproducible benchmark: the “Trojanized” GLN, including per-net attributes and metadata, and fixed train/validation/test splits at the subgraph (COI) level. Since the insertions are function-preserving by construction and approved by equivalence and DFT/scan checks, the resulting set provides industrial-scale, challenge instances that reflect realistic rarity and reconvergence, enabling empirical, head-to-head evaluation of detection methods under the same class imbalance and scale observed in modern SoCs.

#### IV. EXPERIMENTAL RESULTS

The evaluation focuses on whether state-of-the-art detectors trained on modest, publicly available designs can reliably identify Trojans embedded in much larger, structurally different designs with *unseen* trigger–payload characteristics.

Open-source implementations of representative graph and ML-based detectors are selected to reflect the strongest publicly documented baselines. The study includes the following Trojan detectors.

- **HW2VEC** (HOST’21) [9]: an automated framework that extracts RTL or gate-level graphs (AST/DFG/GLN) and feeds a graph-classification backend. In experimental setup, the backend uses a *GNN4TJ* graph classifier [6] following the reference repository configuration file: [19].
- **Netwise** (TCAD’24) [20]: a gate-level, node-wise detector that embeds local neighborhoods into “embedding clouds” and applies scalable convolutions to reduce memory pressure and improve generalization at SoC scale.
- **NetVGE** (TCAD’25) [21]: a KGE+transformer pipeline over weighted variable-dependency graphs that combines an entity transformer (local interactions) with a context transformer (global dependencies) and weighted self-attention emphasizes rarely accessed nodes.

Models are trained on the TrustHub dataset [1] that contains both clean netlists and their Trojanized counterparts.

The evaluation is performed on the proposed larger-scale designs, with samples shown in Table I. These designs are held out from the training and validation sets to ensure a realistic test of generalization, as it mirrors the real-world scenario where a model must identify unseen Trojans in a target SoC whose architecture and its characteristics are not present during training. The ability of the baseline framework to process the generated benchmarks and detect Trojans is shown in Fig. 6. In terms of netlist parsing, all detectors successfully handle designs D1–D3. However, only the KGE-based methods, *NetVGE* [21] and *Netwise* [20], scale to D4–D7, whereas the GNN-based solution fails to process the largest netlists.

##### A. HW2VEC performance at SoCs scale

HW2VEC with the GNN4TJ model is designed to operate on data-flow graphs and aggregate data over local neighborhoods. When applied to gate-level SoCs, achieving a receptive field large enough to connect deep, low-activity triggers to their corresponding payloads requires traversing

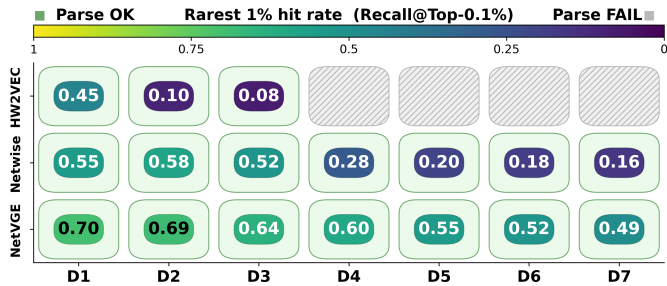


Figure 6. HT detectors capability matrix

numerous hops through high fan-out, reconvergent logic. In such case, the resulting activation/memory footprint grows superlinearly in edge count, and practical mitigation techniques (e.g., neighbor sampling, hard partitioning) sever precisely the long-range dependencies needed for rare-predicate reasoning. On D1, HW2VEC shows capability to detect presence of rarely triggered nodes in design, but does not correctly identify the trigger COI. The sensitivity of the HW2VEC detector to rare nodes drops sharply at D2 and becomes statistically indistinguishable from random chance in the 99–100% rarity tail. The results shows that these constraints manifest as (i) parse/memory failures on D4–D7, and (ii) a collapse of rare-tail recall starting at D2. The distributional gap between TrustHub-scale training graphs and multi-million-edge test graphs further amplifies this degradation.

### B. NetVGE/Netwise performance on large gate-level graphs

In contrast to the GNN model, both KGE-based methods retain *rare-tail* sensitivity by design. *Netwise* aggregates local neighborhoods into embedding clouds and applies scalable set of convolutions, reducing reliance on very deep message passing and preserving rare-node signal through mid-scale designs. As a result, its sensitivity degrades on D4–D7, where triggers and payloads are separated by long, rarely activated, and reconvergent paths within the control and datapath logic.

The *NetVGE* method encodes weighted variable-dependency tuples, embeds them with KGE, and scores them in latent space using a hierarchical transformer. This architecture consists of an entity transformer for local interactions and a context transformer for global dependencies, with a weighted self-attention mechanism that emphasizes rarely accessed nodes. The approach was evaluated on designs D2–D7, where it demonstrated the ability to detect triggers located in the rarest circuit locations (99<sup>th</sup> percentile). As shown in Fig. 6, it continues to localize affected logic regions at a level where other detectors fail to scale, although overall accuracy remains modest (49%–70%).

Both detectors succeed in highlighting *portions* of the affected COIs (rare nodes near the trigger gates) but fall short of localizing the precise trigger logic, underscoring the difficulty of long-range dependencies tracking under extreme class imbalance.

### C. Key observations

Based on the experiments, general-purpose GNNs cannot reliably detect *unseen*, stealthy Trojans at SoC scale when trained on much smaller publicly available benchmarks. This limitation is not only a matter of data volume. Linking rare triggers to their downstream effects requires a long-range view of the circuit, which conflicts with practical memory limits and the graph partitioning strategies used to manage them. These strategies, while necessary, disrupt the logical pathways that a detector must follow. The proposed artifact set therefore serves a dual purpose: it provides SoC-scale challenge instances with *unseen* Trojans, and it highlights the need for model designs that reason hierarchically over large graphs, incorporate testability-aware signals, and preserve long-range relationships without incurring prohibitive resource costs.

## V. CONCLUSION

An automated, scalable methodology is presented for inserting Trojan-like, function-preserving patterns into industrial-scale netlists to stress-test HT detectors under conditions that mirror modern SoCs. The flow converts large gate-level netlists into graphs, mines statistically rare regions using SCOAP testability, and applies local rewrites drawn from a fixed LLM-generated library of trigger–payload templates that leave user-visible behavior unchanged and pass equivalence and DFT/scan checks. A lightweight placement policy, trained on pipeline feedback, selects where to instantiate templates so that accepted insertions are both feasible and statistically stealthy. The resulting benchmarks are labeled in a self-supervised at the net and subgraph levels, enabling reproducible evaluations at scale.

Empirical results indicate that representative state-of-the-art graph-based detectors, trained on smaller public designs, do not generalize to unseen, industrial-size targets populated with unseen trigger–payload families. Precision–recall performance collapses as rarity and reconvergence increase, and attempts to preserve long-range context through a deeper aggregation clash with practical memory and runtime limits. These observations suggest the gap is architectural rather than merely data-driven: detectors must learn to reason over large graphs hierarchically, incorporate testability-aware structure, and retain long-range dependencies without prohibitive resource growth.

The proposed methodology provide a controlled and scalable way to synthesize realistically stealthy challenge instances while keeping I/O behavior intact, narrowing the evaluation gap between academic benchmarks and production-scale designs. Limitations remain: rarity mining relies primarily on SCOAP with sequential cuts, payloads are pass-through analogs rather than truly malicious effects, and experiments focus on a set of open designs. Future work will extend rarity analysis with sequential and path-sensitization signals, broaden template diversity within the same function-preserving envelope, and investigate hierarchical, testability-aware models and hybrid formal/learning strategies that can scale to the connectivity and imbalance regimes of modern SoCs.

## REFERENCES

- [1] H. Salmani, M. Tehranipoor, and R. Karri, "Trust-Hub Hardware Trojan Benchmarks." <https://www.trust-hub.org>.
- [2] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Targeted Translator in FORTRAN," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Kyoto, Japan, 1985.
- [3] Synopsys. (2025) STA Strategies for Fast and Efficient Signoff in Multi-Billion Instance Designs. Describes PrimeTime HyperScale for hundreds of millions to billions of instances. [Online]. Available: <https://www.synopsys.com/articles/sta-strategies-multi-billion-instance-designs.html>
- [4] Z. Pan and P. Mishra, "Automated Test Generation for Hardware Trojan Detection using Reinforcement Learning," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC '21)*. ACM, 2021, pp. 408–413.
- [5] H. Chen, X. Zhang, K. Huang, and F. Koushanfar, "AdaTest: Reinforcement Learning and Adaptive Sampling for On-chip Hardware Trojan Detection," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 2, 2023.
- [6] R. Yasaei, S. Yu, and M. A. A. Faruque, "GNN4TJ: Graph Neural Networks for Hardware Trojan Detection at Register Transfer Level," in *Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.
- [7] D. Cheng, C. Dong, W. He, Z. Chen, and Y. Xu, "GNN4Gate: A Bi-Directional Graph Neural Network for Gate-Level Hardware Trojan Detection," in *Design, Automation & Test in Europe Conference (DATE)*, 2022, pp. 1315–1320, eDAA.
- [8] K. Hasegawa, K. Yamashita, S. Hidano, K. Fukushima, K. Hashimoto, and N. Togawa, "Node-Wise Hardware Trojan Detection Based on Graph Learning," *IEEE Transactions on Computers*, vol. 74, no. 3, pp. 749–761, 2025.
- [9] S. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. A. Faruque, "HW2VEC: A Graph Learning Tool for Automating Hardware Security," in *2021 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 13–23.
- [10] M. Tiwari, X. Li, H. M. G. Wassel, F. T. Chong, and T. Sherwood, "Complete Information Flow Tracking from the Gates Up," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, 2009, pp. 109–120.
- [11] W. Hu, J. Oberg, M. Tiwari, T. Sherwood, and R. Kastner, "Gate-Level Information Flow Tracking for Security Lattices," *ACM Transactions on Design Automation of Electronic Systems*, vol. 20, no. 1, pp. 2:1–2:25, 2014.
- [12] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, and M. Tehranipoor, "Hardware Trojan Detection through Information Flow Security Verification," in *Proceedings of the IEEE International Test Conference (ITC)*. IEEE, 2017.
- [13] M. Du, F. Li, Y. Li, and K. Wei, "A Scalable Hardware Trojan Detection Methodology for CUTs Using Statistical Analysis of Side-Channel Information," in *Cryptographic Hardware and Embedded Systems – CHES 2010*, ser. Lecture Notes in Computer Science, vol. 6225. Springer, 2010, pp. 328–342.
- [14] T. Su, Y. Wang, S. Xu, L. Zhang, S. Feng, J. Song, Y. Liu, Y. Tang, Y. Zhang, S. Li, Y. Guo, and H. Liu, "Improving the Ability of Thermal Radiation Based Hardware Trojan Detection via Noise-Induced Pixel Occupation Enhancement (NICE)," in *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 2024)*. Philadelphia, PA, USA: USENIX Association, Aug 2024. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity24/presentation/su-ting>
- [15] H. Lashen, L. Alrahis, J. Knechtel, and O. Sinanoglu, "TrojanSAINT: Gate-Level Netlist Sampling-Based Inductive Learning for Hardware Trojan Detection," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023, pp. 1–5.
- [16] D. Cheng, C. Dong, W. He, Z. Chen, X. Liu, and H. Zhang, "A Fine-Grained Detection Method for Gate-Level Hardware Trojan Based on Bidirectional Graph Neural Networks (GateDet)," *Journal of King Saud University – Computer and Information Sciences*, 2023, in press.
- [17] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program," in *Proceedings of the 17th Design Automation Conference (DAC)*. ACM, 1980, pp. 190–196.
- [18] Synopsys, Inc. (2024) Formality and Formality Ultra Datasheet. [Online]. Available: <https://www.synopsys.com/content/dam/synopsys/verification/datasheets/formality-and-formality-ultra-ds.pdf>
- [19] AICPS Lab. (2021) HW2VEC `example_gnn4tj.yaml` configuration. [Online]. Available: [https://github.com/AICPS/hw2vec/blob/master/examples/example\\_gnn4tj.yaml](https://github.com/AICPS/hw2vec/blob/master/examples/example_gnn4tj.yaml)
- [20] D. Utyamishv and I. Partin-Vaisband, "Netwise Detection of Hardware Trojans using Scalable Convolution of Graph Embedding Clouds," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3116–3128, oct 2024.
- [21] Y. Popryho, D. Pal, and I. Partin-Vaisband, "NetVGE: Netwise Hardware Trojan Detection at RTL Using Variable Dependency and Knowledge Graph Embedding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.