Expert Systems With Applications 297 (2026) 129355

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

SubG4TJ: A collaborative subgraph classification method based on multidimensional attributes for hardware trojan detection

Xing Hu[a,b], Yang Zhang[a,b,*], Sheng Liu[a,b], Xiaowen Chen[a,b], Yaohua Wang[a,b], Zhenyu Zhao[a,b], Yang Guo[a,b], Keqin Li[c]

[a] College of Computer Science and Technology, National University of Defense Technology, Changsha, 410073, Hunan, China
[b] Key Laboratory of Advanced Microprocessor Chips and Systems, Changsha, 410073, Hunan, China
[c] Department of Computer Science, State University of New York, New York, NY, USA

ARTICLE INFO

Keywords:
Hardware trojans
Less-toggle attribute
Subgraph classification
Graph neural network

ABSTRACT

The proliferation of third-party Intellectual Property cores in Integrated Circuit design has introduced significant security risks, particularly the threat of hardware Trojans (HTs). Existing HT detection methods based on single-gate granularity analyze individual gate attributes but overlook the multi-gate structure of HTs, leading to excessive false positives and heavy manual verification. Comparatively, full-netlist analysis methods, while comprehensive, suffer from poor locating precision and scalability issues in large-scale netlists. Subgraph-based approaches aim to overcome these limitations but often rely on non-feature-based, non-collaborative techniques such as random sampling, to build subgraphs, incurring high computational overhead and low detection accuracy.

To address these challenges, we propose SubG4TJ, a collaborative subgraph-based framework that integrates intrinsic gate attributes with structural connectivity for accurate and efficient HT detection. SubG4TJ identifies a specific class of gates – termed Less-Toggle Gates (LTGs) – that exhibit asymmetric 0/1 controllability, and introduces a concealment-based metric to quantify this property. By combining this intrinsic indicator with inter-gate structural features that capture Trojan interconnectivity, SubG4TJ enables targeted subgraph extraction centered on LTGs, substantially reducing analysis scope. A graph neural network then jointly model nodes' concealment attributes and subgraphs' topology features, enabling collaborative and end-to-end classification of suspicious subgraphs. This joint modeling of controllability asymmetry and structural connectivity allows SubG4TJ to effectively balance detection accuracy, false positive suppression, and runtime scalability. Evaluations in various benchmark circuits demonstrate that SubG4TJ improves true positive rate by up to 13.6 % and achieves a speed-up of 120×, with performance gains increasing in larger designs.

1. Introduction

The Integrated Circuit (IC) design industry has seen a rise in third-party suppliers providing standalone Intellectual Property (IP) cores. This trend is driven by the increasing complexity of IC designs and specialized labor divisions, which attract chip design companies that want to save costs and accelerate development. However, security concerns during IP integration remain critical and cannot be overlooked. Third-party IP cores, provided by external suppliers, inherently exist significant security risks, making them vulnerable to hardware Trojans (HTs). Malicious adversaries can stealthily alter logic gates within an IP core, posing a serious threat to the security and integrity of IC designs.

To address these threats, two mainstream strategies have emerged: protection and detection. Protection-based methods often require intrusive design modifications, such as logic locking or runtime monitoring, which may introduce performance overheads and are difficult to enforce in third-party scenarios where internal design access is limited. In contrast, detection-based approaches provide a more practical solution by analyzing the circuit structure or behavior to identify suspicious components without altering the original design. As a result, HT detection – particularly at the gate level – has become a focal point of hardware security research.

To this end, many researchers have proposed gate-level HT detection techniques. Existing single-gate methods aim to classify each gate

* Corresponding author at: organization = College of Computer Science and Technology, National University of Defense Technology, city = Changsha, postcode = 410073, state = Hunan, country = China.

E-mail addresses: huxing@nudt.edu.cn (X. Hu), zhangyang@nudt.edu.cn (Y. Zhang), liusheng83@nudt.edu.cn (S. Liu), xwchen@nudt.edu.cn (X. Chen), nudtyh@foxmail.com (Y. Wang), zyzhao@nudt.edu.cn (Z. Zhao), guoyang@nudt.edu.cn (Y. Guo), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.eswa.2025.129355
Received 26 June 2025; Received in revised form 8 August 2025; Accepted 11 August 2025
Available online 15 August 2025

as either HT or HT-free, focusing on anomalies in individual gates. However, these methods often generate a high number of false positives and require significant manual effort to verify whether flagged gates are genuinely part of an HT. On the other hand, full-netlist methods evaluate the entire netlist for the presence of HTs. While these methods offer a holistic assessment, they lack the precision needed to localize HTs within the design. Both approaches face scalability challenges when applied to large designs, especially AI-based methods that demand significant computational and memory resources.

To solve the above problems in single-gate and full-netlist detection methods, subgraph-based methods are proposed, which focus on localized sections of the netlist. This approach is particularly advantageous for HT detection because HTs typically consist of both trigger and payload logic, which involve multiple gates and span only a small portion of the entire netlist. Thus, subgraph classification methods can offer a more targeted approach for detecting HTs by focusing on smaller, localized sections of the netlist, where HTs are more likely to reside. However, existing subgraph-based approaches, such as random or uniform sampling, often suffer from inefficiency and lack specificity. These methods typically operate without guidance from HT characteristics, resulting in the indiscriminate generation of subgraphs. As a consequence, they tend to produce an excessive number of irrelevant candidates – especially in large-scale designs – leading to substantial computational overhead.

To highlight HTs from large-scale netlists using subgraph-based methods, it is essential to exploit both intrinsic and structural characteristics of HTs. The intrinsic property refers to the stealthiness of individual gates, often manifested as 'less-toggle' behavior. We propose a concealment metric to capture this attribute across diverse HT structures, enabling the identification of suspicious gates and guiding subsequent subgraph construction. In parallel, structural characteristics describe the connectivity patterns among Trojan-infected gates, which frequently form compact or sparsely connected clusters that differ from the topology of benign logic. To jointly leverage these two dimensions, we propose **SubG4TJ** – a collaborative subgraph classification method based on multidimensional attributes for hardware Trojan detection. SubG4TJ integrates both concealment-based node-level features and inter-gate graph structural relationships to construct multidimensional subgraph representations, which are then classified using a Graph Neural Network (GNN). This approach enables accurate and efficient HT detection by jointly capturing the behavioral stealth of individual gates and their malicious structural organization. By constraining the analysis to LTG-centered regions and embedding both concealment and structural semantics into the GNN, the framework reduces redundant computation, suppresses false positives, and scales effectively with netlist complexity. The key contributions are as follows.

(1) We propose a concealment-based metric to identify LTGs by quantifying the asymmetric 0/1 controllability of logic gates. This metric demonstrates strong robustness across diverse HT structures and effectively distinguishes abnormal gates even in circuits with highly controllable and balanced normal logic, such as AES.
(2) We develop a targeted LTG-aware subgraph extraction strategy that anchors on LTGs to selectively expand structurally correlated regions. By using LTGs as focal points, the method suppresses irrelevant logic propagation and yields compact, high-relevance subgraphs that improve both detection accuracy and computational efficiency.
(3) We propose a collaborative subgraph classification framework that integrates multidimensional attributes – specifically, intrinsic gate concealment attribute and inter-gate structural features – into a unified representation. This joint modeling improves detection robustness, suppresses false positives, and remains compatible with various GNN backbones such as Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT). Experiments demonstrate a true positive rate improvement of up to 13.6 %, along with a speedup of up to 120×.

The paper is structured as follows: Section 2 reviews related work, Section 3 outlines the motivation, Section 4 details our methodology, Sections 5 and 6 present experimental validation and results, and Section 7 concludes the paper.

2. Related works

Detection methods for HTs can be broadly categorized into functional verification analysis, circuit analysis, and machine learning methods.

2.1. Functional verification analysis

Functional verification analysis focuses on examining the functional behavior of the hardware design to identify circuit modules that may potentially contain HTs. Unused UCI [Hicks et al. (2010)] assumes the presence of Trojan logic in inactive signal pairs during testing but is limited to simple circuit paths. FANCI [Waksman et al. (2013)] uses control values to flag suspicious signals, focusing on combinatorial logic, and its extension, the FIGHT Metric, addresses sequential elements by modifying directed graphs. VeriTrust [Zhang et al. (2013)] aims to detect non-redundant inputs through simulation, emphasizing boundary condition checks during verification but struggles with practical application in large-scale designs. FASTrust [Yao et al. (2015)] introduces a feature-based analysis for IP verification, independent of simulation results, and uses multiple functions to detect HTs but faces challenges with new HT types and false positives. These methods highlight the evolving complexity and varied approaches in the field of HT detection.

The above methods utilize functional verification for HT detection. However, comprehensive functional descriptions are often challenging to obtain in third-party IP cores, which results in a higher false positive rate for such methods.

2.2. Circuit analysis

Circuit analysis focuses on detecting HTs by analyzing circuit characteristics, such as their inherently low controllability and minimal switching activity, which are deliberately designed to evade detection. Early detection strategies, such as Salmani's COTD method [Salmani (2017)], utilize these features and are further developed by researchers like Xie [Xie et al. (2017)], Kok [Kok et al. (2019b)], Salmani [Salmani (2022)] and Lo [Lo et al. (2022)]. HuangHuang and He (2020) combines static and dynamic methods to enhance detection using k-means clustering. Su [Su et al. (2021)] links low transition probabilities with poor testability, proposing a trigger signal recognition method for Trojan models with reasonable testability. Subsequent optimizations of the COTD method by Tebyanian [Tebyanian et al. (2021)] and Mehta [Mehta and Popat (2022)] introduce strategic Trojan insertions and a switching rate-based detection approach. Lu [Lu et al. (2021)] employs unsupervised methods like information entropy and density-based clustering with HTDet, while Zou [Zou et al. (2018)] focuses on total state transition probabilities, albeit limited by the need for finite state machine knowledge. Jha [Jha and Jha (2008)] and Popat [Popat and Mehta (2016)] use signal probability signatures for detection during manufacturing phases. Further studies, such as those by Kurihara [Kurihara et al. (2020)] and Chen [Chen et al. (2018)], introduce structured features and a multi-level verification framework to analyze HTs in third-party digital IP cores. Haider's HaTCh [Haider et al. (2019)] method expands HT definitions and employed the EM algorithm for detection. Kok [Kok et al. (2019a)] combines various features to train machine learning classifiers, enhancing detection capabilities. Lastly, references [Dupuis et al. (2015)], [Dupuis et al. (2018)] and [Chen et al. (2025)] use thresholds on switching activity probabilities to differentiate HTs from normal gates.

Circuit analysis identifies rare active signals, which may include both normal circuit signals and Trojan signals. Some strongly concealed signals, despite being part of normal functionality, are flagged as potential hardware Trojan signals, leading to a high false positive rate. This results in increased manual intervention, a problem that becomes even more pronounced in large-scale circuits.

2.3. Machine learning methods

2.3.1. Machine learning methods for detection in single gate granularity

Machine learning methods at the single gate granularity focus on analyzing individual gates within the netlist to identify potential Trojan triggers. Hasegawa [Hasegawa et al. (2025)] introduces a novel node-by-node detection method. Reference [Yamashita et al. (2022)] employs a machine learning method to model the 24 extracted features for hardware Trojan detection. Additionally, techniques [Yasaei et al. (2025, 2022)] to convert register-transfer level (RTL) and gate-level netlists into data flow graphs have been proposed to enhance the detection capabilities.

Research has also taken a hierarchical and adversarial approach to detection. Hepp [Hepp et al. (2022)] develops a method for detecting HTs in unknown hierarchical design modules using machine learning classifiers. Similarly, Hasegawa [Hasegawa et al. (2023)] and Nozawa [Nozawa et al. (2021)] have focused on adversarial training and the generation of hardware Trojan attack instances to improve detection strategies, emphasizing the importance of robust and adaptive detection systems to protect hardware from malicious modifications.

Node classification provides fine-grained detection by predicting the label of each individual node in the netlist. However, this granularity comes with significant challenges. Hardware Trojans often manifest as localized anomalies involving multiple gates, making individual nodes insufficient to capture meaningful patterns. Moreover, in large-scale netlists with millions of nodes, node-level classification becomes computationally prohibitive. The lack of contextual information also makes node classification highly susceptible to noise, leading to a higher risk of misclassification.

2.3.2. Machine learning methods for detection in sub-netlist granularity

Sub-netlist granularity detection methods focus on analyzing localized netlist sections, making them well-suited for detecting HTs that span multiple gates. Yu [Yu et al. (2022)] introduce a system that extracts and classifies netlist path information using a data-driven approach. Techniques like GramsDet [Lu et al. (2019)] and LMDet [Shen et al. (2018)] use deep learning models to detect abnormal patterns within sub-netlists. These methods conceptualize gate sequences as analogous to sentences in a language, where gates are "words" and their connections form "sentences". GramsDet learns embeddings for these sequences, while LMDet applies language models to detect deviations from normal gate sequence behaviors. Muralidhar [Muralidhar et al. (2021)] employs supervised contrastive learning using GNNs to detect triggered HTs by constructing subgraphs for each signal. Verification and testing for hardware Trojan implantation are conducted on specific IP cores available in the open-source library [Home (2025)]. However, the GATE Net method constructs a subgraph for each signal in the gate-level netlist, employing it for learning and signal analysis. The number of subgraphs equals the number of gate nodes in the netlist, and each subgraph's nodes consist of all nodes from the analyzed node to the input. Consequently, this leads to a higher node count in each subgraph. Additionally, Lashen [Lashen et al. (2023)] uses a sampling-based GNN framework to detect and localize HTs by sampling subgraphs from the entire netlist. Other works, such as [Yen et al. (2023)], employ machine learning frameworks to detect and localize HTs by classifying logical paths as Trojan or Trojan-free.

Various detection methods based on sub-netlists have been introduced; however, some of these approaches face notable limitations. They often generate an excessive number of sub-netlists, particularly in large-scale netlists, resulting in increased computational complexity and resource consumption. Additionally, the indiscriminate selection of sub-netlists may capture irrelevant or redundant regions, diverting focus from critical areas associated with hardware Trojans. This lack of targeted filtering not only reduces detection efficiency but also increases false positives, limiting the practicality of these methods in real-world applications.

2.3.3. Machine learning methods for detection in full netlist granularity

Full netlist granularity detection methods consider the entire IC design as a whole, allowing for a holistic analysis of HT presence. Yasaei [Yasaei et al. (2021)] utilizes GNN to extract features from Data Flow Graphs (DFGs), aiming to automate the recognition of hardware Trojan presence by learning the behavior of circuits. While converting a netlist into a single graph may result in a complex and sizable structure, it may pose challenges in achieving precise localization.

Full graph classification focuses on determining whether the entire netlist contains an HT. While this approach is computationally simpler than classifying every individual node, it sacrifices the ability to localize HTs within the design. Without pinpointing suspicious regions, full graph classification is less actionable for hardware verification processes. Furthermore, analyzing the entire netlist as a single entity introduces scalability issues in complex designs, especially when dealing with intricate structures and high gate counts.

3. Motivation

This work assumes a threat model where hardware Trojans (HTs) are inserted into third-party intellectual property (3PIP) cores before integration. In practice, 3PIPs are commonly delivered as firm IPs in the form of gate-level netlists, offering limited visibility into the design internals. We do not assume access to trusted RTL or golden references. Consequently, our approach targets gate-level netlists as the primary abstraction for analysis and detection. This model reflects a realistic and critical pre-silicon scenario in which untrusted 3PIP vendors intentionally embed malicious logic into gate-level deliverables. Accordingly, this paper focuses on detecting such threats directly from gate-level netlists.

The Table 1 illustrates the trigger gate, payload gate, and the total gate count for various hardware Trojan examples. As shown in Table 1, HTs in the netlist consist of only tens to hundreds of gates, representing a small proportion of the total number of gates. As the scale of IP core netlists expands, modeling all gates becomes increasingly challenging, thus making the detection of HTs more difficult.

To highlight HTs within the entire netlist, it is essential to effectively utilize the characteristics of HTs. Concealment, an inherent characteristic of HTs known as 'less-toggle,' is identified as a logical and effective feature for detecting HTs. The introduction of a concealment metric, representing the less-toggle attribute, helps in screening suspicious triggers from the original graph, thus narrowing the search space. Meanwhile, HTs exhibit a special topological structure. Taking the RS232-T1000 as an example, the trigger signal $iCTRL$ displays a large fan-in structure in the input direction and a multi-output structure in the output direction. The unique structural manifestations of HTs, illustrated in Fig. 1, serve as distinctive features that enable the differentiation of HTs from suspicious subgraphs.

By focusing on specific regions of interest within the netlist, our approach captures the structural and contextual information necessary for

Table 1
The size of some existing HTs in Trust-Hub [Trust-Hub.org (2025)] gate-level Netlist [Salmani (2017)].

Trojan	Trigger size (gate no.)	Payload size (gate no.)	Total size (gate no.)
s38147-T100	11	1	5641
s38147-T200	11	4	5644
s38147-T300	15	29	5673
vga_lcd-T100	4	1	69 837

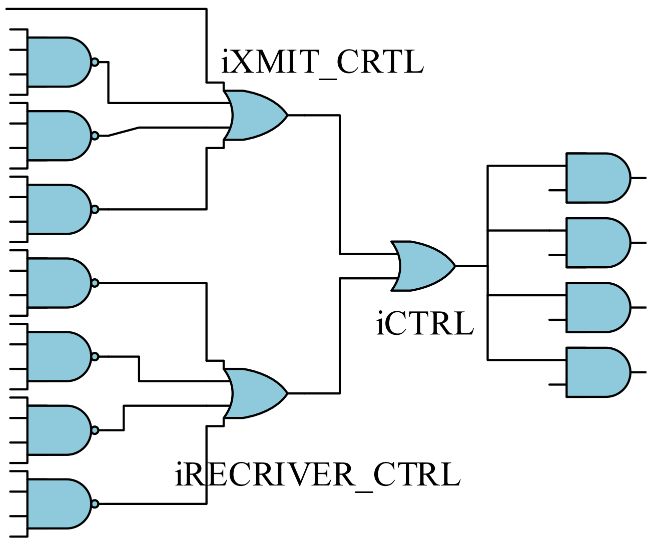


Fig. 1. Structure of an HT example from RS232-T1000.

detecting HTs while maintaining computational efficiency. Unlike node classification, subgraph classification benefits from the additional context provided by surrounding nodes and connections, making it more robust to noise. At the same time, it avoids the scalability challenges of full graph classification by restricting the analysis to smaller, targeted regions.

In conclusion, subgraph classification represents a thoughtful compromise that addresses the limitations of node and full graph classification. By combining the strengths of both methods, it provides an efficient, accurate, and practical approach to detecting hardware Trojans in large-scale netlists.

4. Methodology

In the paper, we propose a novel method named SubG4TJ for HT detection, which is a collaborative detection method based on subgraph classification (illustrated in Fig. 2). This approach entails filtering LTGs based on concealment metric, mining corresponding LTG-aware subgraphs, and utilizing GNN-based network to classify these LTG subgraphs. It is specifically tailored for the identification of HTs with triggers.

To achieve both accuracy and scalability in large-scale netlists, SubG4TJ integrates intrinsic node-level concealment properties with inter-gate structural connectivity. By focusing analysis on concealment-centric regions and modeling these subgraphs using GNNs, SubG4TJ effectively suppresses false positives while preserving high detection precision. This design ensures that subgraphs most likely to contain stealthy or coordinated malicious logic are prioritized during learning, which supports efficient localization and robust classification performance.

4.1. LTG filtering based on concealment asymmetry

HTs are designed with rare trigger conditions, which means that controlling a signal to 0 or 1 is imbalanced, with one state being significantly more difficult to achieve than the other. This imbalance in the controllability of signal states can be leveraged as a key feature to assess the difficulty of triggering HTs, helping to identify potentially malicious circuits.

4.1.1. Limitation of controllability metric

In order to highlight the imbalance between 0/1 controllability and reduce false positives, Su et al. (2021) introduces the use of (CC0/CC1, CC1/CC0) ratios for circuit analysis. Here, CC0 represents the difficulty of setting a signal to 0, while CC1 indicates the difficulty of setting a signal to 1. Initially, the 0/1 controllability of all primary inputs is set to $CC0 = CC1 = 1$.

Controllability, as a metric, estimates the ease or difficulty of driving a signal to 0 or 1 through primary inputs, providing an indication of the testability of circuits. However, for trigger-based HT detection, this metric alone is insufficient. Variations in HT structures lead to inconsistent controllability values, making it unreliable for consistent detection.

For the same HT, such as RS232-T1000, the controllability values can vary significantly depending on the structural implementation. For example, in one structure (shown in Fig. 3(a)), the controllability value is $CC0/CC1(Trigger) = 16.5$, while in another structure (Fig. 3(b)), it reaches $CC0/CC1(Trigger) = 25.9$. These variations highlight the limitations of using controllability metrics for HT detection, as they are inconsistent across different HTs and fail to provide reliable measurements.

4.1.2. Concealment metric

Given the limitation of controllability measurement, we propose a concealment metric to reduce the impact of structural diversity of HTs on detection. The concealment metric incorporates two key components, C_0 and C_1. C_0 assesses the difficulty of setting the output of a gate to 0 (0-concealment) and C_1 evaluates the difficulty of setting the output of a gate to 1 (1-concealment).

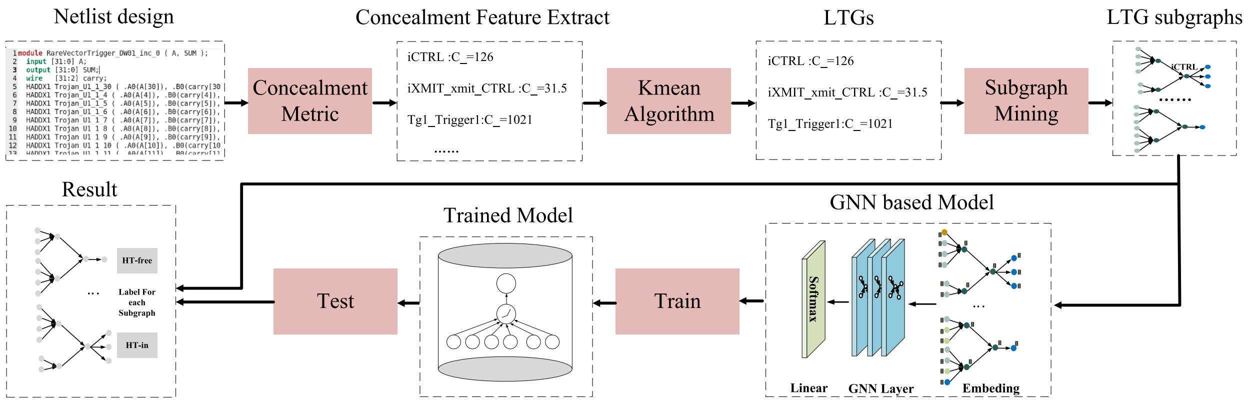
C_0 and C_1 are computed according to the rules outlined in Table 2. Taking AND as an example, while the traditional controllability calculations are $CC0 = \min\{CC0(a), CC0(b)\} + 1$ and $CC1 = CC1(a) + CC1(b) + 1$, the concealment calculations are simpler: $C\_0 = \min\{C\_0(a), C\_0(b)\}$ and $C\_1 = C\_1(a) + C\_1(b)$. It is unnecessary to add 1 (meaning the number of logic gates traversed) to C_0 or C_1 for each gate. These metrics are influenced mainly by the types and inputs of the gates, rather than by the number of gates traversed.

Mathematically, these formulations are derived from Boolean gate logic. For an AND gate with inputs a and b, its output $z = a \cdot b$ evaluates to 0 if *any* input is 0, and only evaluates to 1 if *all* inputs are 1. Therefore, the concealment score C_0 for output logic 0 is governed by the *easiest* input to be set to 0, captured as $C\_0 = \min\{C\_0(a), C\_0(b)\}$. In contrast, to force the output to 1, *all* inputs must be 1, which leads to the aggregated difficulty $C\_1 = C\_1(a) + C\_1(b)$. This logic generalizes to other gate types (e.g., OR, NAND) according to their functional truth tables.

In Table 2, the input controllability values are initialized based on the approach in Wang et al. (2006), where both logic-0 and logic-1 controllabilities for primary inputs are assigned a default value of 1. This assumption ensures consistency in concealment metric computation across different gate types and reflects a uniform input effort for setting logic values during propagation.

Table 2
Calculation formula of concealment metric.

Cell Type	0-Concealment(C_0)	1-Concealment(C_1)
Input	1	1
AND (a,b)	$Min(C\_0(a),C\_0(b))$	$C\_1(a) + C\_1(b)$
$OR(a, b)$	$C\_0(a) + C\_0(b)$	$Min(C\_1(a), C\_1(b))$
$NOT(a)$	$C\_1(a)$	$C\_0(a)$
XOR (a, b)	$Min(C\_0(a) + C\_0(b),$ $C\_1(a) + C\_1(b))$	$Min(C\_0(a) + C\_1(b),$ $C\_1(a) + C\_0(b))$
DFF (D, clk)	$C\_0(D) + C\_0(clk)$ $+C\_1(clk)$	$C\_1(D) + C\_0(clk)$ $+C\_1(clk)$



```
module RareVectorTrigger_DW01_inc_0 ( A, SUM );
  input [31:0] A;
  output [31:0] SUM;
  wire   [31:2] carry;
  HADDX1 Trojan_U1_1_30 ( .A0(A[30]), .B0(carry[30]
  HADDX1 Trojan_U1_1_4 ( .A0(A[4]), .B0(carry[4]),
  HADDX1 Trojan_U1_1_5 ( .A0(A[5]), .B0(carry[5]),
  HADDX1 Trojan_U1_1_6 ( .A0(A[6]), .B0(carry[6]),
  HADDX1 Trojan_U1_1_7 ( .A0(A[7]), .B0(carry[7]),
  HADDX1 Trojan_U1_1_8 ( .A0(A[8]), .B0(carry[8]),
  HADDX1 Trojan_U1_1_9 ( .A0(A[9]), .B0(carry[9]),
  HADDX1 Trojan_U1_1_10 ( .A0(A[10]), .B0(carry[10]
  HADDX1 Trojan_U1_1_11 ( .A0(A[11]), .B0(carry[11]
```

Fig. 2. Overall process of SubG4TJ.

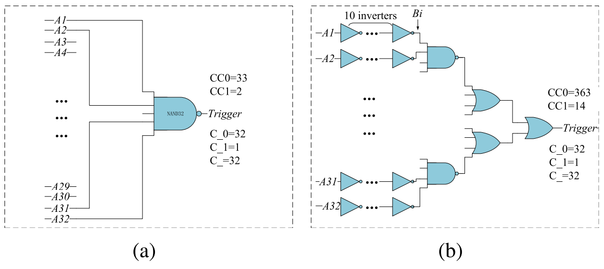


Fig. 3. Controllability and concealment values of HTs with the same function but different gate-level structures.

Using the metric in Table 2, we can compute the C_0 and C_1 values, which represent the concealment of signals being consistently 0 or 1, respectively. While traditional testability theories assume hardware Trojans exhibit high C_0 or C_1 values, the AES encryption algorithm's characteristics challenge this assumption. Due to its multi-round encryption operations, achieving control states where signals are consistently 0 or 1 is inherently difficult. Consequently, many AES signals exhibit high C_0 and C_1 values, leading to their misidentification as hardware Trojans. Since hardware Trojans are typically designed to remain inactive, gates with imbalanced C_0 and C_1 values (e.g., high C_0 and low C_1, or vice versa) are more likely to be part of a trigger. To emphasize the imbalanced C_0 and C_1, we use $C\_$ to represent the concealment metric. For combinational gate and sequential gate, $C\_$ is defined as:

$$C\_ = \frac{max(C\_0, C\_1)}{min(C\_0, C\_1)} \tag{1}$$

The higher $C\_$ values indicate stronger concealment. As demonstrated in Fig. 3, the structure of HTs does not affect the $C\_(Trigger)$ values, indicating that the concealment metric is robust against variations in HT structures. This metric is effective for identifying potential triggers in HTs.

After calculating the $C\_$ values for all gates, these values are input as a one-dimensional dataset into the k-means clustering algorithm, where k is set to 2. We choose k-means over other discriminatory methods because it is an unsupervised, parameter-light algorithm that does not require labeled data or prior distribution assumptions, making it suitable for diverse hardware designs where ground truth labels are unavailable. Compared with supervised classifiers or more complex clustering methods, k-means provides a simple yet effective decision boundary for one-dimensional data with low computational overhead. We further choose k-means instead of a fixed threshold because the distribution of $C\_$ values varies significantly across different designs. A manually defined threshold would require circuit-specific tuning and may not generalize well to unseen datasets, whereas k-means automatically adapts to the actual data distribution, ensuring robustness and eliminating manual parameter selection. This choice of $k = 2$ aligns with our detection objective – to distinguish between gates exhibiting suspicious concealment behavior and those with normal controllability profiles. Specifically, the two resulting clusters correspond to: (i) genuine gates characterized by low concealment values, and (ii) suspicious HT gates that exhibit high concealment values. Such binary separation simplifies downstream subgraph mining and enhances interpretability, while remaining consistent with the assumption that HTs typically exhibit a distinct behavioral footprint. Gates identified with high concealment values are considered suspicious LTGs and are subject to further analysis.

4.2. LTG-Aware subgraph mining

Algorithm 1 Algorithm of k-hop.

```
Require: Graph $G_{all}, source, k_{in}, k_{out}$ //$k_{in} and k_{out}$ means the hop-number from source
Ensure: Subgraph $G$ // subgraph generated from source
1:  $G = G'_0 = source$
2:    for $(i = 0; i < k_{in}; i++)$
3:      if $(G'_i == \emptyset)$
4:        return $G_i$
5:      else
6:        $G'_{i+1} \leftarrow input(G'_i) - G$ // Remove visited items
7:        $G \leftarrow G \cup input(G'_i)$ // Add the inputs of all nodes in $G'_i$ to $G$
8:        $i \leftarrow i + 1$
9:    end for
10: $G''_0 = source$
11:   for $(j = 0; j < k_{out}; j++)$
12:     if $(G''_i = = \emptyset)$
13:       return $G$
14:     else
15:       $G''_{j+1} \leftarrow output(G''_j) - G$; // Remove visited items
16:       $G \leftarrow G \cup output(G''_j)$; // Add the outputs of all nodes in $G''_j$ to $G$
17:       $j \leftarrow j + 1$
18:     end for
19: return $G$
```

To identify potential HTs, we employ the concealment metric to select LTGs from the gate netlist and create subgraphs comprising LTGs along with their neighboring gates and connections. We rank the LTGs in descending order based on their concealment values and iteratively construct subgraphs through the k-hop algorithm (shown in Algorithm 1), which is based on Breadth-First Search (BFS). We choose BFS because it efficiently expands a local neighborhood around each LTG, preserving

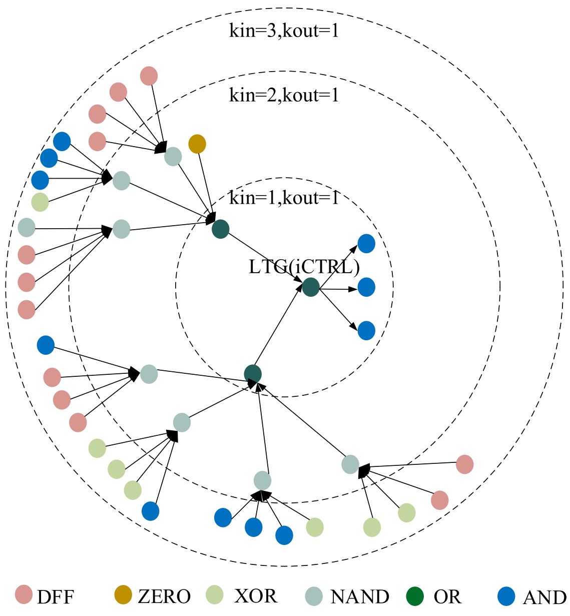


Fig. 4. Subgraphs generated from $ICTRL$ in RS232-T1000.

the contextual connectivity while avoiding excessive growth, making it well-suited for capturing potential HT structures in large netlists.

Each LTG previously used to generate a subgraph is excluded from subsequent subgraph generation to avoid redundancy. The process of k-hop algorithm is detailed as follows: given a subgraph $SubG = (V, E)$, where V represents the set of nodes reachable to LTG within k_{in}, E represents the set of edges between v ($v \in V$). The existence of edge between v_a and v_b means v_a is the input of v_b. The connection is determined from the gate-level netlist. Starting with $k_{in} = 1$, the algorithm adds all input nodes of the LTG to the set V and marks the LTG as visited. Then the algorithm increments k_{in} by 1, adds input nodes of each unvisited node to set V and marks these unvisited nodes as visited. This process continues until the predefined value of k_{in} is reached, or no unvisited nodes remain in the set. Fig. 4 illustrates the subgraphs generated from the LTG ($ICTRL$) in RS232-T1000.

Using the method outlined above, we construct multiple suspicious LTG subgraphs, which are then input into a GNN-based network for training and detecting.

4.3. Subgraph classification with LTG-informed graph neural networks

In this section, we propose LTG-GNN, termed the LTG-Informed Subgraph Classification Using Graph Neural Networks, specifically designed for subgraph classification in HT detection. The network leverages the unique characteristics of LTG nodes to effectively distinguish between subgraphs with and without hardware Trojans. By focusing on LTG nodes and their surrounding structure, the model ensures accurate and interpretable classification results.

To classify the mined LTG subgraphs, we leverage GCN and GAT, two representative methods in the family of GNN. GCN captures local topological patterns through neighborhood aggregation, making it suitable for learning structural regularities of benign and malicious logic. GAT, by contrast, introduces an attention mechanism that adaptively assigns weights to different neighbors, enabling the network to focus on more critical or influential gates – an essential capability for highlighting Trojan-related behaviors. This dual approach helps demonstrate the generalizability and robustness of our LTG-aware subgraph classification framework.

4.3.1. LTG-GNN architecture

The proposed LTG-GNN consists of three main components: an embedding layer, multiple GNN layers, and a linear classification layer, as illustrated in Fig. 2.

4.3.1.1. Embedding layer. The embedding layer is responsible for encoding the nodes of the LTG subgraph and initializing their features. A constructed LTG subgraph, such as the example shown in Fig. 2 (e.g., $LTGsubgraph1$ with nodes G_1, G_2, G_3, and $LTG1$), includes several nodes and edges. Node features incorporate properties such as gate type, concealment value, and hop distance from the LTGs. These features are concatenated to form n-dimensional feature vectors for each node, a dimensionality determined experimentally.

Edges are defined based on the logical connections between nodes, where an edge between gate G_1 and G_2 signifies that G_1 serves as an input to G_2. This layer not only generates node embeddings but also establishes the relationships between nodes within the LTG subgraph.

4.3.1.2. GNN layers. The GNN layers in the proposed LTG-GNN are designed to process subgraph data by capturing both local and global structural information. These layers provide flexibility in employing different GNN architectures, such as GCN and GAT, to adapt to varying detection requirements.

Graph convolutional networks: GCNs [Kipf and Welling (2016)] aggregate node features by computing a weighted sum of neighboring node features, normalized by the degree of each node. For a given node i, the updated feature vector $\mathbf{h}'_i$ is computed as:

$$\mathbf{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}(i)} \frac{1}{\sqrt{d_i d_j}}\mathbf{W}\mathbf{h}_j\right),$$

where $\mathcal{N}(i)$ is the set of neighbors of node i, d_i and d_j are the degrees of nodes i and j, $\mathbf{W}$ is a learnable weight matrix, and σ is a non-linear activation function. GCN layers are effective at capturing global graph structures by stacking multiple layers, but they treat all neighbors equally, which may limit their discriminative power for complex tasks.

Graph attention networks: GAT layers [Velikovi et al. (2018)] enhance the aggregation process by incorporating a self-attention mechanism that assigns different weights to neighbors based on their importance. For a given node i, the attention score α_{ij} between i and its neighbor j is computed as:

$$\alpha_{ij} = \text{softmax}_j\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{h}_i \| \mathbf{h}_j]\right)\right),$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are the feature vectors of nodes i and j, $\|$ denotes concatenation, and $\mathbf{a}$ is a learnable weight vector. Using these scores, node features are aggregated as:

$$\mathbf{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{W}\mathbf{h}_j\right).$$

The GAT layers capture finer-grained relationships by focusing on the most relevant neighbors, making them particularly effective for tasks like hardware Trojan detection.

By leveraging these GNN layers, the LTG-Informed Network processes subgraphs to distinguish between those with and without hardware Trojans, achieving high detection performance across diverse datasets.

4.3.1.3. Linear classification layer. After processing through the GNN layers, the features of LTG nodes are extracted and passed to a fully connected linear layer. The softmax function is applied to classify the subgraph. The classification outcome of the LTG nodes serves as the final classification result for the entire LTG subgraph. By focusing on LTG nodes, the network captures structural anomalies introduced by hardware Trojans, ensuring precise and interpretable classification.

4.3.2. Model training process

The training process follows a supervised learning approach using a dataset of LTG subgraphs, each labeled as either HT-containing or benign. Subgraphs are generated from gate-level netlists, encompassing both benign examples and those embedded with Trojans. Each subgraph is constructed by expanding around LTG nodes and assigned a label based on its characteristics.

To optimize the model, a binary cross-entropy loss function is employed. For a given LTG subgraph with a predicted label $\hat{y}_i$ and ground truth label y_i, the loss is defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right],$$

where N denotes the total number of subgraphs in the batch.

The model is trained using optimization techniques such as stochastic gradient descent, with carefully tuned learning rates and regularization methods to mitigate overfitting and enhance generalization.

4.3.3. Model detection process

Once trained, LTG-GNN is applied for subgraph classification. For each LTG subgraph, the model predicts a label based on the attention-weighted aggregation of node features.

4.4. HT detecting and locating

After performing subgraph classification, the results of the LTG-GNN are used to analyze the entire design for the presence of HTs. Each LTG subgraph is assigned a label $y_i \in \{0, 1\}$, where $y_i = 1$ indicates that the i-th LTG subgraph contains an HT, and $y_i = 0$ otherwise.

Subgraphs labeled as 1 provide critical information for identifying the potential locations of HTs within the netlist. Based on this classification, the design can be categorized as either benign or containing an HT. Furthermore, the subgraphs flagged as containing HTs allow us to pinpoint approximate regions in the netlist where structural anomalies are likely introduced, facilitating targeted verification and mitigation efforts.

This approach not only enables efficient detection of HTs at the subgraph level but also ensures scalability for analyzing large and complex gate-level netlists. By localizing HTs to specific subgraphs, the method reduces the search space for detailed inspections and improves the overall accuracy and interpretability of the detection process.

5. Experimental setup and evaluation indices

5.1. Dataset description

We use two distinct datasets for different purposes in our experiments. The first dataset, Trust-Hub, is a public dataset widely used in HT detection research, and it serves as our test set. This dataset is selected because it is commonly used in related works, allowing us to ensure consistency and comparability with previous studies. The second dataset, generated from open-source IP cores as outlined in Muralidhar et al. (2021), is used for training and validation.

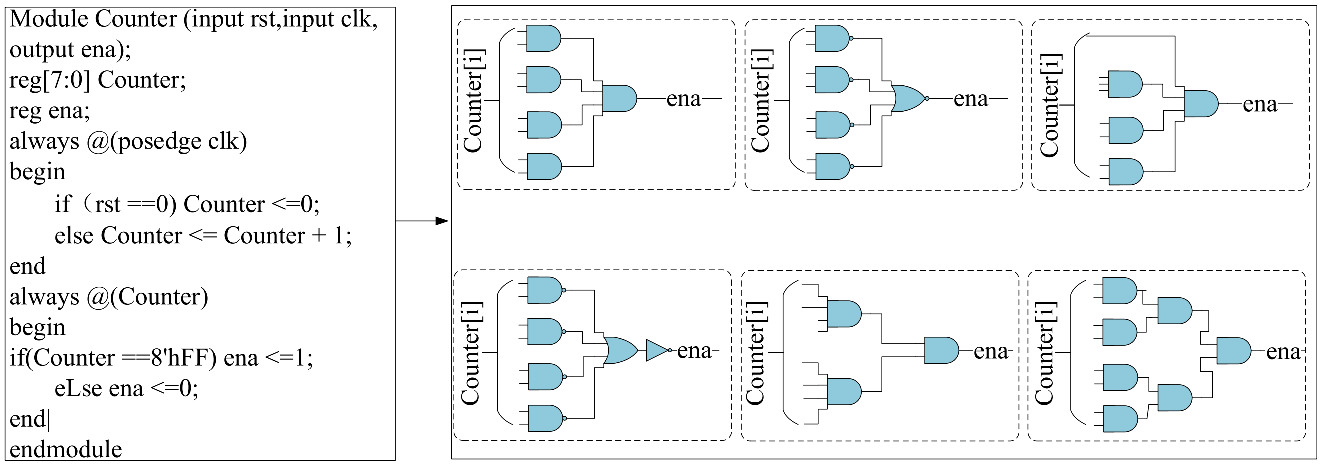
By using Trust-Hub as the test dataset, we ensure consistency with prior research, providing a fair comparison of SubG4TJ's performance. The generated dataset, in contrast, allows us to train and validate the model on a diverse set of hardware designs. To eliminate any potential overlap between the training and test datasets, we make sure that the test set is entirely distinct and contains only data that has not been encountered during the training or validation phases. This separation ensures an unbiased assessment of the model's generalization capability.

5.1.1. Generation of datasets

Public HT datasets (Trust-Hub) are limited by types of HTs and the number of samples available. This limitation compromises the adequacy of training, validation, and testing. To expand the diversity of HT datasets, we automate the insertion of HTs into open-source IP cores. An HT comprises a trigger and a payload. The trigger activates after specific clock cycles or conditions, while the payload disrupts the circuit's functionality upon activation. These HTs are inserted into HT-free circuits, such as ca_prng and i2cSlave from Muralidhar et al. (2021). The RTL code is synthesized into gate-level netlists using synthesis tool. By varying synthesis constraints, we create diverse netlists from the same RTL code, enriching the training dataset with varied HT examples. Fig. 5 illustrates various trigger examples when the trigger condition is a full 8-bit counter. Although only six different examples are shown, our method is capable of generating additional examples with varied gate types and connection relationships.

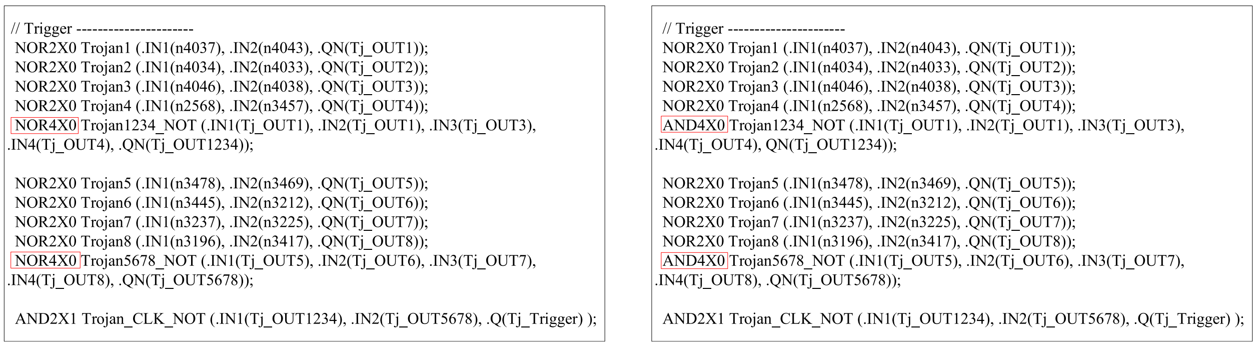
5.1.2. Division of datasets

The IP core dataset is generated by implanting HTs into actual IP cores. It comprises 2550 subgraphs: 1300 HT-inserted and 1250 HT-free. We allocate this dataset for training. Testing is conducted on circuits from the Trust-Hub benchmark, while the model is trained on different circuits and HTs from actual IP cores ([Muralidhar et al. (2021)]). In this scenario, the circuit under testing has not been exposed to the



```
Module Counter (input rst,input clk,
output ena);
reg[7:0] Counter;
reg ena;
always @(posedge clk)
begin
     if（rst ==0) Counter <=0;
     else Counter <= Counter + 1;
end
always @(Counter)
begin
if(Counter ==8'hFF) ena <=1;
     eLse ena <=0;
end|
endmodule
```

Fig. 5. Examples of condition-based triggers.



```
// Trigger ---------------------
 NOR2X0 Trojan1 (.IN1(n4037), .IN2(n4043), .QN(Tj_OUT1));
 NOR2X0 Trojan2 (.IN1(n4034), .IN2(n4033), .QN(Tj_OUT2));
 NOR2X0 Trojan3 (.IN1(n4046), .IN2(n4038), .QN(Tj_OUT3));
 NOR2X0 Trojan4 (.IN1(n2568), .IN2(n3457), .QN(Tj_OUT4));
 NOR4X0 Trojan1234_NOT (.IN1(Tj_OUT1), .IN2(Tj_OUT1), .IN3(Tj_OUT3),
.IN4(Tj_OUT4), .QN(Tj_OUT1234));

 NOR2X0 Trojan5 (.IN1(n3478), .IN2(n3469), .QN(Tj_OUT5));
 NOR2X0 Trojan6 (.IN1(n3445), .IN2(n3212), .QN(Tj_OUT6));
 NOR2X0 Trojan7 (.IN1(n3237), .IN2(n3225), .QN(Tj_OUT7));
 NOR2X0 Trojan8 (.IN1(n3196), .IN2(n3417), .QN(Tj_OUT8));
 NOR4X0 Trojan5678_NOT (.IN1(Tj_OUT5), .IN2(Tj_OUT6), .IN3(Tj_OUT7),
.IN4(Tj_OUT8), .QN(Tj_OUT5678));

 AND2X1 Trojan_CLK_NOT (.IN1(Tj_OUT1234), .IN2(Tj_OUT5678), .Q(Tj_Trigger) );
```

(a) Original netlist of s38417-T300

```
// Trigger ---------------------
NOR2X0 Trojan1 (.IN1(n4037), .IN2(n4043), .QN(Tj_OUT1));
NOR2X0 Trojan2 (.IN1(n4034), .IN2(n4033), .QN(Tj_OUT2));
NOR2X0 Trojan3 (.IN1(n4046), .IN2(n4038), .QN(Tj_OUT3));
NOR2X0 Trojan4 (.IN1(n2568), .IN2(n3457), .QN(Tj_OUT4));
AND4X0 Trojan1234_NOT (.IN1(Tj_OUT1), .IN2(Tj_OUT1), .IN3(Tj_OUT3),
.IN4(Tj_OUT4), QN(Tj_OUT1234));

NOR2X0 Trojan5 (.IN1(n3478), .IN2(n3469), .QN(Tj_OUT5));
NOR2X0 Trojan6 (.IN1(n3445), .IN2(n3212), .QN(Tj_OUT6));
NOR2X0 Trojan7 (.IN1(n3237), .IN2(n3225), .QN(Tj_OUT7));
NOR2X0 Trojan8 (.IN1(n3196), .IN2(n3417), .QN(Tj_OUT8));
AND4X0 Trojan5678_NOT (.IN1(Tj_OUT5), .IN2(Tj_OUT6), .IN3(Tj_OUT7),
.IN4(Tj_OUT8), .QN(Tj_OUT5678));

AND2X1 Trojan_CLK_NOT (.IN1(Tj_OUT1234), .IN2(Tj_OUT5678), .Q(Tj_Trigger) );
```

(b) Modified netlist of s38417-T300

Fig. 6. The original and modified netlist of s38417-T300.

model during training, demonstrating the model's ability to detect HTs in unknown circuits.

5.2. Parameters, environment and evaluation indices

The dataset encompasses 2550 subgraphs, each labeled as either HT-inserted (1) or HT-free (0). Our GAT-based network comprises three GAT layers and one linear layer, featuring a dropout rate of 0.15 for the GAT layer. Throughout training, we configure the batch size at 2000 and train the network for 2000 epochs. A batch gradient descent learning rate of 5e-5 is used to optimize the network parameters.

Our experiment is conducted in PyTorch framework and runs on a server equipped with an i9 processor at 3.0 GHz and 128GB of memory capacity. We utilize an NVIDIA GeForce RTX 3090 graphics card for both training and HT detecting.

We use true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) to calculate the evaluation metrics, namely TPR (True Positive Rate), FPR (False Positive Rate) and F1. The calculation formulas are as follows: $TPR/Recall = TP/(TP + FN)$, $FPR = FP/(FP + TN)$, $Precision = TP/(TP + FP)$, and $F1 = 2 \times (Precision \times Recall)/(Precision + Recall)$.

6. Results and discussions

This section delineates the influence of various factors on the performance and analyzes its overall effectiveness.

6.1. Test sample description

SubG4TJ focuses on detecting concealed HTs, making it applicable only to samples with such characteristics. However, as noted in Krieg (2023), many Trust-Hub examples do not fully exhibit these traits. For instance, s38417-T300 lacks both correctness and stealthiness. To address this, modifications are made to the HT in the original netlist to enhance concealment. Specifically, NOR4X0 gates are replaced with NAND4X0 gates (shown in Fig. 6), which changes the trigger condition from requiring 8 signals to simultaneously be 0 to requiring 16 signals to simultaneously be 0. This significantly increases the difficulty of triggering the HT, thereby improving its stealthiness. The detection results presented in this work are based on these modified examples.

6.2. Impact of concealment metric on screening LTGs

To evaluate the effects of the concealment metric on screening LTGs, we compare the number of original total subgraphs, LTGs, and LTG subgraphs (shown in Fig. 7). Due to the significant variation in the scale of different netlists, we employ a logarithmic coordinate system to represent data. For instance, in RS232-T1000, there are 575 nodes. It is important to note that, due to the decomposition of composite gates into simpler gates, the total gate count may slightly differ from the figures reported in other studies. For instance, an OA21 gate is decomposed into an OR gate and an AND gate, effectively counting as two simpler gates. If subgraphs are constructed for each node without any screening, the total number of subgraphs would amount to 575. This method could significantly increase the number of subgraphs as the node count rises to millions.

With our method, the number of LTGs is reduced to 2, thereby decreasing the number of suspicious gates to 0.348 % of the entire netlist. On average, across 20 netlists, the reduction ratio is 0.157 %, indicating that our method effectively reduces the number of gates involved in the screening process. Additionally, as multiple LTGs can be grouped within a single LTG subgraph, the number of LTG subgraphs is naturally fewer than the number of LTGs. This analysis highlights the efficiency of our method in reducing the number of gates, optimizing the screening process, and enhancing its applicability in large-scale designs.

6.3. Attention weight analysis

Attention mechanisms play a pivotal role in graph neural networks. This section delves into the analysis of attention weights derived from the model.

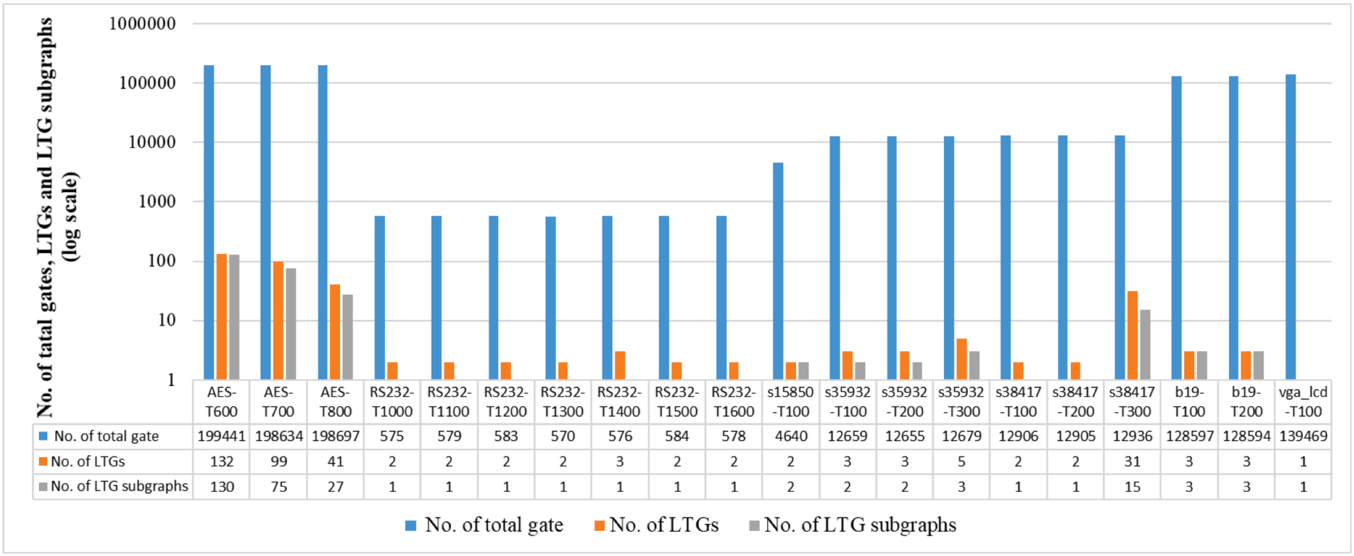
To analyze the behavior of attention weights, we examine a sample subgraph, s35932-T100. Fig. 8(a) illustrates the attention heatmap of this subgraph, where each element represents the attention score between two connected nodes. For enhanced clarity, these heatmap values are annotated onto the s35932-T100 circuit in Fig. 8(b).

The attention weights vary significantly across different node pairs, reflecting the model's ability to prioritize critical connections. The adaptive self-attention mechanism in the GAT layers dynamically allocates importance to critical connections, enabling the network to highlight meaningful patterns while filtering out noise. Among the nodes connected to node 9, nodes 1 and 5 exhibit higher attention weights compared to node 12. These high-attention nodes likely have large fan-ins, as hardware Trojans often rely on multiple signal controls for activation, making such nodes crucial for detection.

We compared the training accuracy trends of the GAT network, which incorporates an attention mechanism, with the GCN network, which does not utilize attention, as illustrated in Fig. 9. The figure demonstrates that while the GAT network converges faster during training, it achieves comparable accuracy to the GCN network during training.

6.4. Impact of k_{in} in k-hop algorithm on performance

In the LTG subgraph, k_{in} and k_{out} denote hop numbers towards input and output directions from LTG, respectively. Nodes towards input indicate HT trigger logic, while output nodes represent payload logic. Since trigger logic involves multiple nodes, $k_{in} \geq 1$. Payload logic



	AES-T600	AES-T700	AES-T800	RS232-T1000	RS232-T1100	RS232-T1200	RS232-T1300	RS232-T1400	RS232-T1500	RS232-T1600	s15850-T100	s35932-T100	s35932-T200	s35932-T300	s38417-T100	s38417-T200	s38417-T300	b19-T100	b19-T200	vga_lcd-T100
No. of total gate	199441	198634	198697	575	579	583	570	576	584	578	4640	12659	12655	12679	12906	12905	12936	128597	128594	139469
No. of LTGs	132	99	41	2	2	2	2	3	2	2	2	3	3	5	2	2	31	3	3	1
No. of LTG subgraphs	130	75	27	1	1	1	1	1	1	1	2	2	2	3	1	1	15	3	3	1

Fig. 7. Comparison between the number of original total gates, number of LTGs and number of LTG subgraphs.

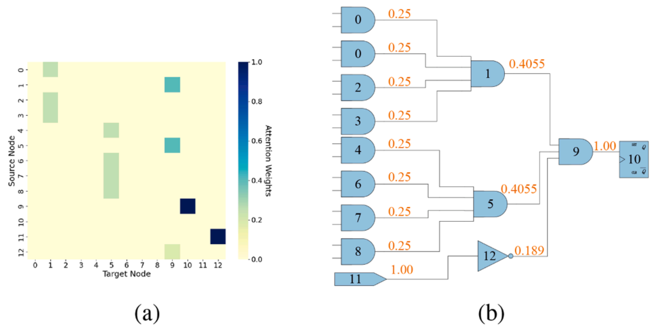


Fig. 8. Visualization of attention weights for subgraph s35932-T100.

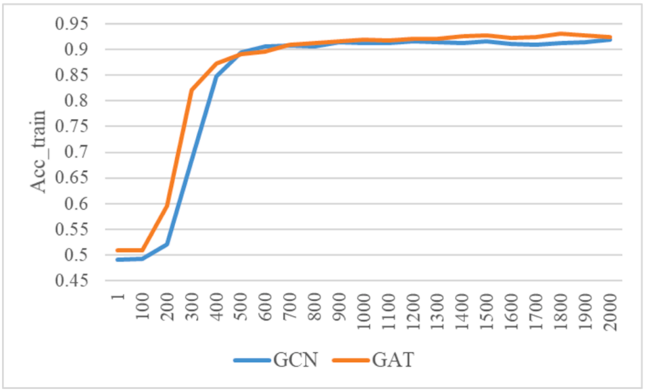


Fig. 9. Training accuracy trends of GAT and GCN networks.

Table 3
Performance of hops used in GAT and GCN.

Model	GAT			GCN		
	F1 (%)	TPR (%)	FPR (%)	F1 (%)	TPR (%)	FPR (%)
Model_hop1	94.59	100.00	10.00	0.00	0.00	0.00
Model_hop2	95.00	95.00	0.36	93.26	93.45	0.71
Model_hop3	91.65	92.48	0.38	91.65	92.48	0.38
Model_hop4	91.63	92.46	0.42	91.63	92.46	0.42
Model_hop5	91.61	92.44	0.45	91.61	92.44	0.45
Model_hop6	90.66	90.84	0.50	91.60	92.42	0.50

typically alters or leaks values, which has no effects on HT detection based on trigger. So k_{out} is set to 1 for simplicity. This study focuses solely on the effects of varying k_{in}.

The construction of LTG subgraphs depends on selecting an appropriate k_{in} value with k-hop algorithm. Different k_{in} values affect the size of subgraphs. We randomly selected approximately 2500 subgraphs as the training set for 2000 epochs, with k_{in} configured from 1 to 6 to gauge trends of model performance.

Model_hop1 to model_hop6 are trained with 1-hop ($k_{in} = 1$) to 6-hop ($k_{in} = 6$) subgraphs, respectively. The results for these models on the Trust-Hub dataset are presented in Table 3.

From Table 3, it is evident that in the detection results of the GAT network, predictions are biased toward 1 (hardware Trojan), while in the GCN network, predictions tend to favor 0 (normal). Consequently, model_hop1 exhibits a significantly higher FPR in GAT and a noticeably lower TPR in GCN compared to other models. This discrepancy arises because subgraphs composed of nodes only 1 hop away from an LTG fail to sufficiently capture the structural complexity of an HT. Analyzing the hierarchical structure of HTs reveals that subgraphs including nodes at least 2 hops away from an LTG are necessary to effectively represent distinct HT patterns.

The performance comparison between GAT and GCN models trained with different hop subgraphs highlights the importance of balancing subgraph size and detection effectiveness. Models trained with

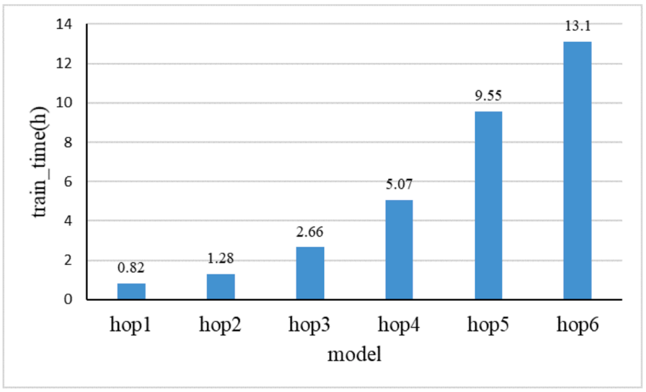


Fig. 10. Training time for GAT models with different hop counts.

2-hop subgraphs consistently achieve comparable performance to those trained with 3-hop to 6-hop subgraphs, effectively capturing the critical structural features of HTs while maintaining robustness in detection.

As the number of hops increases, subgraph size grows, gradually approaching the scale of the full graph. This trend introduces more interfering nodes, resulting in a decrease in TPR and an increase in FPR. The growing subgraph size also increases computational complexity, undermining the advantages of subgraph-based detection by diluting the focus on critical regions. These observations demonstrate the effectiveness of subgraph detection in capturing key HT features while avoiding unnecessary complexity.

Moreover, training time analysis, as shown in Fig. 10, reveals that larger subgraphs require significantly more time to train. In contrast, 2-hop subgraphs strike the optimal balance by providing sufficient information for accurate detection with considerably lower training time.

In summary, the 2-hop GAT model demonstrates the best balance between detection performance, subgraph size, and computational efficiency, validating the effectiveness of subgraph detection in HT detection tasks.

6.5. *Impact of different features used in GAT on performance*

Feature selection is critical in HT detection models. For our experimental analysis, we use various feature configurations to gauge trends in GAT model performance. The subgraphs for both the training and testing datasets are generated using the k-hop ($k\_in = 2$, $k\_out = 1$) algorithm. Table 4 presents the performance of GAT networks using different feature sets, including F1 score, TPR, and FPR. In general, the naming convention fea_Xtype_YC denotes that each node feature vector comprises an X-dimensional one-hot encoding of gate type followed by Y-dimensional concealment features.

The model using only the “TYPE” feature (`fea_26type`) achieves the best performance, with an F1 score of 95.00 %, TPR of 95.00 %, and a low FPR of 0.36 %. The “TYPE” feature represents the gate type of each node, and in our method, there are 26 distinct gate types, including AND, OR, NOT, and other gates. As a result, the feature vector for each node has a dimensionality of 26. This indicates that the “TYPE” feature alone is highly effective for distinguishing HTs from benign circuits. When concealment value features are added with varying bit-lengths, performance degrades. For example, the model `fea_26type_128C` shows an F1 score of 79.34 %, which is significantly lower than the `fea_26type` model. The degradation occurs because, in our method, subgraphs are derived from LTGs where nodes already exhibit high concealment values. These features do not provide additional discriminative power and may even introduce noise that affects classification accuracy.

Thus, while concealment values are useful during LTG filtering, they do not serve as effective features within the GAT network, and their use can harm detection performance. The “TYPE” feature, by itself, is

Table 4
Performance of different features used in GAT.

Model	F1 (%)	TPR (%)	FPR (%)
fea_26type_128C	79.34	80.20	5.00
fea_26type_32C	86.12	87.91	5.00
fea_26type_2C	90.19	91.35	5.36
fea_32C	84.46	85.41	5.00
fea_26type	95.00	95.00	0.36

the most relevant for HT detection, and unnecessary complexity from additional features should be avoided.

6.6. *Performance results of SubG4TJ*

We utilize a GAT network for subgraph classification, where the features of each node are represented using one-hot encoding based on the gate type. Subgraphs are generated with a k-hop algorithm where $k_{in} = 2$ and $k_{out} = 1$.

Tests on 20 benchmark circuits from Trust-Hub (e.g., RS232, S15850, b19, VGA, AES, etc., as shown in Table 5) show that SubG4TJ achieves an average TPR of 95.00 %, an FPR of 0.36 %, and an F1 score of 95.00 %. Notably, our evaluation includes the AES test case, which has not been addressed by other methods, further highlighting the ability of SubG4TJ in handling complex and large-scale netlists. While a few benchmarks exhibit TPR values around 50 %, each contains only a single hardware Trojan instance. In these cases, SubG4TJ successfully localizes at least one subgraph encompassing key portions of the Trojan logic. Lower TPR values arise when certain subgraphs include peripheral Trojan nodes but lack critical trigger or payload components. Such subgraphs may not be flagged as malicious, reflecting SubG4TJ’s conservative design that prioritizes detection of core HT structures to reduce false positives and focus analysis on high-confidence regions.

Table 5 also provides a comprehensive comparison with state-of-the-art methods. Specifically, Yasaei et al. (2021) formulates HT detection as a graph classification task by converting the entire netlist into a Data Flow Graph and applying GNNs. While comprehensive, this global representation limits fine-grained localization. In contrast, Hasegawa et al. (2025), Hasegawa et al. (2023), and Nozawa et al. (2021) adopt node-level detection, where each gate is treated as a node and classified based on extracted features. These methods excel at local anomaly detection but may miss structural patterns of distributed Trojans. Yamashita et al. (2022) uses handcrafted statistical features with traditional classifiers, while Yasaei et al. (2025) introduces a golden-reference-free GNN method based on Data Flow Graphs.

Compared to Yasaei et al. (2021), on the RS232 test set, SubG4TJ achieves significant improvements, increasing the TPR by 13.6 %, reducing the FPR by 5 %, and boosting the F1 score by 9.5 %. SubG4TJ improves TPR by 3.54 % compared to the GAT-based method in Hasegawa et al. (2025), achieving more effective identification of Trojan instances. Although it introduces a modest 0.29 % increase in FPR, this trade-off remains within acceptable bounds. In security-critical scenarios, especially in HT detection, maximizing TPR is often prioritized to avoid undetected threats, as false negatives may lead to severe consequences. Additionally, SubG4TJ demonstrates better performance compared to the methods proposed in Hasegawa et al. (2023), Yamashita et al. (2022) and Yasaei et al. (2025). Taken together, the results demonstrate TPR gains ranging from 0.9 % to 13.6 % across diverse benchmark circuits.

Table 5 compares the detection time of SubG4TJ with the full-graph approach proposed in Hasegawa et al. (2025), which is known for effective node classification. Both methods are evaluated after pre-processing the same netlists into graph data. For smaller test cases, such as RS232, detection times are comparable between the two approaches. However, as netlist size grows (e.g., AES), the detection time for Hasegawa et al.

Table 5
Experimental results on Trust-Hub.

	Subgraph classification			Graph classification			Node classification									detect time (s)	
Trust-Hub	SubG4TJ (ours)			Yasaei et al. (2021)			Hasegawa et al. (2025)		Hasegawa et al. (2023)		Yamashita et al. (2022)		Yasaei et al. (2025)			subG	fullG
benchmark	TPR (%)	FPR (%)	F1 (%)	TPR (%)	FPR (%)	F1 (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)	F1 (%)	SubG4TJ(ours)	Hasegawa et al. (2025)
Ave_RS232	100.00	0.00	100.00	86.4	5.00	90.50	95.60	0.15	99.1	5.25	98.38	1.10	97.1	10.00	93.30	0.001143	0.010143
Ave_AES	83.33	0.00	88.89	–	–	–	–	–	–	–	–	–	-	–	-	0.035333	4.074667
Average_all	95.00	0.36	95.00	–	–	–	91.46	0.07	83.5	5.4	–	–	–	–	–	0.00865	1.0482

(2025) increases sharply, whereas SubG4TJ shows only a slight increase. In summary, we achieve an average reduction in detection time of 120x.

To further evaluate scalability, we tested SubG4TJ on a large-scale accelerator IP comprising 2.5 million gates. All experiments are conducted on the same hardware platform (specified in Section 5), using identical node features, model architecture, and training parameters. The only variable changed between experiments was the graph processing method – SubG4TJ applies localized subgraph extraction while the baseline method performs detection on the entire circuit graph without partitioning. Under these controlled conditions, SubG4TJ completed detection in 0.53 s, whereas the full-graph baseline required 195.41 seconds. This $368\times$ speedup highlights the scalability advantage of subgraph-based processing, especially for large-scale netlists.

This demonstrates SubG4TJ's superior performance in balancing high TPR, low FPR, and time efficiency compared to other methods. These results are attributed to the use of the concealment metric and subgraph classification, which effectively identify HTs with subtle characteristics. The LTG subgraph retains essential HT structures, minimizing noise from unrelated gates. An enhanced GAT-based network further optimizes subgraph feature learning and reduces the influence of unrelated gates, achieving low FPR.

6.7. Complexity analysis

SubG4TJ comprises four sequential steps: Step-1 involves calculating concealment values, Step-2 entails executing k-means clustering, Step-3 involves mining LTG subgraphs using BFS, and Step-4 encompasses executing the GAT-based model. Let n represent the total number of gates in the netlist, and n_{LTGs} represent the number of LTGs, which is smaller than n. The time complexity of Step-1 is $O(n)$, and Step-2 has the time complexity of $O(nkdi)$ where d is the number of data dimensions which is set to 1, k is the number of clusters which is set to 2, and i is the number of iterations needed until convergence. For data with clustering structure, the number of iterations before convergence is usually small, and the results only slightly improve after the first dozen of iterations. Therefore, Step-2 has the time complexity of $O(n)$ [Salmani (2017)]. Step-3 using BFS for a limited depth (k-hop) has a constant time complexity of $O(1)$ per LTG, resulting in a time complexity of $O(n_{LTGs})$. In the GNN model of Step-4, $|V|$ represents the number of nodes in the subgraph, $|E|$ represents the number of edges in the subgraph, F represents the original feature dimension, and F' represents the output feature dimension. The computational complexity of Step-4 is $O(|V| \times F \times F') + O(|E| \times F')$. In total, the time complexity of our method is $O(n) + O(n) + O(n_{LTGs}) + O(|V| \times F \times F') + O(|E| \times F')$. Considering $|V|$ and $|E|$ of the subgraph is much smaller than n, our method offers significant computational savings compared to full-graph approaches.

7. Conclusion

In this paper, we propose SubG4TJ, a collaborative subgraph classification method based on multidimensional attributes for efficient and accurate HT detection in large-scale netlists. Using the intrinsic concealment characteristics of HTs, our method identifies LTGs, significantly reducing the overall search space. To effectively capture local structural attributes indicative of the presence of HT, we introduce an LTG-aware subgraph mining approach. For subgraph classification, we employ a GNN-based classifier (including GCN and GAT). Extensive experimental evaluations demonstrate that SubG4TJ consistently achieves superior detection performance, reflected by TPR improvements ranging from 0.9 % to 13.6 % and achieves up to 120× speedup, outperforming state-of-the-art methods in both accuracy and runtime efficiency. These results underscore SubG4TJ's practical effectiveness and its potential as a robust solution for addressing hardware security concerns in complex, large-scale integrated circuits.

CRediT authorship contribution statement

Xing Hu: Conceptualization, Methodology, Investigation, Writing – original draft; **Yang Zhang:** Software, Validation, Formal analysis, Writing – review & editing; **Sheng Liu:** Writing – review & editing; **Xiaowen Chen:** Writing – review & editing; **Yaohua Wang:** Writing – review & editing; **Zhenyu Zhao:** Writing – review & editing; **Yang Guo:** Writing – review & editing; **Keqin Li:** Writing – review & editing.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant 61832018.

References

Chen, W., Bai, Z., Pan, G., & Wang, J. (2025). A fast modularity hardware trojan detection technique for large scale gate-level netlists. *Computers & Security*, *148*.
Chen, X., Liu, Q., Yao, S., Wang, J., Xu, Q., Wang, Y., Liu, Y., & Yang, H. (2018). Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(7), 1370–1383. https://doi.org/10.1109/TCAD.2017.2748021
Dupuis, S., Flottes, M.-L., Di Natale, G., & Rouzeyre, B. (2018). Protection against hardware trojans with logic testing: Proposed solutions and challenges ahead. *IEEE Design & Test*, *35*(2), 73–90. https://doi.org/10.1109/MDAT.2017.2766170
Dupuis, S., Rouzeyre, B., Flottes, M. L., Natale, G. D., & Ba, P. S. (2015). New testing procedure for finding insertion sites of stealthy hardware trojans. *EDA Consortium*.
Haider, S. K., Jin, C., Ahmad, M., Shila, D. M., Khan, O., & Dijk, M. V. V. (2019). Advancing the state-of-the-art in hardware trojans detection. *IEEE Transactions on Dependable and Secure Computing*.
Hasegawa, K., Hidano, S., Nozawa, K., Kiyomoto, S., & Togawa, N. (2023). R-HTDetector: Robust hardware-trojan detection based on adversarial training. *IEEE Transactions on Computers*, *72*(2), 333–345. https://doi.org/10.1109/TC.2022.3222090

Hasegawa, K., Yamashita, K., Hidano, S., Fukushima, K., Hashimoto, K., & Togawa, N. (2025). Node-wise hardware trojan detection based on graph learning. *IEEE Transactions on Computers*, *74*(3), 749–761. https://doi.org/10.1109/TC.2023.3280134

Hepp, A., Baehr, J., & Sigl, G. (2022). Golden model-free hardware trojan detection by classification of netlist module graphs. In *2022 design, automation & test in Europe conference & exhibition (DATE)* (pp. 1317–1322). https://doi.org/10.23919/DATE54114.2022.9774760

Hicks, M., Finnicum, M., King, S. T., Martin, M., & Smith, J. M. (2010). Overcoming and untrusted computing base: Detecting and removing malicious hardware automatically. In *31st IEEE symposium on security and privacy, SP 2010, 16–19 May 2010, Berleley/Oakland, California, USA*.

Home :: OpenCores (2025). https://opencores.org/.

Huang, K., & He, Y. (2020). Trigger identification using difference-amplified controllability and dynamic transition probability for hardware trojan detection. *IEEE Transactions on Information Forensics and Security*, *15*, 3387–3400.

Jha, S., & Jha, S. K. (2008). Randomization based probabilistic approach to detect trojan circuits. In *2008 11th IEEE high assurance systems engineering symposium* (pp. 117–124). https://doi.org/10.1109/HASE.2008.37

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks.

Kok, C. H., Ooi, C. Y., Inoue, M., Moghbel, M., Baskara Dass, S., Choo, H. S., Ismail, N., & Hussin, F. A. (2019a). Net classification based on testability and netlist structural features for hardware trojan detection. In *2019 IEEE 28th Asian test symposium (ATS)* (pp. 105–1055). https://doi.org/10.1109/ATS47505.2019.00020

Kok, C. H., Ooi, C. Y., Moghbel, M., Ismail, N., Choo, H. S., & Inoue, M. (2019b). Classification of trojan nets based on SCOAP values using supervised learning. In *2019 IEEE international symposium on circuits and systems (ISCAS)* (pp. 1–5). https://doi.org/10.1109/ISCAS.2019.8702462

Krieg, C. (2023). Reflections on trusting trustHUB. In *2023 IEEE/ACM international conference on computer aided design (ICCAD)*.

Kurihara, T., Hasegawa, K., & Togawa, N. (2020). Evaluation on hardware-trojan detection at gate-level IP cores utilizing machine learning methods. In *2020 IEEE 26th international symposium on on-line testing and robust system design (IOLTS)* (pp. 1–4). https://doi.org/10.1109/IOLTS50870.2020.9159740

Lashen, H., Alrahis, L., Knechtel, J., & Sinanoglu, O. (2023). TrojanSAINT: Gate-level netlist sampling-based inductive learning for hardware trojan detection. In *2023 IEEE international symposium on circuits and systems (ISCAS)* (pp. 1–5). https://doi.org/10.1109/ISCAS46773.2023.10181403

Lo, P.-Y., Chen, C.-W., Hsu, W.-T., Chen, C.-W., Tien, C.-W., & Kuo, S.-Y. (2022). Semi-supervised trojan nets classification using anomaly detection based on SCOAP features. In *2022 IEEE international symposium on circuits and systems (ISCAS)* (pp. 2423–2427). https://doi.org/10.1109/ISCAS48785.2022.9937236

Lu, R., Shen, H., Feng, Z., Li, H., Zhao, W., & Li, X. (2021). Htdet: A clustering method using information entropy for hardware trojan detection. *Tsinghua Science and Technology*, *26*(1), 48–61. https://doi.org/10.26599/TST.2019.9010047

Lu, R., Shen, H., Su, Y., Li, H., & Li, X. (2019). Gramsdet: Hardware trojan detection based on recurrent neural network. In *2019 IEEE 28th Asian test symposium (ATS)* (pp. 111–1115). https://doi.org/10.1109/ATS47505.2019.00021

Mehta, U., & Popat, J. (2022). Transition probability-based detection of hardware trojan in digital circuits.

Muralidhar, N., Zubair, A., Weidler, N., Gerdes, R., & Ramakrishnan, N. (2021). Contrastive graph convolutional networks for hardware trojan detection in third party IP cores. In *2021 IEEE international symposium on hardware oriented security and trust (HOST)* (pp. 181–191). https://doi.org/10.1109/HOST49136.2021.9702276

Nozawa, K., Hasegawa, K., Hidano, S., Kiyomoto, S., Hashimoto, K., & Togawa, N. (2021). Generating adversarial examples for hardware-trojan detection at gate-level netlists. *Journal of Information Processing*, *29*, 236–246.

Popat, J., & Mehta, U. (2016). Transition probabilistic approach for detection and diagnosis of hardware trojan in combinational circuits. In *2016 IEEE annual India conference (INDICON)* (pp. 1–6). https://doi.org/10.1109/INDICON.2016.7838895

Salmani, H. (2017). Cotd: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist. *IEEE Transactions on Information Forensics and Security*, *12*(2), 338–350. https://doi.org/10.1109/TIFS.2016.2613842

Salmani, H. (2022). The improved COTD technique for hardware trojan detection in gate-level netlist. In *Proceedings of the great lakes symposium on VLSI 2022* GLSVLSI ’22 (p. 449–454). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3526241.3530835

Shen, H., Tan, H., Li, H., Zhang, F., & Li, X. (2018). Lmdet: A “naturalness” statistical method for hardware trojan detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *26*(4), 720–732. https://doi.org/10.1109/TVLSI.2017.2781423

Su, Y., Shen, H., Lu, R., & Ye, Y. (2021). A stealthy hardware trojan design and corresponding detection method. *IEEE*.

Tebyanian, M., Mokhtarpour, A., & Shafieinejad, A. (2021). Sc-cotd: Hardware trojan detection based on sequential/combinational testability features using ensemble classifier. *Journal of Electronic Testing*, *37*(4), 473–487.

Trust-Hub.org (2025). https://trust-hub.org/#/home.

Velikovi, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks.

Waksman, A., Suozzo, M., & Sethumadhavan, S. (2013). Fanci: Identification of stealthy malicious logic using boolean functional analysis. In *ACM SIGSAC conference on computer & communications security*.

Wang, L. T., Wu, C. W., & Wen, X. (2006). Vlsi test principles and architectures. Elsevier Morgan Kaufmann Publishers.

Xie, X., Sun, Y., Chen, H., & Ding, Y. (2017). Hardware trojans classification based on controllability and observability in gate-level netlist. *IEICE Electronics Express*, *14*(18), 20170682.

Yamashita, K., Kato, T., Hasegawa, K., Hidano, S., Fukushima, K., & Togawa, N. (2022). Effective hardware-trojan feature extraction against adversarial attacks at gate-level netlists. In *2022 IEEE 28th international symposium on on-line testing and robust system design (IOLTS)* (pp. 1–7). IEEE.

Yao, S., Chen, X., Zhang, J., Liu, Q., & Yang, H. (2015). Fastrust: Feature analysis for third-party IP trust verification. In *IEEE international test conference*.

Yasaei, R., Chen, L., Yu, S. Y., & Faruque, M. A. A. (2025). Hardware trojan detection using graph neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *44*(1), 25–38.

Yasaei, R., Faezi, S., & Al Faruque, M. A. (2022). Golden reference-free hardware trojan localization using graph convolutional network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *30*(10), 1401–1411. https://doi.org/10.1109/TVLSI.2022.3191683

Yasaei, R., Yu, S.-Y., & Al Faruque, M. A. (2021). Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 design, automation & test in Europe conference & exhibition (DATE)* (pp. 1504–1509). https://doi.org/10.23919/DATE51398.2021.9474174

Yen, C.-H., Tsai, J.-C., & Wu, K.-C. (2023). Using path features for hardware trojan detection based on machine learning techniques. In *2023 24th international symposium on quality electronic design (ISQED)* (pp. 1–8). https://doi.org/10.1109/ISQED57927.2023.10129300

Yu, S., Gu, C., Liu, W., & O’Neill, M. (2022). Deep learning-based hardware trojan detection with block-based netlist information extraction. *IEEE Transactions on Emerging Topics in Computing*, *10*(4), 1837–1853. https://doi.org/10.1109/TETC.2021.3116484

Zhang, J., Yuan, F., Wei, L., Sun, Z., & Xu, Q. (2013). Veritrust: Verification for hardware trust. In *Design automation conference (DAC), 2013 50th ACM/EDAC/IEEE*.

Zou, M., Cui, X., Shi, L., & Wu, K. (2018). Potential trigger detection for hardware trojans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(7), pp. 1384-1395.