A Comparative Study of Graph Neural Network Approaches for Hardware Trojan Detection*

*2025 CAD Contest Problem A

Yan-Cheng Li
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
likevin1022@gmail.com

Tzu-Chi Huang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
zchuang0203@gmail.com

Chi-Lun Chen
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
chilunchen28@gmail.com

En-Ling Hsiung
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
doo1222.tw@gmail.com

Tsai-Yen Hsieh
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
hihahanaisme@gmail.com

Li-Heng Yang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
steven29061389@gmail.com

***Abstract*—Recent advancements in Machine Learning (ML), particularly Graph Neural Networks (GNNs), have shown significant potential for Hardware Trojan (HT) detection by identifying anomalous patterns in gate-level netlists. This approach eliminates the dependency on a "golden chip" and promises high accuracy. However, challenges such as severe class imbalance and the stealthy nature of Trojans remain. In this paper, we explore and compare four distinct GNN-based frameworks to address these challenges for the 2025 CAD Contest. The first method (Method A) employs a dual-model GraphSAGE system to perform classification at both circuit and gate levels. The second method (Method B) utilizes a dual-branch architecture, fusing node-level BiGCN features with subgraph-level MLP features. The third method (Method C) integrates a BiGCN with a custom Threshold-Aware Focal Loss to combat class imbalance. The fourth method (Method D) implements data augmentation through trojan insertion and employs a BiGCN with Weighted Focal Loss for improved detection. We present the architecture, feature engineering, and post-processing strategies for each method, culminating in a comparative analysis of their effectiveness.**

***Index Terms*—Hardware Security, Machine Learning, GNN, EDA, Hardware Trojan, Comparative Study**

I. Introduction

The modern semiconductor industry heavily relies on a globalized supply chain, where the design and fabrication of Integrated Circuits (ICs) are often distributed across multiple entities. This paradigm, characterized by the extensive use of third-party Intellectual Property (IP) cores in System-on-Chip (SoC) designs, has enabled unprecedented innovation and reduced time-to-market. However, it has also introduced significant security vulnerabilities. Among the most pernicious threats is the insertion of malicious circuitry, commonly known as Hardware Trojans (HTs), by untrusted parties in the design and fabrication flow. An HT can remain dormant during testing phases, only to be activated under specific conditions post-deployment to leak sensitive information, degrade performance, or cause a complete denial-of-service, thereby jeopardizing the security and reliability of critical systems.

Detecting these stealthy modifications before chip fabrication is of paramount importance. The gate-level netlist, a detailed structural representation of the circuit, serves as a critical stage for pre-silicon security verification. Identifying HTs at this stage can prevent the astronomical costs associated with fabricating compromised hardware.

A. Problem Statement

Despite its importance, detecting HTs at the gate-level remains a formidable challenge. Traditional validation methods, such as logic testing and formal verification, often struggle to achieve sufficient coverage to uncover intentionally hidden Trojans. Post-silicon techniques that rely on side-channel analysis require a trusted "golden chip" for comparison, which is often unavailable in a zero-trust supply chain model. Moreover, these physical measurements are susceptible to process variations and measurement noise.

In response to these limitations, Machine Learning (ML), particularly Graph Neural Networks (GNNs), has emerged as a promising direction. By treating the netlist as a graph, these methods can learn to identify anomalous patterns without a golden reference. However, existing ML-based approaches face two primary hurdles:

1) **Severe Class Imbalance:** Trojan gates constitute a minuscule fraction of the total gates, causing models to develop a trivial bias towards the majority (benign) class.
2) **Subtle Structural Signatures:** HTs are designed to be stealthy. Capturing the nuanced structural and contex-

tual relationships that distinguish them from legitimate circuits requires highly expressive models.

B. Paper Organization

To address these challenges, this paper presents a comprehensive exploration of four distinct GNN-based methodologies. Section II reviews related work. Section III details the architecture, feature selection, and post-processing logic for each of our four proposed methods. Section IV describes the experimental setup for each approach. Section V presents the performance of each method and provides a comparative analysis. Finally, Section VI concludes the paper and discusses future work.

II. Background and Related Work

This section provides a foundational understanding of hardware Trojans and reviews the evolution of detection methodologies, culminating in the state-of-the-art that motivates our work.

A. Preliminaries on Hardware Trojans

A Hardware Trojan (HT) is a malicious modification to an IC design, split into a trigger and a payload. The trigger activates the Trojan, which then executes its payload, ranging from leaking data to causing system failure [1]. As noted by Basak et al. [2], gate-level netlists are a prime target for HT insertion and detection.

B. Conventional and ML-based Detection

Conventional detection methods include logic-based testing, which struggles with coverage, and side-channel analysis, which requires a trusted "golden chip". To overcome these limitations, a paradigm shift towards Machine Learning (ML) has occurred [3].

Early ML methods relied on handcrafted features [4], but their performance was limited. The advent of Graph Neural Networks (GNNs) represented a significant leap, as they automatically learn features from the graph structure of the netlist [5]. More advanced architectures like Graph Attention Networks (GATs) have also been explored [6]. The most relevant work to our own introduced bidirectional GNNs [7], acknowledging that a gate's context is defined by both its inputs and outputs. However, this prior work did not fully address the critical challenges of class imbalance and false positive reduction, which forms the research gap our work aims to fill.

III. Proposed Methodologies

All four methods presented in this work adhere to the expected framework architecture provided by the 2025 CAD Contest organizers. This standardized framework ensures a fair comparison across different approaches while allowing sufficient flexibility for methodological innovation. Figure 1 illustrates the official framework structure that guides the development of each method.

We developed and explored five distinct GNN-based frameworks for hardware Trojan detection. Each method is detailed below.

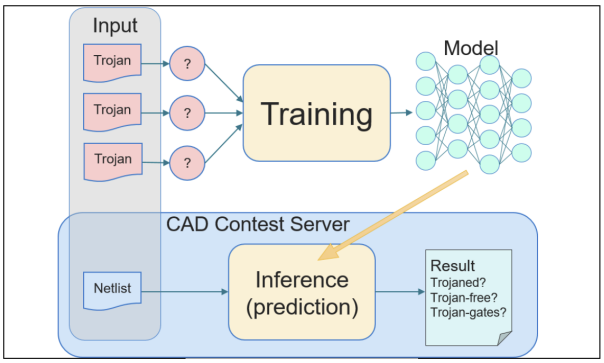

Fig. 1. Expected framework architecture provided by 2025 CAD Contest organizers for Hardware Trojan detection.

A. Method A: Dual-Model Classification with GraphSAGE

The first proposed methodology is designed as a multi-stage pipeline. The core components include (1) feature extraction from the circuit dataset, (2) a dual-model classification system, and (3) a final post-processing and prediction stage.

1) Feature Extraction: In this methodology, we represent each circuit as a graph where each gate is considered a node. For the gate-level analysis, we characterize every node i with a 17-dimensional feature vector, $\mathbf{F}_{gate_i}$. This vector is designed to capture the node's functional, structural, and topological properties. The components of this 17-dimensional vector are defined as follows:

- **Gate Type (10 dimensions):** A one-hot encoding scheme to represent the logical function (NOT, BUF, AND, OR, XOR, NAND, NOR, XNOR, DFF, constant).
- **DFF Port Connectivity (5 dimensions):** A set of five binary flags indicating direct connection to a DFF's 'clk', 'sn' (asynchronous set), 'rn' (asynchronous reset), 'q' (output), or 'd' (data) port. Each flag is 1 if a connection exists, 0 otherwise.
- **Distance to Primary Input (1 dimension):** Topological distance to the nearest PI.
- **Distance to Primary Output (1 dimension):** Topological distance to the nearest PO.

This 17-dimensional vector serves as the input for our classifiers.

2) Model Selection: We employ Graph Neural Networks (GNNs) based on the GraphSAGE architecture. We utilize a dual-model system for classification at two different granularities. Both models share a similar core architecture: multi-layer GNN with 'SAGEConv' layers, 'BatchNorm', a custom node-wise attention mechanism, and residual connections.

a) Model 1: Circuit-Level Classifier: This model classifies the entire circuit graph as "Trojaned" or "Trojan-Free". It processes the graph using 'SAGEConv' layers enhanced with our attention mechanism. After the final GNN layer, a **global mean pooling** operation aggregates all node embeddings into a single graph-level feature vector. This vector is passed through an MLP head for binary classification.

b) Model 2: Gate-Level Classifier: This node classification GNN predicts if each individual gate is "Trojan" or "Benign". It mirrors the GNN architecture (SAGEConv, attention, residuals) but **omits the global pooling layer**. Instead, a final MLP head is applied directly to each node's embedding, producing a per-gate classification.

3) Inference and Post-Processing: First, the 17-dimensional features are extracted. These are fed into the two models: Model 1 produces a circuit-level prediction $P_{circuit}$, and Model 2 produces per-gate predictions P_{gate_i}.

a) Neighbor Voting (Gate-Level Refinement): The initial gate-level predictions are refined using a neighbor voting mechanism to reduce noise. For each gate i, we examine its immediate neighbors. If more than half of the neighbors' predictions contradict the prediction for gate i, P_{gate_i} is flipped. This yields a refined list and count N_{trojan_gates}.

b) Final Decision Logic: A rule-based module combines $P_{circuit}$ and N_{trojan_gates}. The system declares the circuit as **"Trojan-Free"** if either:

1) The circuit-level classifier (Model 1) predicts "Trojan-Free".
2) The total number of trojan gates (N_{trojan_gates}) is less than 15.

Conversely, the circuit is declared **"Trojaned"** only if Model 1 predicts "Trojaned" AND N_{trojan_gates} is 15 or more.

B. Method B: Dual-Branch GNN with Multi-Level Feature Fusion

Our second approach, BiGCN-TIF (Bidirectional Graph Convolutional Network with Topology and Information Fusion), aims to detect and localize hardware Trojans by integrating structural graph analysis and GNNs. The system includes graph construction, model training, and inference post-processing.

1) Graph Construction and Features: The Verilog netlist is converted into a directed graph (gates as nodes, signals as edges).

a) Bidirectional Graph: Two edge index tensors are generated: Forward Edges (`edge_index_fw`, driver to load) and Backward Edges (`edge_index_bw`, load to driver).

b) Multi-Level Feature Extraction: We extract a 41-dimensional feature vector $\mathbf{x}$:

- **Gate-Level:** Gate type one-hot encoding, IO flags.
- **Structural/Topological:** Fanin/fanout counts (2-hop), local depth, DFF presence, and distance features (to PI, PO, DFF).
- **Simulation Statistics:** Logic-1 probability and toggle frequency from Monte-Carlo simulation.

We also introduce subgraph-level contextual features $\mathbf{Z}_s$ by grouping gates into Weakly Connected Components (WCCs) or DFF-based segments.

2) Dual-Branch GNN Architecture: The architecture combines node-level and subgraph-level representations.

a) Node Branch (BiGCN-TIF): This branch consists of three stacked BiGCN_TIF_Layer modules. Each layer uses a pair of GCNConv operators on the forward and backward edge indices. The layer outputs are concatenated to form a 192-dimensional node structural representation $\mathbf{h}_{\text{node}}$.

b) Subgraph Branch: This branch uses a 2-layer Feed-Forward Network (MLP) to encode the statistical subgraph features $\mathbf{Z}_s$ into a 64-dimensional latent space. These embeddings are aligned to all nodes to form the contextual representation $\mathbf{z}_{\text{node}}$.

c) Fusion and Classification: The 192-dim $\mathbf{h}_{\text{node}}$ and 64-dim $\mathbf{z}_{\text{node}}$ are concatenated into a 256-dimensional vector. This fused vector is classified by a two-layer MLP head.

3) Inference and Post-Processing: The model's outputs undergo topology-aware filters.

- **Neighbor Consistency Filter:** Resets a predicted Trojan node if it has no other Trojan neighbors within one hop.
- **Small-Group Pruning:** Connected groups of predicted Trojan gates with size < 5 are discarded.
- **Trojan Count Enforcement:** If the total predicted Trojan count is below 10, all predictions are reset to zero.

C. Method C: BiGCN with Threshold-Aware Focal Loss

Our third framework is designed to overcome key challenges by integrating a bidirectional graph representation, a specialized GNN architecture, a custom loss function, and a targeted post-processing step.

1) Model Selection: We explored multiple architectures, including GCN, GraphSAGE, and a hybrid bidirectional model. To overcome their limitations, we designed a hybrid bidirectional architecture (BiGCN-TIF).

2) Graph Representation and Architecture: We represent the netlist as a graph $G = (V, E)$ where nodes $v \in V$ are gates. For each node, we extract an 18-dimensional feature vector x_v (detailed in Section IV-C). We define forward edges E_{fw} (signal flow) and backward edges E_{bw} (reverse connections).

Our model, `BiGCN-TIF`, stacks three bidirectional GCN layers, processing forward and backward graphs in parallel and fusing the results. A key feature is the dense concatenation of outputs from all intermediate layers,

$$H_{final} = [H^{(1)} \| H^{(2)} \| H^{(3)}],$$

allowing the final classifier to access features from multiple abstraction levels.

3) Training and Post-Processing: To address class imbalance, we propose a **Threshold-Aware Focal Loss**, building upon Focal Loss [8] by adding a dynamic weight for hard-to-classify examples near the decision threshold.

For post-processing, we apply a filter: if the count of predicted Trojan gates in a circuit is less than a hyperparameter $N_{min} = 20$, the entire circuit is reclassified as benign.

D. Method D: Data Augmentation with BiGCN and Weighted Focal Loss

Our fourth approach addresses the limited training data challenge through systematic data augmentation while employing a BiGCN architecture with specialized loss functions.

1) Data Augmentation Strategy: We implement trojan insertion to expand the training dataset. Each trojan definition is parsed to extract gate-level components, which are then inserted into clean benchmark circuits with unique naming ("tj_" prefix) to avoid conflicts. Connection points are strategically selected, and XOR gates combine trojan outputs with victim wires. This process generates approximately 8 augmented circuits per trojan definition, significantly expanding the training set.

2) Feature Extraction: We extract a 30-dimensional feature vector including gate type encoding, DFF indicators, topological metrics (PageRank, betweenness centrality), distance features, and neighborhood analysis patterns (XOR/XNOR concentration, multiple DFF connections, reconvergent fanout).

3) Model Architecture: The model employs a 4-layer BiGCN alternating between SAGEConv and GATConv operations. Forward and backward features are concatenated, normalized, and weighted through attention mechanisms. Residual connections maintain gradient flow.

4) Training Strategy: We use Weighted Focal Loss with parameters $\alpha = 0.111$, $\gamma = 2.0$, and positive class weight 8.0 to handle the 1:8 class imbalance. AdamW optimizer with OneCycleLR scheduling and balanced batch creation (2:1 trojan:clean ratio) are employed. The decision threshold is optimized on validation data every 5 epochs.

E. Method E: Bidirectional GraphSAGE-LSTM with Data Augmentation and Weighted Cross-Entropy

Our approach addresses hardware Trojan detection through systematic data augmentation combined with a novel Bidirectional GraphSAGE-LSTM architecture and class-weighted loss functions.

1) Data Augmentation Strategy: We implement a comprehensive Trojan insertion methodology to expand the limited training dataset. Our augmentation process parses Trojan definitions from text files to extract gate-level components, which are then strategically inserted into clean benchmark circuits with unique naming ('HT_" prefix) to avoid conflicts. The augmentation randomly selects replacement ratios (typically 20% for Trojan gates and 20% for naive replacements) to generate diverse circuit variants, creating approximately 8-12 augmented circuits per original design, significantly expanding the training corpus.

2) Feature Extraction: We extract a comprehensive 21-dimensional feature vector capturing both structural and topological characteristics including graph centrality metrics (degree centrality, betweenness centrality, clustering coefficient), gate connectivity patterns (number of inputs/outputs), node type encoding (is_input, is_output, is_gate), gate family one-hot encoding for 9 gate types (AND, OR, NAND, NOR, NOT, BUF, XOR, XNOR, DFF), and distance-based features (PI: shortest path to nearest input port, PO: shortest path to nearest output port). Selected features undergo normalization to ensure stable training convergence.

3) Model Architecture: The model employs a 3-layer Bidirectional GraphSAGE-LSTM architecture with 256 hidden channels. The bidirectional design processes graph information in both forward and reverse edge directions using separate SAGEConv layers, concatenating the resulting embeddings (512 dimensions) before applying ReLU activation and 20% dropout. Layer-wise embeddings are sequentially fed into an LSTM network to capture temporal dependencies across graph convolution layers. The final MLP classifier includes residual connections and dropout regularization for robust prediction. The complete pipeline framework is illustrated in Figure 2.

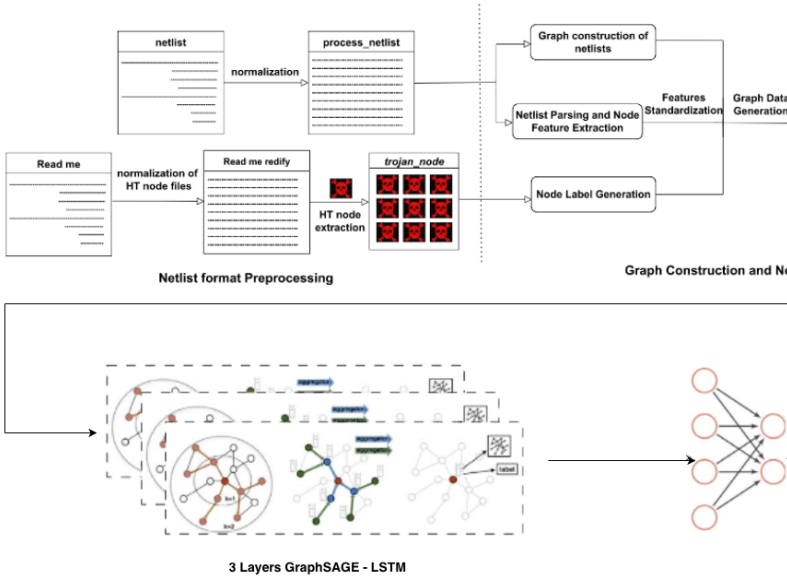

Fig. 2. Bidirectional GraphSAGE-LSTM architecture framework for Method E showing the flow from input features through bidirectional GraphSAGE layers, LSTM processing, and final classification.

4) Training Strategy: We address the severe class imbalance (approximately 1:50 normal:Trojan ratio) using weighted Cross-Entropy loss with class weights [1.0, 50.0]. The Adam optimizer with learning rate 0.001 trains for 1500 epochs with batch size 1. Our training pipeline includes comprehensive validation using both accuracy and F1-score metrics, with NaN loss detection for numerical stability. The model leverages CUDA acceleration when available and implements early convergence monitoring to prevent overfitting.

IV. Experimental Setup

This section details the dataset generation, feature engineering, and training procedures for each of the four methodologies.

A. Setup for Method A

The official dataset provides 20 trojaned circuits and 10 trojan-free circuits. We utilized these 30 base circuits and applied a two-stage data augmentation process to generate a total of 1800 datas for training.

The augmentation process is as follows:

1) **Stage 1: Gate-Level Transformation**
   We first apply randomized, logic-equivalent gate transformations. For each circuit, 10% of its total gates

are randomly selected. These selected gates undergo transformations such as converting `not` and `buf` gates into `xor`, `nand` into `and` + `not`, or `or` into `nor` + `not`, among other rules.
For each of the 30 original circuits, we repeat this randomized process to generate 20 augmented versions. This results in an intermediate set of 600 datas.

2) **Stage 2: Full Logic Transformation**
Next, we take the 600 datas generated in the first stage and create two new, complete variants for each:
   - (a) One variant where all gates in the circuit are converted to `nand`. This creates a new set of 600 datas.
   - (b) One variant where all gates in the circuit are converted to `nor`. This creates a second new set of 600 datas.

Finally, combining the 600 datas from Stage 1, the 600 `nand`-transformed datas, and the 600 `nor`-transformed datas, we obtain the final training set of 1800 datas.

The 17-dimensional features (Section III-A1) were extracted for all gates. The models were trained using PyTorch Geometric, with hyperparameters (e.g., learning rate, batch size) tuned via cross-validation.

B. Setup for Method B

1) Dataset Generation: The final dataset consists of approximately 2,180 netlists ($\approx 2,080$ Trojan-inserted and 100 Trojan-free). Two strategies were used for Trojaned data generation:

1) **RTL-Level Injection:** Adjusting parameters on official RTL Trojan templates and synthesizing them using Cadence GENUS and Synopsys Design Compiler.
2) **Logic-Level Augmentation:** Performing structural transformations on public benchmark cases.

2) Training Procedure: Given the severe class imbalance, Focal Loss is employed ($\alpha \in [0.5, 0.8]$, $\gamma = 2$).

- **Optimizer:** Adam.
- **Learning Rate:** 8×10^{-5}.
- **Early Stopping:** Patience of 100 epochs.
- **Batch Size:** Single-graph batch size (batch size $= 1$).

3) Evaluation Metrics: The final evaluation employs the comprehensive contest-style metric: Total Score $=$ Classification Score $+$ F1 Bonus. F1 Score, Precision, and Recall are used for localization, and Accuracy for overall classification.

C. Setup for Method C

Our experimental methodology was tailored to the competition format, involving a custom training dataset and data-driven feature engineering.

1) Training Dataset Generation: We leveraged the 30 evaluation circuits (with ground truth) to construct a specialized training set.

a) Positive Sample Generation (Trojan Graphs): We extracted the 20 Trojan sub-circuits and performed data augmentation by remapping them with Synopsys Design Compiler using multiple standard cell libraries. This resulted in approx. **360 unique Trojan graph samples**.

b) Negative Sample Generation (Benign Graphs): We extracted a representative set of non-Trojan sub-circuits from the 10 provided clean netlists.

c) Test Set: The final evaluation was performed on the **original, full 30 competition circuits**.

2) Data-Driven Feature Engineering: Our final 18-dimensional feature vector is composed of three categories:

a) Category 1: Compositional Features (9 features): A 9-dimensional one-hot encoded vector for common gate types: `'and'`, `'or'`, `'nand'`, `'nor'`, `'not'`, `'buf'`, `'xor'`, `'xnor'`, `'dff'`. This was justified by a non-uniform distribution of gate types in Trojans.

b) Category 2: Sequential Context Features (5 features): Five binary features to provide pin-level connectivity context for sequential elements: `'is_ck'`, `'is_d'`, `'is_q'`, `'is_rst'`, `'is_set'`.

c) Category 3: Topological Anomaly Features (4 features): Four features capturing structural indicators:

- **`fan_out`:** Count of gates driven by this gate's output. An unusually high fan-out is a classic indicator.
- **`is_reconvergent`:** Flag for gates in reconvergent fan-out structures.
- **`in_isolated_subgraph`:** Flags gates in small, disconnected clusters, which are highly anomalous.
- **`has_tied_inputs`:** Flag for gates with inputs tied to the same signal or constant, an unusual design practice.

3) Evaluation Metrics and Baselines: Evaluation is performed at the circuit-level using Accuracy, Precision, Recall, and F1-Score. We compare our final model against ablated baselines: a **Uni-GCN**, a **BiGCN w/o TIF**, and our model trained with a standard **Cross-Entropy (CE) Loss**.

4) Implementation Details: Implemented using PyTorch/PyTorch Geometric with an Adam optimizer (LR 10^{-3}), batch size of 16, and our custom loss. Inference uses a probability threshold of $\tau = 0.7$ and a post-processing filter of $N_{min} = 20$.

D. Setup for Method D

1) Dataset Generation: Starting from 20 original trojaned and 10 clean circuits, we generated hundreds of augmented circuits through trojan insertion. Clean circuits were randomly assigned to trojan definitions (max 8 per trojan), creating diverse training examples.

2) Training Configuration: Training employed AdamW optimizer with initial learning rate 10^{-4}, weight decay 5×10^{-5}, and OneCycleLR scheduler (max LR 10^{-3}, 30% warmup). Batch size was 64 with 2:1 trojan:clean circuit ratio for balance. The model was trained for 60 epochs with gradient clipping at norm 1.0. Decision threshold was optimized every 5 epochs on validation data to maximize F1 score.

E. Setup for Method E

1) Dataset Generation: Starting from the original circuits, we generated augmented training data through systematic Trojan insertion methodology. Our augmentation process parsed Trojan definitions from text files and inserted gate-level components into clean benchmark circuits with "HT_" prefix naming. Replacement ratios of 20% for Trojan gates and 20% for naive replacements created diverse circuit variants, generating approximately 8-12 augmented circuits per original design to significantly expand the training corpus.

2) Training Configuration: Training employed Adam optimizer with learning rate 10^{-3} for 1500 epochs. Batch size was 1 with graph-level processing due to the nature of circuit-level predictions. The model used 256 hidden channels across 3 GraphSAGE layers with 20% dropout for regularization. Class imbalance (1:50 normal:Trojan ratio) was addressed using weighted Cross-Entropy loss with class weights [1.0, 50.0]. Training included comprehensive validation using both accuracy and F1-score metrics, with NaN loss detection for numerical stability and CUDA acceleration when available.

V. Results and Analysis

This section presents the performance of each framework and provides a comparative analysis.

A. Performance of Method A

1) Official Competition Performance: Method A (Dual-GraphSAGE) performed exceptionally well, successfully identifying **all 20 Trojaned circuits** (100% Recall) while only misclassifying 1 of the 10 clean circuits. This yields a final F1-Score of 97.6% (Table I).

TABLE I
Method A: Final Circuit-Level Performance

Metric	Score
Accuracy	96.7%
Precision	95.2%
Recall (TPR)	100.0%
F1-Score	**97.6%**

2) Gate-Level Detection Consistency: Gate-level analysis showed a high median F1-score of 0.817 across the 20 Trojan cases, indicating reliable localization. The final model achieved significant gains over its pre-tuning variant on challenging cases.

B. Performance of Method B

1) Official Competition Performance: Method A (Dual-Branch) performed exceptionally well, successfully identifying **all 10 clean circuits** (100% Precision) while only misclassifying 1 of the 20 Trojaned circuits. This yields a final F1-Score of 97.4% (Table II).

2) Gate-Level Detection Consistency: Gate-level analysis showed a high median F1-score of 0.9787 across the 20 Trojan cases, indicating reliable localization. The final model achieved significant gains over its pre-tuning variant on challenging cases.

TABLE II
Method B: Final Circuit-Level Performance

Metric	Score
Accuracy	96.7%
Precision	100.0%
Recall (TPR)	95.0%
F1-Score	**97.4%**

C. Performance of Method C

1) Official Competition Performance: Method C (BiGCN w/ Loss) performed exceptionally well, successfully identifying **all 20 Trojaned circuits** (100% Recall) while only misclassifying 3 of the 10 clean circuits. This yields a final F1-Score of 93.0% (Table III).

TABLE III
Method C: Final Circuit-Level Performance

Metric	Score
Accuracy	90.0%
Precision	87.0%
Recall (TPR)	100.0%
F1-Score	**93.0%**

2) Ablation Study: An ablation study (Table IV) confirms the contribution of each design choice, with the final model showing the best F1-Score and lowest false positives (FPs).

TABLE IV
Method C: Ablation Study Performance

Model Variant	Accuracy	F1-Score	# of FPs
Our Method (Final)	**90.0%**	**93.0%**	**3**
BiGCN (Pre-tuning)	83.3%	88.4%	4
Single Directional GCN	80.0%	86.4%	5
2-layer GCN (Baseline)	63.3%	77.6%	10

3) Gate-Level Detection Consistency: Gate-level analysis showed a high median F1-score of 0.688 across the 20 Trojan cases, indicating reliable localization. The final model achieved significant gains over its pre-tuning variant on challenging cases.

D. Performance of Method D

1) Official Competition Performance: Method D (BiGCN w/ Augmentation) achieved strong circuit-level classification with **93.3% accuracy** (28 out of 30 circuits correctly classified). At the gate-level localization across 20 trojaned circuits, the method achieved a precision of 88.3% and recall of 78.6%, yielding an F1-Score of 80.5% (Table V).

The data augmentation strategy successfully improved model generalization across diverse circuit topologies. The Weighted Focal Loss with threshold optimization effectively balanced precision and recall at the gate level, demonstrating competitive performance across various trojan patterns.

TABLE V
METHOD D: FINAL PERFORMANCE

Metric	Score
Circuit Accuracy	93.3%
Gate Precision (avg)	88.3%
Gate Recall (avg)	78.6%
Gate F1-Score (avg)	**80.5%**
Total Score	74.104

E. Performance of Method E

1) Official Competition Performance: Method E (Bidirectional GraphSAGE-LSTM w/ Augmentation) achieved robust circuit-level classification with 83.33% accuracy (25 out of 30 circuits correctly classified) and 90.00% Recall.

TABLE VI
METHOD E: FINAL PERFORMANCE

Metric	Score
Accuracy	83.33%
Precision	85.71%
Recall (TPR)	90.00%
F1-Score	**87.80%**

2) Gate-Level Detection Consistency: Gate-level analysis showed a median F1-score of 0.62 across the 20 Trojan cases. Among cases where detection was attempted, the method achieved varying performance with the best case reaching an F1-score of 0.92. The bidirectional GraphSAGE-LSTM architecture demonstrated particular effectiveness on certain Trojan patterns, while facing challenges with detection sensitivity in other cases, highlighting areas for future model refinement.

F. Comparative Analysis

Here, we compare the five proposed methods (A, B, C, D and E) across several key dimensions based on their respective performance. It is important to note that the primary metrics for Methods A, B, C and E are circuit-level classification scores, whereas the reported Precision, Recall, and F1-Score for Method D are gate-level averages, making a direct comparison of those specific values challenging.

TABLE VII
COMPARATIVE PERFORMANCE OF PROPOSED METHODOLOGIES

Method	Accuracy	Precision	Recall	F1-Score
A: Dual-GraphSAGE	96.7%	95.2%	100%	97.6%
B: Dual-Branch GNN	96.7%	100%	95%	97.4%
C: BiGCN w/ Loss	90.0%	87.0%	100%	93.0%
D: BiGCN w/ Aug	93.3%	88.3%*	78.6%*	80.5%*
E: BiGSAGE w/ Aug	83.3%	85.7%	90.0%	87.8%

*Gate-level average scores, not circuit-level.

a) Analysis of Feature Engineering: The methods employed diverse feature engineering strategies. Method B used the most extensive feature set (41-dim), uniquely incorporating simulation statistics, which could capture dynamic properties at the cost of significant preprocessing overhead. In contrast, Methods A, C, D, and E relied on purely static and topological features. Method A achieved the highest F1-score with the most concise feature set (17-dim), suggesting that a minimal, well-chosen set of topological features was highly effective and potentially less prone to overfitting on this dataset. Method C's data-driven approach (18-dim) targeted known anomaly indicators (e.g., `has_tied_inputs`), reflecting a blend of expert knowledge and statistical analysis. Method D (30-dim) automated the discovery of important nodes through graph centrality metrics like PageRank, offering a scalable alternative to manual feature selection. Method E employed a comprehensive 21-dimensional feature vector that strategically combined graph centrality metrics (degree centrality, betweenness centrality, clustering coefficient) with gate connectivity patterns and distance-based features, achieving a balanced approach between feature richness and computational efficiency.

b) Analysis of Model Architecture: The architectural choices reveal different philosophies. Method A's dual-model system, which separates circuit-level ("if") and gate-level ("where") classification, proved highly effective. This decoupling likely contributed to its superior precision by requiring two distinct forms of confirmation. Methods B, C, D, and E employed single, unified models for node classification. Method B's dual-branch fusion and Method C's dense concatenation of intermediate layers are both powerful techniques for creating rich, multi-scale representations. Method D's hybrid of SAGEConv and GATConv layers attempted to balance neighborhood sampling with attention. Method E introduced a novel Bidirectional GraphSAGE-LSTM architecture that uniquely combines spatial graph convolution with temporal sequence modeling through LSTM layers, enabling the capture of dependencies across graph convolution layers. The success of Method A's simpler, decoupled architecture suggests that for this problem, task-specific models may outperform a single, complex one.

c) Analysis of Post-Processing: All frameworks confirmed the criticality of post-processing to refine raw predictions and reduce false positives. The strategies varied in complexity. Method B employed the most sophisticated, topology-aware filter (consistency, group size, and count), making it theoretically robust. Methods A, C and E used simpler Trojan gate count thresholds (15 and 20, respectively). Method C's higher threshold (20) may explain its lower precision (87.0%) compared to Method A (95.2%), as it is more aggressive in classifying circuits as benign, leading to more false positives among the clean circuits (3 vs. 1). Method D's strategy of dynamically optimizing the decision threshold on a validation set represents a more standard and data-driven machine learning practice, reducing the reliance on hand-tuned heuristics.

d) Analysis of Imbalance Handling: All methods successfully addressed the severe class imbalance, which was crucial for achieving high recall. Methods B, C, and D all used variants of Focal Loss. Method C's novel 'Threshold-Aware Focal Loss' and Method D's 'Weighted Focal Loss' with balanced batching demonstrate advanced strategies to force the model to focus on minority-class examples. Method

E employed a weighted Cross-Entropy loss with class weights [1.0, 50.0], directly addressing the severe 1:50 normal:Trojan ratio through explicit class weighting rather than sophisticated focal mechanisms. Notably, Methods A and C both achieved a perfect 100% circuit-level recall. Method A accomplished this without a custom loss function, relying instead on extensive data augmentation and its dual-model architecture. This implies that architectural design and data volume can be as effective as a specialized loss function in overcoming class imbalance.

VI. Conclusion

In this paper, we conducted a comprehensive study on detecting hardware Trojans at the gate level by developing and comparing five distinct GNN-based methodologies. Our first method (Method A) utilized a dual-model GraphSAGE architecture to decouple circuit and gate-level classification. Our second method (Method B) introduced a dual-branch GNN that fused structural BiGCN features with statistical subgraph features. Our third method (Method C) focused on a BiGCN architecture enhanced by data-driven feature engineering and a custom Threshold-Aware Focal Loss to combat severe class imbalance. Our fourth method (Method D) employed systematic data augmentation through trojan insertion combined with a BiGCN and Weighted Focal Loss. Our fifth method (Method E) introduced a novel Bidirectional GraphSAGE-LSTM architecture that combines spatial graph convolution with temporal sequence modeling, using weighted Cross-Entropy loss and comprehensive data augmentation.

Our experiments, conducted on the 2025 CAD Contest dataset, demonstrate the effectiveness of GNNs for this task. The comparative analysis (Section V-F) highlights the critical impact of choices in feature engineering, model architecture, loss function, and post-processing. Method C, for instance, achieved a perfect recall of 100% and a final F1-Score of 93.0% on the blind test set, validating its approach to imbalance and feature selection. The ablation studies scientifically confirmed the contribution of each component.

This work not only presents five viable solutions to the HT detection problem but also provides a comparative analysis that offers valuable insights into the strengths and weaknesses of different design choices. Future work could involve creating hybrid models that combine the most effective components from each of these five approaches, or applying this methodology to a wider range of public benchmarks.

Acknowledgment

The authors would like to thank the organizers of the 2025 CAD Contest for providing the challenging problem and dataset.

Team Collaboration and Competition Results

This work represents a collaborative effort by six independent students participating in the 2025 CAD Contest (Problem A: Hardware Trojan Detection on Gate Level Netlist). Each team member developed and submitted their own GNN-based methodology. To ensure coherence and foster knowledge sharing, the team conducted weekly meetings throughout the competition period to discuss progress, share insights, and collectively analyze the strengths and weaknesses of each approach.

As a result of these collaborative efforts and individual innovations, the team achieved outstanding recognition at the 2025 CAD Contest. The four methodologies presented in this paper collectively secured one High Distinction Award and four Honorable Mentions in the domestic competition category, demonstrating the effectiveness of diverse GNN-based approaches and the value of systematic comparative analysis in solving hardware security challenges.

References

[1] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Hardware trojan threats: a survey," in *2010 IEEE international conference on computer design (ICCD)*. IEEE, 2010, pp. 7–1.

[2] A. Basak, M. S. W. D'Silva, and S. Bhunia, "Hardware trojans classification for gate-level netlists using multi-layer neural networks," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017, pp. 1–6.

[3] J. Koehler, F. Schellenberg, S.-T. S. Hamdi, and M. A. A. Sanad, "Hardware trojan detection using machine learning: A tutorial," vol. 28, no. 5, 2023, pp. 1–32.

[4] Y. Hao, J. Zhang, and J. Li, "A hardware trojan detection method based on structural features of trojan and host circuits," in *Journal of Physics: Conference Series*, vol. 1651, no. 1, 2020, p. 012117.

[5] G. Wecel, M. A. A. Sanad, and F. Schellenberg, "Hardware trojan detection using graph neural networks," *arXiv preprint arXiv:2204.11431*, 2022.

[6] C. Li, Y. Chen, and X. Li, "Gatrojan: An efficient gate-level hardware trojan detection approach using graph attention networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[7] R. Al-Tawy, A.-H. A. E. Mohamed, and H. M. K. Al-Hassani, "A fine-grained detection method for gate-level hardware trojan based on bidirectional graph neural networks," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 7, pp. 1–13, 2023.

[8] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.