

Black-Box Robustness Probing of Graph Neural Networks for VLSI Circuit Netlists

Rupesh Raj Karn, Johann Knechtel, and Ozgur Sinanoglu
 Center for Cyber Security, New York University, Abu Dhabi, UAE.
 Email: {rupesh.k, johann, ozgursin}@nyu.edu

Abstract—Graph Neural Network (GNN) models are becoming increasingly popular due to their native ability to represent complex integrated circuits as graph data. However, many deployed models remain black boxes with unexamined potential vulnerabilities, including a lack of robustness against perturbations in data distributions. We present a framework for black-box probing for GNN robustness via input-output queries only, utilizing key metrics such as Jacobian, Lipschitz constants, Hessian, prediction margins, robustness radius, and noise stability, relating them all to model performance. We assess various GNN models and seminal architectures, including GraphSAINT, GraphSAGE, GIN, and GAT, all operating on the well-known ISCAS’85 and EPFL benchmarks. We consider gate classification and hardware Trojan detection, the latter being a task that requires excellent robustness by nature. Across node-, subgraph-, and graph-level operation, we find that even highly accurate GNNs can exhibit notable local fragility under perturbations. Overall, our work calls for more stringent consideration of robustness for GNN integration, especially when utilizing third-party service providers, and our framework provides well-defined means for an independent evaluation of this challenge.

Index Terms—Black-box probing, Graph Neural Networks, Hardware Trojan, Jacobian, Sensitivity analysis, ISCAS’85, EPFL

I. INTRODUCTION

The rapid expansion of cloud computing has transformed the deployment of machine learning (ML) models through numerous platforms offering ML as a Service (MLaaS), enabling users to leverage advanced ML capabilities without developing these systems in-house [1], [2], [3]. However, with many service providers available, end users face uncertainty in selecting a model that not only delivers high performance but also exhibits strong robustness against perturbations/variations in the data distribution. This challenge becomes even more important in safety-critical domains such as autonomous vehicles and aerospace systems where reliable node classification [4], graph classification [5], link prediction [6], and related tasks are essential for ensuring the reliable operation of complex systems.

Graph Neural Networks (GNNs) [7], [8] have recently emerged as powerful tools for circuit analysis, where native features such as fan-in, fan-out, and netlist structures are used to reason over circuit components. Despite their high aggregate accuracy, many GNN models deployed via MLaaS [1] are treated as black boxes, with no access to internal parameters or training details. As a result, users cannot directly evaluate the robustness of these systems, even though small perturbations in the input can lead to significant misclassifications [3].

In this paper, we propose a framework for *black-box probing* of GNN robustness in the domain of circuit analysis, an

approach not followed yet in the literature. Our framework solely requires input-output (IO) queries to quantify a range of robustness metrics, enabling the assessment of both local sensitivity and the overall stability of a model’s decision function. Importantly, this framework serves as a practical diagnostic tool for end-users to evaluate and compare third-party ML systems, helping them select the most robust solution for circuit-related applications.

The main contributions of this work are as follows:

- 1) We propose a framework for black-box, IO-only probing of GNNs using state-of-the-art (SOTA) robustness metrics adapted for circuit applications.
- 2) We validate our framework through extensive experiments on established benchmarks, including ISCAS’85 [9] and EPFL [10], for gate classification and Hardware Trojan (HT) detection. For the latter, we also develop and release a new dataset based on TrustHub.
- 3) We discuss how our framework can facilitate the selection of a robust GNN model from MLaaS offerings.
- 4) We provide a full release at https://github.com/rkarn/Probing_GNN_Circuits.

II. BACKGROUND

A. Graph Neural Networks

GNNs enable ML applications in domains with inherently graph-structured data, such as social networks, biological systems, and circuits. Let $G = (V, E)$ denote a graph with node features $X \in \mathbb{R}^{N \times d}$ and adjacency $A \in \mathbb{R}^{N \times N}$. A GNN updates hidden states by aggregating neighborhood information through normalized message passing:

$$H^{(l+1)} = \phi \left(\hat{A} H^{(l)} W^{(l)} \right), \quad (1)$$

where $H^{(0)} = X$, $\hat{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$, D is the corresponding degree matrix required to compute normalized adjacency matrix \hat{A} and ϕ is a non-linear activation. Next, different tasks like node-level classification are realized through readout functions, which aggregate the node embeddings produced after some layers of message passing.

There are various GNN architectures, including the graph convolutional network (GCN) [11], deep graph convolutional neural network (DGCN) [12], GraphSAGE [13], graph isomorphism network (GIN) [14], graph attention network (GAT) [15], etc. The different GNN architectures employ distinct readout functions. For example, GCN and GraphSAGE commonly use the node embeddings from the final layer, whereas GIN

improves representational power by aggregating embeddings from all layers through concatenation or summation, capturing multi-scale structural information.

B. Robustness: Motivation and Metrics

In MLaaS settings, users can only submit IO queries without access to weights, architectures, or training data [1]. Black-box probing is essential here to analyze a model’s sensitivity to structural changes in circuits, revealing vulnerabilities where small perturbations like gate-level tweaks can cause major prediction shifts [3]. With GNNs widely deployed, probing also supports informed model selection [16]; related insights can guide the design of regularization techniques or architecture-level defenses tailored to circuit analysis [17]. For example, if a model exhibits high sensitivity to certain structural patterns like scan chains, domain-aware mitigation strategies such as subgraph normalization or synthesis-invariant training can be introduced by the MLaaS provider.

Next, standard metrics for robustness evaluation are outlined. Together, these metrics provide a multi-faceted view of robustness by revealing how GNN predictions shift under structured or random perturbations across node-, subgraph-, and graph-level classification tasks.

Jacobian Sensitivity [18]: Measures how strongly small changes in input features affect the output. High sensitivity indicates that even minor perturbations in node or subgraph attributes can destabilize predictions.

Local Lipschitz Constant [19]: Characterizes the steepness of the decision boundary around an input. A large value means the model reacts sharply to small feature variations, revealing fragile decision regions in circuit graphs.

Hessian-Based Curvature [20]: Captures the curvature of the model’s output space, reflecting second-order interactions. High curvature suggests predictions can shift dramatically near decision boundaries, especially in critical subcircuits.

Prediction Margin [21]: Represents the confidence gap between the predicted class and the closest alternative. Small margins indicate uncertainty and higher susceptibility to misclassification under perturbations.

Adversarial Robustness Radius [22]: Quantifies the minimum change needed to alter a prediction. A small radius implies that only slight perturbations are sufficient to flip labels, highlighting model vulnerability.

Stability Under Input Noise [23]: Evaluates how consistent the model outputs remain under stochastic noise. Greater instability under noise signals weaker resilience to variations encountered during practical circuit design and testing.

Table I summarizes the expected value ranges for each robustness metric. Low Jacobian, Lipschitz, and curvature values indicate that the GNN reacts gently to perturbations, which is desirable for circuit applications [24], [25], [26], [19],

TABLE I: Robustness metric regimes: ideal vs vulnerable

Metric	Ideal (Robust)	Highly Vulnerable
Jacobian Norm ($\ J_i\ _F$)	Low (< 1)	High (> 3)
Lipschitz Constant (L_i)	Low (< 1)	High (> 3)
Hessian Curvature (λ_{\max})	Low curvature (< 0.1)	Sharp curvature (> 1.0)
Prediction Margin ($M(x_i)$)	Large margin (> 1)	Small margin (< 0.1)
Robustness Radius ($\rho(x_i)$)	Large (> 0.5)	Small (< 0.1)
Noise Stability ($S(x_i)$)	Stable (< 0.1)	Unstable (> 0.5)

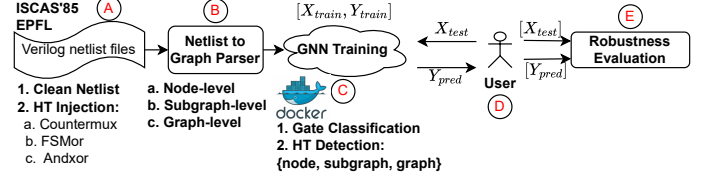


Fig. 1: End-to-End view for black-box probing methodology.

[27]. Conversely, small prediction margins and adversarial radii, or high noise sensitivity, signal fragility and susceptibility to errors in practical circuit deployments [28], [29].

III. METHODOLOGY

Table II positions our work within the broader landscape of robustness analysis in ML and GNN research. Unlike prior works that examine robustness in limited domains (e.g., citation graphs [30], [37], power systems [34], or images [35]), our framework uniquely combines black-box probing, multi-metric quantification, and circuit-aware modeling to assess GNN robustness in VLSI netlists.

A. Overview

The framework’s end-to-end pipeline is illustrated in Fig. 1. It begins with standard *Verilog netlists* obtained from circuit datasets in (A). Next, two parallel mechanisms are employed: in the first, the netlists are directly processed into graph representations, shown in (B); in the second, representative HTs are integrated, after which the modified netlists are also processed, shown in (B). Next, the resulting graph data is split into training and testing sets. GNN models are trained separately for node-, subgraph-, and graph-level operation, shown in (C), for both gate-type classification and HT detection.¹ Each trained model is encapsulated within a Docker container [39] and deployed on a public cloud, to properly emulate an MLaaS environment. The trained model is then applied to the test dataset to generate predictions Y_{pred} , shown in (D), which serve both performance evaluation and robustness assessment. Finally, the predictions are passed to the proposed black-box probing, shown in (E).

B. HT Integration

We adopt well-known templates from TrustHub for representative HTs, namely *Countermux*, *FSMor*, and *Andxor*. Following their respective descriptions, the HTs are integrated into all benchmark netlists. Each HT is devised to remain stealthy under normal conditions while enabling malicious functionality once triggered, as follows. *Countermux*: This HT leverages a multiplexer (MUX) whose select line activates only under rare trigger conditions. Once activated, the HT reroutes signals through a malicious path, thereby altering outputs. *FSMor*: This HT exploits finite state machines (FSMs) by inserting additional hidden states or transitions that cannot be reached during regular operation. When the hidden state is

¹Like HT detection, gate-type classification is critical for real-world security applications. For example, in *netlist reverse engineering*, accurate gate identification underpins higher-level tasks such as sub-circuit recognition [5] and register classification [38], while in *logic locking*, classifying the underlying gates enables attacks and security assessments [4].

TABLE II: High-level comparison with prior art for robustness assessment of ML schemes. \checkmark : supported; \times : not supported

Reference	Model Type	Dataset/Domain	Black-box	Multi-Metric	Circuit-Aware	Key Feature
Zhu [30]	GCN	Citation Graphs	\times	\times	\times	Adversarial GCN regularization
Jin [31]	GCN	Citation Graphs	\times	\times	\times	Graph powering for robustness
Chowdhury [32]	DNN	Circuits	\times	\times	\checkmark	DL analysis for IC testing
Hanif [33]	DNN	Safety Systems	\times	\times	\times	Broad ML reliability survey
Ren [34]	DT/SVM/NN	Power Systems	\checkmark	\times	\times	Robustness index under attacks
Wang [35]	CNN	Images	\checkmark	\times	\times	Lightweight test-time metrics
Geisler [36]	GNN	Citation Graphs	\times	\times	\times	Robust loss functions
Luo [37]	GNN	Citation Graphs	\times	\times	\times	Edge pruning via PTDNet
Wu [3]	GNN	MLaaS (unspecified)	\checkmark	\times	\times	Adversarial fingerprinting
Our Work	Multiple GNNs	Circuits	\checkmark	\checkmark	\checkmark	Sensitivity-driven MLaaS guidance

triggered, the FSM transitions into malicious behavior, such as bypassing security checks or disabling functional logic. *Andxor*: This HT uses AND/XOR gates that remain dormant for most inputs. On rare and specific input patterns, the HT logic is enabled, resulting in corrupted computations.

C. Netlist-to-Graph Parsing and Classification Operations

Each netlist is converted into a graph where nodes represent gates and edges capture the interconnections. Without loss of generality, the netlists are synthesized for AND, OR, NAND, NOR, XOR, XNOR, BUF, and INV/NOT gates. We segment the parsing such that four datasets in CSV format are generated: one for gate-level classification, and three for HT detection at *node*-, *subgraph*-, and *graph*-level, respectively, as follows.

Node-level HT detection: Each gate in the netlist is represented as an individual sample with features such as gate type, fan-in/fan-out, number of neighbors, and its role as a primary or intermediate IO. The classifier distinguishes between HT-infected and clean gates based on these localized attributes.

Subgraph-level HT detection: Connected neighborhoods of gates are grouped to capture localized structures potentially impacted by HT logic. Features include aggregated fan-in/out statistics, distances to IO, and structural motifs that reveal rare connectivity patterns. The classifier operates on these subgraphs to detect HTs embedded in circuit regions/modules.

Graph-level HT detection: The entire circuit netlist is treated as a single graph with global structural and statistical features derived from all nodes and edges. Attributes such as overall gate distribution, global fan-in/out measures, and connectivity depth provide a holistic signature. The classifier determines whether the entire design is HT-free or compromised.

D. Deployment for MLaaS

To simulate a real-world MLaaS environment, we deploy all trained GNNs as standalone services, where virtual users access them as black boxes \textcircled{D} . We embed GNN models with their dependencies like Python scripts, libraries, and pre-trained weights, into Docker containers [39].

Image Creation: A `Dockerfile` defines the environment, including a base image (e.g., `python:3.9`), required packages (e.g., `dgl`, `PyTorch Geometric`, `sklearn`, etc.), and the model files for inference.

Deployment on Public Cloud: Containers are deployed on Amazon Web Services (AWS) via Amazon ECS [40], setting up GNN interfaces through a RESTful API.

Accessing the Deployed Model: The user interacts with the deployed model via HTTPS requests, reflecting the MLaaS paradigm where GNN ML is accessed as a remote service. The

user submits circuit graphs and receives predictions without any control or insight into model internals.

E. Introducing Perturbations Toward Robustness Evaluation

Considering the basic formulation in Sec. II-A, we inject perturbations at different granularities as follows.

At the node level, each feature vector is perturbed:

$$\mathbf{x}'_i = \mathbf{x}_i + \delta\mathbf{x}_i, \quad (2)$$

where $\delta\mathbf{x}_i$ is obtained via Projected Gradient Descent (PGD) [41] with Gaussian initialization, constrained within an L_2 ball of radius ϵ .

At the subgraph level, the concatenated feature matrix $X_S \in \mathbb{R}^{|V_S| \times d}$ of a selected subgraph $V_S \subset V$ is perturbed as

$$X'_S = X_S + \Delta_S, \quad (3)$$

with Δ_S following the same adversarial-stochastic scheme.

At the graph level, the entire feature matrix is modified:

$$X' = X + \Delta_G, \quad (4)$$

where Δ_G combines PGD-driven updates with Gaussian noise under the same norm constraint.

In all cases/levels, the perturbations propagate through the hidden layers and message-passing updates, potentially altering decision boundaries. Even small changes at the node, subgraph, or graph level can flip predictions, thereby exposing fragility in otherwise accurate GNNs.

F. Robustness Evaluation

After repeated inferences on the deployed container, we accumulate a test set of IO pairs, denoted as

$$X_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times d} \quad \text{and} \quad Y_{\text{pred}} \in \mathbb{R}^{N_{\text{test}} \times c},$$

where d is the dimensionality of the node features (recall that each node $v_i \in V$ has $\mathbf{x}_i \in \mathbb{R}^d$) and c is the number of classes.

Next, the standard metrics from Sec. II-B are used in \textcircled{E} . The quality of assessment for each metric is verified using a finite-difference approach to obtain the relative error for each sample. For example, for Jacobian,

$$\text{Relative Error} = \frac{\|\mathbf{J}_i \delta\mathbf{x}_i - (f(\mathbf{x}_i + \delta\mathbf{x}_i) - f(\mathbf{x}_i))\|}{\|f(\mathbf{x}_i + \delta\mathbf{x}_i) - f(\mathbf{x}_i)\| + 10^{-8}} \quad (5)$$

Here, $f(\cdot)$ denotes the trained GNN model's forward function. We apply a small perturbation $\delta\mathbf{x}_i$ sampled from $\mathcal{N}(0, \delta_{\text{fd}}^2 I)$ with $\delta_{\text{fd}} = 10^{-3}$. A similar equation is applied for other robustness metrics.²

²The constant 10^{-8} in Equation 5 is a small numerical stabilizer added

TABLE III: Hyper-parameters for different classification tasks.

Parameter	Gate-Type: Node	HT Detection		
		Node	Subgraph	Graph
GNN Architecture	GCN			
No. of Hidden Layers	2	2	2	2
Feature Dimension (d)	13	26	24	26
Loss Function	Cross-Entropy			
No. of training Samples	48705	171335	471	64
No. of testing samples	12177	36715	102	14
No. of output classes (c)	8	2	2	2
No. of training epochs	50	300	80	180

TABLE IV: Evaluation metrics for different classification tasks.

Metric	Gate-Type: Multi-Class	HT Detection					
		Node		Subgraph		Graph	
		Clean	HT	Clean	HT	Clean	HT
Train Acc. (%)	92.46	100	100	99.36	99.36	98.88	98.88
Test Acc. (%)	92.31	100	100	99.02	99.02	92.86	92.86
Precision	0.89	1	1	0.91	1	1	0.91
Recall	0.92	1	1	1	0.98	0.75	1
F1 Score	0.91	1	1	0.95	0.99	0.85	0.95

TABLE V: Perturbation hyper-parameters across levels. ϵ is the finite-difference step size used for robustness metric computation.

Level	#Samples	ϵ	α	Iterations	Gaussian noise
Node	100	5.0	1.0	40	10^{-3} init
Subgraph	20	4.0	1.0	30	ϵ -scaled init
Graph	14	0.3	0.05	15	$\sigma = 0.1$

IV. EXPERIMENTAL INVESTIGATION

A. General Setup

All code is realized in Python. We use DGL and PyTorch Geometric to construct and train GNNs. The extracted graph datasets and the resulting GNN reflect the complex nature of real-world benchmark circuits. In our reference implementation, we leverage a two-layer GCN; other GNN architectures are covered in Section IV-G. The graph structure is represented by an adjacency matrix A with self-connections included via $\tilde{A} = A + I_N$; further properties are listed in Table III.

B. Perturbation Setup

Following Sec. III-E, we realize adversarial-stochastic perturbations at the node-, subgraph-, and graph-levels. For fair comparison, the same perturbed samples were used across all robustness metrics. Table V summarizes the perturbation hyper-parameters; further setup details are outlined next.

Node-level perturbations: For the test dataset, we selected 100 nodes per class to build a balanced pool. For each selected node, its feature vector \mathbf{x}_i was perturbed according to Equation 2 and Table V. *Subgraph-level perturbations:* We extracted 20 subgraphs per class for the test dataset. Each subgraph’s feature matrix $X_S \in \mathbb{R}^{|V_S| \times d}$ was perturbed following Equation 3 using a PGD routine. *Graph-level perturbations:* We perturbed the entire node-feature matrix $X \in \mathbb{R}^{N \times d}$ of 14 graphs from the test dataset according to Equation 4. For each graph, we applied PGD for 15 steps with step size $\alpha = 0.05$ and budget $\epsilon = 0.3$, followed by addition of Gaussian noise with $\sigma = 0.1$.

to prevent division by zero when the output change is extremely small. The perturbation magnitude δ_{fid} is selected to be small enough to stay within the linear regime of the model’s response. The relative error represents how closely the analytically computed robustness metric matches its finite-difference approximation, serving as a measure of computational accuracy. A small value means the computed metric is accurate and trustworthy, while a large value means there is a noticeable mismatch that could affect the analysis.

C. Results: Baseline Performance

As indicated, we consider four training scenarios: one for gate-type multi-label classification and three for HT detection, with results reported for each binary class. Table IV summarizes the baseline GNN evaluation results after training, showcasing a highly accurate GNN model. Next, this model is perturbed and its robustness is inspected.

D. Results: Baseline Robustness

We compute each robustness metric across node-, subgraph-, and graph-level tasks. Tables VI and VII report the averages and standard deviations across perturbed samples, while Table VIII shows the relative errors (Equation 5). For assessment, these values are to be compared against the robustness thresholds summarized in Table I, as discussed next.

Jacobian Norm: For HT detection (Table VI), infected samples show high Jacobian norms at the node level (2.39 vs. 0.57 clean) and subgraph level (0.99 vs. 0.49 clean), indicating strong local sensitivity, while the graph level is unremarkable ($\approx 10^{-4}$). Relative errors (Table VIII) are small at node level but higher at subgraph/graph levels. For gate-type classification (Table VII), *AND* (7.83) and *XOR* (8.07) gates are most sensitive, while the overall relative error is very low (0.002).

Local Lipschitz Constant: For HT detection, there are much steeper decision boundaries at node level (2.35 vs. 0.55 clean) and subgraph level (0.99 vs. 0.49), whereas graph-level boundaries are regular. Sensitivity for gate-type classification is again highest for *XOR* (6.73) and *AND* (5.84) gates.

Hessian-Based Curvature: For HT detection, the curvature notably rises at the node level (0.014 vs. 0.001 clean), while subgraph and graph values remain unremarkable. Relative errors are large, indicating instability in curvature estimation. For gate-type classification, values are generally low, just for *XOR* (0.122) and *AND* (0.112) slightly higher than others.

Prediction Margin: For HT detection, values spike at the node level (14.44 vs. 0.86 clean) and even more at subgraph level (87.77 vs. 2.77 clean), showing brittle overconfidence, while graph-level margins even drop (0.13 vs. 0.35 clean). For gates, *BUF* (6.73) and *OR* (5.75) show strong margins, whereas *NAND* (1.22) and *NOR* (1.45) exhibit more fragile margins.

Adversarial Robustness Radius: For HT detection, node- and subgraph-level radii saturate at 20.0, but collapse to zero at the graph level, exposing fragility for the latter. For gate classification, *BUF* (0.032) and *XOR* (0.028) are most robust, while *NAND/NOR* are most susceptible.

Stability Under Input Noise: For HT detection, nodes are about four times less stable (0.097 vs. 0.023 clean), whereas subgraph instability rises only lightly (5.46 vs. 4.68 clean) and graph level remains stable. For gate classification, *INV* (0.027) and *BUF* (0.038) are least affected, while *XOR* (0.081) and *AND* (0.069) are most sensitive.

Overall, these results show that HT-infected circuits exhibit systematically higher sensitivity and instability than gate-type classification, especially at the node and subgraph levels. Graph-level robustness can also be fragile for wide-scale modifications induced by consistent perturbations at the lower level.

TABLE VI: Robustness metric values (mean \pm std) for HT detection at node-, subgraph-, and graph-level classification.

Metric	Node		Subgraph		Graph	
	Clean	HT	Clean	HT	Clean	HT
Jacobian Norm	0.5691 \pm 0.2002	2.3851 \pm 0.2951	0.4876 \pm 0.1973	0.9912 \pm 2.6177	0.0001 \pm 0.0002	0.0001 \pm 0.0003
Lipschitz Constant	0.5544 \pm 0.1906	2.3476 \pm 0.3050	0.4876 \pm 0.1973	0.9912 \pm 2.6177	0.0001 \pm 0.0002	0.0001 \pm 0.0003
Hessian Curvature	0.0010 \pm 0.0027	0.0142 \pm 0.0433	0.0085 \pm 0.0151	0.0001 \pm 0.0002	0.0000 \pm 0.0000	0.0000 \pm 0.0000
Prediction Margin	0.8551 \pm 1.1146	14.4402 \pm 1.9893	2.7679 \pm 0.9341	87.7748 \pm 265.9565	0.3489 \pm 0.1507	0.1280 \pm 0.1728
Robustness Radius	20.0000 \pm 0.0000	20.0000 \pm 0.0000	20.0000 \pm 0.0000	20.0000 \pm 0.0000	0.0000 \pm 0.0000	0.0000 \pm 0.0000
Noise Stability	0.0230 \pm 0.0093	0.0971 \pm 0.0260	4.6820 \pm 1.9321	5.4611 \pm 4.0297	0.0009 \pm 0.0008	0.0014 \pm 0.0017

TABLE VII: Robustness metric results for gate-type classification.

Metric	AND	OR	NAND	NOR	XOR	XNOR	BUF	INV
Jacobian Norm	7.827 \pm 1.832	3.783 \pm 0.130	6.422 \pm 2.102	4.702 \pm 1.164	8.068 \pm 1.092	6.430 \pm 2.659	3.422 \pm 0.314	3.757 \pm 0.265
Lipschitz Constant	5.838 \pm 1.382	2.631 \pm 0.074	4.717 \pm 1.542	3.298 \pm 0.982	6.730 \pm 1.045	4.644 \pm 1.843	2.326 \pm 0.124	2.739 \pm 0.253
Hessian Curvature	0.112 \pm 0.035	0.096 \pm 0.029	0.108 \pm 0.031	0.089 \pm 0.028	0.122 \pm 0.035	0.102 \pm 0.033	0.088 \pm 0.025	0.091 \pm 0.027
Prediction Margin	3.459 \pm 1.490	5.745 \pm 0.863	1.223 \pm 0.665	1.451 \pm 0.705	4.835 \pm 1.051	2.073 \pm 1.146	6.734 \pm 0.559	4.567 \pm 0.476
Robustness Radius	0.015 \pm 0.005	0.022 \pm 0.004	0.012 \pm 0.006	0.013 \pm 0.005	0.028 \pm 0.007	0.018 \pm 0.004	0.032 \pm 0.003	0.026 \pm 0.005
Noise Stability	0.069 \pm 0.018	0.046 \pm 0.005	0.060 \pm 0.019	0.046 \pm 0.010	0.081 \pm 0.013	0.054 \pm 0.022	0.038 \pm 0.004	0.027 \pm 0.003

TABLE VIII: Average relative errors for robustness metrics.

Metric	HT			Gate-Type (Node)
	Node	Subgraph	Graph	
Jacobian Norm	0.0013	0.4001	0.3045	0.002
Lipschitz Constant	0.0052	0.2144	0.3045	2.584
Hessian Curvature	0.9360	0.6667	1.4967	0.401
Prediction Margin	0.0000	0.0002	0.7896	0.000
Robustness Radius	0.0000	0.0000	0.0000	0.153
Noise Stability	1.1186	0.0369	0.9988	0.089

TABLE IX: PIER for baseline perturbations. Test samples are same as in Table VI and VII.

Perturbation	HT			Gate-Type (Node)
	Node	Subgraph	Graph	
Table V	84	9.68	28.57	30.68

TABLE X: PIER for HT detection under different perturbations. Other variables remain as in Table V.

Perturbation	Node	Perturbation	Subgraph	Perturbation	Graph
$\epsilon=12; \alpha=2$	100	$\epsilon=5; \alpha=1$	19.38	$\epsilon=20; \alpha=10.2$	21.43
$\epsilon=12; \alpha=6$	83	$\epsilon=5; \alpha=0.4$	9.68	$\epsilon=50; \alpha=10$	57.14
$\epsilon=2; \alpha=0.4$	50.5	$\epsilon=3; \alpha=0.8$	13.67	$\epsilon=50; \alpha=0.9$	42.86

TABLE XI: GNN performance on perturbed samples. Test samples are same as in Table V.

Classification Type	Test Acc. (%)	Precision	Recall	F1 Score
HT Node-level	16.00	0.121	0.160	0.138
HT Subgraph-level	90.32	0.907	0.903	0.904
HT Graph-level	64.28	0.494	0.643	0.559
Gate-type Node-level	69.32	0.678	0.693	0.633

TABLE XII: Interpretation. Relative Deviation indicates how far the metrics are from model-wide averages, in standard deviations ν , shown separately for gate-type “()” and HT detection “[]”.

Metric	Sensitive Gate / HT Level	Observed Behavior	Relative Deviation
Jacobian Norm	(AND, XOR) [Node]	(High input sensitivity); [HT $\uparrow 4\times$ clean (2.39 vs. 0.57)]	(+0.90 ν) [+4.0 ν]
Lipschitz Constant	(AND, XOR) [Node]	(Rapid output changes); [HT $\uparrow 4\times$ clean (2.35 vs. 0.55)]	(+0.83 ν) [+3.9 ν]
Hessian Curvature	(XOR, AND) [Node]	(Sharper decision surface); [HT curvature $\uparrow 14\times$ clean (0.0142 vs. 0.0010)]	(+0.65 ν) [+13.9 ν]
Prediction Margin	(NAND, NOR) [Subgraph]	(Low confidence for certain gates); [HT subgraph spike $\uparrow 30\times$ clean (87.77 vs. 2.77)]	(-1.11 ν) [+29.9 ν]
Robustness Radius	(NAND) [Graph]	(Low perturbation tolerance); [HT graph radius collapsed to 0.0 vs. 20.0 clean]	(-1.15 ν) [- $\infty\nu$]
Noise Stability	(XOR) [Node]	(Output unstable to noise); [HT instability $\uparrow 4.2\times$ clean (0.097 vs. 0.023)]	(+1.27 ν) [+4.2 ν]

E. Results: Impact of Perturbations

For this set of experiments, we propose perturbation-induced error rate (PIER) which quantifies the percentage of test samples whose predictions change under perturbations. PIER provides a complementary perspective to the robustness metrics in Section IV-D, directly measuring how often the trained GNN is fooled by perturbations.

Table IX reports the baseline PIER under the perturbation settings of Table V. For HT detection, node-level classification is by far the most vulnerable, with PIER reaching 84%. This aligns with the high Jacobian norms, steep Lipschitz constants, and unstable noise sensitivity observed earlier (Table VI), confirming that even small perturbations frequently flip node-level predictions. Subgraph-level detection is substantially more robust, with PIER only reaching 9.68%, consistent with the relatively small impact seen for Jacobian and Lipschitz values. Graph-level classification shows medium susceptibility, with 28.57% PIER. For gate-type classification, node-level PIER is 30.68%, indicating moderate sensitivity, but less fragility than HT node-level detection.

Table X further explores PIER under different perturbation budgets. Node-level HT detection remains fragile across all settings, with PIER ranging from 50.5% up to 100%. Subgraph-level detection is relatively robust, with rates ranging between 9.68% and 19.38%. Graph-level detection shows strong dependence on the perturbation scale, ranging from 21.43% to 57.14%, reconfirming its medium vulnerability.

The performance evaluation in Table XI reveals the practical consequences of perturbations.³ For node-level HT detection, accuracy collapses to 16%, with precision, recall, and F1 all nearing zero—this fully confirms the hypothesized significant fragility. In contrast, subgraph-level HT detection maintains high accuracy (90.32%) and strong precision/recall values, corroborating its robustness. Graph-level HT detection drops to 64.28% accuracy, reflecting medium robustness. For gate-type classification, accuracy stabilizes at 69.32% (from 92.31% without perturbations), likewise confirming the medium robustness.

F. Interpretation

The robustness and perturbation results jointly highlight where GNN vulnerabilities are concentrated. For example, Table XII shows that, for gate-type classification, *and* and *xor* gates are consistently fragile, with Jacobian and Lipschitz sensitivities nearly one standard deviation above average and sharper Hessian curvatures, while *nand* and *nor* gates exhibit low prediction margins. For HT detection, node-level classification is dominated by gradient-based fragility, with

³A prediction flip, while counted in PIER, does not always correspond to a classification error; a previously misclassified sample may flip into the correct label, simultaneously increasing both PIER and evaluation accuracy. Conversely, unchanged predictions (not accounted in PIER) may still be incorrect, keeping accuracy relatively low while leaving PIER unaffected. This explains why PIER and accuracies under perturbations do not always sum up to 100%.

TABLE XIII: Comparative results for HT detection (top) and gate-type classification (bottom). Perturbation settings are same as in Table V.

Architecture	Robustness Metric (Average) on Perturbed Samples						Evaluation Metric on Clean Samples				PIER (%)
	Jacobian	Lipschitz	Hessian	Pred. Margin	Robust. Radius	Noise Stab.	Test Acc. (%)	Precision	Recall	F1-score	
GCN (Ours) Node	1.477	1.451	0.008	7.647	20.000	0.0601	100	1.0	1.0	1.0	84
HW2VEC [42] Node	1.467	1.465	0.005	6.549	20.000	0.0598	99.99	0.99	0.99	0.99	46.5
TrojanSAINT [43] Node	3.957	3.955	0.052	20.308	20.000	0.159	97.76	0.979	0.977	0.977	72
GCN (Ours) Subgraph	0.739	0.739	0.004	45.271	20.000	0.000	99.02	0.955	0.99	0.97	9.68
HW2VEC [42] Subgraph	0.351	0.350	0.012	6.061	20.000	2.507	95.50	0.959	0.951	0.9535	6.45
TrojanSAINT [43] Subgraph	3.183	3.182	0.079	4.815	5.769	1.221	97.76	0.9794	0.9776	0.9779	45
GCN (Ours) Graph	0.0001	0.0001	0.000	0.238	0.000	0.0012	92.86	0.955	0.875	0.9	28.57
HW2VEC [42] Graph	0.0001	0.0001	0.000	0.333	0.000	0.0018	92.86	0.9429	0.9286	0.9307	78.57
TrojanSAINT [43] Graph	0.0001	0.0001	0.0001	0.543	0.000	0.0042	100	1.0	1.0	1.0	92.86
GCN (Ours)	5.7233	4.2474	0.1010	3.6833	0.0208	0.0550	92.31	0.89	0.92	0.90	30.68
GraphSAINT [44]	30.6234	24.4172	5.5591	8.6860	1.6717	0.2787	96.68	0.97	0.97	0.97	35.72
GraphSAGE [13], [45]	12.0963	8.6557	0.6771	6.2865	2.5542	0.1121	93.13	0.92	0.93	0.92	39.57
GIN [14], [4]	21.4701	19.4393	1.2139	9.8507	2.4510	0.1851	94.41	0.94	0.94	0.94	47.25
GAT [15], [46]	8.4333	5.8549	0.0525	6.4747	4.1354	0.0779	92.19	0.90	0.92	0.90	38.47

Jacobian and Lipschitz several times higher than clean nodes, an order of magnitude larger Hessian curvatures, and noise stability being much worse. Subgraph-level HT samples show extreme prediction margin spikes, while graph-level detection collapses in robustness radius, dropping to zero for HT graphs.

These results confirm that vulnerabilities differ by granularity: node-level HT detection is most sensitive to gradient metrics, subgraph-level to margin instability, and graph-level to adversarial radius, whereas gate-type vulnerabilities are more localized to specific families. This consolidated view underscores the need for multi-metric evaluation, as no single measure captures the full spectrum of GNN fragility.

G. Results: Comparison with Prior Art Under Perturbations

Table XIII compares the robustness and performance of our representative GCN implementation against relevant prior art, namely HW2VEC [42], TrojanSAINT [43], as well as four SOTA GNN architectures, namely GraphSAINT [44], GraphSAGE [13], GIN [14], and GAT [15].

Our GCN achieves a more balanced robustness–performance tradeoff than both specialized circuit models (HW2VEC, TrojanSAINT) and generic SOTA GNNs (GraphSAINT, GraphSAGE, GIN, GAT). For some examples, node- and subgraph-level HT detection of ours maintains lower sensitivities (Jacobian ~ 1.5 vs. ~ 4.0 in TrojanSAINT) while preserving accuracy, and our gate-type classification achieves moderate accuracy (92.31%) with substantially lower Jacobian/Lipschitz values than GraphSAINT or GIN. The PIER results show further complex, granularity-dependent trade-offs. For HT detection, while our GCN is highly susceptible to node-level perturbations, it demonstrates superior resilience over prior art at the graph level, all while offering superior resilience for gate-type classification in general. This underscores that no single architecture is universally robust, and highlights the importance of selecting models based on the specific task at hand.

V. DISCUSSION

A. Guidance for Robust Model Selection in MLaaS

Our framework provides a systematic way to guide end-users in selecting reliable GNN-based services from multiple MLaaS providers. Metrics with low relative errors (Table VIII), like prediction margin and Jacobian, are most reliable for model selection, while Lipschitz and Hessian should be used as secondary criteria, due to higher variability. The task’s nature and granularity both shift the importance of different metrics: for subgraph classification, smooth embedding transitions make

Jacobian and Lipschitz more informative than margins, while in graph classification, stability measures such as robustness radius and noise robustness dominate.

Resulting guidelines for users are to prioritize margin and Jacobian for node-level HT detection, Lipschitz smoothness for subgraph tasks, and stability metrics for graph-level tasks, all ensuring that robustness assessment is matched to the classification context under black-box constraints.

B. Defense Strategies

Our black-box probing points to opportunities for targeted defenses against perturbations. For gate-type classification, sensitivity-aware defenses such as Jacobian regularization, spectral normalization, and margin-aware calibration can mitigate steep and fragile decision surfaces. For HT detection, node-level tasks require stronger measures like adversarial training, margin boosting, and gradient-constrained regularization to counter extreme sensitivity and high PIERs. At the subgraph level, embedding-space smoothing, contrastive regularization, and confidence calibration are recommended to address overconfident yet fragile predictions. Finally, for graph-level classification, noise-robust training through adversarial augmentation, feature denoising, or dropout-based smoothing can stabilize pooled representations. Toward proper utilization of these various defenses, a granularity-aware robustness enhancement as enabled by our framework is essential.

VI. CONCLUSION

While GNNs are well established in circuit design, prior art has overlooked their robustness, defined as reliability under perturbations / shifts in data distributions. Here, we present a first-of-its-kind, metrics-driven framework for black-box robustness analysis of GNNs applied to circuit netlists.

Our experimental investigation demonstrates that the proposed framework effectively identifies distinct vulnerability patterns in both gate-type classification and HT detection, revealing how robustness characteristics vary across tasks and decision granularities. These insights enable the selection of robust GNN models from MLaaS offerings and can also guide defense efforts. Accordingly, for future work, our framework could be used to systematically analyze recent adversarial works like AttackGNN [47], to better understand their effectiveness and guide the development of more targeted defenses.

Overall, our approach offers actionable insights into architecture-dependent vulnerabilities and MLaaS selection for sensitive GNN applications on circuits.

REFERENCES

- [1] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, “{MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters,” in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 945–960.
- [2] R. R. Karn, P. Kudva, and I. A. M. Elfadel, “Dynamic autoselection and autotuning of machine learning models for cloud network analytics,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1052–1064, 2018.
- [3] B. Wu, X. Yuan, S. Wang, Q. Li, M. Xue, and S. Pan, “Securing graph neural networks in mlaas: A comprehensive realization of query-based integrity verification,” in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 2534–2552.
- [4] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, “Omla: An oracle-less machine learning-based attack on logic locking,” *IEEE Transactions on Circuits and Systems II*, vol. 69, no. 3, pp. 1602–1606, 2021.
- [5] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, “Gnn-re: Graph neural networks for reverse engineering of gate-level netlists,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2435–2448, 2021.
- [6] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, “Muxlink: Circumventing learning-resilient mux-locking using graph neural network-based link prediction,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 694–699.
- [7] H. Hu, M. Yao, F. He, and F. Zhang, “Graph neural network via edge convolution for hyperspectral image classification,” *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2021.
- [8] R. R. Karn and O. Sinanoglu, “Benchmarking backdoor attacks on graph convolution neural networks: A comprehensive analysis of poisoning techniques,” in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2024, pp. 149–174.
- [9] M. C. Hansen, H. Yalcin, and J. P. Hayes, “Unveiling the iscas-85 benchmarks: A case study in reverse engineering,” *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.
- [10] L. Amari, P.-E. Gaillardon, and G. De Micheli, “The eplf combinational benchmark suite,” in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.
- [11] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *ICLR*.
- [12] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in *AAAI*, 2018.
- [13] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *NIPS*, 2017.
- [14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in *ICLR*, 2019.
- [15] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in *ICLR*, 2018.
- [16] K. M. Patel, “Machine learning as a service (mlaaS) selection for iot environments,” Ph.D. dissertation, Curtin University, 2024.
- [17] A. Abomakhelb, K. A. Jalil, A. G. Buja, A. Alhammadi, and A. M. Alenezi, “A comprehensive review of adversarial attacks and defense strategies in deep neural networks,” *Technologies*, vol. 13, no. 5, 2025.
- [18] A. Ezertas and S. Eyi, “Performances of numerical and analytical jacobians in flow and sensitivity analysis,” in *19th AIAA Computational Fluid Dynamics*, 2009, p. 4140.
- [19] M. Jordan and A. G. Dimakis, “Exactly computing the local lipschitz constant of relu networks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 7344–7353, 2020.
- [20] Z. Yao, A. Gholami, Q. Lei, K. Keutzer, and M. W. Mahoney, “Hessian-based analysis of large batch training and robustness to adversaries,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [21] C. Zheng, V. Malbasa, and M. Kezunovic, “Regression tree for stability margin prediction using synchrophasor measurements,” *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1978–1987, 2012.
- [22] C. Qin, J. Martens, S. Goyal, D. Krishnan, K. Dvijotham, A. Fawzi, S. De, R. Stanforth, and P. Kohli, “Adversarial robustness through local linearization,” *Advances in neural information processing systems*, vol. 32, 2019.
- [23] X. Li, P. Yang, Y. Gu, X. Zhan, T. Wang, M. Xu, and C. Xu, “Deep active learning with noise stability,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 12, 2024, pp. 13 655–13 663.
- [24] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, “Sensitivity and generalization in neural networks: an empirical study,” in *International Conference on Learning Representations (ICLR)*, 2018.
- [25] D. Jakubovitz and R. Giryes, “Improving dnn robustness to adversarial attacks using jacobian regularization,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 514–529.
- [26] A. Virmaux and K. Scaman, “Lipschitz regularity of deep neural networks: analysis and efficient estimation,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [27] S.-M. Moosavi-Dezfooli, A. Fawzi, J. Uesato, and P. Frossard, “Robustness via curvature regularization, and vice versa,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9078–9086.
- [28] G. Elsayed, D. Krishnan, H. Mobahi, K. Regan, and S. Bengio, “Large margin deep networks for classification,” *Advances in neural information processing systems*, vol. 31, 2018.
- [29] Y. Wang, B. Dong, K. Xu, H. Piao, Y. Ding, B. Yin, and X. Yang, “A geometrical approach to evaluate the adversarial robustness of deep neural networks,” *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 19, no. 5s, pp. 1–17, 2023.
- [30] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, “Robust graph convolutional networks against adversarial attacks,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1399–1407.
- [31] M. Jin, H. Chang, W. Zhu, and S. Sojoudi, “Power up! robust graph convolutional network via graph powering,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, 2021, pp. 8004–8012.
- [32] A. B. Chowdhury, B. Tan, S. Garg, and R. Karri, “Robust deep learning for ic test problems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 183–195, 2021.
- [33] M. A. Hanif, F. Khalid, R. V. W. Putra, S. Rehman, and M. Shafique, “Robust machine learning systems: Reliability and security for deep neural networks,” in *2018 IEEE 24th international symposium on on-line testing and robust system design (IOLTS)*. IEEE, 2018, pp. 257–260.
- [34] C. Ren and Y. Xu, “Robustness verification for machine-learning-based power system dynamic security assessment models under adversarial examples,” *IEEE Transactions on Control of Network Systems*, vol. 9, no. 4, pp. 1645–1654, 2022.
- [35] J. Wang, J. Chen, Y. Sun, X. Ma, D. Wang, J. Sun, and P. Cheng, “Robot: Robustness-oriented testing for deep learning systems,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 300–311.
- [36] S. Geisler, T. Schmidt, H. Şirin, D. Zügner, A. Bojchevski, and S. Günnemann, “Robustness of graph neural networks at scale,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 7637–7649, 2021.
- [37] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang, “Learning to drop: Robust graph neural network via topological denoising,” in *Proceedings of the 14th ACM international conference on web search and data mining*, 2021, pp. 779–787.
- [38] S. D. Chowdhury, K. Yang, and P. Nuzzo, “Reignn: State register identification using graph neural networks for circuit reverse engineering,” in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.
- [39] D. Merkel *et al.*, “Docker: lightweight linux containers for consistent development and deployment,” *Linux j*, vol. 239, no. 2, p. 2, 2014.
- [40] P. Acuña, “Amazon ec2 container service,” in *Deploying Rails with Docker, Kubernetes and ECS*. Springer, 2016, pp. 69–98.
- [41] G. Chen, D. Li, and J. Zhang, “Iterative gradient projection algorithm for two-dimensional compressive sensing sparse image reconstruction,” *Signal Processing*, vol. 104, pp. 15–26, 2014.
- [42] S.-Y. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. Al Faruque, “Hw2vec: A graph learning tool for automating hardware security,” in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 13–23.
- [43] H. Lashen, L. Alrahis, J. Knechtel, and O. Sinanoglu, “Trojansaint: Gate-level netlist sampling-based inductive learning for hardware trojan detection,” in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.
- [44] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, “GraphSAINT: Graph sampling based inductive learning method,” in *International Conference on Learning Representations*, 2020.
- [45] K. Huang and C. Chen, “Subgraph generation applied in graphsage deal with imbalanced node classification,” *Soft Computing*, vol. 28, no. 17, pp. 10 727–10 740, 2024.
- [46] A. G. Vrahatis, K. Lazaros, and S. Kotsiantis, “Graph attention networks: a comprehensive review of methods and applications,” *Future Internet*, vol. 16, no. 9, p. 318, 2024.
- [47] V. Gohil, S. Patnaik, D. Kalathil, and J. Rajendran, “Attackgmn: red-teaming gmn in hardware security using reinforcement learning,” in *33rd USENIX Conference on Security Symposium*, 2024.