

# Interpretable Graph Neural Networks for Fault Detection in 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**—This work presents a framework for accurate and interpretable fault detection in digital circuit netlists using multiple state-of-the-art Graph Neural Network (GNN) architectures. Targeting three representative fault types—stuck-at, bridging, and glitch—we formulate the detection task as a multi-label node classification problem. Using a robust parsing pipeline, we construct graph datasets from the ISCAS85 and EPFL benchmarks, embedding both structural attributes and novel relational features. Results demonstrate that most GNNs achieve over 90% accuracy, with the proposed relational features consistently boosting performance. Furthermore, we leverage these relational features for model interpretability, successfully highlighting features most relevant to circuit faults.

**Index Terms**—Graph Neural Networks, Faults, Gate-level Netlists, Interpretability, Similarity Score, SCOAP Controllability

## I. INTRODUCTION

Fault detection in digital circuits remains a major challenge in pre- and post-silicon validation [1], [2]. Conventional techniques like ATPG [3] require extensive pattern generation, simulation, and post-processing; as circuits scale to millions of gates, these workflows become increasingly slow, resource-intensive, and susceptible to human error [4], [5]. To overcome these limitations, we recast fault detection as a *graph perturbation problem* [6], [7], injecting realistic stuck-at, bridging, and glitch faults directly into circuit netlists and training a Graph Neural Network (GNN) [8] to identify and localize faults from structural and relational features without requiring simulation loops. This shift enables a scalable, data-driven framework that generalizes across designs. Moreover, while existing interpretability tools such as GNNExplainer [9] or gradient-based saliency methods provide post-hoc insights, our approach derives interpretability intrinsically by leveraging the relational features constructed during training, eliminating the need for external explainers.

The source code and an extended version of this work are available at [10]. Our contributions are:

- 1) A comprehensive evaluation of various state-of-the-art GNN architectures—GCNN [11], GraphSAGE [12], GAT [8], HybridGAT [10], MPNN [13], and APPNP [14]—for multi-class fault detection across the ISCAS85 and EPFL benchmark circuits.
- 2) A model-agnostic interpretability framework that leverages relational features to elucidate the GNN’s fault predictions.

## II. PRELIMINARIES

Digital circuits are susceptible to a variety of fault types during manufacturing, deployment, and operation. Among

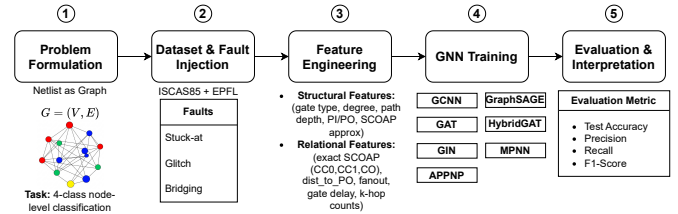


Fig. 1: Fault detection: End-to-end block diagram.

the most common are *stuck-at faults* [15], where a net is permanently stuck at a logic 0 or 1 state, typically due to shorts or opens; *bridging faults* [16], caused by unintended connections between nets that create logical interference; and *transient (glitch) faults* [17], which are short-lived erroneous transitions induced by noise, crosstalk, or power fluctuations.<sup>1</sup>

## III. METHODOLOGY

Fig. 1 outlines the proposed pipeline. Each netlist is represented as a directed graph whose nodes denote logic elements and whose edges represent signal flow, with both structural and relational features extracted for learning. Fault detection is posed as a supervised node-level classification task over four categories—clean, stuck-at, bridging, and glitch. A GNN is trained to map each node to its corresponding class label, using cross-entropy with class weighting to address imbalance, enabling the model to capture both local structure and global circuit context.

We use the ISCAS85 and EPFL benchmarks [20], [21], parsing each Verilog design into its graph representation and generating clean and faulted variants. For every benchmark, we systematically inject stuck-at [15], glitch [17], and bridging faults [16] by modifying selected nets or net pairs according to configurable parameters, producing one clean instance and three corresponding faulted instances per circuit. This process yields a diverse dataset that reflects realistic fault conditions while preserving original circuit structure.

**Feature Engineering.** We construct two complementary feature sets—structural and relational—to provide each node with a rich representation of its logical role and graph context. Structural features are derived directly from the netlist topology and include gate type, in/out/total degree, path depth, primary input/output indicators, fan-out information, and approximate SCOAP (Sandia Controllability/Observability Analysis Program) metrics [22], forming the initial node and edge feature matrices for all GNN models. To enhance this

<sup>1</sup>Other notable categories include *delay faults* [18], arising from path timing violations due to increased gate or interconnect delays, and *open faults* [19], where broken connections result in undefined signal levels.

TABLE I: Representative relational-feature explanations. Similarity scores  $\hat{s}_c(r_i)$  are normalized over classes. Differences are reported as feature  $(r_{i,j}, \mu_{c^*,j}, |r_{i,j} - \mu_{c^*,j}|)$  in last column.

Case	$\hat{s}_c(r_i)$ (Top similarities)	$\mathcal{E}(r_i)$ (Top relational feature differences, examples)
Correct bridging	bridging: 0.995; stuck_at: 0.005; others: 0.0	cc0_in_mean: (10.843, 9.409, 1.434); cc1_out_mean: (10.788, 9.441, 1.347) — elevated controllability consistent with bridging.
Correct clean	clean: 0.556; glitch: 0.309; stuck_at: 0.135	out_k3: (21.141, 0.008, 21.133); in_k1: (0.789, 0.043, 0.746) — large $k$ -hop expansions yet still closest to clean.
Misclassified bridging → glitch	glitch: 0.809; clean: 0.191; others: 0.0	cc_in_mean: (-0.577, -0.731, 0.155); in_k1/2/3 match glitch exactly — relational overlap explains the error.
Correct stuck_at	stuck_at: 0.963; bridging: 0.035; others: $\approx 0$	cc1_out_mean: (10.788, 4.927, 5.861); cc0_in_mean: (-0.100, 4.959, 5.059) — extreme controllability asymmetries typical of stuck_at.

TABLE II: Cohen’s  $d$  and ROC–AUC over clean samples, for a subset of numeric features across fault types. Relational features in red and structural in blue.

Feature	Glitch		Bridging		Stuck-at	
	$d$	AUC	$d$	AUC	$d$	AUC
cc1_exact	-0.1801	0.4688	0.0905	0.6755	0.0700	0.5950
cc0_exact	0.0856	0.5695	0.0888	0.6676	0.0219	0.4938
path_depth	-0.1147	0.4795	-0.0197	0.4999	-0.1141	0.4797
out_degree	-0.0026	0.4970	-0.0011	0.4989	-0.0026	0.4970
fanout	-0.0026	0.4970	-0.0011	0.4989	-0.0026	0.4970
in_degree	-0.0141	0.4971	-0.0057	0.4985	-0.0141	0.4971
total_degree	-0.0052	0.4957	-0.0021	0.4983	-0.0052	0.4956
gate_delay	-0.0151	0.4968	-0.0151	0.4968	-0.0193	0.4968
fanout_norm	-0.0295	0.4967	-0.0332	0.4959	-0.0020	0.4970
dist_to_PO	-0.8263	0.2922	-0.8746	0.2812	-0.8303	0.2939
co_exact	-0.7628	0.2922	-0.8001	0.2812	-0.7655	0.2939

baseline, we incorporate relational features that capture multi-hop dependencies and testability context by computing exact SCOAP scores through forward–backward propagation, updated path-depth and distance-to-output measures, refined fan-out statistics, and nominal gate-delay estimates based on gate types and injected faults. These relational descriptors encode richer structural variation across the circuit, significantly improving fault separability and overall GNN performance.

**Interpretability.** We analyze the relational features directly to understand why a node is classified as faulty or clean. We first construct a relational “prototype” for each class by averaging the relational attributes across all training nodes of that class. Each test node is then compared with these prototypes by measuring how similar its relational profile is to each class signature, producing a normalized similarity score that indicates which class it most closely aligns with. To explain an individual prediction, we examine which relational attributes differ most from the predicted class prototype, identifying the dimensions that contribute most strongly to the node’s relational alignment or deviation. This yields a transparent, feature-level explanation: nodes are classified according to how their relational testability and structural-context measures match class-level patterns, and their most influential attributes are explicitly surfaced without requiring external post-hoc explainers.

#### IV. EXPERIMENTS

**Feature Analysis.** To evaluate how well individual features separate faulty from clean nodes, we compute ROC–AUC [23] scores for key numeric attributes across stuck-at, glitch, and bridging classes. Structural features alone perform poorly, with AUC values clustered near 0.5 and negligible Cohen’s  $d$ , confirming their limited discriminative value. In contrast, relational

TABLE III: Test accuracy [%] using structural features only vs. structural + relational features.

Model	Structural Only	Structural + Relational
GCNN	49.71	99.14
GraphSAGE	48.75	99.83
GAT	49.59	99.93
HybridGAT	33.96	99.30
MPNN	48.92	99.75
APPNP	49.36	99.59

features—particularly exact controllability/observability metrics such as cc1\_exact (AUC 0.6755 for bridging)—consistently exceed structural features by capturing neighborhood interactions, reachability, and multi-hop testability effects. This significant gap validates the inclusion of relational descriptors, which provide richer, more class-separable representations essential for effective multi-class fault detection.

**GNN Evaluation.** We assess the performance of all GNN architectures under the two feature settings to quantify the contribution of our relational descriptors (Table III). Using only structural features, all models perform poorly, with accuracies clustered near 50% and failing to reliably distinguish most fault types—highlighting the inadequacy of purely local, topology-based attributes. When relational features are added, test accuracy increases dramatically across all architectures, with GAT achieving the highest score of 99.93% and GraphSAGE, MPNN, HybridGAT, and GCNN all exceeding 99%. These results clearly demonstrate that relational features provide the essential contextual information required for robust fault classification, enabling GNNs to generalize beyond the limited discriminative power of structural attributes alone.

**Interpretability.** We apply the prototype-based interpretability method from Section III to assess how relational features explain GNN decisions. Table I shows that correct predictions correspond to high similarity with the appropriate class prototype (e.g., bridging at 0.995, stuck-at at 0.963), while misclassifications arise when a node lies closer to another class in relational space (e.g., a bridging node aligned with glitch at 0.809). The accompanying top feature deviations highlight which relational attributes drive these outcomes: bridging and stuck-at nodes exhibit large controllability differences, clean nodes show atypical but non-faulty  $k$ -hop expansions, and misclassified cases share relational signatures with the wrong prototype. Overall, these examples demonstrate that prototype similarity provides a compact global explanation, while the largest relational deviations offer precise feature-level insight.

## REFERENCES

- [1] D. Lin, T. Hong, Y. Li, S. Kumar, F. Fallah, N. Hakim, D. S. Gardner, S. Mitra *et al.*, “Effective post-silicon validation of system-on-chips using quick error detection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1573–1590, 2014.
- [2] P. Mishra and F. Farahmandi, *Post-Silicon Validation and Debug*. Springer, 2019, vol. 301.
- [3] K.-T. Cheng and A. Krstic, “Current directions in automatic test-pattern generation,” *Computer*, vol. 32, no. 11, pp. 58–64, 1999.
- [4] M. Fieback, G. C. Medeiros, L. Wu, H. Aziza, R. Bishnoi, M. Taouil, and S. Hamdioui, “Defects, fault modeling, and test development framework for rrams,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, pp. 1–26, 2022.
- [5] G. Thakur, S. Jain, and H. Sohal, “Current issues and emerging techniques for vlsi testing-a review,” *Measurement: Sensors*, vol. 24, p. 100497, 2022.
- [6] S. Sajadmanesh, A. S. Shamsabadi, A. Bellet, and D. Gatica-Perez, “Gap: Differentially private graph neural networks with aggregation perturbation,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3223–3240.
- [7] B. Wang, J. Jia, X. Cao, and N. Z. Gong, “Certified robustness of graph neural networks against adversarial structural perturbation,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1645–1653.
- [8] C. Sun, C. Li, X. Lin, T. Zheng, F. Meng, X. Rui, and Z. Wang, “Attention-based graph neural networks: a survey,” *Artificial intelligence review*, vol. 56, no. Suppl 2, pp. 2263–2310, 2023.
- [9] A. Longa, S. Azzolin, G. Santin, G. Cencetti, P. Liò, B. Lepri, and A. Passerini, “Explaining the explainers in graph neural networks: a comparative study,” *ACM Computing Surveys*, vol. 57, no. 5, pp. 1–37, 2025.
- [10] R. Karn, “fault\_netlist\_gnns: Interpretable graph neural networks for fault detection in circuit netlists,” Sep. 2025. [Online]. Available: [https://github.com/rkarn/fault\\_netlist\\_gnns](https://github.com/rkarn/fault_netlist_gnns)
- [11] 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.
- [12] P. Lu, C. Jing, and X. Zhu, “Graphsage-based multi-path reliable routing algorithm for wireless mesh networks,” *Processes*, vol. 11, no. 4, p. 1255, 2023.
- [13] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Message passing neural networks,” in *Machine learning meets quantum physics*. Springer, 2020, pp. 199–214.
- [14] J. Wang, X. Zhao, P. Jin, C. Yang, B. Li, and H. Zhang, “Storyline generation from news articles based on approximate personalized propagation of neural predictions,” in *International Conference on Database Systems for Advanced Applications*. Springer, 2023, pp. 37–52.
- [15] Z. Navabi, “Digital system test and testable design,” *Springer doi: https://doi.org/10.1007/978-1-4419-7548-5*, 2011.
- [16] M. L. Bushnell and V. D. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer, 2002.
- [17] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, “Modeling the effect of technology trends on the soft error rate of combinational logic,” in *Proceedings International Conference on Dependable Systems and Networks*. IEEE, 2002, pp. 389–398.
- [18] P. Nuzzo and A. Sangiovanni-Vincentelli, “Robustness in analog systems: Design techniques, methodologies and tools,” in *2011 6th IEEE International Symposium on Industrial and Embedded Systems*. IEEE, 2011, pp. 194–203.
- [19] D. Addala, P. Teja, and S. Saxena, “63 fault simulation algorithm for transistor single stuck short faults,” *Intelligent Circuits and Systems*, p. 404, 2021.
- [20] 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.
- [21] L. Amarú, P.-E. Gaillardon, and G. De Micheli, “The eplf combinational benchmark suite,” in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.
- [22] C. Ravikumar and H. Joshi, “Scoop-based testability analysis from hierarchical netlists,” *VLSI Design*, vol. 7, no. 2, pp. 131–141, 1998.
- [23] G. Naidu, T. Zuva, and E. M. Sibanda, “A review of evaluation metrics in machine learning algorithms,” in *Computer science on-line conference*. Springer, 2023, pp. 15–25.