GRAND: A Graph Neural Network Framework for Improved Diagnosis

Hongfei Wang, *Member, IEEE,* Ziqiang Zhang, Hongcan Xiong, Dongmian Zou, *Member, IEEE,* Yu Chen, and Hai Jin, *Fellow, IEEE,*

***Abstract*—The pursuit of accurate diagnosis with good resolution is driven by yield learning during both early bring-up and production excursions. Unfortunately, fault callouts from diagnosis tools often render poor resolution that hinders the follow-up failure analysis. In this work, we propose a method that significantly improves diagnosis. By modeling the logic circuits under test as graphs, the method employs graph neural networks to determine each fault candidate from the diagnosis callout as either the true fault or the false candidate. This novel deep learning method mainly makes full use of circuitry topology with underlying structural information, which was largely ignored or insufficiently analyzed by previous approaches. Other contributions include the finding of the dependency among candidates that can be leveraged to improve diagnoses. Extensive experiments on various benchmark circuits including industrial designs demonstrate that the diagnostic resolution can be improved by 4.51× compared with a fault simulator-based diagnosis tool, and increased by 5.98× compared with one state-of-the-art commercial diagnosis tool. Moreover, experiments also reveal that our method can successfully identify 62.96% of true candidates that were originally not given high priority by the commercial tool (non top-scoring candidates). This means our method can rectify the existing commercial diagnosis for better characterizing failure Pareto, in addition to boost diagnostic resolution.**

***Index Terms*—diagnosis, resolution, candidate dependency, graph neural networks, machine learning, transfer learning**

I. Introduction

Chipmakers are always concerned about yield, especially when faced with surging demand during this worldwide shortage of semiconductors. Yield refers to the fraction of chips passing the tests out of the entire batch of fabricated chips. Failing chips from any test stage decrease the yield. Defect identification with characterization of failure mechanisms is therefore highly demanded towards achieving profitable yield levels, during both early ramping and volume production. For this aim, design houses, along with semiconductor foundries, usually perform logic diagnosis on the failing chips prior to the more expensive and time-consuming physical failure analysis (PFA). As a software-based analytics, diagnosis uses the applied test patterns, the circuitry description, and the fail data. A typical commercial diagnosis tool returns one or more defects inside failing ICs, with possible failure behaviors (described by fault models) and netlist locations. Effective diagnosis increases the chance of success for performing PFA on the selected failing ICs, especially those fabricated through modern manufacturing process nodes.

An ideal diagnosis has two important facets, i.e., remarkable accuracy and good resolution. Accuracy means the actual faulty sites are indeed included in the callouts. Good resolution, a key metric for evaluating diagnosis quality, means the number of reported defect candidates is small. A perfect resolution is one correct candidate per defect. This best-case scenario is termed as a 100% precise call or a home run [1].

The definitions of resolution vary among literature. Generally, they can be put into two categories. In this work, resolution is defined as the number of candidates in the fault callouts reported for each defect, which is consistent with [2–5]. The range is ≥ 1, the smaller the better. Conversely, resolution can be defined as the inverse form, which is the ratio of each defect to the number of reported candidates [1][6]. The range is (0, 1], meaning the larger the better. For both definitions, 1 is the perfect resolution.

Yield learning requires good diagnostic resolution for post-silicon simulation and debug. First, good resolution helps guide a successful PFA. Due to the limited and expensive nature of PFA, only a handful of failing chips per week from a production lot can eventually go through PFA. Besides, PFA is destructive to a chip, which suggests that there is no second chance if the targeted defect is not verified or caught by PFA. Prior to allocating PFA resources appropriately, failing chips should be adequately diagnosed to reveal possible silicon sites regarding manufacturing defects. Second, good diagnostic resolution is beneficial for the follow-up failure analysis, especially during volume diagnosis [7–9] to uncover the root causes for plotting a failure Pareto. Third, diagnosis can be leveraged to measure the effectiveness of test generation [10][11], preventing potential failures (functional bugs or manufacturing defects) from escaping into the released integrated silicon system.

Poor diagnostic resolution may happen due to various

Manuscript received May 16, 2005; revised May 16, 2005. This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62172173. Corresponding authors: Hongfei Wang and Dongmian Zou.

Hongfei Wang, Hongcan Xiong, and Yu Chen are with School of Cyber Science and Engineering, Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Huazhong University of Science and Technology, Wuhan, 430074, China (e-mail: hongfei@hust.edu.cn; xhc333@hust.edu.cn; chenyu998@hust.edu.cn)

Ziqiang Zhang and Hai Jin are with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China (e-mail: zhangziqiang@hust.edu.cn; hjin@hust.edu.cn)

Dongmian Zou is with the Zu Chongzhi Center for Mathematics and Computational Sciences, Data Science Research Center, Division of Natural and Applied Sciences, Duke Kunshan University, Kunshan, Jiangsu, 215316, China (e-mail: dongmian.zou@duke.edu)

reasons, such as aliasing risk [9][12]. The effectiveness of production test is also among the reasons. Because chip testing is expensive, the amount of applied tests and test-response data-volume are usually limited to save cost, although they should be sufficient to allow a quality diagnosis [13]. In addition, using EDA tools to perform physical design as well as post-silicon simulation of modern chips are becoming increasingly challenging nowadays. Expertise and tool costs can become the barrier [14][15]. On the other hand, the tools and their fault models may be immature due to the fast development of semiconductor technology.

Previous methods improve diagnostic resolution by optimizing the quality and ordering of test patterns [11][16][17], the collected amount of test-response data [18] and failure logs [19]. Commercial tools have been developed to gather more relevant details in addition to logic netlist for better resolution, such as cell-aware information and layouts. More recently, machine learning (ML)-based approaches have been proposed for **improving diagnostic resolution (IDR)** [1–5, 7, 8, 13, 18]. One big advantage of these ML approaches is that ML is good at fast modeling and building statistical models to reason correlations among observed symptoms, physical defects, fault models, etc. They can be hard for reasoning from first principles.

In this work, we propose a framework called **GRAND** (**GRA**ph **N**eural networks for **D**iagnosis) that significantly boosts both diagnostic resolution and accuracy. GRAND mainly makes full use of circuitry topology with underlying structural information, which was either ignored or insufficiently analyzed by previous approaches. Graph neural networks (GNNs) are employed and properly configured to determine each fault candidate from the diagnosis callout as either the true fault or the false candidates. False candidates are then discarded to improve diagnostic resolution. Extensive experiments demonstrate that the diagnostic resolution can be improved by 4.51× compared with an in-house diagnosis tool, and increased by 5.98× compared with one state-of-the-art commercial diagnosis tool. Moreover, experiments also reveal that GRAND can successfully identify a significant portion of candidates that were originally not given high priority by the commercial tool (contained in the fault callouts but marked with low match scores).

GRAND is the first GNN-based methodology for IDR. GNN combines the advantages of graph analytics and deep learning. Graph contains geometric and topological information. With circuit modeled as graphs, deep learning can have complete and direct access to the nets inside chips, allowing a closer examination to each candidate. Previously, the granularity of deep learning in test and diagnosis is at die or device level [20] [21]. Deep learning effectively automates most part of feature extraction to learn more rich features (called embeddings), which are more expressive towards candidate classification. Other main technical contributions are:

- The finding that there exists dependency among candidates. It is quantified to derive initial node features that is later used by GNNs to identify true candidates.
- A comprehensive exploration of GNN architectures. We analyze and compare major GNN algorithms, configured with directed/ undirected/ bidirected graphs, and with different layer numbers to find out the best suitable one for improving diagnosis.
- In addition to boost diagnostic resolution, GNN modeling in GRAND creates more opportunities for improved diagnosis. For example, GNN-enabled transfer learning is explored to demonstrate that failure information from other chips or defects can be shared for better diagnoses. Moreover, GRAND can rectify the imprecise candidate scoring rendered by existing commercial diagnosis.

The rest of this paper is organized as follows. In Section II, preliminary material about graph neural networks is introduced. Section III describes how to use GRAND to analyze the candidates produced by diagnostic tools and determine their true/ false property. Section IV describes the ability of GRAND to handle some real-world cases. Section V presents the experiment results. Section VI concludes the paper.

II. Background

For a failing chip, software-based logic diagnosis produces a list of net locations that are assumed to be suspect of defected site(s). These nets, oftentimes listed with additional description such as fault models and match scores, are called *candidates*. A *match score* is a ratio based fraction (≤ 1), calculated as a function of the following counts: test pass simulation fail (TPSF), test fail simulation pass (TFSP), and test fail simulation fail (TFSF) [4, 5, 9, 22]. Intuitively, the score reflects how well fault model-based simulation results match the actual tester responses.

The problem of IDR is to reduce the number of top-scoring candidates produced by logic diagnosis, while retaining the ones that do give clues to actual defects. This task can be modeled as a binary classification problem in ML. The object is to classify each candidate as either true or false. The true label means the candidate net is indeed where the defect locates; a false label means the corresponding candidate points to somewhere else. Eliminating false candidates from the fault callout improves resolution, and saves efforts from failure analysis in the downstream flow (such as PFA).

To shorten the time-to-market during the early bring-up stage, volume diagnosis is expected to identify the root causes that explain the majority of the yield losses. Post-silicon debug is sometimes more concerned with particular types of defects than others. Previous examples include cell defects [23] and bridges [24], though they are design and process dependent and should be decided on a case by case basis. In addition, resolving systematic issues from manufacturing or design also accelerate yield ramping.

Existing methods extract neighborhood information of a candidate for IDR [4, 6, 25, 26]. The neighbors of a candidate refer to the nets that are close or relevant to this candidate. Specifically, nets that are in proximity to the candidate are termed as *physical* neighbors. In practice, they are the nets within the radius of a predetermined distance (such as 45 nm) centered at the candidate site from the same metal or silicon layer. Comparatively, *logical* neighbors are the side inputs and drivers of a candidates. Neighborhood information can

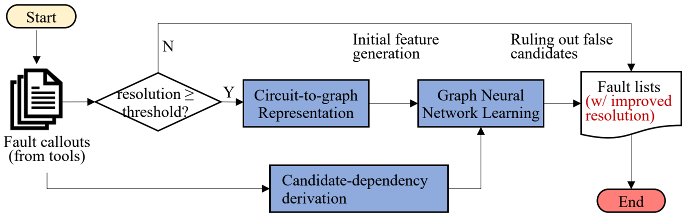

Start
N
Initial feature generation
Ruling out false candidates
resolution ≥ threshold?
Y
Circuit-to-graph Representation
Graph Neural Network Learning
Fault lists (w/ improved resolution)
Fault callouts (from tools)
Candidate-dependency derivation
End

Fig. 1. Work flow diagram. GRAND starts with the fault callouts produced by a diagnosis tool. If the resultant resolution is above a certain threshold, GRAND is invoked. It transforms circuit netlists to graph representations. Next, candidate dependency and initial node features are derived. Customized GNN is then trained to compute the true/ false labels of candidates for IDR.

be leveraged to identify false candidates. A defect often has an impact on both the logical netlist and the physical layout. In most cases such impact does not limit to one single net (i.e., the true candidate). Neighbors of a true candidate may exhibit particular behaviors if stimulated by certain input test patterns. Therefore, neighborhood can help verify the existence of a defect at a candidate site.

Graphs are natural representations of many real-world entities. In EDA field, previous works have attempted to model objects as graphs in order to find design and test solutions, such as hypergraph partitioning in VLSI design [27][28], Boolean circuit manipulation [29], hardware reverse engineering [30], and test-point insertion [31].

Graph neural networks (GNNs) have emerged as a new and hot research topic in ML for the past couple of years. Previously, most data utilized in ML has to be in the Euclidean domain [32]. Graphs are typical non-Euclidean[33], calling for new and adapted deep learning paradigms. GNN algorithms adopt techniques from both deep learning and graph theory, and quickly attract much attention. They have been applied to optimize design and test solutions from frontend to backend flow [34], including tier partitioning [35], timing model selection [36], testability analysis [37][38], placement and routing[39, 40], and power estimation [41].

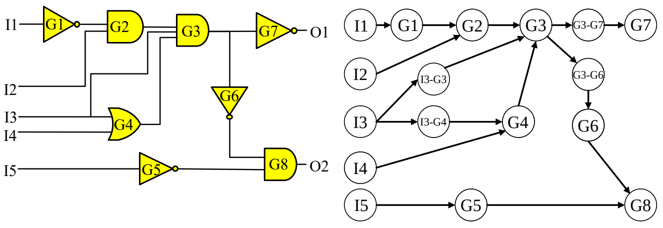
To the best of our knowledge, GRAND is the first GNN-based EDA technique proposed for diagnosis. GRAND is entirely different from all the existing works using GNNs, including the adjacent research area on test [34][37][38]. We name a few among the many differences. For example, unlike the method in [37] uses directed graphs to classify a netlist node as difficult-to-observe or easy-to-observe, GRAND uses undirected graphs to determine whether a candidate is true or false. In addition, GRAND is powered up by advanced ML strategy (transfer learning) for more potential tasks. The method in [38] builds pre-trained models for downstream tasks. Such interest of pre-trained models is similar from domains like natural language processing. However, pre-training does not fit into diagnosis. Controllability and observability are rather static for testability analysis. A fault (defect) may change the netlist (layout) of a circuit drastically into an entirely different one with hard-to-predict output behavior, making pre-training effort vanish into thin air. Section III has more details on GRAND's methodology.

III. GNN Modeling and Learning

This section describes GRAND. As depicted in Fig. 1, GRAND functions as an adds-on toolkit, rather than being orthogonal to the established approaches based on diagnosis tools. GRAND improves diagnoses by analyzing the input fault callouts produced by a diagnosis tool, hence can be easily incorporated into industrial post-silicon debug flow.

A. Circuits to Graphs

A circuit can be represented as a directed or an undirected graph by GRAND, as illustrated by Fig. 2. Nets correspond to the edges in a graph. Primary inputs (PIs) and logic gates are represented by nodes. Aside from this, for branch structures, GRAND creates an extra virtual node on each branching net to distinguish between them and the stem. Fig. 2 (a) shows a circuit and Fig. 2 (b) shows how to model the circuit into a directed graph. If the arrows are ignored, the graph becomes an undirected one. Gate $G3$ has two branches connected to $G6$ and $G7$. Hence two virtual node are added as $G3 \rightarrow G6$ and $G3 \rightarrow G7$, respectively.



(a) Circuit (b) Graph

Fig. 2. Example of representing a logic circuit as a graph.

Unlike [38] and [37] where circuits are put into directed graphs only, GRAND also considers undirected graphs. This is intuitively because for test-point insertion or automatic test patter generation (ATPG) problems, signals are propagated forwards from primary inputs towards outputs. For diagnosis, however, neighborhood nets are considered meaning information from backward direction is also required. Hence, both directed and undirected graphs are examined in GRAND.

In effect, creating virtual nodes for branch structures as the above is the same as a diagnostic tool does [22]. Based on the produced diagnosis reports, it treats branches separately as if they were logic gates by themselves. Some fault simulators and ATPG tools also have similar treatments. Except for the branch structures, everything else in the logic circuitry is remained unaltered in GRAND. The work in [38] modifies a reconvergence structure in a circuit by adding a skip connection. GRAND does not adopt such design because it is apt for testability analytics but not for diagnosis[1].

B. Node Features and Candidate Dependency

After converting logic circuits into graphs, GRAND constructs the initial features based on circuit structure, fault simulation results, and diagnosis reports for each node in a

[1] Otherwise an added skip connection itself can become a candidate in the fault callout.

TABLE I
INITIAL NODE FEATURES SPECIFICATION

No.	Feature	Description
1	depth_input	Maximum No. of gates passed by from the inputs of the circuit to this gate
2	depth_output	Minimum No. of gates passed by from this gate to the outputs of the circuit
3	candidate_flag	Indication of whether the node is a candidate or not
4	consistent_num	Sum of the number of consistent states of a candidate in the diagnosis report
5	inconsistent_num	Sum of the number of inconsistent states of a candidate in the diagnosis report
6	candidates_num	No. of candidates included in a diagnosis report
7	dependency_level	No. of candidates in the longest dependency chain starting from this candidate
8	dependency_num	No. of dependency chains starting from this candidate

TABLE II
EXAMPLE OF INITIAL NODE-FEATURE GENERATION FOR CANDIDATES

Candidate locations	Logic neighbors	Failing states	#Failings	Passing states	#Passings	Initial node features
G1	I1, I2	0 1	2	0 1	3	1 3 1 6 5 4 4 2
		0 0	1	1 0	5	
G2	I2, I3→G3, G1, G4	0 0 1 1	5	1 0 0 1	6	2 2 1 11 0 4 3 2
G3	I3→G3, G2, G4	0 0 0	10	1 1 1	1	3 1 1 11 0 4 0 0
G4	I3→G3, I3→G4, I4, G2	0 1 0	7	1 0 1	4	1 2 1 11 0 4 3 2

graph. Altogether eight features are designed to form an eight-dimensional vector. Table I describes the features. Table II gives an example of the initial node features calculated based on the circuit in Fig. 2. In this example, the initial fault diagnosis reports four suspects $\{G1, G2, G3, G4\}$ as the candidates, produced by a set of 11 tests.

depth_input and depth_output describe the logic levels of a gate, represented by a node according to Section III-A. The depth_input of a gate is the same as *level* or *level number* defined in [12], which is the maximum number of gates of its multiple inputs, starting from the circuit PIs to this gate. For example, the depth_input of $G1$ is 1, whereas the depth_input of $G2$ is 2 (the longer one of the two paths starting from $I1$ and $I2$). Similarly, depth_output of a gate is the minimum number of gates from its output the primary outputs of the circuit. Because virtual nodes are created for branches, their depth_input and depth_output are the same as their stem.

candidate_flag provides an indication of whether the node is a candidate or not. If yes, this feature bit is set to 1. Otherwise, it is 0. candidates_num is the total number of candidates included in a diagnosis report.

consistent_num and inconsistent_num specify the stats for logical neighborhood states [4, 5, 25, 26]. The logical neighborhood of a candidate are its drivers and side inputs. The candidates and their corresponding logical neighbors are listed in the first two columns in Table II. For a candidate, a neighborhood state is a set of logic values on its neighborhood gates, given an input test pattern. If a test passes (fails), the state on the candidate's logical neighborhood is a passing (failing) state. If a state appears in both the passing states and failing states, the state is inconsistent. On the contrary, if a state only appears in either failing states or passing states, the state is consistent. consistent_num and inconsistent_num thus record the numbers of consistent and inconsistent states occurring at each candidate, respectively.

Previous works show that, in most cases, the same neighborhood state should not exist for both passing and failing states [4, 6, 25, 26]. While consistent neighborhood states do not guarantee a candidate to be true, inconsistent states provide a strong indication of the existence of false candidate. Hence GRAND leverages them for IDR.

GRAND establishes the concept of *candidate dependency* for the first time. If a candidate x exists in the logical neighborhood of another candidate y, there is a dependency chain defined as $x \rightarrow y$. GRAND also quantifies such dependency so that it can be used for GNN-based IDR. The dependency_level of a candidate is the number of candidates in the longest dependency chain starting from it. The dependency_num of a candidate is the number of dependency chains. For example, there are two dependency chains that start from candidate $G2$, which are $G2 \rightarrow G4 \rightarrow G3$ and $G2 \rightarrow G3$. The former is the longer one containing three candidates, hence dependency_level = 3, dependency_num = 2.

For nodes that are non-candidates, depth_input and depth_output remain the same as the first two rules in Table I. candidate_flag is set to 0. Although the rest five features are originally defined for candidate nodes, they still need to be assigned values so as to allow node-information propagation and aggregation over a graph. Here they are generated via a random sampling from an uniform distribution with range 0 to 0.1 ($\sim U(0, 0.1)$). Other configured distribution that generates small and random numbers, such a random Gaussian $G(0.05, 0, 1)$, will also do.

In practice, missing values or incomplete logs may happen due to various reasons, from inadequate file manipulation to fail data collection exceeding ATE buffer size. To treat incomplete data or fault callouts that are fragmented, one may resort to general-purpose feature engineering methods, such as label imputation and feature-selection-based techniques described in [42], to reconstruct the feature matrix for handling the missing-value problems.

C. Candidate Classification via Graph Neural Networks

Two most commonly used GNN algorithms are presented in this section relating to candidate classification for IDR. Some frequently used notations are provided in Table III.

TABLE III
FREQUENTLY USED NOTATIONS RELATED TO GNNS

$\mathbf{x}_i^{(l)}$	The l-th layer embedding of node i.
$X^{(l)}$	Layer-l embeddings of N nodes in a graph, $X^{(l)} = \left\{\mathbf{x}_1^{(l)}, \mathbf{x}_2^{(l)}, \cdots, \mathbf{x}_N^{(l)}\right\}$.
A	The adjacency matrix of the graph.
D	A diagonal matrix $D = D_{ii} = \sum_j A_{ij}$.
I_N	An $N \times N$ identity matrix.
$\mathbf{W}$	A weight matrix to be learned through GNN training.
$\mathcal{N}(i)$	A set consists of neighbors of node i.
L	A Laplacian matrix, $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$.
Λ	A diagonal matrix whose diagonal elements are eigenvalues of Laplacian matrix L.

(1) Graph Convolutional Network (GCN)

Graph Convolutional Network (GCN) [43] makes use of convolution operation. The idea is motivated by convolutional neural networks, which excels at building multiple network layers to extract useful information from data. The convolution operation on graph is defined as:

$$\mathcal{F}_\theta * \mathbf{x} = U \mathcal{F}_\theta U^\top \mathbf{x} \tag{1}$$

where $\mathcal{F}_\theta$ is a filter function with θ as the parameter, and $*$ denotes the convolution operation. $\mathbf{x}$ represents the feature values of a node in the graph. U is the eigendecomposition matrix of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^\top$, U^T is the transpose matrix of U.

Performing Laplacian matrix decomposition on large graphs can be expensive. To ease the computation burden, the filter function $\mathcal{F}_\theta$ can be approximated by Chebyshev polynomial $T_k(\mathbf{x})$ [44]. The polynomial approximation allows Eq. (1) to be reformulated as $\mathcal{F}_{\theta'} * \mathbf{x} \approx \theta'_0 T_0(\tilde{L})\mathbf{x} + \theta'_1 T_1(\tilde{L})\mathbf{x}$, where $T_0(\mathbf{x}) = 1$, $T_1(\mathbf{x}) = \mathbf{x}$, $\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$. $\lambda_{\max}$ means the maximum in all eigenvalues of L. Further reduction of computation can be done by setting $\lambda_{\max} = 2$ and $\theta'_0 = -\theta'_1 = \theta$, which leads to a more concise formulation as

$$\mathcal{F}_\theta * \mathbf{x} \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) \mathbf{x} \tag{2}$$

By replacing $(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})$ in Eq. (2) with $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ ($\tilde{A} = A + I_N$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$), the output of a graph with N nodes of dimensionality d can be collectively represented in an iterative form,

$$X^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X^{(l)} \mathbf{W}^{(l)} \right) \tag{3}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{d \times S}$ is a matrix of filter parameters θ's; l denotes the l^{th} layer; $\sigma(\cdot)$ is a ReLU activation function. The features of a node after layer-by-layer convolution is a vector of length S, called a *node embedding*. The output $X^{(l+1)}$ consists of a set of vectors in embedding space of $N \times S$, which are the deep features learned by the network model through training.

In GRAND, GCN is configured with two convolutional layers to handle undirected graphs, by setting $l = \{0, 1\}$. $l = 0$ means the initial feature vector of node i. Nodes within a two-hop logical neighborhood are considered. In this way, the neighborhood scope is extended in a controlled way. We set $d = 8$ for the initial eight-dimension feature according to Section III-B, and $S = 32$ for node embedding length. The obtained node embedding $X^{(l+1)}$ are fed into a fully connected layer for binary classification task to determine the true/ false label of a candidate node.

For a binary classification problem, the loss function of GCN (and GraphSAGE) is the cross-entropy given as,

$$Loss = -\sum_{i=1}^{N} \{ y_i \ln(P_i) + (1 - y_i) \ln(1 - P_i) \} \tag{4}$$

where N is the number of nodes in graph, y_i is the true/ false label of node i, and P_i is the probability GCN predicts node i to be 1. P_i is calculated as,

$$P_i = \frac{\exp(\mathbf{W}^{fc} \mathbf{x}_i^{last} + b_i)_1}{\exp(\mathbf{W}^{fc} \mathbf{x}_i^{last} + b_i)_0 + \exp(\mathbf{W}^{fc} \mathbf{x}_i^{last} + b_i)_1} \tag{5}$$

$\mathbf{W}^{fc}$ are the weights for the fully connected layer with a size of 2×32. $\mathbf{x}_i^{last}$ is the last layer embedding of node i of size 32×1. b_i is a bias of size 2×1. The subscript 1/0 denotes the output from the fully connected layer is 1 or 0, respectively.

(2) Graph SAmple and aggreGatE (GraphSAGE)

The second GNN algorithm on our menu is Graph SAmple and aggreGatE (GraphSAGE) [45]. Unlike GCN, which requires the computation of a huge Laplacian matrix L derived from the entire graph, GraphSAGE examines the local topology with node attributes during training. GraphSAGE is thereby more efficient for handling large graphs. Besides, it is able to analyze both directed and undirected graphs.

GraphSAGE considers the dynamic process of node-information propagation and aggregation on a graph. Given a circuitry represented by a graph following the modeling in Section III-A, the feature information of a node i can be propagated through its edge connections so that other adjacent nodes will not only possess their own information but also receive the information from node i. As the propagation progresses, a wider range of nodes will receive the information from node i. During one iteration of propagation (called one hop in GNN), a node aggregates the information, which is either the initial features (d dimension) or the aggregated values from the previous iteration(s). Such process is repeated until the accumulated iteration numbers exceeds a pre-determined limit or the node feature values are convergent. The final result is the node embeddings that can be fed into a fully connected layer for classification.

The output of GraphSAGE can be defined as

$$\mathbf{x}_i^{(l+1)} = \mathbf{W}\mathbf{x}_i^{(l)} + \mathbf{W}_N \cdot \mathrm{mean}_{\mathbf{x}_j \in \mathcal{N}(i)} \, \mathbf{x}_j^{(l)} \tag{6}$$

where $\mathbf{x}_i$ is the feature vector of the node i; $\mathbf{x}_j$ are the neighbors of node i; $\mathbf{W}$ and $\mathbf{W}_N$ are two weight matrices. `mean` is an aggregation operator, which sums up the values of nodes in the neighborhood $\mathcal{N}(i)$ of node i, and computes an averaged value. Similar to GCN, we found a two-layer GraphSAGE model ($l = \{0, 1\}$) best suitable for the candidate classification task here.

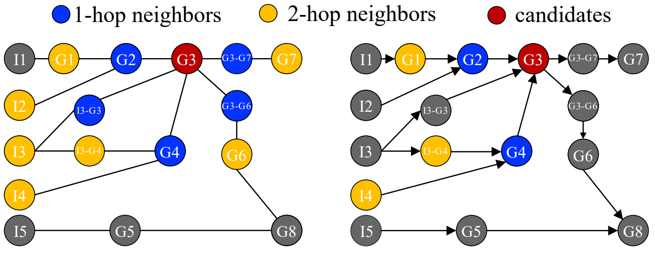
D. Design Exploration of GNNs for Diagnosis

GNNs enable deep learning to play a full part in studying geometric data that describes non-Euclidean domains, such as graphs [32]. The success of GNNs in many applications has two significant implications towards IDR tasks. (1) We can train powerful deep learning models for prediction tasks using citcuit-to-graph data. (2) Useful features containing topological information can be extracted in a more automatic and effective fashion.

Previously, designing features manually based on statistics or heuristics takes a significant portion of the workflow in ML-based diagnostic enhancement [2–5, 18]. This is non-ideal since in an efficient EDA flow, "no human in the loop”, "self-driving tools and flows” , and "24-hour turnaround time” are desirable [15]. Aside from human involvement, manually created features encode much belief from the designers or engineers. As a result, they can be biased or incomplete, especially when most of these features are countings [4][18].

GNN algorithms start with a few initial features as described in Section III-B, and subsequently obtain more features (i.e., the node embeddings) based on topological structures and deep learning operation. The feature extraction part demands much less manual effort and is comparatively more objective.

Fig. 3 illustrates two GNN algorithms, using the example from Fig. 2. Suppose node $G3$ is a candidate, which is treated as a target node marked in red. One-hop and two-hope nodes from the target node are colored in blue and yellow, respectively. Untouched nodes are denoted by gray. GraphSAGE in Fig. 3 (b) shows its sampling functionality by choosing two out of the five neighbors. GCN is originally developed for undirected graphs. On the other hand, GraphSAGE can work on both directed and undirected graphs. For directed graphs, the collection of neighbors only contains the predecessors of the target node $G3$ while its fan-outs are excluded.



(a) GCN (b) GraphSAGE

Fig. 3. GCN uses undirected graphs. GraphSAGE can work on both directed and undirected graphs. Hops extend the neighborhood scope of a candidate, from logical neighbors to including topological neighbors.

In addition to both undirected and directed versions of GraphSAGE (where undirected GraphSAGE is better from experiments), we also examine the bidirectional GraphSAGE. The experiments indicate that undirected graphs are the most efficient models that consider messages from both directions. Also, they focus on connectivity and topology information instead of directional information. GCN and GraphSAGE are explored because they represent two typical examples of GNNs. GCN emphasizes local topology since each node receives messages from all its neighbors. On the other hand, GraphSAGE takes a sampling approach and therefore the features are more global.

Topological neighbors in GNN algorithms are different from logic neighbors, in terms of node adjacency and propagation scope. Based on Table II, logic neighbors of $G3$ include $G2$, $G4$, and $I3$ (the $I3 - G3$ branch). GCN visits all of $G3$’s adjacent nodes. These five nodes are one-hop away colored by blue in Fig. 3 (a). GraphSAGE samples two of them as illustrated in Fig. 3 (b). Besides, nodes are two-hops away are also considered. Their feature values affect the target node through message passing, and are affected by the target node as well. GNN extends the neighborhood scope of a candidate by hopping to encompass all the relevant nets that may help identify whether the candidate is true or false.

GRAND choose to use two layers in the GNN architectures. A larger search hop does not help in the diagnosis problem. Large hops jump out of the scope of neighborhood, but only nearby neighborhood information of a suspect can really help verify the existence of a defect at a candidate site or not. Please refer to [4–6, 25, 26] for the explanations and reasoning regarding neighborhood sites and scopes in failure diagnosis. Such fact is also evidenced by experiments in Table V. One more advantage of our two-layer GNN architectures is related to the over-smoothing issue. Smoothing is the nature of GNNs as long as they follow the message-passing regime. In particular, even in directed or bidirected graphs, the propagation of messages will dilute the effective information contained in each node feature [46][47]. In any case, our GNNs do not suffer from over-smoothing since two-layer GNN architectures are employed rather than multi-layers.

IV. Failure Information Sharing

Up to this point, all previous discussion is based on the assumption that there is a sufficient amount of labeled data samples for training GNN models. However, labeled samples are not always available, especially in test and diagnosis. Verifying the true/ false label of each candidate can be expensive or simply unfeasible. Even without considering the resources and development time, PFA may result in a failure, providing little information to label candidates. Hence, it is desirable to empower GRAND with the ability to draw inferences from other definite failure knowledge, where candidates and their true/ false labels are available and correct.

For this aim, transfer learning is introduced to expand the capability of GRAND. In a generic setting of transfer learning, data samples come from two domains. The one has original labels for samples is the source domain. The one lacking of labels is the target domain. The purpose is to train a ML model that predicts the labels for the target-domain samples. The task is difficult because typically supervised learning requires the presence of labels for training. Directly obtaining labels for the target domain is costly or impossible. Therefore, the

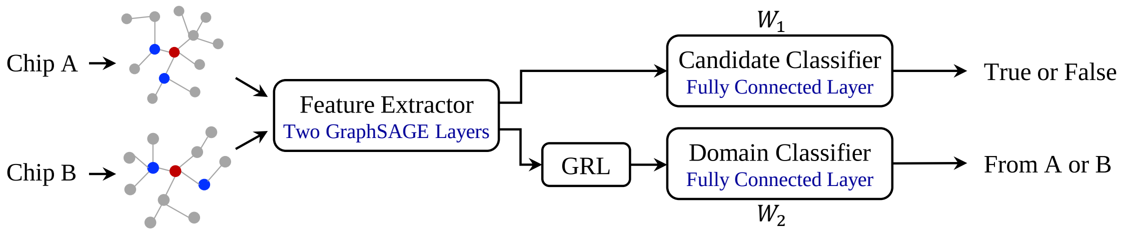


Fig. 4. Work flow of training G-DANN for failure knowledge sharing. Data inputs from the source domain include the adjacency matrices of the graphs, the initial feature vectors, the labels of the source domain (true/ false of all candidates), and the created labels denoting that they are from the source domain. Data inputs from the target domain have the same categories except that they possess no candidate labels. Two classifiers are trained. The training objective for Candidate Classifier is to accurately determine the true or false labels of candidates. Domain Classifier is trained to find similarities in Chip A and B so as not to distinguish between them.

motivation of transfer learning is to *share* the labeled samples from the source domain. Knowledge from the source domain is shared to allow training in the target domain. One basic assumption for transfer learning is that the two domains share similar distributions. Learning knowledge in one domain with sufficient training data (sample-label pairs) is useful towards training models for the other domain where sample labels are insufficient or unavailable.

Two cases are considered in this work that require transfer learning. (1) The candidate labels for one chip (Chip A) are available, but not for the other (Chip B). (2) For the same chip, candidate labels for a particular defect type (such as bridge) are available, but not for the other defects. For (1), Chip A and B do not have to be different designs. They may refer to different lots of dies belonging to one chip design as well.

GRAND employs Domain-Adversarial Neural Network (DANN)[48] as the algorithmic framework to tackle the above problems. DANN is built upon convolutional neural networks (CNNs), first proposed for transfer learning using data in Euclidean space. We leverage the framework and adapt it into non-Euclidean space to handle graph data, by substituting CNNs for GNNs. In the following of this paper, we use G-DANN to refer to our GNN adapted DANN model that works on graph data using GraphSAGE-based architecture, not the original one built upon CNNs [48].

Fig. 4 depicts the high-level flow of G-DANN during the training stage. G-DANN consists of four parts: the Feature Extractor for constructing embeddings, the Candidate Classifier for predicting required labels, the Domain Classifier for determining the data sources, and the gradient reversal layer (GRL). Among the four modules, the function formed by Feature Extractor and Candidate Classifier is essentially the same as the GraphSAGE in Section III-C (2), determining the true or false labels of candidates.

G-DANN assigns additional domain labels according to sample sources. For the example in Fig. 4, two class labels (0's and 1's) are assigned to Chip A and B, respectively, to distinguish their domains (source or target). The Domain Classifier module learns to classify these candidate samples based on their appended domain labels, as either A or B.

The Feature Extractor starts with the initial node features to obtain vectors in the high-dimensional feature space, which are the extracted embeddings. The Candidate Classifier and Domain Classifier are both configured with fully connected layers, predicting the node labels and data domains, respectively. The Candidate Classifier takes as input the embeddings, and outputs a set of predicted labels indicating whether a node (meaning a candidate) is a true fault or not. The Domain Classifier takes the same inputs, and outputs the labels that indicate which domain the corresponding inputs of the embeddings are derived from.

The optimization function integrating the training goals of the two classifiers is shown by Equation (7):

$$arg\min_{\mathbf{W}} F\left(\mathbf{E}_s\mathbf{W}_1, \mathbf{Y}\right) - \lambda F\left(\mathbf{E}_s\mathbf{W}_2, \mathbf{0}\right) - \lambda F\left(\mathbf{E}_t\mathbf{W}_2, \mathbf{1}\right) \tag{7}$$

where $\mathbf{E}_s$ and $\mathbf{E}_t$ are embeddings from source domain and target domain, respectively. $\mathbf{Y}$ is a vector of true/ false labels for candidates in source domain. $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices (both of size 32×2) for candidate classifier and domain classifier, respectively. λ is a parameter between 0 and 1. $F(\cdot)$ denotes the loss function. We choose cross-entropy loss here, but any reasonable cost function (such as the mean square error) is eligible as well. The first F measures the loss of the candidate label classification. The second and third F computes the loss from source and target domain-label classification, respectively.

The gradient reversal layer (GRL) functions as a pivot in transfer learning. During the forward propagation, GRL serves as an identity transformation. There is no change in the flow as if it does not exist. However, during the backpropagation, GRL multiplies the gradients by the coefficient $-\lambda$ and passes the processed gradients to Feature Extractor. Such operation reverses the gradient direction with scaling effect ($0 < \lambda \leq 1$), producing domain-invariant features that capture two distribution's commonality. The idea is when Domain Classifier cannot judge whether the input graph is from the source domain or the target domain, the knowledge learned from the source domain can be used to help label the data samples in the target domain.

Once the training stage is complete, the G-DANN can be deployed for IDR. Two of the four modules as shown in Fig. 4, the Feature Extractor and the Candidate Classifier, are selected to determine each fault candidate from the diagnosis callout as either the true fault or the false candidate. Their usages are the same as the GraphSAGE in Section III-C (2). The rest two modules, the GRL and the Domain Classifier, are not involved during the prediction stage.

V. Experiments

A. Setup and Evaluation Metrics

Experiments are performed on a Linux workstation, equipped with three 16GB GPUs, two eight-core 3.30GHz CPUs, and 128GB RAM. GRAND is programmed in Python using PyTorch library v1.9.0 [49], PyTorch Geometric library v2.0.2 [50], and sklearn library v1.0.1 [51].

Thirty-two benchmark circuits from ISCAS'85, ISCAS'89, ITC'99, IWLS'05, LGSynth'91 [52], EPFL [53], and openSPARC [54] are used to demonstrate the viability of GRAND for various designs. The injected fault types include single stuck-at line fault (SSL), multiple stuck-at line fault (MSL), bridges (including AND, OR, and dominant types), and front end faults. Altogether, a total number of 155,717 failing chips with 290,957 randomly injected faults are created to form the failing population. Failing chips of each benchmark are randomly divided into three parts by 8:1:1, namely training, validation, and test set. Models are trained and tuned by training and validation sets, respectively. Ten-fold cross validation is performed on the test set for evaluation.

Two diagnosis tools are used to produce the fault callouts, as the starting point of GRAND. The first tool (denoted as Tool A) is a fault simulator-based diagnosis tool that supports arbitrary failing behaviors and fault modeling. The second tool (Tool B) is a widely adopted, state-of-the-art commercial one. In addition to produce candidates that explain (part of) the failings, Tool B also provides match scores.

To systematically gauge the performance of GRAND, we devise an evaluation checklist (EC), on which the metrics are listed in a descending order based on their priority. Model performances are evaluated and compared following this EC: (1) Count; (2) Diagnostic resolution; (3) F2 Score; (4) AUC.

While diagnostic resolution is defined in Section I, the *Count* value for a failing chip is defined as,

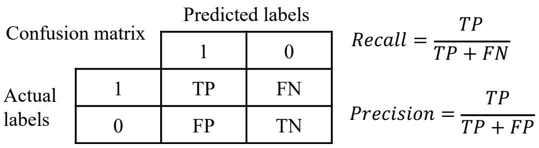
$$\text{Count} = \begin{cases} 1, & \text{at least one true fault is found;} \\ 0, & \text{none of the true fault(s) is found.} \end{cases} \tag{8}$$

Count indicates whether a diagnosis run is successful or not. Because the majority of candidates are false, if a ML-based prediction model assigns label 0 to nearly all of its candidates in an arbitrary fashion, the resolution is good but makes little effort of catching the true faults.

F2 Score takes into account both recall and precision in evaluating prediction models. The metric can be better explained by using a confusion matrix shown in Fig. 5. The problem of IDR can be formulated as a binary classification task mentioned in Section II. F_β Score is defined as,

$$\text{F}_\beta \text{ Score } = \frac{\left(1+\beta^2\right) * Precision * Recall}{\beta^2 * Precision + Recall} \tag{9}$$

where β is a coefficient reflecting that the weight of recall is β times the weight of precision. In the problem of IDR, because of the imbalanced data (number of false candidates is far more than that of true ones), it is important not to miss the true faults. The β is set to 2 in Equation (9) for F2 Score. Unlike F1 score that treats recall and precision equally, F2 score penalizes the false negative term in recall more heavily, avoiding leaving out true faults.



Confusion matrix

		Predicted labels	
		1	0
Actual labels	1	TP	FN
	0	FP	TN

$$Recall = \frac{TP}{TP+FN}$$

$$Precision = \frac{TP}{TP+FP}$$

Fig. 5. The confusion matrix presents four combinatory cases in the binary classification problem, considering the actual candidate labels with the model prediction results. The four categories include true positive (TP), false positive (FP), false negative (FN), and true negative (TN).

Area Under Curve (AUC) is more robust to imbalanced data-class distribution, compared with prediction accuracy $(= \frac{TP+TN}{TP+FN+FP+TN})$. We give two examples to illustrate EC. (1) Suppose there is only one true fault in 100 candidates, and a model predicts three potential faults where the true fault is among one of them. Precision=0.3333, Recall=1.0000, F2 Score=0.7143, accuracy=0.9800, AUC=0.9899, diagnostic resolution=3 and Count=1. (2) Suppose there is only one true fault in 100 candidates. A model predicts one candidate, but is not the true fault. Precision=0, Recall=0, F2 Score=0, accuracy=0.98, AUC=0.4949, diagnostic resolution=1 and Count=0.

The IDR task itself is a Pareto optimization problem. GRAND seeks to find both accurate diagnosis and good resolution on Pareto optimal fronts. EC is more comprehensive and effective for the evaluation than a single statistic can be.

B. Diagnostic Resolution Improvement

We first determine which GNN algorithm works best for IDR. The two GNN algorithms share the same architecture frame and hyperparameters, as shown in Table IV.

TABLE IV
GNN Hyperparameter Description

Parameters	Value
# epochs	800
learning rate	0.01
batch size	128
optimizer	Adam
activation function	ReLU
# initial node features (input dimensionality)	8
node embedding size (hidden dimensionality)	32
output dimensionality	2
loss weight	1:10
sample ratio	1:20

For GNN, eight features are initialized for each node as the input. GNN thus extracts 32 deep features as the node embeddings. They are fed into the fully connected layer to produce two output probability for true and false, respectively. Each algorithm implements two GNN layers with one fully connected layer. The algorithmic difference is the operation and configuration in the GNN convolutional layers, as formulated by Eq. (3) and (6).

Table V compares multiple methods. A benchmark circuit is chosen from each suite as the representative. We choose random forest (RF) and support vector machine (SVM) as baselines to be compared. Both RF and SVM are classic ML methods with huge popularity in EDA applications, including the test field [1–5, 13, 18, 24, 37, 55]. In particular, RF and SVM have been used to improve the efficiency and resolution of diagnosis [1–5, 24, 55]. In our experiments, the inputs to the two algorithms are eight initial features rather than the 32 node embeddings extracted by GNNs. We set the number of decision trees in the RF to 100, the number of features sampled and considered at each split is three, and the minimum leafnode size is one. A Gaussian kernel is used with SVM, with two parameters configured as C=1 and γ=`scale`.

Recall that two of the eight initial node features (`dependency_level` and `dependency_num` in Table I) are calculated by examining the dependency among fault candidates. They are expected to be informative towards GNN modeling and prediction. We also conduct ablation study to validate the efficacy of such dependency, by setting these two features to 0's. The rest procedure is exactly the same as GraphSAGE with undirected graphs. The results are given by *w/o_dependency* in Table V.

The fourth and fifth columns in Table V give the diagnostic resolution (DR) by GRAND (including RF) and Tool A, respectively. Unlike GCN, which is applied to undirected graphs only, GraphSAGE are applicable to both directed and undirected graphs. We append *dir* and *undir* to differentiate between them. To further evaluate the effects of directional information [56] [57], we follow the bidirectional GraphSAGE setting in [57] for an ablation test. The results are reported by *Bidirectional* in each group. Another set of ablation tests are performed to implement larger number of layers in order to examine if a larger search hop helps the IDR task. They are listed by *-3 layers*, *-4 layers*, and *-5 layers* in Table V, meaning GraphSAGE configured with 3 to 5 layers, respectively.

GraphSAGE-undir with two layers is the best in Table V. RF is significantly worse than GNN methods for the first metric Count. Both SVM and RF lack accuracy in identifying true candidates, classifying too many candidates as false ones, which are also evidenced by their poor F2 Scores. Such false negative error is overkill, ruling out a substantial amount of true faults. Thus, the two baseline algorithms (RF and SVM) cannot compete with GNNs. Among the GNN methods, GraphSAGE with undirected graphs outperforms the others, considering all four metrics on EC. It has the highest Count (except *frg2*) and the F2 Score. For AUC, GraphSAGE-undir is the best on two circuits, and second best on the rest three. For the ablation tests on bidirectional information and hop scopes reflected by layer numbers, the results cannot compete with GraphSAGE-undir with two layers either. It is not surprising that an increased number of layers hinders the performance, because of the over-smoothing phenomenon of GNNs [46][47]. We also observed from experiments that runtime, from both training and prediction, increases linearly with the number of layers. For the ablation study on candidate dependency, the results are significantly worse than the three GNN methods, especially the GraphSAGE-undir, demonstrating the effectiveness of the two derived dependency features. Therefore, in the following experiments GRAND uses GraphSAGE as the representative of GNNs for IDR.

TABLE V
ALGORITHM COMPARISON

Circuits	Algorithms	Count	DR[†]	DR_Tool A[‡]	F2 Score	AUC
cavlc	GCN	0.9355	1.7782	5.2702	0.8111	0.9208
	GraphSAGE-dir	0.9301	2.6694		0.6997	0.8532
	GraphSAGE-undir	0.9382	1.8065		0.8196	0.9136
	-3 layers	0.9217	2.6609		0.6985	0.8761
	-4 layers	0.9391	2.6304		0.7187	0.9028
	-5 layers	0.9298	1.9649		0.7664	0.9315
	Bidirectional	0.8596	3.8070		0.5531	0.7610
	RF	0.6102	1.4193		0.5201	0.7804
	SVM	1.0000	9.2730		0.5546	0.3983
	w/o_dependency	0.8900	2.3700		0.7094	0.8840
s1488	GCN	0.9125	2.8906	12.0906	0.6958	0.9301
	GraphSAGE-dir	0.9125	2.5719		0.7178	0.9300
	GraphSAGE-undir	0.9438	2.6344		0.7375	0.9389
	-3 layers	0.9335	2.8478		0.6956	0.9036
	-4 layers	0.9284	2.9987		0.6718	0.8991
	-5 layers	0.9258	2.9412		0.6914	0.9022
	Bidirectional	0.8418	2.9566		0.5848	0.8525
	RF	0.5125	1.1625		0.4583	0.8849
	SVM	0.8200	6.4500		0.4710	0.8440
	w/o_dependency	0.9300	3.3750		0.6407	0.8897
frg2	GCN	0.9767	1.7636	5.3372	0.8676	0.9597
	GraphSAGE-dir	0.9767	1.7907		0.8557	0.9749
	GraphSAGE-undir	0.9690	1.3953		0.9120	0.9690
	-3 layers	0.9335	1.5081		0.8515	0.9670
	-4 layers	0.9520	1.4760		0.8675	0.9541
	-5 layers	0.9355	1.4113		0.8613	0.9638
	Bidirectional	0.9516	2.5927		0.7398	0.8887
	RF	0.8604	1.1400		0.8235	0.9626
	SVM	0.2778	2.3056		0.1705	0.6356
	w/o_dependency	0.6757	2.0541		0.5793	0.8354
b12	GCN	0.9000	2.2469	12.0344	0.7385	0.8810
	GraphSAGE-dir	0.8750	2.0375		0.7207	0.8604
	GraphSAGE-undir	0.9063	2.0031		0.7580	0.9047
	-3 layers	0.9000	2.4250		0.6585	0.8794
	-4 layers	0.8800	2.2400		0.6752	0.8961
	-5 layers	0.8700	1.6500		0.6751	0.8873
	Bidirectional	0.8527	3.1094		0.5444	0.7647
	RF	0.5875	0.9750		0.5398	0.8512
	SVM	0.8650	8.4800		0.5223	0.7603
	w/o_dependency	0.8900	2.1900		0.6546	0.8517
DMA	GCN	0.9500	1.6250	5.4750	0.7879	0.8654
	GraphSAGE-dir	0.9750	1.8750		0.8051	0.8910
	GraphSAGE-undir	0.9500	1.3125		0.8253	0.9229
	-3 layers	0.9500	1.3625		0.8039	0.9235
	-4 layers	0.9500	1.3500		0.7913	0.9130
	-5 layers	0.9750	1.4500		0.8147	0.9335
	Bidirectional	0.9250	2.4750		0.7002	0.7225
	RF	0.8500	1.5750		0.6876	0.9280
	SVM	1.0000	9.3000		0.6636	0.6565
	w/o_dependency	0.9500	1.8625		0.7892	0.9185
l2b	GCN	0.9135	2.2035	5.9091	0.7222	0.8522
	GraphSAGE-dir	0.8235	2.4412		0.6598	0.8060
	GraphSAGE-undir	0.9545	2.4318		0.7232	0.8253
	-3 layers	0.9535	3.5349		0.6559	0.7301
	-4 layers	0.9048	3.1071		0.6389	0.7840
	-5 layers	0.8372	2.8605		0.6029	0.7124
	Bidirectional	0.8140	2.6977		0.5542	0.6304
	RF	0.5909	1.7045		0.4090	0.7518
	SVM	1.0000	7.6429		0.5219	0.5670
	w/o_dependency	0.8387	2.1344		0.7271	0.8242

[†] DR = Diagnostic resolution by GRAND. [‡] Diagnostic resolution by Tool A.

Table VI shows GRAND's performance on benchmarks. Starting from the second column to the seventh column, the metric values are given following the EC order. Results of diagnostic resolution are further divided into three columns: **DR** for the resolution obtained by GRAND, **DR_Tool A** for the resolution calculated using the diagnosis report produced by Tool A, and **Ipv** showing the diagnostic resolution improvement comparing GRAND over the tool.

To mimic production diagnosis, if the resolution in the diagnoses produced by Tool A is already below a threshold and

TABLE VI
RESULTS OF GRAND ON BENCHMARKS, USING GRAPHSAGE WITH CIRCUITS MODELED AS UNDIRECTED GRAPHS

Circuits	#Failings	#Defects	#Tests	Coverage	Count	Count_Tool A	DR	DR_Tool A	Ipv	F2	AUC	Tool A*	Trng†	Pred‡
ctrl	1370	2330	29	91.598%	0.9545	0.8220	1.5341	7.3977	4.82×	0.8561	0.9514	0.06	1.28	0.76
int2float	833	1435	82	100.000%	0.9815	0.8098	1.5185	5.7037	3.76×	0.8590	0.9481	0.06	2.74	0.79
dec	1721	5087	256	100.000%	0.9000	0.9678	2.6850	5.8400	2.18×	0.6718	0.8428	0.19	3.12	1.14
cavlc	2995	4577	153	100.000%	0.9382	0.8672	1.8065	5.2702	2.92×	0.8196	0.9136	0.45	13.74	0.83
priority	2182	2726	279	86.332%	0.9881	0.8102	2.5119	19.9464	7.94×	0.8666	0.9724	24.84	6.18	0.64
adder	3578	5821	249	100.000%	0.9875	0.9783	1.3906	5.6219	4.04×	0.9262	0.9841	1.70	5.34	0.70
i2c	6482	9292	603	99.859%	0.9500	0.9817	1.9500	8.4600	4.34×	0.8117	0.9549	3.33	14.79	0.81
bar	3748	5350	908	100.000%	0.9900	0.8939	1.4250	8.8650	6.22×	0.9212	0.9835	25.68	7.10	0.90
square	505	755	89	99.988%	1.0000	0.9957	1.0238	6.9048	6.74×	0.9960	0.9980	57.69	11.55	1.40
sin	1567	2058	167	93.678%	0.9114	0.9677	1.6709	15.6203	9.35×	0.8120	0.9489	20.29	6.10	1.07
c499	2408	4402	78	99.700%	0.9342	0.8571	4.3158	50.0625	11.60×	0.6422	0.9116	0.25	4.01	0.60
s1488	4487	6836	116	100.000%	0.9438	0.9704	2.6344	12.0906	4.59×	0.7375	0.9389	0.30	11.13	0.78
s4863	2758	3431	70	100.000%	0.9700	1.0000	1.1650	5.1000	4.38×	0.9177	0.9682	2.18	4.49	1.06
s5378	6205	8485	640	99.121%	0.9600	0.9732	2.9100	6.7250	2.31×	0.6994	0.8509	12.62	14.16	0.79
s6669	5818	7177	56	100.000%	0.9400	0.9968	2.3600	6.0900	2.58×	0.7549	0.8940	10.38	2.32	1.09
s9234	8880	12696	951	93.460%	0.8600	0.9826	2.6850	5.4550	2.03×	0.6439	0.7483	7.97	3.12	1.12
s13207	8447	10347	1651	98.462%	0.9119	0.9880	3.4843	8.5031	2.44×	0.6110	0.8210	38.57	16.47	0.92
s15850	9528	11279	1622	96.682%	0.9563	0.9909	2.4281	6.6094	2.72×	0.7354	0.8456	60.07	6.10	0.89
s35932	8997	14123	29	91.600%	1.0000	1.0000	1.2273	7.2121	5.88×	0.9476	0.9914	33.45	3.42	0.34
x1	5049	10501	413	98.859%	0.8500	0.9697	2.4400	18.2950	7.50×	0.6688	0.8455	6.56	2.27	1.00
pair	4768	5875	476	95.050%	0.9800	0.9869	2.2675	7.3975	3.26×	0.7827	0.9226	1.75	16.09	0.68
frg2	1297	2011	779	92.935%	0.9690	0.9787	1.3953	5.3372	3.83×	0.9120	0.9690	1.06	9.40	0.90
i10	978	1372	989	88.963%	0.9796	0.9847	2.2296	14.5510	6.53×	0.8177	0.9489	4.67	4.04	0.85
des	7078	11475	1822	95.078%	0.8800	0.9951	2.7275	11.2500	4.12×	0.6457	0.8929	24.30	17.68	1.06
b12	13311	18184	115	100.000%	0.9063	0.9648	2.0031	12.0344	6.01×	0.7580	0.9047	2.51	10.0	0.80
b14	3020	3555	2254	99.250%	0.9773	0.9722	1.9205	6.0795	3.17×	0.8049	0.8776	37.98	11.94	1.01
b15	3543	3856	2294	96.303%	0.9559	0.9861	2.5000	6.4706	2.59×	0.7440	0.8871	56.40	4.44	1.05
b17	3664	42340	563	97.960%	0.9355	1.0000	3.0968	9.6613	3.12×	0.6177	0.8371	42.73	6.35	0.39
b22	3658	40930	505	98.550%	1.0000	1.0000	3.0625	10.7917	3.52×	0.6389	0.8569	53.67	4.64	0.25
DMA	9574	11547	318	92.820%	0.9500	1.0000	1.3125	5.4750	4.17×	0.8253	0.9229	32.87	7.71	0.57
DSP	10444	12436	517	99.510%	0.9630	1.0000	1.5000	5.0000	3.33×	0.7529	0.8800	74.89	10.06	0.38
l2b	6824	8668	3006	99.706%	0.9545	0.8948	2.4318	5.9091	2.43×	0.7232	0.8253	18.61	4.03	1.10

*Runtime of this tool is measured by hours. †Training time is measured by minutes. ‡Prediction time is measured by milliseconds.

TABLE VII
IMPROVEMENTS OVER A COMMERCIAL DIAGNOSIS TOOL

Circuits	Count	Count_Tool B	DR	DR_Tool B	Ipv	F2	AUC	Tool B*	Trng	Pred	#Chips	#Identified	Ratio
ctrl	0.9474	0.9922	1.4737	8.8947	6.04×	0.8860	0.9633	1.02	0.71	0.81	16	10	0.6250
int2float	0.9231	1.0000	1.0000	8.6923	8.69×	0.9231	0.9538	1.80	0.60	0.82	4	2	0.5000
dec	1.0000	1.0000	1.8485	7.3758	3.99×	0.8904	0.9469	1.43	0.80	0.25	1	1	1.0000
cavlc	0.9787	0.9996	1.0426	8.0638	7.73×	0.9681	0.9981	6.49	1.48	0.75	6	3	0.5000
priority	0.9310	0.9869	1.3103	23.2414	17.74×	0.8770	0.9903	31.64	4.02	0.66	120	49	0.4083
adder	0.9833	0.9800	1.1167	15.3833	13.78×	0.9619	0.9970	4.77	1.34	0.97	759	509	0.6706
i2c	0.9867	0.9993	2.5800	10.9500	4.24×	0.7774	0.9786	15.12	7.49	0.73	7	6	0.8571
bar	0.9100	0.9999	2.3200	14.0900	6.07×	0.7476	0.9438	25.61	4.17	0.90	19	19	1.0000
square	1.0000	0.9788	1.2308	6.8846	5.59×	0.9663	0.9945	10.52	6.92	1.43	109	46	0.4220
sin	0.9362	1.0000	1.1915	11.3262	9.51×	0.9024	0.9380	160.88	14.67	1.94	33	15	0.4545
c499	0.9467	0.8739	2.9893	7.6000	2.54×	0.6859	0.8136	2.81	1.65	0.70	47	22	0.4681
s1488	0.9412	0.8437	1.8824	8.4706	4.50×	0.8109	0.9745	9.72	0.44	0.16	95	63	0.6632
s4863	0.8667	0.8550	3.5333	16.8000	4.75×	0.6151	0.8809	14.25	0.30	0.18	624	490	0.7853
s5378	0.8167	0.8477	3.4500	37.3667	10.83×	0.5584	0.9069	21.72	1.25	0.60	688	343	0.4985
s6669	0.8833	0.8666	2.6667	26.1417	9.80×	0.6959	0.9395	21.33	1.46	0.64	861	668	0.7758
s9234	0.8833	0.9059	2.7083	20.0833	7.42×	0.6821	0.9028	34.04	2.88	0.73	336	202	0.6012
s13207	0.8800	0.8708	1.4800	9.2000	6.22×	0.8088	0.9725	30.97	1.09	0.29	118	68	0.5763
s15850	0.9167	0.8632	2.1333	14.2917	6.70×	0.7875	0.9425	33.35	3.04	0.74	230	107	0.4652
s35932	0.8667	0.8570	3.0083	17.1208	5.69×	0.6274	0.8667	13.50	6.68	0.89	198	167	0.8483
x1	0.8833	0.9903	3.5500	13.7000	3.86×	0.6291	0.7812	214.58	2.63	0.63	61	34	0.5574
pair	0.9023	0.9998	2.8305	12.3276	4.36×	0.6781	0.8993	13.51	11.77	0.68	6	3	0.5000
frg2	0.9459	0.9947	3.8108	11.4865	3.01×	0.6746	0.8411	12.11	3.56	0.75	8	1	0.1250
i10	0.9098	0.9972	3.2669	13.1241	4.02×	0.6453	0.8996	16.85	15.81	0.71	18	14	0.7778
des	0.9706	0.9999	2.9265	11.2574	3.85×	0.7799	0.8413	46.01	12.86	0.89	23	11	0.4783
b12	0.9500	0.9946	2.5917	5.8750	2.27×	0.7264	0.6579	13.31	0.74	0.59	43	29	0.6744
b14	0.9000	0.9987	2.7000	7.1333	2.64×	0.6695	0.7416	19.63	4.28	0.99	37	28	0.7568
b15	0.9167	0.9993	3.0000	5.9750	1.99×	0.6694	0.6545	31.30	3.91	0.83	31	21	0.6774
b17	0.9167	0.8961	1.2417	6.3417	5.11×	0.8771	0.9748	197.25	11.22	0.88	215	210	0.9767
b22	0.9000	0.9248	2.4333	12.5500	5.16×	0.7266	0.9124	77.43	6.89	0.86	1434	882	0.6151
DMA	0.8667	0.9586	4.0583	13.5833	3.35×	0.5779	0.8013	38.30	11.68	0.88	68	59	0.8676
DSP	0.8667	0.9534	3.8167	22.4083	5.87×	0.5761	0.8655	175.81	21.73	0.87	75	26	0.3467
l2b	0.9262	0.9991	2.8065	10.9677	3.91×	0.6834	0.8916	63.69	6.12	0.93	146	99	0.6781

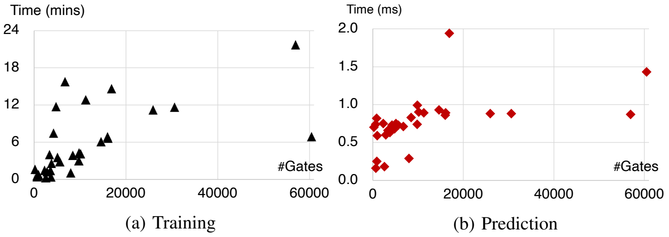
*Runtime of this tool is measured by minutes.

deemed acceptable, there is no more need to invoke GRAND for further analysis, as shown in Fig. 1. Here the threshold is five, the same as previous work. Tool A is able to identify at least one true fault from 95.58% of the failing chips, with an averaged diagnostic resolution of 9.87. Comparatively, on average, GRAND attains accurate diagnoses on 94.93% of the chips (reflected by Count) with a resolution of 2.18. GRAND improves diagnostic resolution by 4.51×.

The last three columns in Table VI give the execution time of Tool A and runtime overhead by applying GRAND. On average, it takes 10.88% additional time to train GRAND models after Tool A finishes the diagnosis. For a failing chip, GRAND takes no more than 1.40 milliseconds on average to complete the task of IDR. It is thereby feasible to incorporate GRAND into a post-silicon debug flow. GRAND will not incur computation burden that affects time-to-market duration.

Table VII presents GRAND's performance compared with Tool B. All circuits, including the sequential ones, are synthesized using OSU035 (compatible with TSMC 180 nm process)[58]. Similar to Table VI, GRAND improves resolution for all circuits. On average, the commercial Tool B has a diagnostic resolution of 13.08, while GRAND achieves 2.44. GRAND improves diagnostic resolution by 5.98×.

The runtime overhead in Table VII again reveals that GRAND executes the task of IDR efficiently, as shown in Fig. 6. Similar to the results in Table VI, while both the training and prediction time are positively associated with the circuit sizes, the training time also depends on the amount of data samples used, which are the simulated failing chips in the experiment. The longest training time is $<$ 22 minutes for the circuit *DSP*. Scaling issue does not exist during prediction stage, considering the fact that except for two circuits, GRAND takes no more than 1 millisecond to finish the IDR task. The longest prediction time GRAND takes is 1.94 milliseconds. It takes 26.82% additional training time after Tool B finishes the diagnosis. We analyze the memory consumption of GNN models. GRAND for above experiments has the same model size, which is 16.95KB. According to Eq (6), the dimension of input $\mathbf{x}_i^{(0)}$ and $\mathbf{x}_j^{(0)}$ are both 8×1, and the dimension of output $\mathbf{x}_i^{(1)}$ is 32×1. The GraphSAGE parameters $\mathbf{W}$ and $\mathbf{W}_N$ are matrices with dimension 32×8, suggesting that the sizes of trained models are independent of the input scale.



(a) Training (b) Prediction

Fig. 6. Plots of training and prediction time using GRAND. The x-label gives the numbers of gates in each circuit under diagnosis.

F2 Score is primarily used for algorithm comparison and model parameterization as in Table V. It functions on a macro level weighing precision and recall, while imposing heavier penalty to combat false negatives (true faults determined as false candidates). On average, the F2 Score is 0.7789 and 0.7466 for Table VI and Table VII, respectively. AUC is robust to imbalanced data distribution, a better choice than the generic accuracy given that the number of true faults are much less than that of false candidates. The AUC is 0.9074 and 0.8901 for Table VI and Table VII, respectively. The fact that GRAND is able to determine the true or false nature of a candidate accurately proves the validity of hyperparameter setting (such as loss weight and sample ratio).

Fig. 7 further characterizes the diagnoses produced by Tool B. In addition to report suspects, Tool B also provide match scores to the candidates. In most cases considered, true faults exist in the top-scoring candidates. However, for 6.67% of the the total number of 152,308 failing chips from 32 benchmark circuits, true faults exist in non top-scoring candidates. Specifically, the bars in Fig. 7 indicate the accumulated numbers of failing chips that have true faults actually found in the 1st, 2nd, 3rd, ... ,etc, rank of match scores. This exception suggests that, if only selecting the top-scoring candidates from the diagnoses, true faults can hardly be found for these 10,157 failing chips in later failure analysis. Even worse, using these top-scoring but incorrect candidates to guide PFA is very likely to fail, wasting resources and time.

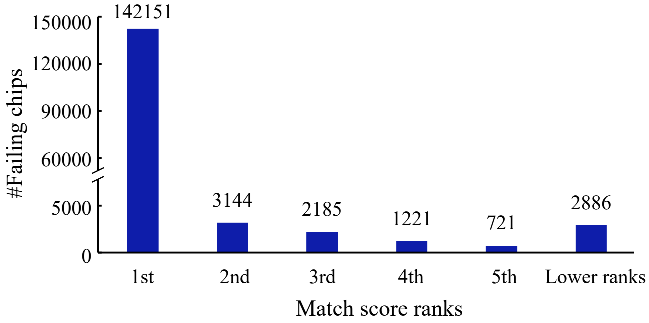
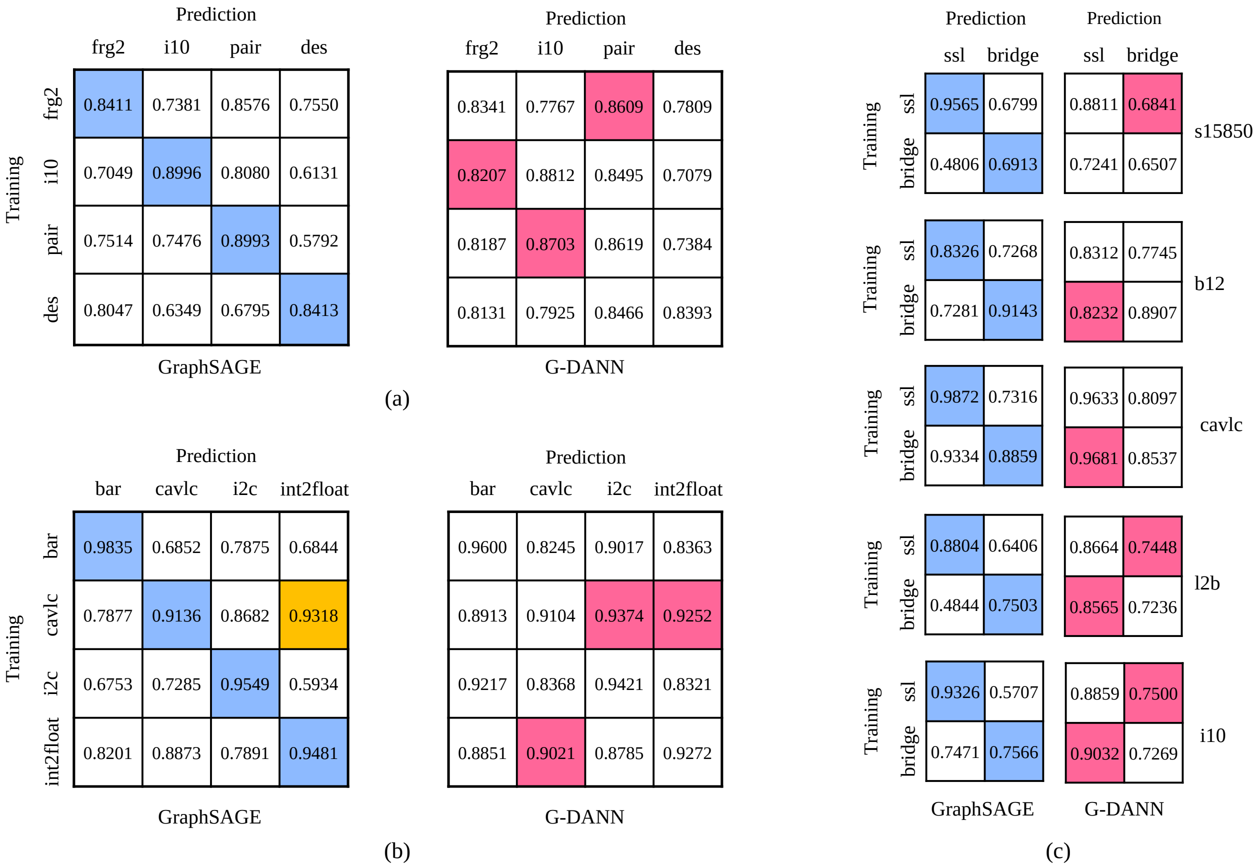


Fig. 7. Altogether, 10,157 chips (sum of all bar counts except the first 142,151 ones) have true faults not given top mach scores in the commercial diagnoses. True faults may slip through the net and go undetected, due to the imperfection in match scores.

The fourth part in Table VII presents the results of using GRAND to handle the problem caused by imperfect scoring. Column **#chips** gives the numbers of chips whose true faults exist in non top-scoring candidates. The last two columns, **#Identified** and **Ratio**, give the number of chips and their corresponding ratio, which are correctly identified by GRAND for their true faults. GRAND achieves a correction ratio ≥0.5 on 20 of the 32 circuits, successfully catches 62.96% of the true faults hiding within the non top-scoring candidates. GRAND can effectively rectify the existing, commercial diagnoses. Please note the last columns in Table VII are collected from existing experiment results, focusing on recording the faults with non top scores. No extra training or test is required.

C. Failure Information Sharing

Fig. 8 presents the results on two cases considered, measured by AUC values. (1) The candidates labels for one circuit



(a)

GraphSAGE

Training \ Prediction	frg2	i10	pair	des
frg2	0.8411	0.7381	0.8576	0.7550
i10	0.7049	0.8996	0.8080	0.6131
pair	0.7514	0.7476	0.8993	0.5792
des	0.8047	0.6349	0.6795	0.8413

G-DANN

Training \ Prediction	frg2	i10	pair	des
frg2	0.8341	0.7767	0.8609	0.7809
i10	0.8207	0.8812	0.8495	0.7079
pair	0.8187	0.8703	0.8619	0.7384
des	0.8131	0.7925	0.8466	0.8393

(b)

GraphSAGE

Training \ Prediction	bar	cavlc	i2c	int2float
bar	0.9835	0.6852	0.7875	0.6844
cavlc	0.7877	0.9136	0.8682	0.9318
i2c	0.6753	0.7285	0.9549	0.5934
int2float	0.8201	0.8873	0.7891	0.9481

G-DANN

Training \ Prediction	bar	cavlc	i2c	int2float
bar	0.9600	0.8245	0.9017	0.8363
cavlc	0.8913	0.9104	0.9374	0.9252
i2c	0.9217	0.8368	0.9421	0.8321
int2float	0.8851	0.9021	0.8785	0.9272

(c)

	Training	GraphSAGE ssl	GraphSAGE bridge	G-DANN ssl	G-DANN bridge
s15850	ssl	0.9565	0.6799	0.8811	0.6841
	bridge	0.4806	0.6913	0.7241	0.6507
b12	ssl	0.8326	0.7268	0.8312	0.7745
	bridge	0.7281	0.9143	0.8232	0.8907
cavlc	ssl	0.9872	0.7316	0.9633	0.8097
	bridge	0.9334	0.8859	0.9681	0.8537
l2b	ssl	0.8804	0.6406	0.8664	0.7448
	bridge	0.4844	0.7503	0.8565	0.7236
i10	ssl	0.9326	0.5707	0.8859	0.7500
	bridge	0.7471	0.7566	0.9032	0.7269

Fig. 8. GRAND's performance on failure information sharing. An AUC value in the (i,j) entry of a matrix is obtained from a prediction model trained on the ith circuit, and tested (evaluated) on the jth circuit. Those diagonal matrix cells filled by the **blue** color are the results using GraphSAGE for training and test on the same circuit or fault type, without using transfer learning. They are the baseline and reference for comparison. A **red** cell in the (i,j) entry $(i \neq j)$ denotes that a model trained by G-DANN has similar performance ($\sim -3\%$ of AUC) compared with the blue (j, j) entry of the matrix on the left (holding AUC results of models by GraphSAGE). The **yellow** cell means the exception, where GRAND by GraphSAGE outperforms the transfer learning-based option.

type are available, but not for the other. Fig. 8 (a) and (b) show the results on circuits from LGSynth'91 and EPFL suites, respectively, for this case. (2) For the same circuit, candidate labels for a particular defect type are available, but not for the other defects. Fig. 8 (c) is for such scenario, using SSL and bridges as two fault types.

In Fig. 8 (a), the left (right) matrix shows the results using GraphSAGE (G-DANN). The left matrix (GraphSAGE) is the baseline for comparison, while the right matrix (G-DANN) employs transfer learning to improve GraphSAGE. An AUC value in the (i,j) entry of a matrix is obtained from a prediction model trained on the ith circuit, and tested (evaluated) on the jth circuit. For example, the AUC value is 0.8703 for the $(3,2)^{th}$ entry in the right G-DANN matrix, referring to a prediction model trained by G-DANN on labeled samples from circuit *pair* and evaluated on *i10*.

Three colors, blue, red, and yellow, are used to differentiate the results from comparison, as the Fig. 8 caption explains. For example, consider the $(2,4)^{th}$ entry in the G-DANN matrix in Fig. 8 (b). The AUC value 0.9252 is obtained by training on *cavlc* and tested on *int2float* via G-DANN. The cell is highlighted by red because it is close to the $(4,4)^{th}$ entry 0.9481 (highlighted by blue as the baseline) in the GraphSAGE matrix on the left, whose training and test are both performed on *int2float*. The $(2,4)^{th}$ entry in the GraphSAGE matrix is highlighted by yellow, showing the only exception in our experiment where GraphSAGE outperforms G-DANN.

Experiments validate two facts. First, transfer learning has played its part. In all experiments reported in Fig. 8, the AUC values of the diagonal entries from the left GraphSAGE matrix are better than those from the right G-DANN matrix. Comparatively, for the off-diagonal entries, G-DANN is significantly superior to GraphSAGE, with only one exception highlighted by yellow. This means that, powered by transfer learning, G-DANN outperforms GraphSAGE for tasks requiring labels. Note that G-DANN can largely mitigate the loss of lacking labels, but cannot compete in scenarios where labels do exist. Second, failure information from other designs or fault types can be borrowed to help identify true/ false candidates when actual labels are lacking towards building ML models. GRAND succeeds in six cases for sharing failure information across different types of chips, demonstrated by the red entires in Fig. 8 (a) and (b). For sharing failure information across different fault types, GRAND performs well on seven out of 10 cases shown by Fig. 8 (c).

VI. Conclusion

Post-silicon debug calls for good diagnostic resolution to guide PFA and speed up yield learning. In this work, we present GRAND (**GRA**ph **N**eural networks for **D**iagnosis) for improving diagnostic resolution (IDR). Unlike previous works using GNNs for design- [35][36] and test- [37][38] related purposes, GRAND is the first work leveraging GNNs for fault diagnosis, expanding the role of GNN as a novel ML paradigm in diagnostic analytics.

GRAND begins by modeling the circuits as graphs, and computes initial node features with information from logical neighborhood, circuitry topology, and candidate dependency. Rich features represented by node embeddings are then extracted via deep learning using graphs. Graph neural networks are then geared towards identifying the true/ false candidates. Experiments demonstrate that the diagnostic resolution can be improved by 4.51× to 5.98× compared with existing diagnosis tools. Besides IDR, GRAND can adjust existing commercial diagnoses by successfully identify 62.96% of true faults that are likely to be ignored because they were originally not ranked in the first place among all candidates. Finally, GRAND can make use of failure information from other designs or failures to better understand failure mechanisms where labeled true/ false candidates are lacking.

References

[1] S. Mittal and R. Blanton, "A deterministic-statistical multiple-defect diagnosis methodology," in *IEEE VLSI Test Symposium (VTS)*, 2020.

[2] Q. Huang, C. Fang, S. Mittal, and R. D. Blanton, "Towards smarter diagnosis: A learning-based diagnostic outcome previewer," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 25, no. 5, pp. 1–20, 2020.

[3] Q. Huang, C. Fang, and R. D. Shawn Blanton, "LAIDAR: Learning for accuracy and ideal diagnostic resolution," in *IEEE International Test Conference (ITC)*, 2020, pp. 1–10.

[4] Y. Xue, O. Poku, X. Li, and R. D. Blanton, "PADRE: Physically-aware diagnostic resolution enhancement," in *IEEE International Test Conference (ITC)*, 2013, pp. 1–10.

[5] Y. Xue, X. Li, and R. D. Blanton, "Improving diagnostic resolution of failing ICs through learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 6, pp. 1288–1297, 2016.

[6] X. Yu and R. D. Blanton, "Improving diagnosis through failing behavior identification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 10, pp. 1614–1625, 2012.

[7] S. Chillarige *et al.*, "Machine learning driven throughput optimization of volume diagnosis methodology," in *IEEE International Test Conference India*, 2020, pp. 1–8.

[8] X. Huang *et al.*, "Adaptive NN-based root cause analysis in volume diagnosis for yield improvement," in *IEEE International Test Conference (ITC)*, 2021, pp. 30–36.

[9] M.-T. Wu, C.-S. Kuo, J. C.-M. Li, C. Nigh, and G. Bhargava, "Improving volume diagnosis and debug with test failure clustering and reorganization," in *IEEE International Test Conference (ITC)*, 2021, pp. 251–259.

[10] Y.-T. Lin and R. D. Blanton, "Test effectiveness evaluation through analysis of readily-available tester data," in *IEEE International Test Conference (ITC)*, 2009, pp. 1–10.

[11] I. Pomeranz, "New targets for diagnostic test generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 10, pp. 3035–3043, 2020.

[12] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*. New York, USA: Springer, 2002.

[13] H. Wang and K. He, "Improving test and diagnosis efficiency through ensemble reduction and learning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 24, no. 5, pp. 1–26, 2019.

[14] OpenROAD, "The OpenROAD Project." [Online]. Available: https://theopenroadproject.org

[15] T. Ajayi *et al.*, "Invited: Toward an open-source digital flow: First learnings from the OpenROAD project," in *56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–4.

[16] N. Wang, I. Pomeranz, B. Benware, M. E. Amyeen, and S. Venkataraman, "Improving the resolution of multiple defect diagnosis by removing and selecting tests," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2018, pp. 1–6.

[17] Y. Huang, J. Janicki, and S. Urban, "Non-adaptive pattern reordering to improve scan chain diagnostic resolution," in *IEEE European Test Symposium (ETS)*, 2019, pp. 1–6.

[18] H. Wang et al., "Test-data volume optimization for diagnosis," in *ACM/IEEE of Design Automation Conference (DAC)*, 2012, pp. 567–572.

[19] S. Tanwir, S. Prabhu, M. Hsiao, and L. Lingappan, "Information-theoretic and statistical methods of failure log selection for improved diagnosis," in *International Test Conference (ITC)*, 2015, pp. 1–10.

[20] H. Wang, Z. Wu, and W. Liu, "CNN-based data-model co-design for efficient test-termination prediction," in *IEEE European Test Symposium (ETS)*, 2022, pp. 1–6.

[21] C. Chen et al., "CNN-based stochastic regression for IDDQ outlier identification," in *IEEE VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[22] Synopsys, "TetraMAX ATPG User Guide," 2018. [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/TetraMAXATPG.aspx

[23] S. Mittal and S. Blanton, "NOIDA: Noise-resistant intra-cell diagnosis," in *IEEE 36th VLSI Test Symposium (VTS)*, 2018, pp. 1–6.

[24] J. E. Nelson, W. C. Tam, and R. D. Blanton, "Automatic classification of bridge defects," in *IEEE International Test Conference (ITC)*, 2010, pp. 1–10.

[25] R. Desineni, O. Poku, and R. D. Blanton, "A logic diagnosis methodology for improved localization and extraction of accurate defect behavior," in *IEEE International Test Conference (ITC)*, 2006, pp. 1–10.

[26] R. D. Blanton, W. C. Tam, X. Yu, J. E. Nelson, and O. Poku, "Yield learning through physically aware diagnosis of IC-failure populations," *IEEE Design & Test of Computers*, vol. 29, no. 1, pp. 36–47, 2012.

[27] D. A. Papa and I. L. Markov, "Hypergraph partitioning and clustering," in *Handbook of Approximation Algorithms and Metaheuristics*, 2007.

[28] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 7, no. 1, pp. 69–79, 1999.

[29] J. Sweeney, R. Purdy, R. D. Blanton, and L. Pileggi, "CircuitGraph: A python package for Boolean circuits," *Journal of Open Source Software*, vol. 5, no. 56, p. 2646, 2020.

[30] J. Baehr, A. Bernardini, G. Sigl, and U. Schlichtmann, "Machine learning and structural characteristics for reverse engineering," *Integration*, vol. 72, pp. 1–12, 2020.

[31] P. Krishnamurthy, A. B. Chowdhury, B. Tan, F. Khorrami, and R. Karri, "Explaining and interpreting machine learning CAD decisions: An IC testing case study," in *ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, 2020, pp. 129–134.

[32] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[33] W. Cao, C. Zheng, Z. Yan, and W. Xie, "Geometric deep learning: progress, applications and challenges," *Science China Information Sciences (SCIS)*, vol. 65, no. 2, pp. 126 101:1–126 101:3, 2022.

[34] D. S. Lopera, L. Servadei, G. N. Kiprit, R. Wille, and W. Ecker, "A comprehensive survey on Electronic Design Automation and graph neural networks: Theory and applications," *to appear at ACM Transactions on Design Automation of Electronic Systems (TODAES)*, pp. 1–25, 2022.

[35] Y.-C. Lu *et al.*, "TP-GNN: A graph neural network framework for tier partitioning in monolithic 3D ICs," in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[36] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu, "Understanding graphs in EDA: From shallow to deep learning," in *Proceedings of the International Symposium on Physical Design (ISPD)*, 2020, pp. 119–126.

[37] Y. Ma *et al.*, "High performance graph convolutional networks with applications in testability analysis," in *Proceedings of the 56th Annual Design Automation Conference (DAC)*, 2019, pp. 1–6.

[38] M. Li, S. Khan, Z. Shi, N. Wang, Y. Huang, and Q. Xu, "Representation learning of logic circuits," *arXiv preprint arXiv:2111.14616*, 2021.

[39] R. Kirby, S. Godil, R. Roy, and B. Catanzaro, "CongestionNet: Routing congestion prediction using deep graph neural networks," in *IFIP/IEEE*

27th International Conference on Very Large Scale Integration (VLSI-SoC), 2019, pp. 217–222.

[40] Z. Xie, R. Liang, X. Xu, J. Hu, Y. Duan, and Y. Chen, "Net2: A graph attention network method customized for pre-placement net length estimation," in *26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 671–677.

[41] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph neural network inference for transferable power estimation," in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[42] S. Jin, F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu, "Efficient board-level functional fault diagnosis with missing syndromes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 35, no. 6, pp. 985–998, 2016.

[43] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[44] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[45] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 1025–1035.

[46] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, vol. 34, no. 04, 2020, pp. 3438–3445.

[47] H. Shi, J. Gao, H. Xu, X. Liang, Z. Li, L. Kong *et al.*, "Revisiting over-smoothing in BERT from the perspective of graph," in *International Conference on Learning Representations (ICLR)*, 2022.

[48] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1180–1189.

[49] https://pytorch.org/docs/1.9.0/.

[50] https://pytorch-geometric.readthedocs.io/en/2.0.2/.

[51] https://scikit-learn.org/stable/.

[52] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," Microelectronics Center of North Carolina, Tech. Rep., 1991.

[53] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic Synthesis (IWLS)*, 2015.

[54] https://www.oracle.com/servers/technologies/opensparc-overview.html.

[55] S. Mittal and R. Blanton, "LearnX: A hybrid deterministic-statistical defect diagnosis methodology," in *IEEE European Test Symposium (ETS)*, 2019.

[56] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate operation delay prediction for FPGA HLS using graph neural networks," in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–9.

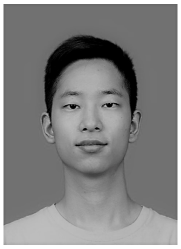
[57] Z. He, Z. Wang, C. Bail, H. Yang, and B. Yu, "Graph learning-based arithmetic block identification," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–8.

[58] https://vlsiarch.ecen.okstate.edu/, Oklahoma State University system on chip (SOC) design flows.

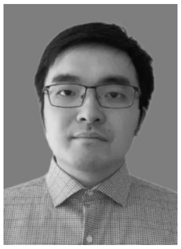
Hongfei Wang (Member, IEEE) received the M.S. degree in computer engineering from Carnegie Mellon University (CMU), Pittsburgh, PA, USA, in 2010, and the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2020.

He is currently an Associate Professor with the School of Cyber Science and Engineering, HUST. Before joining HUST, he worked with Intel Research Lab, Pittsburgh, PA, USA, and also with the Advanced Chip Test Laboratory, CMU. His current research interests include statistical optimization of test and diagnosis solutions for digital systems, hardware design for machine learning systems, and artificial intelligence for enhancing VLSI variability and reliability.

Ziqiang Zhang received the B.S. degree in software engineering from Huazhong University of Science and Technology, Wuhan, China, in 2020. He is currently a graduate student with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interest is VLSI test and diagnosis.



Hongcan Xiong received the B.S. degree in software engineering from Huazhong University of Science and Technology, Wuhan, China, in 2021. He is currently a graduate student with the School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China. His research interest is graph neural networks and deep learning.



Dongmian Zou (Member, IEEE) received the B.S. degree from Chinese University of Hong Kong, Hongkong, China, in 2012, and the Ph.D. degree from University of Maryland, Maryland, USA, in 2017. He is currently an Assistant Professor of the Division of Natural and Applied Sciences, Duke Kunshan University, Jiangsu, China. His current research interests include machine learning, graph neural network, and signal processing.

Yu Chen received the B.S. degree in computer science from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2021. He is currently a graduate student with the School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China. His research interest is VLSI test.

Hai Jin (Fellow, IEEE) received the Ph.D. degree in computer engineering from the Huazhong University of Science and Technology (HUST), China, in 1994.

He is a Cheung Kung Scholars Chair Professor with the School of Computer Science and Technology, HUST. He worked with The University of Hong Kong, Hong Kong, from 1998 to 2000, and as a Visiting Scholar with the University of Southern California, Los Angeles, CA, USA, from 1999 to 2000. He has coauthored 15 books and published over 600 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security. Dr. Jin was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. In 1996, he was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz in Germany. He is a Fellow of CCF and a member of ACM.