Robust Deep Learning for IC Test Problems

Animesh Basak Chowdhury, Benjamin Tan, *Member, IEEE,* Siddharth Garg, and Ramesh Karri, *Fellow, IEEE*

***Abstract*—Numerous machine learning (ML), and more recently, deep learning (DL) based approaches, have been proposed to tackle scalability issues in electronic design automation, including those in integrated circuit (IC) test. This paper examines state-of-the-art DL for IC test and highlights two critical unaddressed challenges. The first challenge involves identifying fit-for-purpose statistical metrics to train and evaluate ML model performance and usefulness in IC test. Our work shows that current metrics do not reflect how well ML models have learned to generalize and perform in the domain-specific context. From this insight, we propose and evaluate alternative metrics that better capture a model's likely usefulness in the IC test problem. The second challenge is to choose an appropriate input abstraction so as to enable an ML model to learn robust and reliable features. We investigate how well DL for IC test techniques generalize by exploring their robustness to perturbations that alter a netlist's structure but do not alter its functionality. This paper provides insights into challenges via empirical evaluation of the state-of-the-art and offers guidance for future work.**

***Index Terms*—Machine Learning, Deep Learning, Adversarial Perturbations, VLSI Testing**

I. Introduction

The increasing complexity of integrated circuits (IC) poses challenges throughout the design flow, including scalability problems in verification and test. To overcome such challenges, researchers have proposed machine learning (ML) based techniques (ML) [1]–[4], and more recently, emerging techniques using deep learning (DL). For example, in lieu of lithography simulation, designers can use a deep neural network (DNN) to detect design hotspots [5] in a physical layout. In physical design, designers can use DNNs to estimate routability [6]. The power of DL comes from its ability to approximate functions [7], [8], offering faster test turnaround compared to heuristic and analytic techniques [3], [9]. To harness this capability, test engineers need to map IC test problems to "learning-based" ML problems. This requires an in-depth understanding of the IC test domain for selecting appropriate features from a design as inputs to an ML model.

Since ML/DL is *data-driven*, care must be taken to avoid problems such as the potential fragility of trained models. Otherwise, poorly trained models incur additional manual effort when deployed in a system. In a typical Electronic Design Automation (EDA) design life-cycle, such situations might arise after deploying ML models with poor predictive and classification performance. If ML models are blindly trusted, manual effort for re-doing the work using conventional methods can undo the efficiency gains purported by adopting ML-in-the-loop. The entire production cycle and the timeline for chip manufacturing is delayed. It is therefore critical for designers to properly assess the "quality" of a model.

In this paper, we study two challenges in using DL for IC testing that arise from a critical analysis of state-of-the-art works in this area. First, we observe that prior approaches tend to use conventional ML metrics such as accuracy, F1 score, or mean square error to train and *assess* the trained models; in typical workflows, a model is deployed only after it is deemed "fit-for-purpose" using these metrics. *However, are these metrics meaningful with respect to the final (IC testing) objective?* We show that regular ML metrics can mislead and often poorly correlate with test-specific metrics. We investigate the reasons for this mismatch and argue for the use of new ML metrics in model evaluation. From these insights we formulate loss functions tuned to the IC testing context.

The second challenge we address is the selection of appropriate features to train generalized and robust DL models for IC testing. Specifically, we investigate whether models that learn from features used in state-of-the-art works are robust to small perturbations of the input. Robustness provides some evidence of how well the trained models generalize to new inputs. Adversarial "attacks" have been demonstrated in various image-based settings [10] and have extended to ML-based CAD, including DL-based lithographic hotspot detection [11]. While past studies in adversarial ML invoke arbitrary modifications of images, the inputs for IC test are structured—one might assume that this affords better robustness. *However, is DL for IC test robust?* Our study sheds light on this question.

In exploring these two challenges, we provide new insights into the practical application of DL in the domain of IC testability. Our studies reveal potential shortcomings while applying DL-based techniques in IC testing on public datasets. Precisely, this work systematically investigates two important aspects of DL in the context of IC testing:

1) **Robustness of Metrics:** Do classical metrics for classification and regression adequately measure performance of ML/DL models, used for solving IC testability problems, as implied by the literature?
2) **Robustness of Models:** Are approximate testability measures, used in state-of-the-art DL for IC test, adequate for models to meaningfully learn "IC testability"?

Paper Organization From our study, we identify future research perspectives and open challenges at the intersection of DL and IC test. In Section II we present some helpful background concepts. Readers familiar with DL principles and common ML evaluation metrics can skip this section. In Section III, we explore related work in ML for IC test. We present our motivating application in Section IV. We conduct robustness analyses in our case studies in Section V and Section VI, providing insights into our research questions. We present discussion in Section VII. Section VIII concludes.

A. B. Chowdhury, B. Tan, S. Garg, and R. Karri are with the Department of Electrical and Computer Engineering, New York University, Brooklyn, NY, 11201 USA. E-mail: {abc586, benjamin.tan, siddharth.garg, rkarri}@nyu.edu

II. Preliminaries

This section outlines some helpful background material. Readers familiar with these concepts may continue to Section III, where we focus on the intersection of ML and test.

A. Traditional ML Techniques

The field of ML broadly encapsulates the study of techniques for "training" a mathematical model on data to make predictions on new, unseen samples. Traditional ML techniques include decision trees [12], support vector machines (SVMs) [13], Bayesian networks [14], clustering [15] and so forth. These techniques require some level of feature engineering and domain expertise to appropriate map into ML problems. We encourage readers to peruse Bishop's work [16] for more details about these traditional techniques and metrics. Recently, researchers have explored techniques based on organizing "artificial neurons" into complex, "deep" architectures—such DNNs demonstrate state-of-the-art classification performance in domains such as image classification, natural language processing, and drug discovery.

B. Artificial Neural Networks (ANN)

(ANNs) are a collection of "neurons" and are regarded as universal function approximators [7]. This technique is inspired by the structure and functionalities of a biological neural network. An ANN architecture consists of one input layer, one (or more) hidden layers and an output layer[1]. Typically, for a feed-forward ANN, every node in a layer is connected to every other node in the next layer. At every node, the node inputs are multiplied with a "learned" weight. These weighted inputs are summed with an additional bias. Finally, the output is passed through a non-linear activation function (e.g., ReLU or sigmoid). These outputs are inputs to the next layer. The ANN can be mathematically described as:

$$\mathbf{y^k} = F\left(\sum_{i=0}^{m} \mathbf{w_i^k x_i} + \mathbf{b_i^k}\right) \quad (1)$$

- x_i is i^{th} input value of *m*-dimensional input vector.
- w_i^k is i^{th} weight value of *m*-dimensional weight vector, corresponding to k^{th} node of next layer.
- b_i^k is the bias term.
- ***F*** represents the activation function.
- y^k is the kth output node of next layer.

C. Graph Convolution Network (GCN)

There are many specialized architectures of DNNs, including convolutional neural networks (CNN) and recurrent neural networks (RNN), each tailored for a class of problems. The GCN [17] operates on graphs. A GCN performs two tasks in each hidden layer: aggregation, followed by encoding. In aggregation, each node collects information from its neighborhood and combine it with its own information. As hidden layers are added to the GCN, the k-hop neighborhood expands, aggregating neighborhood information into each node. In the encoding step, a non-linear activation function (e.g., ReLU) is applied to the inputs to produce the layer outputs. In the output layer, the encoded information for each node goes through a softmax layer for classification. The softmax layer squashes the scores of each node in the range of $[0, 1]$ which represents the probabilistic value of each class of classifier output.

Formally, the GCN takes as input a Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X})$, where $\mathbf{V}$ denotes nodes in the graph and $\mathbf{E}$ denotes the connectivity among nodes. $\mathbf{X}$ denotes node attributes, where $\mathbf{X} \in \mathbb{R}^{\mathbf{n}\times\mathbf{d}}$ is the feature matrix with each node v having feature $\mathbf{x}_v \in \mathbb{R}^{\mathbf{d}}$. Initially, the input layer information $\mathbf{h_v^0}$ are the node features itself (Eq. 2). Encoding of nodes at kth hidden layer is denoted by $\mathbf{h_v^k}$. The aggregation (Eq. 3) and encoding (Eq. 4) steps as defined by Kipf *et al.* [17] are:

$$\mathbf{h_v^0} = \mathbf{x_v} \quad (2)$$

$$\mathbf{g_v^k} = \sum_{\mathbf{u}\in\mathbf{N(v)}\bigcup\mathbf{v}} \frac{\mathbf{h_v^k}}{\sqrt{\mathbf{N(u)}}\sqrt{\mathbf{N(v)}}} \quad (3)$$

$$\mathbf{h_v^{k+1}} = \sigma(\mathbf{W_k} \times \mathbf{g_v^k}) \quad (4)$$

D. Typical Metrics for Evaluating Classification Performance

There are numerous metrics for evaluating the performance of a trained ML model. For binary classification problems:

Precision is the ratio of true-positive (TP) and predicted positive samples (true-positive + false-positive). Higher precision signifies fewer false-positives generated by the model.

$$\mathbf{Precision} = \mathbf{TP}/(\mathbf{TP} + \mathbf{FP}) \quad (5)$$

Recall is the ratio of true and actual positive samples (true-positive + false-negative). High recall indicates that most of the actual positive samples are correctly predicted as positive.

$$\mathbf{Recall} = \mathbf{TP}/(\mathbf{TP} + \mathbf{FN}) \quad (6)$$

F1 score is the harmonic mean of precision and recall and indicates a balance between the two. For an imbalanced dataset, lower F1 score may indicate poor precision or recall.

$$\mathbf{F1} = \mathbf{2TP}/(\mathbf{2TP} + \mathbf{FP} + \mathbf{FN}) \quad (7)$$

E. Non-linear Regression

Non-linear regression is a statistical analysis tool in which the relationship between variables is hypothesized to be non-linear. The hypothesis function $\mathbf{Y}$ is a non-linear function of independent variables or input features $\mathbf{x}$, represented as:

$$\mathbf{Y} = f(\mathbf{x}, \theta^*) + \epsilon \quad (8)$$

$\mathbf{x}$ is *d*-dimensional input vector. θ^* is *d*-dimension weight vector. ϵ is the bias term. $\boldsymbol{f}$ is non-linear function.

F. Metrics for Evaluating Regression Performance

For evaluating the performance of a non-linear regression model, it is important to understand how well the model fits the training dataset. We describe two well-known statistics typically used to measure goodness-of-fit for regression models.

[1] Often, a network with multiple hidden layers is referred to as a DNN.

Mean Squared Error (MSE) is the mean of the difference between predicted output values compared to the actual values of the observations, squared. A low MSE indicates that the model has fitted well on the training dataset, but does not preclude model over-fitting. It is important to do k-fold cross validation to generalize the regression model over test data.

$$\mathbf{MSE} = \frac{\mathbf{1}}{\mathbf{N}}\sum_{\mathbf{i=0}}^{\mathbf{N}}(\mathbf{y^{i}_{actual}} - \mathbf{y^{i}_{predicted}})^{\mathbf{2}} \tag{9}$$

G. k-fold Cross-validation

k-fold cross-validation is typically used to estimate average generalization error of an ML model. The data is initially divided into k-bins and training is performed using data from $k-1$ bins, and tested on the kth bin. The process is repeated k times; each time a new bin is considered as the test set. One of the key features of cross-validation is that each data sample is considered once as part of the test set and $k-1$ times as part of the training set—the prediction error is averaged over k times. The performance of cross-validation heavily depends on choice of K and set size α. The cross validation error (CVE) is typically computed as shown in Eq. 10.

$$\mathbf{CVE}(\alpha) = \frac{\mathbf{1}}{\mathbf{K}}\sum_{\mathbf{i=0}}^{\mathbf{K}}\mathbf{E_i}(\alpha) \tag{10}$$

III. Recent Advances in ML for IC Test

IC test researchers face long-standing problems in IC post-fabrication defect detection and diagnosis in large circuits, giving rise to design-for-test (DFT) and design-for-debug (DFD) architectures. Increased number of gates and complexity in an IC have led to scalability issues in using heuristic approaches for inserting DFT/DFD architectures along with increased time in entire design processes [9], [18], [19]. Hence, ML techniques have attracted interest in the IC test community, prompting application to NP class IC test problems. ML in IC test [20], [21] is nascent, motivating us to identify and seek insights into challenges that inform ongoing and future studies.

A. Machine Learning in Test and Diagnosis

For testability analysis and test point insertion (TPI), researchers have used a variety of ML-based techniques like non-linear classification and regression. The primary motivation for adoption is scalability and reducing time-bandwidth for DFT. In general, an ML model is first trained "offline" on previously acquired data, incurring a one-time training cost. In prior works [18], [19], [22], [23], authors propose classical ML approaches like SVMs and Naïve Bayes for chip and board-level functional fault diagnosis. The techniques use large scale volumetric failure log data to identify possible sites of chip failure. In scan-chain diagnosis, techniques include unsupervised learning for reasoning based diagnosis [24].

Pradhan *et al.* [19] implement non-linear regression by applying support vector regression (SVR) for estimating detectability loss in the presence of X-values. They formulate the problem of detectability loss estimation as a prediction problem. The input features to the model include structural features extracted from the directed acyclic graph (DAG) representation of the netlist. For model evaluation, they use the coefficient of determination as the metric to compare the predicted output versus actual output. It should be noted, however, that metrics like coefficient of determination are not always a good fit for evaluating non-linear regression models [25], and therefore, more caution is required for evaluating their efficacy. These works demonstrate that ML can be used as a drop-in replacement for heuristic-based solutions in IC testing and diagnosis problems.

B. Deep learning for IC Testability and Diagnosis

In recent years, techniques using ANNs (including DL) have emerged. In prior work [3], [4], researchers use ANNs to assist iterative TPI in circuits. The ANN model is trained to predict the improvement in fault coverage of a netlist when a node is chosen as a possible test point. The inputs to the ANN include COP [27], the gate type of the node itself, and its neighboring nodes. The results show that ANN-based techniques speed up TPI while yielding better coverage results when compared with TPI of conventional COP-based heuristics plus fault-simulation [29]. In another work [9], TPI is formulated as a node classification problem and solved using a graph convolution network (GCN) [17]. More specifically, the problem is formulated as a semi-supervised node classification in DAGs. The input features to the model are the logic level and the SCOAP [28] metrics for each node in the netlist. The nodes are labeled as *easy-to-observe* or *hard-to-observe*, based on the output of a commercial DFT tool. The authors show that the GCN-based approach surpasses the performance of the commercial DFT tool with respect to number of test points inserted with comparable coverage using fewer test patterns. Table I, summarizes ML solutions for TPI and diagnosis.

Although DL-based applications appear to out-perform conventional DFT-based heuristics, we observe that there are no intuitions as to why this is the case, especially as DL models are trained using labels provided by DFT tools (as their true or *ground-truth* values). Furthermore, the application of DL is new in IC test and diagnosis, and none of the aforementioned works consider problems of robustness, address wider considerations for model generalizations, or contextualize limitations of existing ML-metrics. We seek insights in these areas.

IV. Investigating State-of-the-art DL for TPI

We consider the problem of *circuit testability* as our motivating application for DL in IC test. The testability of an IC is defined as the *hardness* of *controllability* and *observability* of stuck-at faults (s-a-fs) at every node in a netlist. The motivation to adopt DL-based *TPI* over conventional heuristics+fault-simulation-based solutions is to reduce time for generating quality test points while being scalable to large circuits. TPI involves two parts: (1) identifying locations to insert test points and (2) evaluating the effect of a test point at a given location (to maximize coverage, minimize test points, or both).

We investigate approaches that use DL-based formulations of TPI for improving fault coverage of the netlist. The first

TABLE I
PRIOR WORK ON ML FOR IC TEST AND DIAGNOSIS

Prior work	Target Application	Publicly available benchmarks	Input Features	ML-model evaluation metric	Open-source Implementation
Pradhan *et al.* [19]	Detectability loss	✓	DAG topological features [26]	R2-score	✗
Sun & Millican [3] / Millican *et al.* [4]	TPI	✓	COP [27]	MSE	✓
Ma *et al.* [9]	TPI	✗	SCOAP [28]	F1 score	✗

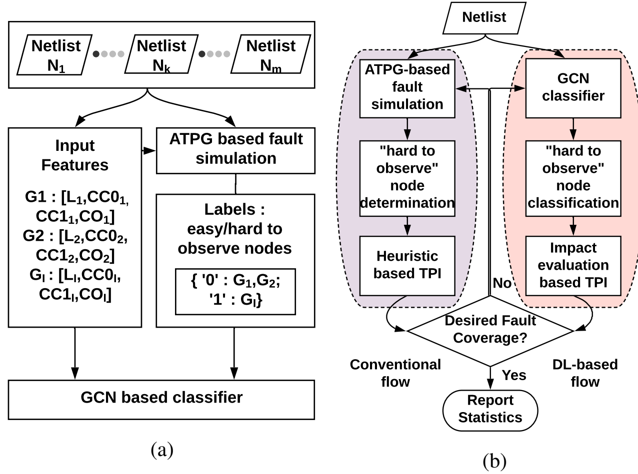


Fig. 1. DL-based (a) training flow, (b) classification for GCN-based TPI [9].

uses DL-based models for classification to identify "hard-to-observe" nodes as candidates for TPI. The second uses DL-based models for regression to predict the improvement in fault coverage given a candidate test point.

A. Case Study 1: Using DL for Classification

1) Problem Formulation: Our first case study is GCN-based TPI by Ma *et al.* [9]. Assuming a single s-a-f model, a netlist is considered highly testable if sufficient test vectors are available to excite s-a-f faults (i.e., *controllability*) and propagate the effect to any primary output (i.e., *observability*) for all nodes in the netlist.

Ma *et al.*'s formulation [9] focuses on observability improvement (this can be extended to consider controllability). The goal of the GCN is to identify nodes in the netlist where automatic test pattern generation (ATPG) algorithms fail to propagate the fault effect to primary outputs. Such nodes are classed as *hard-to-observe*. The objective of TPI is to add test points in the netlist and aid ATPG algorithms propagate s-a-f of *hard-to-observe* nodes to primary outputs.

Definition 1 (Testability as a Classification Problem)**.** *Given a set of netlists with nodes annotated as easy-to-observe or hard-to-observe, train a classifier that accurately predicts hard-to-observe nodes for a new netlist.*

2) DL-based Flow: We illustrate the DL-based classification and training flow in Fig. 1. Ma *et al.* [9] propose a multi-stage classification using GCNs as building blocks for inserting test points into a circuit. The GCN-based approach was chosen primarily for two main reasons: (1) GCNs are a natural match to a circuit netlist (given that a netlist is a DAG), and (2) the approach purportedly scales to large netlists.

Each node in the circuit is characterized using a four dimensional feature vector $[LL, CC0, CC1, CO]$, as shown in Fig. 1. LL is the logical level of a node in a circuit, while $CC0$, $CC1$ and CO are the SCOAP parameters for 0-controllability, 1-controllability, and observability of each node in the circuit, respectively. The ground-truth labels for each node ("hard-to-observe" or "easy-to-observe") are obtained from an industrial DFT tool. The GCN classifier consists of K-layers of aggregation and encodes node information up to the K-hop neighborhood for each node. This process produces a 128-dimensional feature vector for each node. These vectors are passed on to a fully-connected layer for classification.

To alleviate the problem of training on imbalanced datasets, Ma *et al.* propose a multi-stage classification approach, where GCNs are connected in series. A fraction of the nodes that are predicted as *easy-to-observe* (i.e., negative samples) with high confidence by the first GCN are "ignored" by GCNs in the subsequent stages. These negative samples are pruned over various stages and the *hard-to-observe* (positive samples) are reported as candidate locations for TPI. Test points are obtained iteratively by selecting the top-ranked hard-to-observe nodes where inserting a test point would provide the most impact in covering other hard-to-observe nodes in the netlist. Finally, the coverage improvement is measured after TPI at the "hard-to-observe" locations as identified by the GCN and compared against the coverage improvement after TPI performed by DFT tools.

B. Case Study 2: Using DL for Regression

1) Problem Formulation: One objective for TPI is to improve fault-detection capability when using random test vectors. This is explored in Sun and Millican's work [3], where an ANN is used to predict the fault coverage improvement of candidate nodes for TPI. This approach aims to avoid costly fault-simulations to ascertain the exact fault coverage improvement in every iteration.

Definition 2 (Testability as a Regression Problem)**.** *Given a set of netlists, randomly select a set of nodes. For each node, generate a sub-circuit and measure fault coverage improvement before and after inserting a test point at that node. Train a model to predict fault coverage improvement of a test point inserted at any given node in a netlist.*

2) DL-based Flow: We illustrate in Fig. 2 the ANN-based training process as proposed by Sun and Millican [3]. In

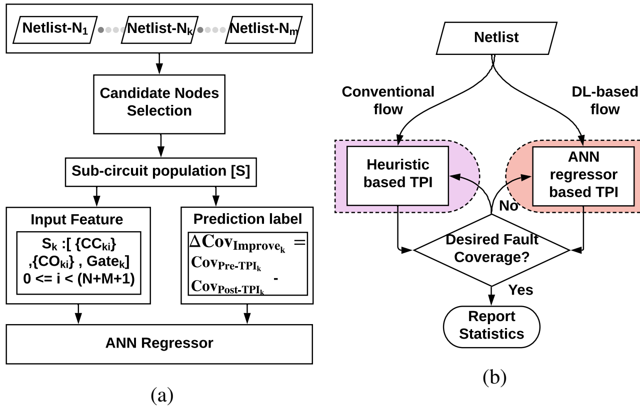


Fig. 2. (a) Training flow for ANN-based coverage change prediction and (b) DL-based regression, as proposed in prior work [3]

this flow, an ANN performs non-linear regression to predict improvement in the overall fault detectability of a netlist when test points are inserted at a given node in the netlist. The ANN model accepts an input of fixed dimension $2 \times (N + M) + 3$ representing a sub-circuit, where M and N are arbitrary values chosen by the designer. For nodes with fewer than M or N nodes in their fan-in/fan-out cone, the authors use COP controllability and observability values of primary inputs and outputs. During inference, the model iteratively predicts fault coverage improvement for a test point inserted at each node. This identifies the top k nodes where an inserted test point might improve fault coverage. Test points are inserted until some objective is met (e.g., coverage goal or number of test points).

To train the ANN, fault-simulation is performed on the netlists using random test vectors. Nodes are selected at random from the netlists as training data. Sub-circuits are generated using M and N nodes from a node's fan-out and fan-in cone, respectively. Fault coverage improvement of the sub-circuit is measured before and after inserting a test point and the process is repeated for each training node. As shown in Fig. 2(a), the input features of a node comprise COP controllability and observability parameters of nodes in its sub-circuit and its driving gate. Coverage is determined by checking the percentage of faults stimulated against a target fault list. Sun and Millican [3] report that their ANN-based TPI outperforms conventional COP-based TPI heuristic [29] in terms of random test pattern fault coverage improvement.

C. Identifying Limitations and Challenges

1) DL-model Robustness: Firstly, approximate testability measures like *SCOAP and COP* are the predominant input feature for DL-based TPI techniques [3], [4], [9]. However, when preparing inputs for DL-based flows, these techniques do not capture either the fault-simulation information (dynamic/runtime) of the entire circuit or precise Boolean value information flow (static/SAT-based encoding) from primary inputs to outputs in the DL input representation. These static and dynamic analyses of a circuit determine the optimal test points to be inserted. Although such analyses are time consuming on large circuits, (something that DL-based techniques seek to avoid) they capture the actual "hardness" of testability.

Conventional TPI and test generation algorithms like the D-algorithm, FAN, PODEM and HITEC/PROOFs rely on information flow, topological features, and fault-simulation of the network [30] *in addition* to approximate testability measures. In part, the use of complementary features (alongside SCOAP/COP) addresses decades-old critiques regarding the inaccuracies and coarseness of relying solely on SCOAP/COP testability metrics [31]. In fact, Savir [32] showed that good controllability and observability values do not necessarily guarantee good testability. As state-of-the-art DL-based techniques rely predominantly on approximate testability measures to learn and predict node testability, whether a model trained with these generalizes well needs investigation.

2) Robustness of Metrics for Model Quality: An important part of any ML-based workflow is evaluation of the trained model's *quality*; if the model appears to perform well in its intended classification/regression role, it can be deployed in the wider system. This has implications when trying to assess the suitability of a model. If a model makes quality predictions, decisions made on those predictions should produce good results in the overall objective.

Take the GCN-based approach [9] for identifying "hard-to-observe" nodes—a model that classifies such nodes accurately should inform proper TPI. In general, the prevailing *metric* for assessing the quality of ML models are classification accuracy and F1 score, and these are used as part of training (in formulating loss functions). However, when it comes to ML for IC test, we contend these metrics poorly reflect the quality of the final ML-model guided solutions. As discussed earlier, the solution space of TPI is NP-complete and no DFT tool guarantees an optimal TPI solution. In the GCN-based approach [9], the model is attempting to learn the heuristics of a particular DFT tool by training on data labels produced by that tool. In a sense, this is a *proxy objective* to the true objective of learning to improve testability. The accuracy and F1 score of TPI of the GCN model is *one* way to characterize the capability of the model to approximate the proprietary solution of a DFT tool.

For a new (unseen) netlist, used to validate a trained model, a model with good accuracy and F1 score indicates that it can replicate the heuristic. Conversely, a poor accuracy and F1 score should imply that the model is not doing well. As such, does a poor score mean that "bad" models offer no utility? As we will show in Section VI, the inability of a DL model to *exactly* replicate an industrial DFT heuristic does not guarantee that the ML-guided TPI results in poor/no improvement in fault coverage. This raises a broader question: are conventional metrics able to guide the model to learn "testability"? Since, the end objective of DL models is to improve fault coverage, the loss function of DL model should incorporate metrics that correlate well with fault coverage improvement of a netlist.

A different line of argument can be made in the case of the ANN-based approach [3]. As opposed to DFT-based heuristics for inserting test points, the ANN model estimates the effect of an inserted test point (as would be measured by true value fault-simulation); in regression, metrics such as MSE provide

the measure of prediction accuracy. In principle, because decisions are made on (estimates of) the impact of inserting a test point directly—as predicted by the model—ANN-based TPI might work better than DFT heuristics to identify test point locations and still avoid the cost of performing actual fault-simulation [3]. Thus, if one wants to use the predicted change in fault coverage to guide TPI, one would expect that good predictive performance, indicated by low MSE, is *necessary* for meaningful insertion. However, as we will see in Section VI, even where the MSE of a model is high, it turns out that one might still be able to use the ANN to select good locations for TPI. In other words, it is not necessary that the exact predicted change in fault coverage has to be accurate. Instead, a model that predicts fault coverage improvement with good *rank correlation*, as an alternative metric, provides a better guide for deciding which test points to insert.

V. Experimental Setup and Model Baseline

To explore the complexities and pitfalls of using DL for IC test, we design experiments to provide insight into the challenges discussed earlier. For each case study:

1) We analyze the baseline results after training models on public IC datasets using the approaches proposed in the literature. We investigate DL-based TPI and compare it with TPI performed by an industrial DFT tool.
2) We investigate robustness. To do this, we craft a redundant pattern that, once inserted into a netlist, should not affect the testability of the circuit, but throw-off the model (i.e., the pattern can be seen as analogous to an adversarial perturbation). We explore the effectiveness of our "attack" to draw insights into model robustness.

For these studies, we implement the multi-stage GCN [9] and ANN [3], described in Section IV.

Circuits We begin with benchmark circuits from the publicly available ISCAS, ITC, and EPFL benchmarks [33]–[36]. We obtain SCOAP/COP features and labels for the circuits using Mentor's Tessent 2019.1 DFT tool. We perform experiments on an Intel Xeon 3.2GHz processor, with 32 GB RAM and 12 GB NVIDIA GTX 1080 Ti GPU. All the benchmarks are fully scanned and synthesized using ABC [37]. We use the Cadence 90 nm library for synthesis. We use PyTorch to implement our GCN and ANN models. Table II presents general dataset statistics.

A. Case Study 1: GCN-based TPI

1) Experiment dataset: The objective of GCN-based TPI is to insert additional observation points in the netlist at "hard-to-observe" nodes. To prepare training data, we obtain ground-truth labels for the nodes in the public benchmark circuits using the industrial DFT tool. As the DFT tool reports that all s-a-f are covered without requiring additional test points for several benchmarks, we experiment only on those circuits which have *hard-to-observe* nodes. This set has 11 circuits: *log2, mem_ctrl, voter, div, arbiter, b21, b17, b18, s38584, b20 and b22*. This dataset has ~325,000 nodes.

TABLE II
Benchmark Characteristics

Circuit	Total Nodes	# POS Samples	# NEG Samples	Max. Level	Max. CC0	Max. CC1	Max. C0
log2	25970	339	25631	372	162118	160947	68861
mem_ctrl	49269	480	48789	115	182	291	334
voter	14712	137	14575	71	747965	741794	2218709
div	68924	758	68166	4373	1122	1218	26751
arbiter	12223	89	12134	89	255	10	586
b17	27814	300	27514	86	1739	1466	1640
b18	82528	824	81704	129	2416	2416	2799
b20	12489	159	12330	67	1046	1024	1386
b21	13073	175	12898	66	984	1071	1410
b22	19026	212	18814	66	1149	1163	1579
s38584	13894	124	13770	24	124	153	159

2) GCN architecture and training: We perform 3-stage classification using a GCN with K=3. For each classifier, the number of hidden layers, fully connected layers and their associated complexities are the same as those proposed by Ma *et al.* [9]. We use 300 epochs for end-to-end training, 100 epochs per GCN stage. As the dataset is highly imbalanced (the number of positive samples—hard-to-observe nodes—is $\sim$0.8% of the dataset), we use a weighted loss function to incur a greater penalty for misclassifying the minority class (i.e., *hard-to-observe* nodes). We set the weights based on the ratio of positive to negative samples in the dataset to ensure high recall of the output. We use the class weight ratio for negative to positive samples as 0.01, 0.03, and 0.05, for each of the three respective classifier stages. We use min-max normalization on our dataset as a pre-processing step. All other parameters are consistent with those reported in that paper.

3) GCN baseline performance: We use *k*-fold cross-validation, using nine circuits for training, one circuit for validation and the remaining one circuit for testing. We compare the quality of TPI results generated by DFT tool against those produced by the GCN-based prediction. The results of *k*-fold cross-validation, GCN classification, and attained coverage are reported in Table III. We show the true number of *hard-to-observe* (positive label) nodes (as reported by DFT tool), the total number of positive nodes as classified by the GCN model (in a single-shot query), the true-positive/negative node classifications (TP/TN) and false-positive/negative node classifications (FP/FN), for each circuit. The best F1 score is 0.48. Many circuits have low F1 scores between (0 – 0.48). As our train/test split is performed at the circuit-level, different train/test trials use different numbers of nodes. Although, the accuracy of the GCNs appears high ($\geq 94\%$ for the excluded circuits in a train/test trial), there is a wide range of F1 scores. We will dig into this apparent contradiction in Section VI.

B. Case Study 2: ANN-based TPI

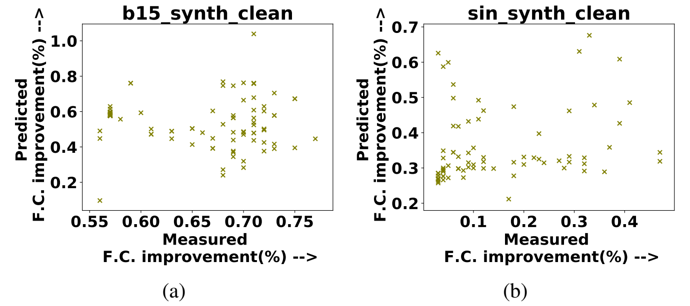
1) Experiment dataset: ANN-based TPI aims to improve the fault coverage of a netlist when using random test vectors. We use large public benchmarks on which the coverage using random vectors is comparably lower than that achieved by using test vectors from ATPG to more clearly characterize any effects on fault coverage. We use 10 circuits from the benchmark set: *arbiter, b17, b18, b19, b21, log2, mem_ctrl,*

TABLE III
CLASSIFICATION AND FAULT COVERAGE RESULTS USING DFT TOOL VS. GCN-BASED APPROACH. ORIG. REFERS TO THE GCN TRAINED WITH THE TRADITIONAL F1 SCORE, ALT. REFERS TO THE GCN TRAINED USING THE PROPOSED 1-HOP F1 SCORE. NB: F1 SCORE FOR ALT. IS 1 HOP F1 SCORE.

Circuit	# Hard-to-observe			# Nodes Classified								Overall Metrics				Fault Coverage using ATPG (in %)			
				True-Pos.		False-Pos.		True-Neg.		False-Neg.		Accuracy		F1 score		No TPI	DFT-TPI	GCN-TPI	
	DFT	Orig.	Alt.	Orig.	Alt.	Orig.	Alt.	Orig.	Alt.	Orig.	Alt.	Orig.	Alt.	Orig.	Alt.			Orig.	Alt.
b20	159	308	318	112	133	196	185	12134	12145	47	25	0.98	0.98	0.48	0.72	96.02	99.44	98.34	98.97
b22	212	430	400	153	165	277	235	18537	18558	59	47	0.98	0.99	0.48	0.74	94.87	99.45	97.25	98.64
log2	339	364	342	5	168	359	174	25272	25227	334	171	0.97	0.99	0.01	0.65	85.26	92.37	93.41	93.88
memctrl	480	514	462	69	274	445	288	48344	48401	411	206	0.98	0.99	0.14	0.71	95.71	98.39	98.41	98.65
voter	137	134	134	0	84	134	81	14541	14594	137	53	0.98	0.99	0	0.77	87.31	95.35	95.33	95.35
div	757	189	410	4	305	185	105	67982	68062	753	452	0.99	0.99	0.01	0.71	99.31	99.69	99.33	99.52
arbiter	119	66	107	7	49	59	58	12045	12046	112	80	0.94	0.98	0.08	0.52	99.8	100	99.91	99.96
b21	175	251	181	63	95	188	86	12710	12812	112	80	0.98	0.98	0.3	0.72	99.28	99.89	99.89	99.89
b17	300	571	571	45	177	526	451	26988	27063	255	123	0.97	0.98	0.1	0.47	98.14	99.37	99.12	99.25
b18	824	1653	1400	442	503	1211	897	80493	80807	382	321	0.98	0.98	0.36	0.58	99.24	99.59	99.65	99.75
s38584	124	192	203	12	73	180	130	13590	13640	112	51	0.98	0.98	0.08	0.57	98.19	98.53	98.24	98.5

TABLE IV
REGRESSION RESULTS USING THE ANN-BASED APPROACH. ORIG. REFERS TO ANN TRAINED USING MSE LOSSES, ALT. REFERS TO USE OF RANK CORRELATION FOR LOSSES.

Circuits	# TP	Valid. Loss		Test Loss		Fault coverage with random vectors(%)			
		Orig.	Alt.	Orig.	Alt.	No TPI	DFT-TPI	ANN-TPI	
								Orig.	Alt.
arbiter	119	0.085	0.32	0.271	0.45	20.39	49.49	33.67	42.43
b15	84	0.052	0.19	0.061	0.23	79.34	87.45	83.27	85.16
b17	278	0.025	0.2	0.168	0.32	76.8	86.23	81.35	85.98
b18	821	0.048	0.17	0.142	0.33	80.76	89.97	82.51	86.78
b19	1251	0.046	0.26	0.048	0.35	77.49	91.45	89.19	90.48
b21	130	0.029	0.24	0.125	0.28	87.76	95.54	91.35	93.12
log2	249	0.03	0.09	0.149	0.14	85.43	88.52	86.67	88.72
memctrl	327	0.049	0.25	0.157	0.31	54.42	78.25	70.08	75.52
sin	47	0.091	0.31	0.095	0.37	94.26	98.96	95.24	96.21
s38584	135	0.052	0.25	0.059	0.32	96.41	98.87	97.53	97.45



(a) (b)

Fig. 3. Scatter plot of fault coverage gain predicted by the ANNs

voter, sin, and s38584; on these circuits, the random vector fault coverage is at least, 2% lower than the ATPG-based fault coverage.

2) ANN architecture and training: We prepare the ANN model with input dimension of 101. We choose fan-in (N)/fan-out cone (M nodes as explained in Section IV-B) for all the netlists. All netlists are flattened and expanded with 2-input gates as in [3]. The ANN has an input layer, a hidden layer of 128 nodes and an output layer with a single node. We use MSE as the loss function and use Adam optimizer for model training. We use min-max normalization of the COP values as a preprocessing step. We use the sigmoid function as the activation function, as in [4]. We train the network for 100 epochs using k-fold cross-validation, using 8 netlists in training, 1 netlist for validation, and 1 netlist for test.

3) ANN baseline performance: We report the performance of the trained models in Table IV and Fig. 3, comparing the fault coverage improvement of the ANN to the TPI performed by the DFT tool using random test vectors. For a fair comparison, we make sure that the number of test points inserted are the same for both TPI mechanisms. The fault coverage achieved using ANN based TPI on the benchmarks is within 2-3% of the coverage using industrial DFT-based TPI. We obtain the k-fold cross-validation error of the models by averaging MSE on validation circuits and compare this to the MSE computed on querying the model with test circuits. The cross-validation loss ranges from 0.025 to 0.091.

VI. EXPERIMENTAL ROBUSTNESS STUDY AND INSIGHTS

A. Investigation I: A Closer Look at Baseline Behavior

As shown in Table III and Table IV, the various baseline models exhibit low F1 and MSE scores. At first sight, this suggests that the models, trained on the publicly available datasets, might not generalize well, and should thus lead to poor performance in guiding TPI. However, the fault coverage obtained by GCN and ANN-based TPI does improve coverage over the baseline and the results are comparable to the improved fault coverage from industrial DFT tool-based TPI. This raises questions about the validity of conventional ML metrics for assessing the quality of the models, at least in terms of their likely usefulness in informing TPI. To seek insights into this mismatch between ML metrics (accuracy, F1 score, MSE) and domain-specific test metrics (fault coverage improvement, number of test points inserted), we now examine the DL-based TPI in more detail.

1) Examining GCN-based TPI: Consider the GCN model evaluated on netlists *log2* and *voter* (Table III) where the (Orig.) F1 scores are almost 0. While the GCN model's output differs from "true" labels set by the DFT tool, the attained fault coverage is comparable (in fact, marginally better) to the DFT-based approach. We make two key observations:

TABLE V
ANALYSIS OF GCN-BASED CLASSIFICATION. TPR: TRUE-POSITIVE RATE.

Circuits	GCN-based Test Point Insertion					
	TPR	% of FP near FN			F1 Score	
		1-hop	2-hop	>2-hop	Orig.	1-hop
b20	0.704	58	12	30	0.48	0.70
b22	0.721	55	8	37	0.48	0.67
log2	0.014	67	12	21	**0.01**	**0.66**
mem_ctrl	0.144	52	22	26	**0.14**	**0.60**
voter	0	45	50	5	**0.0**	**0.44**
div	0.005	18	43	39	**0.01**	**0.07**
arbiter	0.058	41	22	37	**0.08**	**0.33**
b21	0.360	57	26	17	0.30	0.61
b17	0.150	62	11	27	**0.10**	**0.47**
b18	0.536	71	8	21	0.36	0.55
s38584	0.09	36	25	39	**0.08**	**0.49**

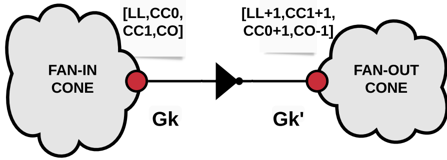


Fig. 4. Many false +ves in the vicinity of nodes maybe classed as false -ves.

- Accuracy and F1 score for model evaluation is misleading with respect to the model's usefulness in the *IC test application*. Even with low F1 scores, GCN-based TPI was able to insert useful test points, where the coverage boost is comparable to that achieved by the DFT tool.
- Most false-positive nodes classified by GCN are in 1- or 2-hop vicinity of true-positive and false-negative nodes.

In fact, we notice that where the GCN is "wrong" (i.e., it misses a desirable test point location or adds a test point somewhere else), most of the "incorrect" test points are within the vicinity of the *hard-to-observe* nodes. We report the density of false-positive (easy-to-observe classified as hard-to-observe) nodes in the neighborhood of false-negatives (hard-to-observe classified as easy-to-observe) in Table V.

Our experimentation reveals that over 60% of the GCN's false-positive classifications, on average, are within the 2-hop neighborhood of *hard-to-observe* nodes. From Table V, we observe that even though the F1 score of the model evaluated on *voter* is 0, 95% of the false-positive insertions were near the false-negative nodes. Adding test points where the GCN suggests improves the coverage from 87.31% to 95.33%, comparable with the coverage achieved by the DFT tool.

This suggests that the GCN is indeed learning *some* sense of *testability* based on local neighborhood information (as represented by the aggregation of SCOAP values). Additionally, while inspecting the TPI produced by GCN, we found that the presence of certain gates in the netlist, like buffers and inverters, do not impact testability measures and therefore, the feature representation of neighboring nodes are quite similar.

Consider the example in Fig. 4. We can see that faults at node G_k and G_{k+1} are equivalent faults (here, test vectors detecting *stuck-at-0* at G_k will also detect *stuck-at-1* at G_{k+1} and vice versa); the DFT tool labels G_k as a "hard-to-observe" node. Industrial DFT tools use fault collapsing to merge equivalent faults, outputting nodes from the list of collapsed faults which are "hard-to-observe". Therefore, DFT tools recommend inserting a test point at G_k. At present, the GCN is not trained to identify equivalent faults. It classifies both nodes as "hard-to-observe" since both nodes have similar controllability and observability values and are in the vicinity.

Insights: In this setting, when a model achieves a high F1 score, we can surmise that the model has learned to approximate the heuristic of a DFT tool to identify "hard-to-observe" nodes. Conversely, one might surmise that a model with a low F1 score has poor classification ability and hence TPI using such models yield negligible improvement in coverage. However, our investigation reveals that even with low F1 scores, we get comparable fault coverage with respect to a standard DFT tool. This is a conundrum: when training a set of models, how can one gauge that a model is fit-for-purpose without "trying it out" by employing costly fault-simulation to ascertain its performance?

The mismatch stems from how the testability problem is cast to a learning problem: identifying "hard-to-observe" nodes and inserting test points to improve fault coverage are different problems. While one can improve fault coverage by inserting test points at "hard-to-observe" nodes, test points can be inserted at other locations *near* "hard-to-observe" nodes to get comparable or better fault coverage, compared to DFT tools. There is a clear gap: the ML models are trained using loss functions for a proxy objective (approximating the DFT tool) not the primary objective (identifying the best test points to improve fault coverage). The fact that a model appears to perform **poorly on the task for which it has been trained** should raise skepticism as to whether it will work "well" in the broader problem.

If the F1 score is inadequate for evaluating a model, *how can one train and evaluate a model to better reflect the desired goal?* This is where domain expertise is useful. In this GCN-based classification setting, we note that, during training, the model should not be penalized for classifying a node as "hard-to-observe" in the vicinity of the (true) "hard-to-observe" nodes. Thus, one needs to devise a "*closeness metric*" which reflects that a test point within the "vicinity" of the hard-to-observe node can aid observability.

Exploring an alternative metric: We propose and investigate a new metric, the "1-hop F1 score" (Algorithm 1) and tune the binary cross-entropy loss function for training:

$$\mathbf{Loss} = \sum_{\mathbf{x}} \mathbf{y}(\mathbf{x}) \log \mathbf{G}(\mathbf{x}) + (\mathbf{1} - \mathbf{y}(\mathbf{x})) \log(\mathbf{1} - \mathbf{G}(\mathbf{x}))$$

$$y(x) = \begin{cases} 1, & \text{if } x \in \{\text{hard-to-obs.} \cup \text{1-hop neighbor}\} \\ 0, & \text{otherwise} \end{cases}$$

We train the GCN model using our proposed loss function and evaluate it using the 1-hop score. The results of these alternative models are shown in Table III (*Alt.* columns) and compared against the original models in Fig. 6. Our proposed

Algorithm 1: Relaxed 1-hop F1 score metric

Data: Netlist DAG N; Training Data : $X \in \mathbb{R}^{n \times d}, Y \in [0,1]^n$, Model parameters : θ, GCN model function : $G(\theta; x)$
Result: Relaxed 1-hop F1 score
$TP(N), FP(N), TN(N), FN(N) \leftarrow 0$;
for *node* $i \in N$ **do**
  if $G(\theta, x_i) == 1$ & $y_i == 1$ **then**
    $TP(N) \leftarrow TP(N) + 1$
  if $G(\theta, x_i) == 0$ & $y_i == 0$ **then**
    $TN(N) \leftarrow TN(N) + 1$
  if $G(\theta, x_i) == 1$ & $y_i == 0$ **then**
    $i_{1-hop} \leftarrow$predecessor(i) $\bigcup$ successor(i)
    if $\exists k$ *s. t.* $k \in i_{1-hop}$ & $y_k == 1$ **then**
      $TN(N) \leftarrow TN(N) + 1$
    else
      $FP(N) \leftarrow FP(N) + 1$
  if $G(\theta, x_i) == 0$ & $y_i == 1$ **then**
    $i_{1-hop} \leftarrow$predecessor(i) $\bigcup$ successor(i)
    if $\exists k$ *s. t.* $k \in i_{1-hop}$ & $G(\theta, x_k) == 1$ **then**
      $TP(N) \leftarrow TP(N) + 1$
    else
      $FN(N) \leftarrow FN(N) + 1$
  Return F1 score based on computed confusion matrix.

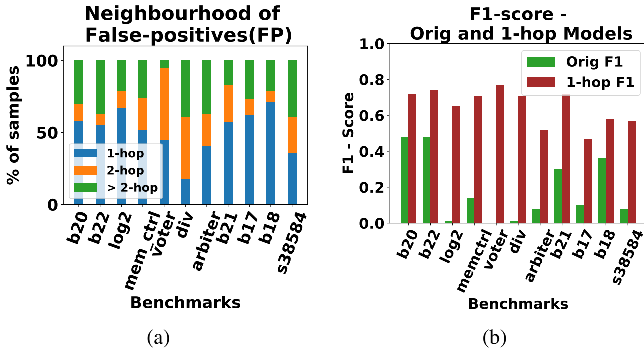


Fig. 5. (a) Percentage of false-positive nodes in the vicinity of false-negatives. (b) Model performance trained using traditional metrics v/s "1-hop F1 score".

metric better reflects the fault coverage improvement. During cross-validation, practitioners can use the 1-hop F1 score to select the best model. Furthermore, the 1-hop F1 score can used as part of early stopping criteria for training the model.

2) Examining ANN-based TPI: There are no well-established metrics to measure the "goodness-of-fit" for a non-linear regression model [38]. Instead, the MSE when predicting with test data via k-fold cross-validation error on training data is used as an *indicator* of model performance. Comparable MSE across validation data and test data suggests that a model fits well for the given data distribution and should work well on real (unseen) data. However, studies [38] have highlighted that a large MSE does not necessarily imply a bad model, especially when the dataset has outliers. As in the GCN case, ML metrics do not capture the whole story.

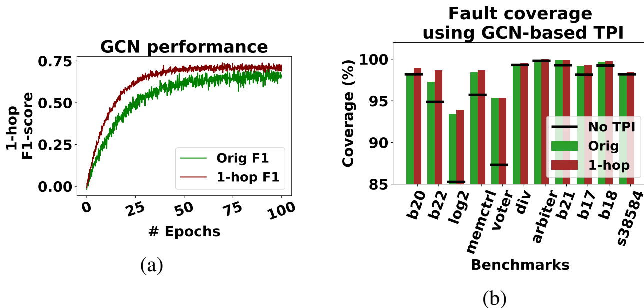


Fig. 6. (a) Performance of GCN-based TPI models trained using traditional metrics and "1-hop" F1 score (b) Fault coverage using GCN-based TPI models trained with traditional metrics and "1-hop" F1 score.

For instance, using the ANN-based approach, we obtain fault coverage comparable to the DFT tool-based TPI, for *log2* and *memctrl*, despite the MSEs being almost 3× the cross-validation loss. There is no correlation between the fault coverage obtained by the ANN-based TPI and model's MSE. This raises a serious concern: one cannot be sure to pick the "best" model using the MSE during cross-validation. There lies a possibility that picking the model with lowest MSE from cross-validation does not guarantee maximum fault coverage improvement on test data.

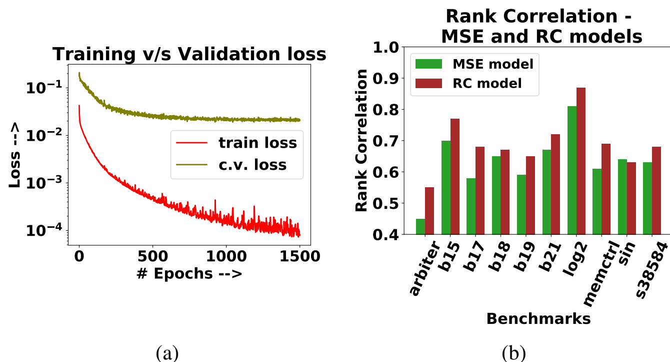
Insights: Fault coverage improvement depends on how the iterative TPI algorithm is guided by the predictions from the ANN. For this TPI problem formulation, instead of analysing the performance of the regression model using traditional metrics, we find that *rank correlation* is a more useful metric for evaluating the prediction quality. Consider a scenario where one model, $\mathcal{M}$ exhibits no errors in its predictions for 90% of the nodes while the other predictions have high errors. The overall MSE of $\mathcal{M}$ may be lower than another, $\mathcal{M}'$ whose predictions, while imperfect, correctly reflect the relative ordering of the nodes in terms of their impact on improving fault coverage. Even though the MSEs of the predicted scores are higher for $\mathcal{M}'$, the rank correlation between predicted and actual fault coverage improvement from the candidate nodes are close. In the iterative ANN-based TPI, $\mathcal{M}'$ guidance can provide higher fault coverage despite its higher MSE, as the algorithm uses a rank-based approach to insert test points.

Exploring an alternative metric: We use the *Spearman Ranking Correlation Coefficient* (ϕ) [39] to capture the correlation between the candidate-node list obtained from the ANN predicted fault coverage improvement and the actual improvement. This metric captures the monotonic improvement of bi-variate rank order variables and measures the correlation, where $\phi = +1$ denotes perfect association of ranks and $\phi = 0$ denotes no correlation [39]. A "strong correlation" ($0.6 < \phi < 1$) suggests that the iterative TPI will insert test points at nodes identified by ANN prediction in a rank order akin to the top *k* locations obtained using fault-simulation. The new loss function for the model is:

$$\mathbf{Loss_{rank-correlation}} = \mathbf{1} - \phi(\mathbf{R_{FC_P}}, \mathbf{R_{FC_A}})$$

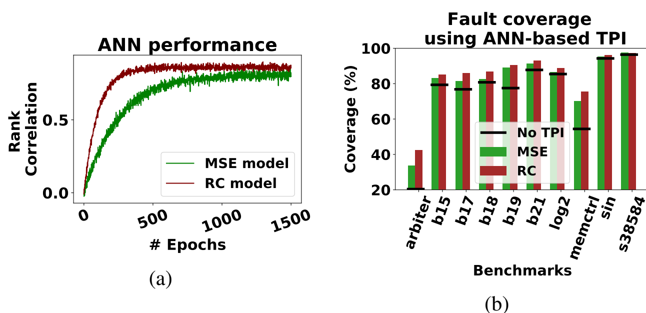
$$\text{where, } \phi(\text{Spearman correlation}) = \frac{cov(R_{FC_P}, R_{FC_A})}{\rho_{R_{FC_P}} \rho_{R_{FC_P}}}$$

R_{FC_P} denotes the rank order of the node based on fault coverage improvement predicted by the ANN, R_{FC_A} is the rank order based on fault coverage improvement. $Cov(.)$ and ρ denote co-variance and standard deviation values respectively. We train the ANN model using our proposed loss function and



(a) (b)

Fig. 7. (a) Training/validation loss while performing k-fold cross validation to train ANN regressor. (b) *Spearman Ranking Correlation* for ANN- v/s fault simulated top k rank locations with high fault coverage improvement.



(a) (b)

Fig. 8. (a) Performance of models with loss function using mean squared error (mse) and spearman rank correlation (rc) (b) Fault coverage using ANN-based TPI models trained with MSE and rc.

illustrate the *Spearman Correlation Coefficients* for both ANN models in Fig. 7. The model trained using rank correlation shows better rank correlation values than the model trained using MSE. In Fig. 8(a), we show that models trained with the rank correlation metric achieve better rank correlation compared to the MSE-based models in fewer training epochs, indicating that our proposed loss function leads to better models for TPI. Furthermore, Fig. 8(b) shows that the alternative model captures the likely fault coverage improvement better than the MSE-based original models. These results are shown in Table IV (Alt. columns). For comparing regression models targeted towards TPI insertion, rank correlation offers better insights and is more meaningful than MSE.

3) Discussion: From this investigation, it is clear that classical ML metrics and loss functions do not provide meaningful measures of how a trained model will perform in the IC test application; selecting models in the typical "ML" way is not a robust approach. For the GCN-based approach, while "good" classification scores suggest good approximations of industrial DFT-based TPI, "bad" scores do not provide much information on the quality of TPI. As DFT tools are heuristics-based, there might be numerous, disjoint solution sets (i.e., lists of nodes for TPI) which improve fault coverage. For the ANN-based approach, the prediction inaccuracy (as indicated by MSE) is less meaningful than the rank correlation. This raises the question: *are classical loss function and metrics adequate to ascertain the effectiveness of a model in the context of IC testability?* Our findings suggest: no.

The 1-hop F1 score and the Spearman Correlation show a better fit for exploring DL-based IC test problem formulations. Moving forward, the community should identify domain-informed metrics and standards to train and evaluate ML models, rather than using and reporting traditional metrics. The models should be trained and evaluated either directly on test-based metrics (fault coverage, number of test-patterns), with caveats well-explained, or use "tweaked" ML metrics which capture effectiveness of the model in improving testability. We encourage close scrutiny of models that appear to work "well" in a solution but appear to perform "poorly" on the task for which the model has been trained.

B. Investigation II: Probing the robustness of DL-based TPI

State-of-the-art DL-based TPI use approximate testability measures as input features. We explore robustness of the trained models, especially given that the approximate testability measures are unreliable in the presence of re-convergent fan-outs and redundant circuit structures [31]. The motivation for this is twofold: (1) robustness suggests that a model is learning meaningful concepts and is better equipped to handle "corner cases" and (2) fragility carries security concerns if the model acts as a "drop-in" replacement for fault-simulation (e.g., an adversary could sabotage the design by reducing fault coverage if they can manipulate the ML-based TPI to omit test points or provide an inflated estimate of testability).

To evaluate robustness, we devise a test using a *Redundant re-convergent fan-out* pattern (RRF) pattern that, when inserted, does not change functionality or testability. A robust model should be able to accommodate the pattern (as DFT tools can). The RRF pattern is akin to a "semantically meaningful perturbation" [11] as it perturbs the approximate testability values in the neighborhood and fan-out cone.

While security is not a main focus of this work, one can frame our investigation as seeing if an adversary can use the RRF pattern to coax the model into overlooking nodes that are "hard-to-observe" as a form of sabotage. A change in functionality is too easily caught, so the RRF pattern must somehow exploit the ML model so that it overestimates testability. To design the pattern, we first observe that TPI near the primary inputs/outputs are very rare. Thus, we design the RRF so that the feature values resemble those of a primary input's neighborhood. Other "adversarial" patterns can be designed using internal nodes of the netlist. Our evaluation shows that the insertion of RRF patterns ($\sim$5 gates) in a netlist of $\geq$ 20,000 nodes does not change the parameters like path delay, power consumed and area.

1) RRF Pattern Insertion: We start with a DAG of a circuit—as in Fig. 9(a)—and use an internal node and a primary input to create an RRF. We identify a node where the trained DL model would normally insert a test point. To create the RRF pattern, we choose the internal node $G7$ and

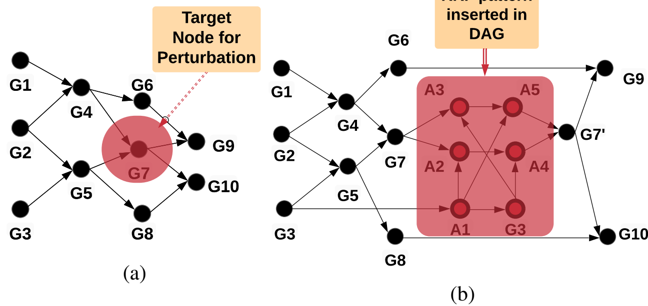


Fig. 9. (a) G7 is chosen for perturbation. (b) DAG after RRF insertion.

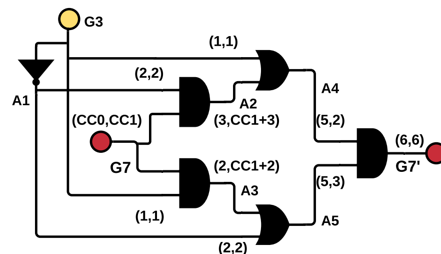


Fig. 10. An Example Redundant Re-convergent Fanout (RRF) Pattern

primary input $G3$ (Fig. 9(a)). The output at $G7'$ is $G7$:

$$A1 = \neg G3 \qquad A2 = A1 \wedge G7 \qquad A3 = G3 \wedge G7$$
$$A4 = G3 \vee A2 \qquad A5 = A1 \vee A3 \qquad G7' = A4 \wedge A5$$
$$G7' = (G3 \vee ((\neg G3) \wedge G7)) \wedge (\neg G3 \vee (G3 \wedge G7))$$
$$= (G3 \vee G7) \wedge (\neg G3 \vee G7) = G7$$

To make this perturbation, any primary input can be considered. We create a set of RRF patterns (an example is shown in Fig. 10). After RRF insertion, we use the DL tool to analyze the netlist and see if it fails to insert a test point at the node-under-attack. If the DL model is robust and properly incorporates "neighborhood" knowledge (e.g., k-hop neighbors for classification or the fan-in/out for regression), these changes should be "ignored".

Consider Fig. 10. Node $G7$ with SCOAP controllability values ($CC0$,$CC1$). $G3$ has SCOAP values (1,1) as it is a primary input. The SCOAP values of $A2$ and $A3$ become (3,C1+3) and (2,C1+2) respectively as the *0-controllability* value is dominated by primary input and AND gates. Similarly, at $A4$ and $A5$, we obtain SCOAP values of (5,2) and (5,3) due to *1-controllability* domination by primary inputs. Finally, $G7'$'s SCOAP value is (6,6). Therefore, for any internal node, introducing a redundant pattern changes the SCOAP values to (6,6) without improving testability, since $G7'$ has no contribution from a primary input. Fig. 9(b) represents the modified DAG after insertion of an RRF pattern. DFT tools identify the RRF pattern as redundant, and thus behave appropriately by adding observation points as required.

2) Robustness of the GCN approach: We report GCN classification performance on *clean* test circuits in Table VI. To test robustness, we take a test circuit and create two variants. The first variant (targeted) has RRF patterns at five random *hard-to-observe* locations determined by the DFT tool. In the second variant (random), the RRFs are inserted in five random locations. For clean and variant circuits, we use DFT tool-based and GCN-based TPI to find test point locations, measuring and comparing the resulting fault coverage.

As shown in Fig. 11, TPI in netlists with RRF patterns has reduced fault coverage. Inserting RRF patterns at the (mispredicted) "hard-to-observe" nodes negligibly improves the fault coverage over the original circuit. For RRF patterns inserted at random locations, there is up to 50% degradation in fault coverage compared to clean circuits. This degradation

TABLE VI
FAULT COVERAGE AFTER GCN-BASED TPI FOR CIRCUITS WITH/WITHOUT RRFS. C: CLEAN, T: TARGETED VARIANT, R: RANDOM VARIANT

Circuit	1-hop F1			Fault Coverage (%)			Improvement (δ%)		
	C	T	R	C	T	R	C	T	R
b20	0.72	0.46	0.59	98.97	96.78	97.34	2.95	0.76	1.32
b22	0.74	0.51	0.6	98.64	95.68	96.72	3.77	0.81	1.85
log2	0.65	0.42	0.56	93.88	87.04	89.51	8.62	1.78	4.25
mem_ctrl	0.71	0.39	0.58	98.65	96.33	97.1	2.94	0.62	1.39
voter	0.77	0.36	0.52	95.35	89.36	90.53	8.04	2.05	3.22
div	0.71	0.29	0.35	99.52	99.33	99.42	0.21	0.02	0.11
arbiter	0.52	0.15	0.32	99.96	99.84	99.92	0.16	0.04	0.12
b21	0.72	0.31	0.5	99.89	99.39	99.56	0.61	0.11	0.28
b17	0.47	0.19	0.31	99.25	98.43	98.75	1.11	0.29	0.61
b18	0.58	0.19	0.39	99.75	99.42	99.6	0.51	0.18	0.36
s38584	0.57	0.29	0.41	98.5	98.31	98.39	0.31	0.12	0.2

occurs as "hard-to-observe" nodes in the neighborhood of the RRF pattern are "missed" by the fragile DL model. RRF patterns drastically drop the 1-hop F1 scores, pointing to a reduction in TPIs near desirable nodes. From these results, we find that the GCN-based approach is not robust.

3) Robustness of the ANN-based approach: We insert RRF patterns randomly at $< 1\%$ of the locations in a netlist to see if the ANN predictions are robust to these insertions. We use DFT tool-based and ANN-based TPI to insert observation points on clean and RRF pattern-inserted variants. We generate 10K random test patterns and measure fault coverage of the circuits. From Fig. 12, the trends are in line with results of our GCN investigation. The industrial DFT tool identifies redundant logic sub-circuits. The coverage improvement reported by applying 10K random vectors pre- and post-insertion of RRFs

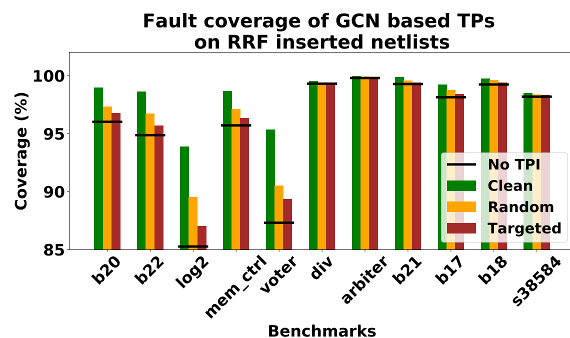


Fig. 11. ATPG fault coverage degradation post insertion of RRF patterns using GCN-based TPI on our benchmark netlists.

TABLE VII
ANN-BASED TPI PERFORMANCE WITH RRF PATTERNS

Benchmarks	Number of TPs	DFT based TPI fault coverage(%)		ANN based TPI fault coverage(%)	
		clean	perturbed	clean	perturbed
arbiter	119	49.49	49.49	42.43	28.47
b15	84	87.45	87.45	85.16	79.45
b17	278	86.23	86.23	85.98	80.46
b18	821	89.97	89.97	86.78	83.31
b19	1251	91.45	91.45	90.48	79.98
b21	130	95.54	95.54	93.12	91.18
log2	249	88.52	88.52	88.72	85.89
memctrl	327	78.25	78.25	75.52	62.42
sin	47	98.96	98.96	96.21	96.05
s38584	135	98.87	98.87	97.25	96.48

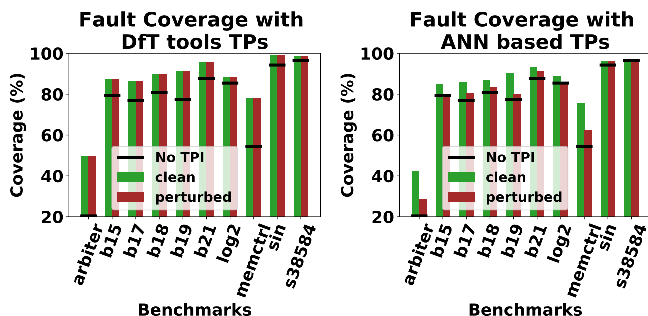


Fig. 12. Fault coverage degradation using 10K random test-patterns post insertion of RRF patterns using ANN-based TPI. Fault coverage on DFT based TPI (left), ANN based TPI (right)

remains almost the same. In contrast, there is a drastic fall in fault coverage improvement of the ANN-based TPI when RRFs are present, as shown in Table VII. Clearly, the ANN-based approach is also not robust.

VII. DISCUSSION

A. Further Insights into Robustness

In both problem settings, our trained models perform poorly on netlists with RRF patterns. To improve robustness, we use data augmentation and retraining by including RRF pattern-added variants into the training data and training new models. Table VIII reports the fault coverage improvement when performing TPI with the new models. As can be seen in Fig. 13, TPI with the new models results in better fault coverage.

The degradation of the fault coverage achieved following DL-based TPI in the presence of RRFs provides interesting avenues for exploration of robustness. In a sense, the RRFs can be seen to exploit a structural bias, in that nodes near primary inputs are generally "easier" to test; by manipulating SCOAP values to resemble such nodes, we are able to confuse the ML models. Thus, one should analyse circuits for structural biases before training models and redress these in some way. DL-based techniques often pick up inherent biases in training data and perform poorly when such biases are absent in the test data. Recent work [11], [40] demonstrates attacks on DL models through "poisoning" the training dataset using meaningful bias/patterns, pointing to proactive augmentation of the

TABLE VIII
FAULT COVERAGE AFTER GCN-BASED TPI (TRAINED WITH RRF PATTERNS). C: CLEAN, T: TARGETED VARIANT, R: RANDOM VARIANT

Circuit	1-hop F1			Fault Coverage (%)			Improvement ($\delta\%$)		
	C	T	R	C	T	R	C	T	R
b20	0.67	0.65	0.65	98.75	98.29	98.29	2.71	2.27	2.27
b22	0.69	0.67	0.68	98.33	98.19	98.19	3.46	3.32	3.32
log2	0.59	0.58	0.59	91.89	91.89	91.89	6.03	6.03	6.03
mem_ctrl	0.68	0.64	0.65	98.32	98.14	98.14	2.61	2.43	2.43
voter	0.71	0.69	0.69	93.97	93.78	93.78	6.66	6.47	6.47
div	0.62	0.62	0.62	99.49	99.49	99.49	0.18	0.18	0.18
arbiter	0.52	0.52	0.52	99.96	99.96	99.96	0.16	0.16	0.16
b21	0.63	0.63	0.61	99.77	99.77	99.72	0.49	0.49	0.44
b17	0.44	0.44	0.4	99.2	99.2	99.15	1.06	1.06	1.01
b18	0.55	0.55	0.55	99.71	99.71	99.71	0.47	0.47	0.47
s38584	0.52	0.52	0.52	98.47	98.47	98.47	0.28	0.28	0.28

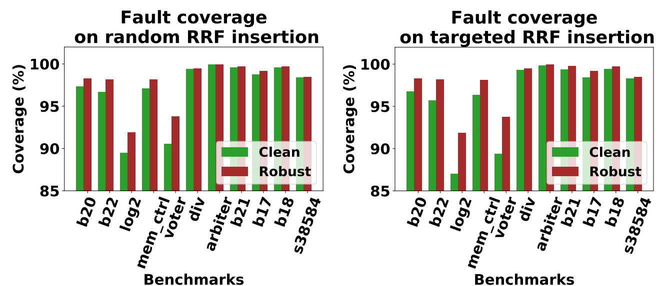


Fig. 13. Fault coverage obtained using TPI from clean network and adversarially trained robust network post insertion of RRF patterns. Fault coverage on randomly insertion in netlists(left), targeted insertion in netlists (right)

dataset, as in recent work [41]. However, more fundamentally, the effectiveness of inserting RRF patterns shows that testability metrics like SCOAP should be complemented with other, *reliable* features, perhaps explicit structural features.

B. Insights into the Dataset

As shown in Section V, cross-validation of the GCN classifiers resulted in wide-ranging F1 scores. To understand the reasons behind this, we take a closer look at the dataset distribution. Typically, a well trained model performs best on data of the same distribution as that on which it was trained. There are different ways in which on might consider circuits to be "similar", such as topological similarity or functional similarity. There is no study which quantifies the distribution of data-points sampled from the various netlists.

Given the role of SCOAP in the ML approaches we studied, we plot statistical information to show the variation of node characteristics within and across circuits in Fig. 14. $CC1$ and CO vary by orders of magnitude across circuits. While *log2* and *div* have similar SCOAP parameter distributions, they are outliers relative to the others. The distribution of SCOAP values for *easy-to-observe* and *hard-to-observe* nodes lies in a similar range for all b-series circuits. When we examine the models trained on the b-series circuits and tested on b-series, the F1 score is higher than where the model was trained on the b-series circuits and tested on the "outliers" (i.e., F1 scores when testing on b-series circuits are in the range 0.1–0.48 and when testing on outliers, ~ 0). From this we can infer that the

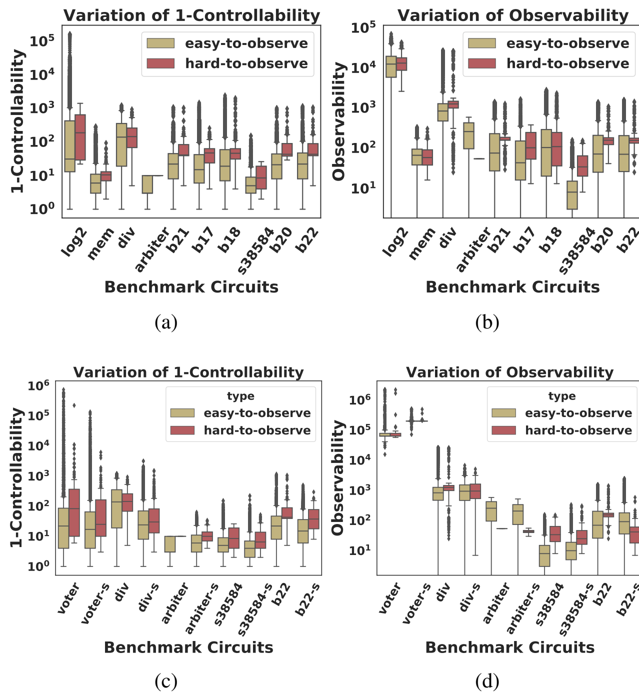


Fig. 14. (a) Distribution of SCOAP 1-controllability, (b) Distribution of observability, (c) Side-by-side plot of SCOAP-1 controllability with siblings from [26] and (d) Side-by-side plot of observability with siblings from [26].

ML models perform better on "sibling" circuits with similar node feature distributions.

Prior work [26] produces circuit variants of the ITC'99 and EPFL benchmarks by randomly substituting gates. We plot the feature distributions of each circuits alongside an example "sibling" circuit in Fig. 14(c) and (d). As the node feature distributions resemble the original circuits, we expect that the ML models will perform similarly on the siblings. This insight can guide practitioners to check if the distribution of the training data resembles the test data.

C. Study Limitations

Our evaluation relies on labels obtained from a commercial tool flow (Mentor Tessent ATPG tool, ABC logic synthesis tool and Cadence technology library). The logic synthesis tool plays a key role in determining the network structure and the testability of various nodes. As in Ma et al.'s work [9], we use a DFT tool's outputs as the ground truth for classification. However, we note that different DFT tools will produce different labels for nodes given the same circuit because different tools use different heuristics to guide TPI. A motivation for adopting ML-based approaches is the promise of achieving useful fault coverage improvement while avoiding expensive simulation costs. From a practicality standpoint, DFT tools provide an adequate "gold standard" to emulate; ML-based TPI that improves coverage at least as good as a DFT tool demonstrates viability. The comparison of the ML-based approaches to industrial DFT tools provide a useful empirical context for how well ML-based approaches can work. As such, our findings support the notion that ML can achieve comparable results to industrial tools and achieve improved results after adopting our proposed metrics. In reality, the truly "hard-to-observe" nodes, as identified using fault-simulation might not be the same as those determined as "hard-to-observe" by DFT tools. This is why care should be taken to define exactly what an ML model is trying to predict. In experiments, we report fault coverage after TPI as measured by fault-simulation *alongside* the other ML metrics to capture how the models perform, noting that our chosen DFT tool is only one of many. Also, hyper-parameter settings play a key role in the attainable model quality[2]. While our experiments use GCNs and ANNs, our insights into approximate testability measures and dataset limitations extend to other ML approaches.

VIII. Conclusions and Future Directions

From a critical exploration of the state-of-the-art in using DL for IC test, we identified two areas that required further investigation: metric robustness and model robustness. We studied the behavior of trained GCN and ANN models and found that existing ML-based evaluation metrics fail to capture scenarios where a model's outputs may be useful for testing while ostensibly performing poorly in the task on which the model was trained. We also found that robustness of DL models is lacking, particularly under "adversarial" settings with redundant circuitry able to cause the models to predict poorly. Drawing from these insights, future work includes deeper examination and evaluation of ML models and investigations into design abstractions and features that can complement or supplant approximate testability measures like SCOAP as inputs for increased robustness as well as comparisons of different models/architectures on common objectives/benchmarks. Ideally, the community should coordinate an effort to create metrics to assess dataset quality alongside open-sourcing implementations. Our work should not be taken as a narrow critique of DL-based techniques for IC test, nor as a general critique of ML. Instead, this work provides insights that will assist advancements in this field.

Acknowledgement

This work was supported in part in part by NSF grant # 1526405 and # 1801495, ONR grant # N00014-18-1-2058, NSF career award and NYU/NYUAD CCS. B. Tan and R. Karri are supported in part by ONR Award # N00014-18-1-2058. S. Garg is supported in part by an NSF CAREER Award and NSF Award # 1801495. R. Karri is supported in part by the NYU/NYUAD CCS.

References

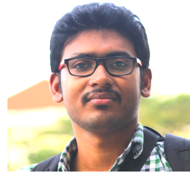
[1] R. Pan, Z. Zhang, X. Li, K. Chakrabarty, and X. Gu, "Black-Box Test-Coverage Analysis and Test-Cost Reduction Based on a Bayesian Network Model," in *IEEE VLSI Test Symposium*, 2019.

[2] H. Dhotre, S. Eggersglüß, K. Chakrabarty, and R. Drechsler, "Machine Learning-based Prediction of Test Power," in *IEEE European Test Symposium*, 2019, pp. 1–6.

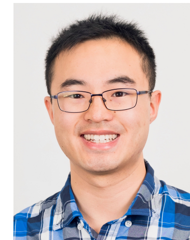
[3] Y. Sun and S. Millican, "Test Point Insertion Using Artificial Neural Networks," in *IEEE Computer Society Symp. on VLSI*, 2019.

[2] to aid reproducibility, we will release our implementation.

[4] S. K. Millican, Y. Sun, S. Roy, and V. D. Agrawal, “Applying Neural Networks to Delay Fault Testing: Test Point Insertion and Random Circuit Training,” in *IEEE Asian Test Symposium*, 2019.
[5] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, “Layout Hotspot Detection With Feature Tensor Generation and Deep Biased Learning,” *IEEE Transactions on CAD.*, vol. 38, no. 6, pp. 1175–1187, Jun. 2019.
[6] A. F. Tabrizi, N. K. Darav, L. Rakai, I. Bustany, A. Kennings, and L. Behjat, “Eh?Predictor: A Deep Learning Framework to Identify Detailed Routing Short Violations from a Placed Netlist,” *IEEE Transactions on CAD.*, pp. 1–1, 2019.
[7] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, 1989.
[8] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. S. Dickstein, “On the expressive power of deep neural networks,” in *Int. Conf. on Machine Learning*, 2017, pp. 2847–2854.
[9] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, “High Performance Graph Convolutional Networks with Applications in Testability Analysis,” in *IEEE/ACM Design Automation Conference*, 2019.
[10] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, “Robust physical-world attacks on deep learning models,” *arXiv preprint arXiv:1707.08945*, 2017.
[11] K. Liu, H. Yang, Y. Ma, B. Tan, B. Yu, E. F. Y. Young, R. Karri, and S. Garg, “Adversarial Perturbation Attacks on ML-Based CAD: A Case Study on CNN-Based Lithographic Hotspot Detection,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 5, Aug. 2020.
[12] J. E. Nelson, W. C. Tam, and R. D. Blanton, “Automatic classification of bridge defects,” in *IEEE International Test Conference*, 2010, pp. 1–10.
[13] Z. Li, J. E. Colburn, V. Pagalone, K. Narayanun, and K. Chakrabarty, “Test-cost optimization in a scan-compression architecture using support-vector regression,” in *IEEE VLSI Test Symposium*, 2017.
[14] L. R. Gómez, A. Cook, T. Indlekofer, S. Hellebrand, and H.-J. Wunderlich, “Adaptive bayesian diagnosis of intermittent faults,” *Journal of Electronic Testing*, vol. 30, no. 5, pp. 527–540, 2014.
[15] N. Sumikawa, M. Nero, and L.-C. Wang, “Kernel based clustering for quality improvement and excursion detection,” in *IEEE International Test Conference*, 2017, pp. 1–10.
[16] C. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. Springer-Verlag, 2006.
[17] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
[18] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu, “Board-level functional fault diagnosis using multikernel support vector machines and incremental learning,” *IEEE Transactions on CAD.*, 2014.
[19] M. Pradhan, B. B. Bhattacharya, K. Chakrabarty, and B. B. Bhattacharya, “Predicting X-Sensitivity of Circuit-Inputs on Test-Coverage: A Machine-Learning Approach,” *IEEE Transactions on CAD.*, 2018.
[20] H.-G. Stratigopoulos, “Machine learning applications in ic testing,” in *IEEE European Test Symposium*, 2018, pp. 1–10.
[21] M. Pradhan and B. B. Bhattacharya, “A survey of digital circuit testing in the light of machine learning,” *WIREs Data Mining and Knowledge Discovery*, vol. n/a, no. n/a, p. e1360.
[22] S. Jin, F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu, “Efficient Board-Level Functional Fault Diagnosis With Missing Syndromes,” *IEEE Transactions on CAD.*, pp. 985–998, June 2016.
[23] S. Mittal and R. S. Blanton, “Learnx: A hybrid deterministic-statistical defect diagnosis methodology,” in *IEEE European Test Symposium*, 2019, pp. 1–6.
[24] Y. Huang, B. Benware, R. Klingenberg, H. Tang, J. Dsouza, and W.-T. Cheng, “Scan Chain Diagnosis Based on Unsupervised Machine Learning,” *IEEE Asian Test Symposium*, pp. 225–230, 2017.
[25] A.-N. Spiess and N. Neumeyer, “An evaluation of r 2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a monte carlo approach,” *BMC pharmacology*, vol. 10, no. 1, p. 6, 2010.
[26] M. N. Mondal, A. B. Chowdhury, M. Pradhan, S. Sur-Kolay, and B. B. Bhattacharya, “Fault Coverage of a Test Set on Structure-Preserving Siblings of a Circuit-Under-Test,” in *IEEE Asian Test Symposium*, 2019.
[27] F. Brglez, “On Testability Analysis of Combinational Networks,” in *IEEE Int. Symp. Circuits and Systems*, 1984.
[28] L. H. Goldstein and E. L. Thigpen, “SCOAP: Sandia controllability/observability analysis program,” in *Design Automation Conf.*, 1980.
[29] H.-C. Tsai, C.-J. Lin, S. Bhawmik, and K.-T. Cheng, “A hybrid algorithm for test point selection for scan-based bist,” in *IEEE/ACM Design Automation Conference*, 1997, pp. 478–483.
[30] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer, 2004, vol. 17.
[31] L. M. Huisman, “The reliability of approximate testability measures,” *IEEE Des. Test. Comput.*, 1988.
[32] J. Savir, “Good controllability and observability do not guarantee good testability,” *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1198–1200, 1983.
[33] D. Bryan, “The ISCAS’85 benchmark circuits and netlist format,” *North Carolina State University*, vol. 25, p. 39, 1985.
[34] F. Brglez, D. Bryan, and K. Kozminski, “Notes on the ISCAS’89 Benchmark Circuits,” *MCNC*, 1989.
[35] F. Corno, M. S. Reorda, and G. Squillero, “RT-level ITC’99 benchmarks and first ATPG results,” *IEEE Des. Test. Comput.*, 2000.
[36] L. Amarú, P.-E. Gaillardon, and G. De Micheli, “The EPFL combinational benchmark suite,” in *Int. Workshop on Logic Synthesis*, 2015.
[37] R. Brayton and A. Mishchenko, “ABC: An academic industrial-strength verification tool,” in *Int. Conf. on Computer Aided Verification*, 2010.
[38] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. John wiley & sons, 2005, vol. 589.
[39] C. Spearman, “The proof and measurement of association between two things.” 1961.
[40] T. Gu, B. Dolan-Gavitt, and S. Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” *arXiv preprint arXiv:1708.06733*, 2017.
[41] K. Liu, B. Tan, G. R. Reddy, S. Garg, Y. Makris, and R. Karri, “Bias Busters: Robustifying DL-based Lithographic Hotspot Detectors Against Backdooring Attacks,” *arXiv:2004.12492 [cs, stat]*, Apr. 2020.



Animesh Basak Chowdhury received his M.Tech. in Computer Science from Indian Statistical Institute in 2016. Currently, he is a doctoral candidate at the NYU Centre for Cybersecurity. His research interests include Secure Electronics Design Automation (EDA) and Deep Learning. Prior to joining the Ph.D. program, he spent three years as a researcher at Tata Research Development and Design Centre (TRDDC), India.



Benjamin Tan (S’13-M’18) received the BE(Hons) degree in Computer Systems Engineering, in 2014, and the Ph.D. degree, in 2019, both from the University of Auckland. Since 2019, he has been a Research Assistant Professor working in the NYU Center for Cybersecurity, New York University, NY, USA. His research interests include hardware security, electronic design automation, and machine learning. He is a member of the IEEE and ACM.

Siddharth Garg received the B.Tech. degree in electrical engineering from IIT Madras and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, in 2009. He was an Assistant Professor with the University of Waterloo, from 2010 to 2014. In 2014, he joined New York University (NYU) as an Assistant Professor. His general research interest includes computer engineering, more particularly secure, reliable, and energy-efficient computing. He was a recipient of the NSF CAREER Award, in 2015.

Ramesh Karri (SM’11-F’20) received the B.E. degree in electrical and computer engineering from Andhra University, Visakhapatnam, India, in 1985, and the Ph.D. degree in computer science and engineering from the University of California San Diego, San Diego, CA, USA, in 1993. He is currently a Professor of Electrical and Computer Engineering with New York University (NYU), Brooklyn, NY, USA. He co-directs the NYU Center for Cyber Security. He has authored or coauthored more than 240 articles in leading journals and conference proceedings. His current research interests include hardware cybersecurity include trustworthy ICs; processors and cyberphysical systems; security-aware computer-aided design, test, verification, validation, and reliability; nano meets security; hardware security competitions, benchmarks and metrics; biochip security; and additive manufacturing security.