Explaining and Interpreting Machine Learning CAD Decisions: An IC Testing Case Study

Prashanth Krishnamurthy
prashanth.krishnamurthy@nyu.edu
New York University
Brooklyn, USA

Animesh Basak Chowdhury
abc586@nyu.edu
New York University
Brooklyn, USA

Benjamin Tan
benjamin.tan@nyu.edu
New York University
Brooklyn, USA

Farshad Khorrami
khorrami@nyu.edu
New York University
Brooklyn, USA

Ramesh Karri
rkarri@nyu.edu
New York University
Brooklyn, USA

ABSTRACT

We provide a methodology to explain and interpret machine learning decisions in Computer-Aided Design (CAD) flows. We demonstrate the efficacy of the methodology to the VLSI testing case. Such a tool will provide designers with insight into the "black box" machine learning models/classifiers through human readable sentences based on normally understood design rules or new design rules. The methodology builds on an intrinsically explainable, rule-based ML framework, called Sentences in Feature Subsets (SiFS), to mine human readable decision rules from empirical data sets. SiFS derives decision rules as compact Boolean logic sentences involving subsets of features in the input data. The approach is applied to test point insertion problem in circuits and compared to the ground truth and traditional design rules.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Hardware** → **Design for testability**; • **Information systems** → *Decision support systems.*

KEYWORDS

Interpretable Machine Learning; IC Testing; Test-Point Insertion

ACM Reference Format:
Prashanth Krishnamurthy, Animesh Basak Chowdhury, Benjamin Tan, Farshad Khorrami, and Ramesh Karri. 2020. Explaining and Interpreting Machine Learning CAD Decisions: An IC Testing Case Study. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Increasing complexity in Integrated Circuit (IC) design has driven the exploration of Machine Learning (ML) techniques (from traditional models [14] through to state-of-the-art Deep Learning (DL) [12]) for accelerating the Computer-Aided Design (CAD) flow [3]. ML has shown promise for predicting downstream impacts of design choices in earlier stages of design. ML techniques learn model parameters from existing data to make accurate predictions. In a typical ML-based CAD flow, designers make decisions based on the "black box" outputs produced by the trained model, often without any insight into the factors that led to a given classification or prediction. If an ML model was able to *explain* its prediction making, the guidance would help designers use their domain expertise to address design problems—for targeting root-causes—or increase confidence in the model behavior.

Designers use their domain expertise upfront for data curation and feature engineering, but are given few insights into how the input feature values contributed to the ultimate prediction. To address this, and with wider implications for accountability and fairness [13], studies in explainable and interpretable ML explore techniques for post hoc explainability, to provide insight into black box decisions (e.g., feature importance [6]), and desiderata for models that are intrinsically explainable (e.g., linear regression). All works in MLCAD (including 20+ papers in ML focused on IC test [14]) treat the trained models as black boxes with no explicit focus on explainability and human interpretability. **This gap provides an opportunity for new insights into ML-based CAD: Explain and interpret ML-based decisions.**

Specifically, we address this problem in the context of ML-based classifiers [16] for predicting likelihoods that certain circuit elements (nodes/locations, sub-circuits, etc.) satisfy specific properties (difficulty-of-testing in this paper) in a computationally efficient manner (by using ML) than a more direct circuit-based analysis of the properties. Crucially, along with the ML-based evaluation of circuit elements, we seek to generate (in an automated manner) human-readable explanations and interpretations of the decision logic of the ML classifier. Additionally, we seek to generate context-sensitive human-readable explanations, i.e., when the ML classifier identifies multiple circuit elements as likely to satisfy a certain property, a separate human-readable explanation is generated for *each* circuit element to explain why it was classified as such.

For this purpose, we propose the use of an intrinsically explainable, rule-based ML framework, called Sentences in Feature Subsets (SiFS) and investigate: (1) classification accuracy in predicting node testability on public netlist datasets and (2) qualitative benefits of this human-readable methodology. SiFS [10] enables automated generation of compact decision rules, that are comprised of Boolean logic sentences from subsets of features in the input data. SiFS is a general methodology applicable to a variety of ML tasks such as, for example, anomaly detection in network data [9].

We seek to enhance ML-based CAD flows with *explanations* that can further inform design decisions *alongside* the ML model predictions. While we focus on an IC test problem in this study, SiFS can be used throughout the CAD flow where model interpretability can offer insights to humans in the loop. Our contributions are:

- A study of SiFS, an ML methodology that can be used throughout the CAD flow for learning human-readable classification rules, in an Electronic Design Automation (EDA) problem.
- Detailed experimental evaluation on an IC testing case study, showing up to 95% accuracy in prediction of "hard-to-test" nodes in a netlist (comparable to recent related works), but with the added bonus of model interpretability.
- Insights into the opportunities provided by explainability and interpretability in MLCAD, informing new studies in this domain.

Section 2 describes background and related work. Section 3 details the SiFS approach and our adaptation of it the IC test problem. Section 4 presents our experimental setup and results. Section 5 provides insights and discussions arising from our study.

2 BACKGROUND AND RELATED WORK

2.1 ML based Approaches for Test

In this study, we focus on IC testability, a popular area for recent studies in ML CAD [14]. Researchers have formulated the Test Point Insertion (TPI) problem as an ML problem for various objectives like improving fault coverage or reducing the number of test vectors.

In this work, we focus on identifying hard-to-test locations in a netlist corresponding to the *stuck-at* fault model. Identifying such faulty sites provide insight into appropriate locations for inserting test points to improve testability. Traditionally, testability measures like SCOAP [7] estimate the "hardness" of line testability in linear time. However, such estimations are often coarse and do not work well in presence of various circuit structures like re-convergent fanouts [8]. Estimating exact line testability is an NP-class problem and therefore can only be identified by exhaustive circuit simulation over the input space. Given the impracticality of exhaustive simulation for industrial designs, ML approaches seek to produce accurate predictions at a fraction of the computational cost.

Recent works have used Neural Network (NN) based approaches. For example, Ma *et al.* use Graph Convolution Networks (GCNs) to identify "easy to observe" and "hard to observe" nodes. Their intuition is that the natural graph structure of a netlist aligns well with the graph-centric GCN formulation such that the ML model will learn meaningful features implicitly. In an alternative approach, Sun and Millican propose artificial NNs for predicting TPI quality by estimating the change in fault coverage [15]. These approaches use approximate testability measures such as SCOAP [7] as the predominant input feature and learn complex, inexplicable non-linear combinations of these features, together with those of neighboring nodes, for predictions. Both these approaches achieve a sizable speed-up over conventional fault-simulation based approaches with reasonable predictive capability. Our proposed work aims for a more interpretable ML model while achieving comparable accuracy.

2.2 Explainability and Interpretability

There is growing interest in explainability and interpretability given the proliferation of "big data" and the potential for complex models (i.e., DL approaches), especially when used in systems with high risk/impact [13]. Interpretability is beneficial for guiding and auditing decision making; understanding how different characteristics of an input (i.e., features) affect the output provides insight into root causes, enables "sanity checks", and aids model debugging.

Post hoc techniques aim to design "interpretable" approximations of complex models, typically with trade-off between accuracy and interpretability [11], or to apply statistical reasoning to determine feature importance [6]. Part of the interpretability challenge stems from DL models learning to extract features automatically without any constraint that the learned features are meaningful to humans.

In contrast, *intrinsically* explainable models such as linear regression and decision trees require upfront feature engineering by a human expert. Such models offer a level of interpretability (e.g., "impact" of a human-understandable input feature can be understood, in part, by examining learned weights). However, approaches like linear regression struggle to accommodate complexity by oversimplifying the decision space. A more advanced approach, such as SiFS, can derive more complex decision boundaries in feature space by learning *multiple* rules; different input samples can "activate" different rules, providing a context-aware explanation of a decision.

3 PROPOSED APPROACH

The structure of the explainable ML approach is shown in Figure 1. While this methodology can be applied to obtain human-readable explanations in a wide range of problems in the CAD domain and beyond, we focus on the TPI problem in this paper.

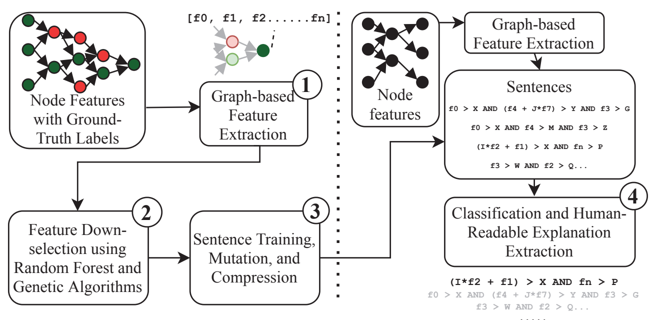

Figure 1: Proposed methodology for explainable ML CAD decisions. Training (left) and Inference (right) stages.

3.1 Problem Description

We focus on identifying hard-to-test locations in a combinational circuit netlist corresponding by means of ML based classification.

Problem: *Given a node from a circuit netlist, and a set of node features, classify the node as "hard-to-test" or otherwise.*

A combinational circuit is a a graph $G(V, E)$, comprising a set of nodes V that are interconnected by edges E, where the edges between nodes are the connections between logic gates. We define a node as "hard-to-test" when the number of test vectors that cover a stuck-at fault there, n, as a fraction of a pre-determined number of random test vectors, N, is below some threshold (i.e., when $n/N < h$). We set threshold h to 0.1 in our experiments in Section 4. More precisely, for each node, the ground truth consists of two numbers `L0` and `L1`, which indicate the percentages of test inputs to the circuit that are able to detect a stuck-at-0 fault and a stuck-at-1 fault, respectively. Nodes with small `L0` and/or `L1` values are hard-to-test since test inputs are less likely to be able to detect faults at that node in the circuit. The ground truths can be determined by running fault simulation using random test vectors.

3.2 Feature Extraction

The first aspect of our approach is feature extraction (① in Figure 1). For each node in the circuit, we derive four features: SCOAP [7] controllability 0 and 1, observability, and logic depth from furthest PI ([`CC0`, `CC1`, `C0`, `LL`]). From these circuit-level features for the nodes, we use the graph structure of the circuit to extract additional features to facilitate ML-based classification. The additional features aggregate characteristics from the local neighborhood of each node as well as capture connectivity to the primary inputs and outputs of the circuit. These features (computed for each node) are:

- `pct_connected_inputs`: % input pins for which there is ≥ 1 path to node
- `pct_connected_outputs`: % output pins where there is ≥ 1 path from node to the pin
- `min_dist_inputs`: Shortest path to node from any input pin
- `min_dist_outputs`: Shortest path from node to any output pin
- `n_paths_from_inputs`, `n_paths_to_outputs`: # distinct paths from any input pin to node and from node to any output pin
- `n_in`, `n_out`: # predecessor nodes and # successor nodes
- `mean_CC0_in`, `mean_CC1_in`, `mean_CO_in`, `mean_LL_in`: Average `CC0`/`CC1`/`CO`/`LL` feature value among predecessors
- `max_CC0_in`, `max_CC1_in`, `max_CO_in`, `max_LL_in`: Largest `CC0`/`CC1`/`CO`/`LL` feature value among predecessors
- `min_CC0_in`, `min_CC1_in`, `min_CO_in`, `min_LL_in`: Smallest `CC0`/`CC1`/`CO`/`LL` feature value among predecessors
- `mean_CC0_out`, `mean_CC1_out`, `mean_CO_out`, `mean_LL_out`: Average `CC0`/`CC1`/`CO`/`LL` feature value among successors
- `max_CC0_out`, `max_CC1_out`, `max_CO_out`, `max_LL_out`: Largest `CC0`/`CC1`/`CO`/`LL` feature value among successors
- `min_CC0_out`, `min_CC1_out`, `min_CO_out`, `min_LL_out`: Smallest `CC0`/`CC1`/`CO`/`LL` feature value among all successors

Since enumerating all paths to input/output pins is computationally expensive, the distinct path based features are computed with a saturation S (defined in our experiments to be 10). With the additional features above, the feature vector for each node is of length 36.

3.3 ML Formulation and Data Filtering

Since the aim is to find hard-to-test locations, we are interested in down-selecting nodes to determine good candidates for TPI. Thus, the ML system does not need to estimate actual `L0` or `L1` values, or distinguish between nodes with very high `L0` or `L1` values and nodes with medium-sized `L0` or `L1` values. Instead, the ML system should be able to identify accurately nodes with low `L0` or `L1` values. Thus, we train ML classifiers to predict nodes that would satisfy any of the following four properties:

- `L0` $< h$, i.e., nodes at which stuck-at-0 faults are hard-to-test
- `L1` $< h$, i.e., nodes at which stuck-at-1 faults are hard-to-test
- `L0` $< h$ AND `L1` $< h$, i.e., nodes at which both stuck-at-0 faults and stuck-at-1 faults are hard-to-test
- `L0` $< h$ OR `L1` $< h$, i.e., nodes at which either stuck-at-0 faults or stuck-at-1 faults are hard-to-test.

Since a value of `L0` or `L1` slightly larger than h does not indicate that the node is "easy-to-detect," we define another threshold $\overline{h}$ (set to 0.2 in our experiments) and discard training data for which `L0` or `L1` values are in the interval $[h, \overline{h}]$. We train separate ML classifiers to detect each of the four node properties above.

As the ML objective is to identify hard-to-test nodes, in particular where nodes are classified by the model with high confidence. We set a threshold (0.75 in the experiments in Section 4) on the confidence value to down-select nodes in the test data (i.e., nodes for which the continuous-valued likelihood outputs of the classifier ≤ 0.25 or ≥ 0.75 are retained). For experimental purposes, we measure accuracy of the classifiers by considering these nodes.

3.4 Human-Readable Interpretable ML

The SiFS methodology [9, 10] answers the following questions:

- Which features in the data are relevant for classification?
- What combinations of features (i.e., combinations of inequalities involving subsets of features) can be used to classify?
- Given an input data point, what is the best context-sensitive explanation for classification of that data point?

For this purpose, we use a multi-step architecture as shown in Figure 1. The methodology is applied to the four node property classification tasks (i.e., four different labels in a multi-label classification context) defined in Section 3.3. For brevity, we describe the methodology considering one ML task, i.e., considering one label.

② **Feature Down-Selection:** To find the subset of features that are most relevant for the classification task, we use a random forest and genetic algorithm based approach. A random forest classifier offers an efficient feature utility estimator and a genetic algorithm searches the space of possible feature subsets based on classification accuracy obtained using the pruned feature vectors.

Starting with a population of randomly initialized candidate feature subsets, we train random forest classifiers using training data restricted to each of the candidate feature subsets. The population of candidate feature subsets is iteratively evolved using random mutation and cross-over operators based on the classification performance of the random forest classifiers and the feature utilities they estimate. The process is iterated until a stopping condition is met (e.g., max number of epochs and/or compactness of feature subset). While any classification algorithm can be the guiding heuristic

for this iterative feature pruning, a random forest has the advantage of having optimized off-the-shelf implementations (e.g., [1]) that provide feature utility estimates (feature "importances") computed from the training data.

③ **Sentence-Based Classifier Training:** From the reduced set of features down-selected as discussed above, a SiFS classification model is defined as an OR-combination of a set of "sentences." Each sentence is defined as an AND-combination of "words." Each word is an inequality involving a subset of features. The choice of features for each word is represented using mask variables. The mask variables for the inequalities in each sentence are trained along with the coefficients that appear in the inequalities using stochastic gradient descent and genetic algorithms. Denoting the subset of features down-selected as discussed above by $f_1, \ldots, f_{n_f}$, with a normalization for zero mean and unit variance (with the scale and shift parameters for the normalization computed from training data). the multi-sentence classifier model can be written as

$$s_j = \wedge_{k=1,\ldots,n_w} \left(\sum_{i=1}^{n_f} m_{i,j,k} c_{i,j,k} f_i \geq c_{(n_f+1),j,k} \right) \ , j = 1, \ldots, n_s \quad (1)$$

$$y = \vee_{j=1,\ldots,n_s} s_j \quad (2)$$

where n_s is the number of sentences in the model, n_w is the number of words in each sentence, and y is the binary output of the classifier. In (1), $m_{i,j,k}$ are the binary "mask" variables and $c_{i,j,k}$ are the coefficients in the inequalities. To enable gradient-based learning of the coefficients $c_{i,j,k}$, the "hard" inequalities and Boolean combinations in (1) are softened to a differentiable relaxation given by:

$$\tilde{s}_j = \Pi_{k=1}^{n_w} \left\{ \sigma \Big(\sum_{i=1}^{n_f} m_{i,j,k} c_{i,j,k} f_i - c_{(n_f+1),j,k} \Big) \right\} \ , j = 1, \ldots, n_s \quad (3)$$

$$\tilde{y} = \max\{\tilde{s}_1, \ldots, \tilde{s}_{n_s}\} \quad (4)$$

$$y = 0 \text{ if } \tilde{y} \leq 0.5 \text{ and } 1 \text{ if } \tilde{y} > 0.5. \quad (5)$$

In (3), σ denotes the sigmoid function given by $\sigma(a) = \frac{1}{1+e^{-a}}$. The gradient-based learning of the coefficients $c_{i,j,k}$ uses randomly drawn mini-batches and a loss function defined as a combination of the binary cross-entropy, a penalty term for false positives, and a regularizer to penalize sizes of the coefficients $c_{i,j,k}$.

In concert with the gradient-based learning of the coefficients $c_{i,j,k}$, the mask variables are updated by a genetic algorithm. The mask variable updates happen after some number of epochs of the gradient-based updates. For the genetic updates, we define a "goodness" of a sentence $\tilde{s}_j$ based on the true positive rate of the sentence, a penalty on the false positive rate of the sentence, and a measure of the sentence complexity. We measure sentence complexity as the weighted sum of the effective word lengths in the sentence and the number of distinct features in it. The sentence population is randomly initialized. The sentences are sorted by decreasing goodness and we iteratively construct a subset of "best" sentences by adding sentences that contribute true positive classifications of the training data on top of those classified by the sentences in the subset. The genetic update of the mask variables $m_{i,j,k}$ is performed by drawing extra sentences based on goodness and by introducing random mutations to generate the next generation of sentences.

The gradient-based and genetic updates are iterated until a stopping condition (e.g., preset max. number of epochs). The sentence population is pruned to retain the best subset (i.e., sentences with the highest goodness) and compressed to remove word redundancies (i.e., inequalities). The compression considers instances where (i) inequalities are redundant given physically relevant upper and lower bounds for the features in the input data and (ii) multiple inequalities that are approximately the same (as measured by the angle between the corresponding hyperplanes in the feature space).

④ **Extraction of Context-Sensitive Human-Readable Explanation:** Each sentence in the multi-sentence SiFS classifier model estimates a sub-volume of an overall volume in feature space within which lie some data points of a given class, C. The estimated volume in feature space corresponding to C is the union of the sub-volumes defined by each sentence. The model labels an input data point as being in C when it "triggers" any of the learned sentences. From the subset of activated sentences for a data point, the "best" sentence for that data point is picked based on the activation value of the sentence (i.e., $\tilde{s}_j$ from (3)) and the goodness of the sentence in the context of the genetic updates. This best sentence provides the human-readable explanation and is reported as the context-sensitive identification of a subset of features and combinations of features as a Boolean combination of inequalities.

3.5 Decision Making

As discussed in Section 3.3, the ML classifier identifies nodes that are likely to be hard-to-test. Hence, these nodes are good candidates for test point insertion. Therefore, to iteratively improve testability of the circuit, the proposed ML methodology can be used as a guiding heuristic for an iterative TPI algorithm. Such an iterative algorithm can, for example, consider a top $k\%$ of the nodes identified by the ML methodology as hard-to-test, and then after inserting output pins at each of these nodes, rerun the ML methodology. In this paper, we focus on the core ML problem of detecting hard-to-test points and explaining in a human-readable way the decision logic for these classifications. Hence, for brevity, we do not consider explicitly further the iterative TPI application of the methodology.

4 EXPERIMENTAL SETUP AND RESULTS

4.1 Experimental Overview

We apply SiFS on a set of benchmark circuits (summarized in Section 4.2). We apply the feature extraction and ML algorithms discussed in Section 3 to obtain an explainable ML system for the four classification tasks outlined in Section 3.3. The experimental results are summarized in terms of the obtained classification performance in Section 4.3 and in terms of human-readable explanations in Section 4.4. High accuracy, precision, etc., in detecting hard-to-test nodes are attained while providing compact human-understandable, context-sensitive explanations of the decision as combinations of inequalities among subsets of features in the input data.

4.2 Netlist Datasets from Benchmark Circuits

We use circuits from the publicly available ISCAS and EPFL benchmarks [2, 4, 5]. They are listed in Table 1 ranging from small (231-node iscas85 c499) to large (14712-node epfl voter). These circuits were randomly split into training and testing datasets.

- Training: 15 circuits from Table 1 except the five for testing.

- Testing: epfl (router, voter), iscas89 (s15850, s838, s9234).

We obtain the SCOAP features and L0/L1 for the circuits using Mentor's Tessent 2019.1 DFT tool. Figure 2 summarizes some SCOAP statistics for the circuits. All benchmarks are scanned and synthesized using ABC. To obtain ground-truth testability scores for stuck-at-0 and stuck-at-1 faults (L0/L1, respectively), we generated 1 million random test vectors for each circuit and identified the % of test vectors that can detect the faults for each node in netlist.

We do not use some parts of a circuit for training and the rest of the circuit for testing. Instead, we use separate circuits for training and testing. This ensures that the ML system is not learning incidental artifacts of a circuit, but is learning physically relevant tie-ins between the semantically meaningful features (in Section 3.2) and the "hard-to-test" properties (in Section 3.3).

The training dataset (i.e., the set of all nodes from all the circuits used for training) has 35091 nodes. As discussed in Section 3.3, the nodes in the training dataset are down-selected to focus the ML training on the nodes that offer the most insights into the hard-to-test properties. We retain those nodes from the training data whose ground-truth L0 and L1 labels are in either a low- or a high-magnitude range. The down-selected training dataset has 30540 nodes. The testing dataset has 21208 nodes. For each of the nodes in both datasets, the four node-level features (CC0, CC1, C0, and LL) are computed and additional features based on the graph structure of the circuit are constructed as discussed in Section 3.2 yielding a 36-element feature vector for each node.

Table 1: Circuits used for experimental studies.

Category	Netlist
iscas85	c432, c499, c1908, c3540, c7552
iscas89	s526, s832, s838, s1494, s9234, s15850, s38584
epfl	arbiter, cavlc, decoder, i2c, int2float, priority, router, voter

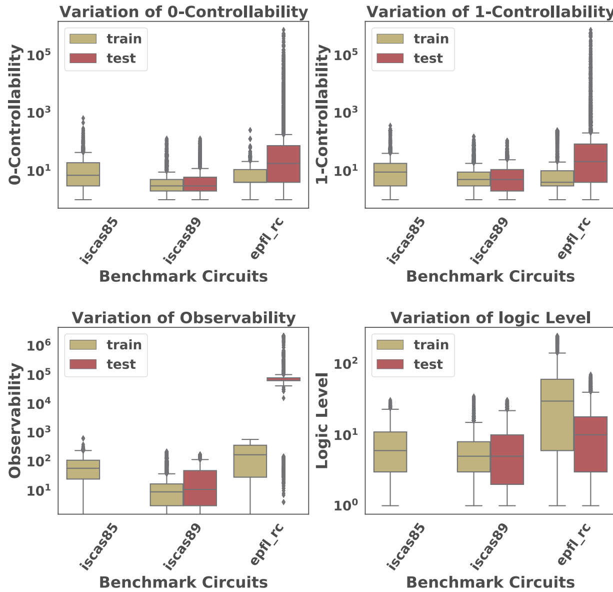

Figure 2: Distribution of SCOAP testability metrics.

Table 2: Classification performance on the testing dataset.

Task	#down-sel.	Accuracy	Precision	Recall	F1
Low L0	12949	0.898	0.898	0.990	0.942
Low L1	5718	0.910	0.884	0.955	0.918
Low L0 AND Low L1	3806	0.929	0.857	0.914	0.884
Low L0 OR Low L1	12645	0.951	0.963	0.980	0.971

4.3 Classification Performance

As discussed in Section 3.3, we train a classifier for each of the four classification tasks: classifying a node as hard-to-test for stuck-at-0 faults, hard-to-test for stuck-at-faults, hard-to-test both types, and hard-to-test for either type. As discussed in Section 3.3, the continuous-valued activation outputs (which can be viewed as confidence values or likelihoods in a [0,1] interval) are used to down-select nodes in the testing dataset to those nodes that are classified as TRUE or FALSE (for the particular classification task addressed by the classifier) with confidence higher than the threshold. This down-selection is performed separately for each of the four classification tasks. The classification performance is then measured on these down-selected nodes using the ground-truth labels computed as defined in Section 4.2. The classification performance is summarized in Table 2. The numbers of down-selected nodes for each of the four classification tasks are provided in Table 2.

Since the primary objective of our ML classifier is to find good TPI candidates, we next analyze to what extent nodes flagged as hard-to-test with high confidence by the ML classifiers are indeed hard-to-test (based on ground-truth labels). We sort the testing dataset nodes (separately for each of the four classification tasks) in the order of decreasing confidence values of the TRUE label. Then, for multiple values of n, we plot the number of the top-n nodes that are indeed hard-to-test (Figure 3). Comparing with the ideal case where all top-n nodes are indeed hard-to-test (black dotted line in Figure 3), we see a strong correlation between confidence values and actual difficulty of testing. This is especially the case for the "Low L0 OR Low L1" task, which is the most important for the TPI application since those nodes are hard to test for at least one of the stuck-at-0 and stuck-at-1 fault scenarios.

4.4 Human-Readable Explanations

As discussed in Section 3.4, the "best" sentences can be extracted for each test data point that is classified as TRUE for the particular binary classification task. For discussion, we examine a selection of sentences that are picked as best for a majority of the test data points for classification tasks defined in Section 3.3. These sentences are not the only sentences used in classification, other nodes may be detected by other sentences. These other sentences would be similar in structure, but possibly with different numbers of words in the sentences and with different coefficients and mask variables. The features in the explanations correspond to selected features from the list in Section 3.2. To interpret the explanations, we qualitatively consider the relative weights of the features and the combination of features in the words of each sentence. These offer insights into how the ML classifier makes some of its decisions.

ML Explanation for Low `L0` (hard-to-test stuck-at-0):

```
word 1: (-0.011*f1+0.53*f2+1*f3 > -0.38)
word 2: AND (0.98*f1+0.32*f2+1*f3-2.5*f4 > -0.091)
word 3: AND (-0.11*f1+1*f2+0.039*f3+0.33*f4 > -0.052)
where f1 = CO, f2 = mean_CO_in,
f3 = pct_connected_outputs, f4 = CC1
```

Test Expert Interpretation: From word 1, the classifier recognises that nodes connected to a high number of primary outputs (i.e., as f3 increases) with immediate predecessors that have low observability (i.e., as f2 increases) are possibly hard-to-test. Word 2, suggests the classifier has learned that a node is possibly hard-to-test in situations where controllability decreases (i.e., as f4 increases) but only if this is not balanced out by its and its neighbors' observability (i.e., f1, f2, f3 should be relatively larger than f4). In word 3, the node is possibly hard-to-test if it has lower controllability (i.e., as f4 increases) along with lower observability (i.e., as f1 increases) with less consideration given to % of primary outputs in a node's fanout.

ML Explanation for Low `L0` AND Low `L1`:

```
word 1: (+0.54*f2+0.24*f3-1.7*f6 > -1)
word 2: AND (-0.091*f5-0.11*f7+1*f9 > 0.017)
where f1 = min_CO_out, f2 = min_dist_outputs,
f3 = pct_connected_outputs, f4 = pct_connected_inputs,
f5 = min_LL_out, f6 = n_paths_to_outputs,
f7 = mean_CC1_in, f8 = mean_CC0_in, f9 = n_out
```

Test Expert Interpretation: In word 1, the classifier learned to recognize that even though a node is close to the primary outputs (i.e, with decreasing f2) or is connected to many outputs (i.e., with increasing f3), it does not reflexively mean that the node is easy-to-test. A higher number of paths from the node to primary outputs (i.e., with increasing f6) is important for making the node easier to test. From word 2, we can interpret that the model spots nodes with a high fanout (i.e., with increasing f9) as hard-to-test if the mean controllability of its predecessors is low (i.e., with increasing f7).

5 DISCUSSION AND CONCLUDING REMARKS

We explored the potential for explainable ML in CAD, with promising results in IC testing. For the classification objective (detecting hard-to-test nodes) we considered, for this proof-of-concept application of the human-readable ML methodology, all the feature elements are physically relevant to the classification tasks. To some extent, the features selected through our down-selection and training processes to form human-readable sentences are somewhat random and can vary between different runs and train/test splits. Nevertheless, the generated sentence-based explanations are more human-understandable than arbitrary combinations of the 36 features and more human-interpretable than other classifiers such as decision trees, random forests, and neural networks.

Selecting features that are understandable and physically relevant for classification remains challenging. Where some features are detected as not relevant for a classification task, our approach eliminates such features leaving behind vital human-interpretable clues as features that are the most relevant. In turn, it provides insights to engineers when making design decisions, such as highlighting areas of a circuit to focus on.

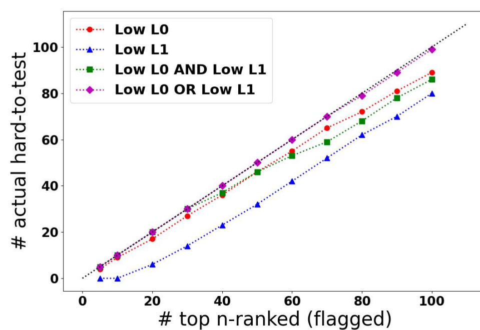

Figure 3: # of top-ranked flagged nodes in test data that are actually hard to test based on threshold-based ground truth.

ACKNOWLEDGMENTS

This work was supported in part by ONR Award # N00014-18-1-2672 and DARPA grant # FA8750-20-1-0502. B. Tan and R. Karri are supported in part by ONR Award # N00014-18-1-2058. R. Karri is supported in part by the NYU/NYUAD Center for Cyber Security.

REFERENCES

[1] Retrieved, Aug. 2020. scikit-learn random forest classifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Int. Workshop on Logic Synthesis*.
[3] Duane S. Boning, Ibrahim (Abe) M. Elfadel, and Xin Li. 2019. A Preliminary Taxonomy for Machine Learning in VLSI CAD. In *Machine Learning in VLSI Computer-Aided Design*. Springer International Publishing, Cham, 1–16. https://doi.org/10.1007/978-3-030-04666-8_1
[4] Franc Brglez, David Bryan, and Krzysztof Kozminski. 1989. Notes on the ISCAS'89 Benchmark Circuits. *MCNC* (1989).
[5] David Bryan. 1985. The ISCAS'85 benchmark circuits and netlist format. *North Carolina State University* 25 (1985), 39.
[6] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. 2019. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *Journal of Machine Learning Research* 20, 177 (2019), 1–81. http://jmlr.org/papers/v20/18-760.html
[7] Lawrence H Goldstein and Evelyn L Thigpen. 1980. SCOAP: Sandia controllability/observability analysis program. In *Design Automation Conf.* https://doi.org/10.1109/DAC.1980.1585245
[8] Leendert M Huisman. 1988. The reliability of approximate testability measures. *IEEE Des. Test. Comput.* (1988). https://doi.org/10.1109/54.9272
[9] P. Krishnamurthy, F. Khorrami, S. Schmidt, and K. Wright. 2020. Machine Learning for NetFlow Anomaly Detection with Human-Readable Annotations. (2020). Under review for journal publication.
[10] P. Krishnamurthy, A. Sarmadi, and F. Khorrami. 2019. Explainable Classification by Learning Human-Readable Sentences in Feature Subsets. (2019). Under review for journal publication.
[11] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 4765–4774. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf
[12] Yuzhe Ma, Haoxing Ren, Brucek Khailany, Harbinder Sikka, Lijuan Luo, Karthikeyan Natarajan, and Bei Yu. 2019. High Performance Graph Convolutional Networks with Applications in Testability Analysis. In *Design Automation Conf.* https://doi.org/10.1145/3316781.3317838
[13] Christoph Molnar. 2019. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/.
[14] Manjari Pradhan and Bhargab B. Bhattacharya. 2020. A survey of digital circuit testing in the light of machine learning. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (March 2020), e1360. https://doi.org/10.1002/widm.1360
[15] Y. Sun and S. Millican. 2019. Test Point Insertion Using Artificial Neural Networks. In *IEEE Comp. Society Symp. on VLSI*. https://doi.org/10.1109/ISVLSI.2019.00054
[16] Ryan J. Urbanowicz and Jason H. Moore. 2009. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications* 2009 (Jan. 2009), 1:1–1:25. https://doi.org/10.1155/2009/736398