# Improving Diagnosis Efficiency via Machine Learning

Qicheng Huang, Chenlei Fang, Soumya Mittal and R. D. (Shawn) Blanton
Advanced Chip Testing Laboratory (www.ece.cmu.edu/~actl/)
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213
{qichengh, chenleif, soumyami, rblanton}@andrew.cmu.edu

***Abstract*—Logic diagnosis, the process of identifying and locating possible defects in failing integrated circuits, is a key step in yield learning for both technology development and high-volume manufacturing. However, resources can be easily wasted if diagnosis results in no meaningful information, or if the type of diagnostic result is not actionable. It would therefore be very beneficial to have a comprehensive preview of diagnostic outcomes beforehand, which allows diagnosis resources to be prioritized in a more reasonable and effective way. In this work, a methodology is developed to predict whether a fail log for a given design will result in a diagnosis outcome that is meaningful for the purpose at hand. Specifically, the aim is to predict the time required for diagnosis, and whether diagnosis produces any defect candidates, and if so, are those candidates the result of logic failure or chain failure. Random Forest classification algorithm is used for prediction. Experiments on a 28nm test chip and a high-volume 90nm part illustrate that the methodology can provide accurate prediction results (0.95+ precision, 0.9+ recall and F1-score, and 0.96+ AUC on average) when two classes are balanced, and satisfactory results (0.95+ recall and 0.98+ AUC on average) when two classes are imbalanced.**

## I. Introduction

Yield learning, the task of identifying and mitigating sources of yield loss, remains an integral and important task in developing an advanced semiconductor technology. It remains important as technology development transitions to high-volume manufacturing. For both development and high-volume manufacturing, yield learning for logic presents unique challenges because of the virtually unlimited layout patterns that are created by new cell libraries and the techniques employed by evolving place-and-route algorithms. Under both scenarios, logic-circuit diagnosis for both locating and characterizing failure is a key first step in yield learning.

Logic diagnosis takes as input a logic-circuit description, the applied set of test patterns, and a fail log for a tested fabricated instance of the circuit. A fail log is a straightforward tabulation of the primary and scan outputs that have failed for all or a subset of the test set. Traditionally, the quality of a diagnosis is measured in terms of its resolution and accuracy. Resolution is the number of possible locations or defects where the actual failure can reside, and a diagnosis is deemed accurate if one of the reported locations/defects corresponds to the actual failure. Increasingly important is the amount of time required to perform diagnosis, and whether or not diagnosis will report any meaningful information. For example, Fig. **1**(a) shows a histogram of the runtime required to diagnose on 4,235 fail logs for a test chip fabricated at the 28nm technology node. The data has been partitioned into three sets: blue represents cases where diagnosis reports a set of locations (i.e., logic failure), yellow represents cases of scan-chain failure (i.e., chain failure), and red indicates those cases where diagnosis fails to report any locations (i.e., no result). The total amount of runtime to perform diagnosis on these fail logs exceeds a million seconds (12 days) on a state-of-the-art server, and the percentage of fail logs that report no meaningful information (the red portion of the plot) is 54%.

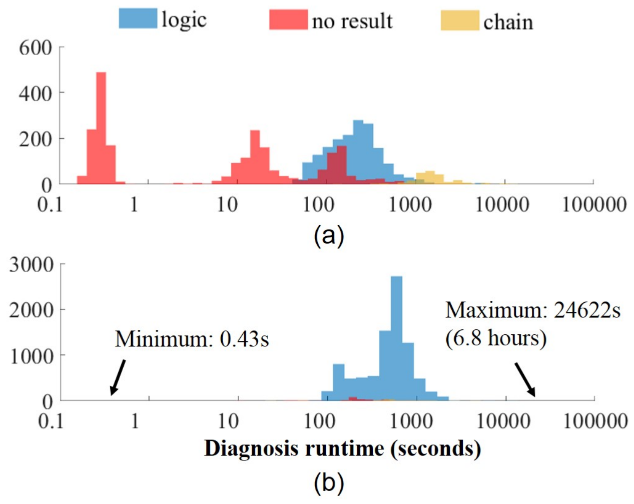


**Figure 1**. Histograms of diagnosis runtime for (a) a 28nm test chip and (b) a high-volume 90nm chip.

We have surveyed nearly a dozen experts in industry concerning the amount of diagnosis performed during technology development and high volume manufacturing. These experts include representatives from foundries, integrated device manufactures, and fabless design houses. Several of them indicated that the number of diagnoses performed per week per design during technology development can be 10,000, and one indicated that number exceeds 100,000. Given the amount of diagnoses during technology development that can lead to no actionable information (54% in Fig. **1**(a)), it would be very beneficial to know beforehand which failure logs would result in "no result", and those that result in either logic circuit or chain failure since different yield learning techniques can be brought to bear.

Fig. **1**(b) plots the data for a 90nm high-volume part. Although the number of diagnoses that result in no actionable information (3%) and chain failure (1.2%) is much less than the test chip, the runtime for a diagnosis can vary significantly, ranging from 0.43 to 24,622 seconds. In a high-volume scenario, it is typically the case that failures have been categorized via a failure pareto as illustrated in Fig. **2**. Yield learning often targets the largest bar in the pareto because identifying and remedying the root cause associated with this source of yield loss results in the largest gain in yield. But as already demonstrated, resources can be easily wasted if diagnosis results in no meaningful information, or if the type of result is not actionable (e.g., chain failure). Similar to technology development, it would therefore again be very beneficial to know beforehand which fail logs during high-volume manufacturing would result in actionable information. Moreover, for a given set of fail logs, like those associated with the largest source of yield loss as indicated by a pareto, a constraint on diagnosis resources, and with all other things being equal, it is prudent to choose fail logs that have short runtime. This conclusion is congruent with our survey because practitioners that are handling large numbers of diagnoses indicate that sampling must be employed, implying that there is indeed limited resources for diagnosis.

Based on the aforementioned, the objective of this work is to develop a methodology for predicting whether a fail log for a given design will result in a diagnosis outcome that is meaningful for the purpose at hand. Specifically, our aim is to predict the time required for diagnosis, and whether diagnosis produces any defect candidates, and if so, are those candidates the result of logic failure or chain failure. Our preliminary analysis reveals that there is no simple correlation between fail-log features and runtime or diagnosis success. For example, Fig. **3** plots diagnosis runtime versus the number of failing outputs for both the test chip and high-volume part of Fig. **1**(a) and **1**(b), respectively. So we have employed *machine learning* (ML) to uncover higher-dimensional correlations between diagnosis outcomes and fail log features. For chain failures, special tests called flush tests [1] are typically used to ascertain chain integrity before further diagnosis – circuits failing in flush test are considered to have scan-chain failures. However, a flush test is not always in 100% correlation with chain failure. For example, it is possible that a flush test misses a scan-cell internal stuck-open fault [2] or an intermittent fault [3]. On the other hand, circuits with faults outside the scan chains can also fail a flush test. We will show in Section III that a considerable portion of circuits that fail the flush test are later diagnosed as either a logic failure or unsuccessfully diagnosed. These circuits may suffer from compound defects [4], which means there exist both chain defects and logic defects. Such a situation can be too complex for available diagnosis tools to obtain a reasonable diagnostic result, or simply lead a tool to explain the failures with simpler logic faults (rather than chain faults). In this case, directly classifying the chip as having chain defects due to the flush test failure will miss the opportunity to find logic faults or waste time in un-diagnosable fail logs. ML can therefore be used to improve the accuracy of chain failure prediction, on top of the flush-test results.

There has been prior work that has used ML in diagnosis. In [5], a random forest is used to predict whether a diagnosis outcome from a given fail log is due to a bridge defect. In

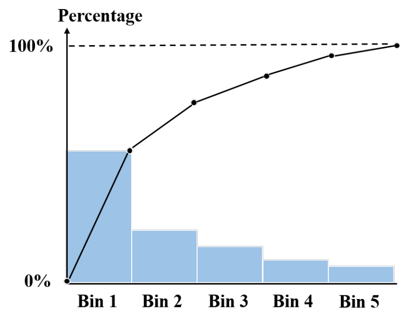


**Figure 2**. Typical failure pareto for high-volume part.

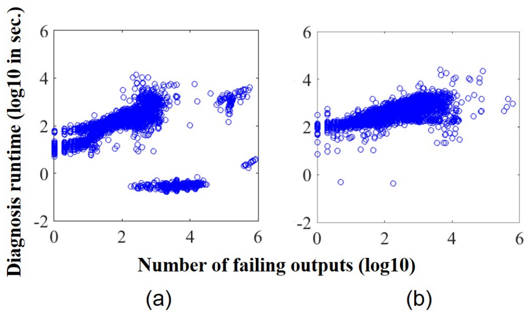


**Figure 3**. Plot of diagnosis runtime versus the number of failing outputs for (a) a 28nm test chip and (b) a high-volume 90nm chip.

[6], ML predicts in real time how much fail data should be collected for a high-quality diagnosis. In [7-9], the authors use a learned support-vector machine model based on volume diagnostics to improve diagnostic resolution. Volume diagnostics and ML techniques are also used to check the efficacy of DFM rules and to identify systematic defects in [10, 11], and [12, 13] respectively. The authors of [14] apply neural networks to match the faulty behavior of a chip with certain fault models. The authors of [15] identify defective scan chain cells by using unsupervised learning. In [16], hierarchical clustering is used to group similar fail logs and to help identify systematic defects. Except for [6], the work proposed here is a complement to all aforementioned work because the goal is to generally predict the outcome of diagnosis before diagnosis is executed. The work in [6] has a similar goal but it is used to govern the decision of whether collecting more fail data will lead to higher-quality diagnosis result, while the work here assumes that the fail data of a given fail log is fixed. But given that both approaches have somewhat differing goals, both approaches, in theory, can be combined to achieve both sets of goals.

The rest of this paper is organized as follows. Section II describes the details of the proposed methodology; Section III demonstrates the efficacy of our method in experiments that use the 28nm logic test chip and a high-volume 90nm part of Fig. **1**(a) and **1**(b), respectively. Finally, Section IV concludes the paper and provides directions for future work.

## II. Methodology

To facilitate efficient diagnosis, we aim to predict three aspects of diagnosis: (i) diagnosis success; (ii) failure type (logic vesus chain); (iii) runtime. These three aspects provide a comprehensive preview of diagnostic results, and allows diagnosis resources to be prioritized in a more reasonable and effective way.

- **Diagnosis success:** We consider a diagnosis run to be successful if it reports one or more candidate(s) for each defect within a failing circuit. According to survey results, a majority of the practitioners indicated that they have encountered cases of unsuccessful diagnosis runs, and three indicated diagnosis fails to report any candidates for more than 50% of the time . With the capability to predict diagnosis success, diagnosis resources can be efficiently allocated. For example, fail logs that would result in the reporting of zero candidates can altogether be ignored or scheduled lastly.
- **Failure type:** For a fail-log diagnosis that does result in candidates, the failure type can be broadly divided into two classes: scan-chain and logic. If the diagnosis tool reports candidates for one or more defects in the scan-chains, then the failure type is deemed to be of type chain; otherwise the failure type is deemed to be of type logic. The yield learning methods brought to bear for scan-chain and logic failures can be quite different, so knowing the failure type beforehand can enable the proper allocation of resources. For example, one may want to initially focus on chain failures because of the existence of state-of-the-art chain debug capabilities like those described in [17].
- **Runtime:** Based on whether the predicted runtime for a given fail log is long (i.e., exceeds a user-defined threshold), it can be decided whether it is worthwhile to run the diagnosis, or if it is more prudent to order fail logs by running the most promising and fast diagnosis first. In the survey conducted, 84% of respondents indicated that diagnosis runtime that exceeds 1,000 seconds is too long. In addition, survey respondents that indicated the need to sample the large number of weekly fail logs could effectively increase their sample size by using those fail logs predicted to have acceptable runtime.

In this section, we first formulate the mathematical problem, then introduce the ML algorithm and the features we exploit. In the two scenarios we are considering (i.e., technology yield ramp and high-volume manufacturing), it is unavoidable to encounter imbalanced data. For example, for a chip during high-volume manufacturing, the ratio of scan-chain failure or failed diagnosis can be extremely low. So we also discuss the measures for such cases, including how to choose proper evaluation metrics and how to control the trade-off between the imbalanced classes.

### A. Problem Formulation

Three classifiers $C_1, C_2$ and $C_3$ are trained to predict diagnosis success, failure type and runtime, respectively. As shown in Fig. **4**, $d$ features are extracted from each fail log and

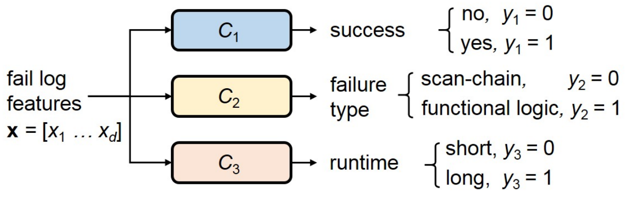


**Figure 4**. Based on the features derived from a given fail log, three classifiers $C_1, C_2$ and $C_3$ are trained to predict three aspects of diagnostic results, including diagnosis success, failure type and runtime. **x** is a vector containing $d$ features. $y_1, y_2$ and $y_3$ are three discrete variables indicating the prediction results.

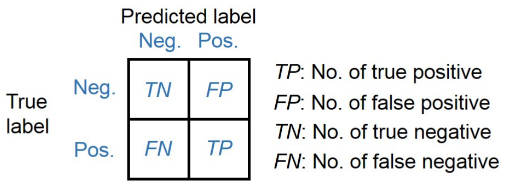


**Figure 5**. An illustration of the confusion matrix of a binary classifier.

represented as a $d$-dimensional vector **x**. The predictive results of three classifiers are encoded as three binary variables $y_1, y_2$ and $y_3$, respectively. The variable concerning runtime, $y_3$, can also have more than two values if a user wants to predict certain ranges that the runtime falls in, instead of simply being short or long. The threshold that defines the range of the bins can be decided by the user preference, or judiciously derived from the runtime distribution estimated from training data.

We divide all the available data samples into a training set $\mathbf{D}_{train} = \{\mathbf{X}_{train}, \mathbf{Y}_{train}\}$ and a test set $\mathbf{D}_{test} = \{\mathbf{X}_{test}, \mathbf{Y}_{test}\}$. The training data include the features extracted from the already diagnosed fail logs (i.e., $\mathbf{X}_{train}$) and the diagnosis information extracted from the corresponding diagnostic results (i.e., $\mathbf{Y}_{train}$). $\mathbf{X}_{test}$ represents the information extracted from fail logs that have not yet been diagnosed. The classifiers aim to take in $\mathbf{X}_{test}$ and generate prediction results that best match the ground truth, namely, $\mathbf{Y}_{test}$ that results from running diagnosis on the fail logs corresponding to $\mathbf{X}_{test}$.

*Precision, recall, F1-score* and *AUC* [18] are used to evaluate the matching extent between the predicted and the true labels (i.e., the values of $y_1 \sim y_3$). Precision, recall and F1-score can be directly computed from a *confusion matrix*. Fig. **5** shows a confusion matrix for a binary classifier. $TP, FP, TN$ and $FN$ represent the number of samples with different predicted and true labels, as shown in Fig. **5**. Precision, recall and F1-score are then defined in equations (1)-(3):

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1\text{-}score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}. \tag{3}$$

Precision represents the probability that (a randomly selected) sample predicted positive is truly positive, while recall represents the probability that a (randomly selected) truly positive sample is correctly predicted. F1-score is the harmonic mean of precision and recall. Its value will be low if either precision or recall is low. These three metrics in (1)-(3) evaluate the classification results for each class.

For the evaluation of overall classification performance, *accuracy* is a commonly-used metric, which is defined as the ratio of correctly-predicted samples. However, it is not used here, because the error of the minority class can be easily masked by the majority class when two classes are imbalanced. Instead, AUC (i.e., area under the curve) is a more reasonable indicator of the overall performance, since it is not sensitive to imbalanced classes. AUC also works as a complementary metric to precision, recall and F1-score by exposing additional performance information. The details of the imbalanced-class case and AUC are introduced in Section II.D.

## B. Classification Algorithms

As the No Free Lunch Theorem [19] claims, there is no one best algorithm that does best in all cases. Therefore, several ML algorithms that we believe are suitable for the classification task are applied, and the one with the best performance is selected. From straightforward plots of samples, the decision boundary is expected to be highly nonlinear. Hence, we consider several supervised learning methods that are capable of classifying non-linearly separable data, such as *Kernel Support Vector Machine* [20, 21] and *Random Forest* (RF) [22]. Since RF outperforms other algorithms in the experiments, here we mainly give a brief introduction of RF.

RF is increasingly popular in recent years because of its good performance. Essentially it is an ensemble method based on decision trees. A decision tree learns a tree-structured model from the training samples, with each leaf representing a classification result. For each internal node, one feature is selected as the optimal split criteria at the current level. There are several metrics to measure the quality of a split. In this work, we use Gini impurity, which is commonly used in decision tree algorithms such as CART [23]. Gini impurity measures the probability that a sample in a set has the correct label if it is labeled randomly with the distribution of the overall set. Suppose we have $M$ possible classes and $p_i$ denotes the fraction of samples that belong to class $i$. Gini impurity $g$ is then computed as:

$$g = \sum_{i=1}^{M} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{M} p_i(1 - p_i) = 1 - \sum_{i=1}^{M} p_i^2. \tag{4}$$

Gini impurity reaches its minimum (zero) when all samples in the node fall into a single target category. At each node, the weighted sum of Gini impurity $g_{ws}$ of the left and right children after a split is computed as

$$g_{ws} = \frac{N_l}{N_c} g_l + \frac{N_r}{N_c} g_r, \tag{5}$$

where $N_c$ is the number of samples at the current node, $N_l$ and $N_r$ are the sample number in the left and right children of the current node, and $g_l$ and $g_r$ are the Gini impurity of the left and right children, respectively. The feature resulting in the minimal $g_{ws}$ is then chosen for the current split.

While a decision tree is easy to implement and interpret, it is not robust. A small change in the training samples can result in a totally different tree. A single decision tree is also prone to over-fitting the training set. A RF overcomes these disadvantages with ensemble learning [22]. A RF is an ensemble of decision trees and it has two degrees of randomness. Firstly, the training samples for each tree is generated from bootstrap sampling (random sampling with replacement) of the whole training set. In this way, each tree has a different subset of training data, although drawn from the same distribution. Secondly, when searching for the optimal split at each node, only a subset of all features are selected. A RF performs the final classification by taking a majority vote over the predictions made by each tree. From this process, a RF achieves much lower variance [24] than a single decision tree, at the expense of slightly increasing the fitting bias [24].

For RF, several hyper-parameters need to be carefully tuned for best prediction performance. These hyper-parameters control the structure of RF and thus determine the complexity of the overall mathematical model. An over-simple model has limited representational capacity to learn enough information from all the data (i.e., under-fitting), while an over-complex model tends to fit noise/outliers (i.e., over-fitting). The hyper-parameters we concern about are:

- The number of trees in a forest.
- The number of features to consider for each tree. With a higher value of this number, the bias of each tree decreases. However, the trees will be more similar and the randomness of the forest will decrease along with a higher risk of over-fitting.
- The maximum depth of each tree or the minimum number of samples required to split an internal node. It is also used to control the trade-off between the bias of each tree and the risk of over-fitting.

In practice, the values of the hyper-parameters can be determined by cross validation [25].

## C. Features

Besides suitable ML algorithms, using proper features is also a key factor for efficient classification. We extract 30 features that we believe represent important characteristics of fail logs (Table I).

**Feature Description**

The features mainly concern two attributes of a fail log: failing patterns and failing outputs. Suppose there are $n$ primary/scan outputs in a circuit, and a total of $m$ test patterns in the complete test set. The authors in [16] propose to use two vectors with dimensions $m$ and $n$ as *signatures* to represent each distinct fail log. These features, however, are not practical for today's complex ICs. Modern ICs usually have tens of thousands of outputs and are tested with thousands of test patterns, which produces an incredibly high-dimensional feature space. Directly using the data as features can easily cause over-fitting for ML. Therefore, it is necessary to find a

TABLE I
FEATURES EXTRACTED FROM FAIL LOGS.

| No. | Feature | Feature description |
|---|---|---|
| 1 | num_fail_pattern | number of failing patterns. |
| 2 | num_fo | total number of failing outputs (total number of failures). |
| 3 | num_uniq_fo | number of unique failing outputs. |
| 4 | max_fo | maximum number of failing outputs per failing pattern. |
| 5 | min_fo | minimum number of failing outputs per failing pattern. |
| 6 | mean_fo | average number of failing outputs per failing pattern. |
| 7 | num_fo_0 | total number of failing outputs having error value 0. |
| 8 | num_uniq_fo_0 | number of unique failing outputs having error value 0. |
| 9 | max_fo_0 | maximum number of failing outputs having error value 0 in one pattern. |
| 10 | min_fo_0 | minimum number of failing outputs having error value 0 in one pattern. |
| 11 | mean_fo_0 | average number of failing outputs having error value 0 in one pattern. |
| 12 | num_fo_1 | total number of failing outputs having error value 1. |
| 13 | num_uniq_fo_1 | number of unique failing outputs having error value 1. |
| 14 | max_fo_1 | maximum number of failing outputs having error value 1 in one pattern. |
| 15 | min_fo_1 | minimum number of failing outputs having error value 1 in one pattern. |
| 16 | mean_fo_1 | average number of failing outputs having error value 1 in one pattern. |
| 17 | num_fail_sc | number of failing scan chains. |
| 18 | fo_only_0 | number of failing outputs that only have error value 0. |
| 19 | fo_only_1 | number of failing outputs that only have error value 1. |
| 20 | fo_both | number of failing outputs that have error value both 0 and 1. |
| 21 | fail_pattern_0 | number of failing patterns that only have failures with error value 0. |
| 22 | fail_pattern_1 | number of failing patterns that only have failures with error value 1. |
| 23 | fail_pattern_both | number of failing patterns that have failures with error value 0 and 1. |
| 24 | pattern_single_sc | number of patterns that fail for only one scan chain. |
| 25 | pattern_multi_sc | number of patterns that fail for more than one scan chain. |
| 26 | mean_pattern_sc | nhe average number of scan chains each failing pattern fail for. |
| 27 | sc_single_pattern | number of scan chains that fail for only one pattern. |
| 28 | max_output_sc | maximum number of failing outputs per scan chain. |
| 29 | diff_max_output_sc | the difference between num_fo and max_output_sc. |
| 30 | fail_flush_test | whether the chip fails for flush test. |

smaller set of higher-level features that are representative of the fail-log information.

Obvious features pertinent to diagnosis, such as the number of failing patterns, the number of (total) failing outputs and the number of *unique* failing outputs are extracted. If a certain output fails for three times with respect to different test patterns, the number of failing outputs is three while the number of unique failing outputs is one. To characterize the behavior of a faulty circuit under different failing patterns, the max/min/average number of failing outputs for each pattern is also computed. In addition, the above calculations are repeated for failing outputs with failure values 0 and 1 respectively, as a decomposition of the total number of failures.

On the other hand, currently most integrated circuits are equipped with multiple scan chains to reduce test cost. Logic elements connected to the same scan chain are in close proximity, so a scan chain provides some physical information concerning logic elements. For example, suppose two scan chains A and B are connected to logic elements that are not in proximity, and thus share no common elements. If both A and B contain faulty outputs, it is an indicator of possible existence of multiple defects. Hence, feature 17 (num_fail_sc) is added among the collection of features. It is also intuitive that since a defective scan chain exhibits a long sequence of failing bits, a feature such as no. 28 (max_output_sc) and no. 24 (pattern_single_sc) may be helpful in identifying such defects. An additional feature, no. 30, reflects the flush-test outcome is also used, since flush-test failure is a strong indicator of chain failures (despite its imperfection mentioned in Section 1). In this way, we include the information from flush test for chain-failure prediction. The ML prediction result is expected to be more accurate than just the flush test, due to the additional information from other features.

It is obvious that some features in Table I are correlated, but we do not apply a feature dimension reduction process (e.g., principal component analysis) before training. The reasons are (i) the feature dimension is not high compared to the number of training samples; (ii) a RF can automatically identify important features during the learning process by feature selection for splitting; (iii) using the features directly are more convenient for model interpretation, especially for RF.

## Feature Importance

After the classifiers are trained, the quantification of feature importance provides crucial information to understand the trained models. We should note that the goal of calculating the feature importance is not to select a compact set of features for training, but to find important variables highly related to the objective variable for interpretation.

Feature importance can be conveniently calculated from a RF. Gini impurity reduction and permutation importance are two commonly-used metrics to measure feature importance [26]. In this paper, we use Gini impurity reduction bacause it can be conveniently calculated from the training of a RF. It measures how much each feature reduces the weighted impurity in a tree. The weighted Gini impurity reduction $Gini_{wr}$ is defined as:

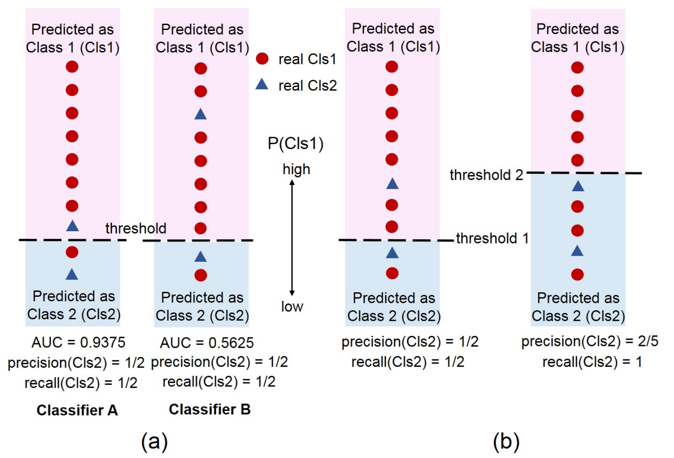


**Figure 6**. Ten samples are sorted according to their probability score of being in class C1 as reported by different classifiers. (a) Precision and recall of the two classifiers are the same, however Classifier A has higher AUC and better discriminating power. (b) With the same probability score ranking, different threshold values lead to different precision and recall.

$$Gini_{wr} = \frac{N_c}{N}(g_c - g_{ws}), \tag{6}$$

where $N$ is the total number of training samples, $N_c$ is the number of samples at the current node, $g_{ws}$ is defined in Eq.(5), and $g_c$ is the Gini impurity of the current node. For a RF, the impurity reduction from each feature is averaged and then are ranked according to this measure. In order to eliminate the mutual importance reduction effect among correlated features, we also use Recursive Feature Elimination (RFE) algorithm [27] to select the most important features.

## *D. Imbalanced Data*

During high-volume manufacturing, because the process is typically mature and thus yielding appropriately, it is less likely that multiple, interactive defects exist within a single chip. As a result, faulty behavior exhibited by a failing chip can be comprehensively diagnosed by commercial tools. This leads to few cases for which diagnosis fails to report any candidate for defects (i.e., diagnosis failure). In addition, it can also be true that few failures solely involve the scan chains especially when the amount of chain die area and layout-pattern irregularity is limited. Such cases lead to *imbalanced data*, that is, two classes that have extremely different sizes. Imbalanced data make model training and evaluation difficult. Here, we discuss the importance of using the metric AUC for imbalanced data, and demonstrate the trade-off between precision and recall for the minority class.

### Evaluation Metric

When two classes are not completely separable (i.e., two classes have overlap within their feature space), it is usually difficult to achieve high precision and recall simultaneously for the minority class. The reason is, in order to obtain a higher precision for the minority class, the boundary must be biased towards the other side to include more minority-class samples. At the same time, it inevitably includes more samples from the majority class, leading to a lower precision.

In order to evaluate the classifier performance in a more meaningful manner, we use the Area Under the Curve (AUC) metric. AUC refers to the area under Receiver Operating Characteristic (ROC) curve [28]. It represents the probability that a random positive sample ranks higher than a negative sample, and is not sensitive to imbalanced data. It is also a one-number statistic that adequately encodes the prediction goal, thereby allowing different classifiers to be easily compared. The AUC calculation uses the fact that many ML algorithms not only generate a classification result, but also a probability of one sample being labeled as positive. For example, in a RF, the class probability of an input sample is calculated as the mean predicted probabilities of the trees in the forest. The predicted probability of a tree is the fraction of samples of the same class in a leaf.

Fig. **6**(a) illustrates a case where AUC works as a complementary metric to precision and recall by providing additional performance information they fail to show. The ten samples in Fig. **6**(a) include eight samples of Class 1 (Cls1) and two samples of Class 2 (Cls2), an imbalanced case of 80% versus 20%. Classifiers A and B are applied to the samples and report the probability of each sample being in class Cls1. The samples are sorted according to the probability score. Those above a given threshold (shown as a dashed line in Fig. **6**) are classified as Cls1 and the ones below as Cls2. Since the minority class only has two samples, both classifiers have the same low values for precision and recall (both 0.5). However, it is obvious that classifier A performs better than classifier B, because for classifier A, the two Cls2 samples have probabilities more associated with a Cls2 prediction. By slightly increasing the threshold of Classifier A, both the recall and precision for Cls2 can be improved. However, we cannot observe the same improvement if we raise the threshold to the same extent for Classifier B. This difference of prediction performance is reflected in their AUC scores:

$$\mathrm{AUC_A} = 0.9375, \mathrm{AUC_B} = 0.5625.$$

Thus, classifier B performs much worse in separating the two classes.

Since AUC is not sensitive to imbalanced classes and is a one-value metric summarizing the classification capability, we use it to evaluate prediction performance, in addition to the precision, recall and F1-score. AUC is also used as the optimization goal when choosing hyper-parameters during cross validation.

### Trade-off Control

Because satisfactory precision and recall of a minority class cannot be simultaneously achieved when classes are imbalanced and not completely separable, it is desirable to control the trade-off between recall and precision of the minority class. Therefore, two strategies that effectively adjust the trade-off are described.

The first strategy adjusts the weights of the two classes. RFs are sensitive to imbalanced data. A RF may ignore the minority

TABLE II
CHARACTERISTICS OF TWO INDUSTRIAL EXAMPLES.

| **Name** | Chip 1 | Chip 2 |
|---|---|---|
| **Chip type** | Test chip | High volume chip |
| **Technology** | 28nm | 90nm |
| **No. of standard cells** | 4.4 million | 9.3 million |
| **No. of scan chains** | 12 | 103 |
| **Test set size** | 500 | 1000 |
| **No. of failing chips** | 4235 | 9301 |
| **Fail-log data collection time** | 70 days | 325 days |

TABLE III
THE NUMBER OF FAIL LOGS THAT FAIL THE FLUSH TEST.

| **No. of fail logs** | | **Chip 1** | **Chip 2** |
|---|---|---|---|
| **fail for flush test** | diagnosed as chain-failure | 230 | 112 |
| | diagnosed as logic-failure | 44 | 11 |
| | diagnosis unsuccessful | 1035 | 36 |

TABLE IV
PREDICTION RESULTS FOR CHIP 1.

**1 – Diagnosis Success**

| Confusion matrix | | Predicted 0 | Predicted 1 | Count | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|---|---|---|
| True labels | 0 | 1003 | 136 | 1139 | 0.94 | 0.88 | 0.91 | 0.96 |
| | 1 | 66 | 913 | 979 | 0.87 | 0.93 | 0.90 | |

**Label 0:** diagnosis failure **Label 1:** diagnosis success

**2 – Failure Type**

| Confusion matrix | | Predicted 0 | Predicted 1 | Count | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|---|---|---|
| True labels | 0 | 115 | 6 | 121 | 0.83 | 0.95 | 0.89 | 1.00 |
| | 1 | 23 | 1974 | 1997 | 1.00 | 0.99 | 0.99 | |

**Label 0:** scan-chain failure **Label 1:** not scan-chain failure

**3 – Runtime**

| Confusion matrix | | Predicted 0 | Predicted 1 | Count | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|---|---|---|
| True labels | 0 | 1591 | 163 | 1754 | 0.99 | 0.91 | 0.95 | 0.98 |
| | 1 | 16 | 348 | 364 | 0.68 | 0.96 | 0.80 | |

**Label 0:** short runtime **Label 1:** long runtime

class if the samples are too few to have much influence on the weighted Gini impurity calculation. The most direct solution for imbalanced data is to simply adjust the size of either class through over-sampling, down-sampling or the synthesis of new samples using interpolation [29]. Here, class weight adjustment is employed. That is, all the samples in the minority class are assigned a higher weight than the majority class when calculating the Gini impurity for a RF. If we set the weight ratio of minority and majority class to be $N$:1, it is equivalent to every sample in the minority class being re-sampled $N$ times. In this way, the classifier adds priority to the minority class leading to an improvement in minority-class prediction performance. This improvement for the minority class comes at the expense of more errors for the majority class. As a result, the minority class will have a higher recall but lower precision. By adjusting the weights assigned to each class, trade-off becomes straightforward.

Another strategy is to change the probability threshold for samples to be labeled positive. As shown in Fig. **6**(b), if we raise the threshold, more samples will be classified as Cls2 (minority class). But at the same time, more samples from Cls1 (majority class) are mistakenly classified as Cls2. Consequently, the recall of Cls2 increases at the expense of precision.

The effects of both strategies are proved in Section III.C.

## III. EXPERIMENT

In this section, we describe the details of two experiments that involve a logic test chip fabricated during technology development, and a second chip fabricated during high-volume production.

### A. *Setup*

Two types of industrial chips (Chip 1 and Chip 2) are used to demonstrate the efficacy of using learned models to predict diagnosis information. Characteristics of the two chips are listed in Table II. Chip 1 is fabricated in a 28nm process, and serves as a test chip for yield ramping. Chip 2 is fabricated in a 90nm mature process at high-volume. A substantial number of fail logs from both chips are obtained from industrial partners and each fail log is diagnosed using a commercial diagnosis tool.

For both examples, half of the data samples are used as training data and the remaining are used for test. Various ML algorithms are used in the training for each of the three classifiers, namely, C1 for predicting diagnosis success, C2 for predicting failure type (i.e., logic vs. chain failure) and C3 for predicting runtime. For all three classifiers, a RF has better performances than other algorithms for all the evaluation metrics, thus only the results of RF are presented. To deal with imbalanced data, we assign a *balanced weight* to each class, which is inversely proportional to the counts of samples in that class. The probability threshold for dividing two classes is set as the default value (0.5) for C1 and C3. For C2, we set the threshold a little bit higher (0.6) to make the recall of chain failure as close as to the flush-test result for comparison purposes.

### B. *Chip* 1

Table IV-1 shows the prediction results of C1. Chip 1 is a test chip fabricated in an immature process with many unknowns that lead to complex failures. As a result, diagnosis fails to report any defect candidates for 2,283 out of 4,235 fail logs, which exceeds more than 50% of all the available logs. From Table IV-1, it is observed that the precision and recall for both classes range from 87% to 94% and the AUC is as high as 0.96. This means C1 can effectively predict whether the diagnosis of a certain fail log tends to terminate with failure (i.e., no defect candidates).

The performance of C2 is given in Table IV-2. The ratio of scan-chain failure to all the samples is quite low (5%), which results in significant data imbalance. However, C2 achieves high precision and recall for both classes. The near perfect

TABLE V
PREDICTION RESULTS FOR CHIP 2.

**1 – Diagnosis Success**

| Confusion matrix | | Predicted 0 | Predicted 1 | Count | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|---|---|---|
| True labels | 0 | 119 | 21 | 140 | 0.29 | 0.85 | 0.43 | 0.96 |
| | 1 | 292 | 4219 | 4511 | 1.00 | 0.94 | 0.96 | |

**Label 0:** diagnosis failure **Label 1:** diagnosis success

**2 – Failure Type**

| Confusion matrix | | Predicted 0 | Predicted 1 | Count | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|---|---|---|
| True labels | 0 | 57 | 0 | 57 | 0.77 | 1.00 | 0.87 | 1.00 |
| | 1 | 17 | 4578 | 4594 | 1.00 | 1.00 | 1.00 | |

**Label 0:** scan-chain failure **Label 1:** not scan-chain failure

**3 – Runtime**

| Confusion matrix | | Predicted 0 | Predicted 1 | Count | Precision | Recall | F1-score | AUC |
|---|---|---|---|---|---|---|---|---|
| True labels | 0 | 1132 | 51 | 1183 | 0.91 | 0.96 | 0.93 | 0.99 |
| | 1 | 108 | 3360 | 3468 | 0.99 | 0.97 | 0.98 | |

**Label 0:** short runtime **Label 1:** long runtime

AUC also indicates the excellent classification capability of C2. The results also show that ML provides a more accurate prediction of diagnosed scan-chain failure than solely using flush test. In Table III, we show the diagnostic results of fail logs that fail the flush test. For Chip 1, all 230 fail logs diagnosed as chain failures are picked out by the flush test, indicating a 100% recall. However, 44 fail logs are diagnosed as logic-failure and 1,035 unsuccessfully diagnosed fail logs are also included, resulting in a low precision of 17.6%.

For runtime classification, 300 seconds is the threshold delineating *long* and *short* diagnostic execution/runtime. More than 80% of the samples have a short runtime. As shown in Table IV-3, C3 achieves a very high AUC of 0.98 and good precision and recall for both classes, except the precision of the minority class is slightly deteriorated due to the class imbalance.

## C. Chip 2

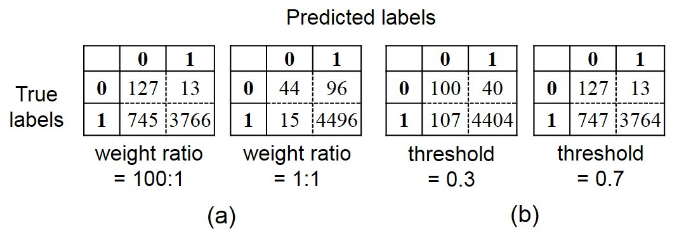
Chip 2 is an actual product chip manufactured in high-volume, with an imbalanced-data problem much more severe than Chip 1. Because the manufacturing process and design are both more mature, it is less likely that a failing chip exhibits faulty behavior that leads to diagnosis failure. This is evidenced by the fact that only 281 out of 9301 (3%) fail logs results in a reporting of zero defects, i.e., diagnosis failure. In addition, only 112 chips are diagnosed to have scan-chain failures.

The prediction results of the three classifiers for Chip 2 are listed in Table V. As described in Section II.D, it is difficult to obtain both high precision and recall for the minority class, as evidenced in Table V-1 and V-2. However, AUC values of these two classifiers are very high at 0.96 and 1.00, respectively. The prediction of runtime is however quite accurate and has an AUC value of 0.99. Similar to Chip 1, the results in Table III show that flush test has a low precision

TABLE VI
THE PREDICTED AND REAL COUNTS OF EIGHT DIFFERENT LABELS FOR CHIP 1 AND CHIP 2.

| No. | D | S | T | Chip 1 $\text{Cnt}_p$ | Chip 1 $\text{Cnt}_r$ | Chip 2 $\text{Cnt}_p$ | Chip 2 $\text{Cnt}_r$ | Comments |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 45 | 0 | Diagnosis is predicted to fail, but also predicted to have scan-chain defects. Such cases never happen in reality. |
| 2 | 0 | 0 | 1 | 8 | 0 | 18 | 0 | |
| 3 | 0 | 1 | 0 | 1002 | 1101 | 231 | 100 | Diagnosis is predicted to fail, and runtime is short. |
| 4 | 0 | 1 | 1 | 59 | 38 | 117 | 40 | Diagnosis is predicted to fail, and runtime is long. |
| 5 | 1 | 0 | 0 | 0 | 0 | 1 | 8 | Successful chain diagnosis with short runtime. Optimal outcome. |
| 6 | 1 | 0 | 1 | 112 | 121 | 10 | 49 | Successful chain diagnosis with long runtime. |
| 7 | 1 | 1 | 0 | 605 | 653 | 963 | 1075 | Successful logic diagnosis with short runtime. Optimal outcome. |
| 8 | 1 | 1 | 1 | 332 | 205 | 3266 | 3379 | Successful logic diagnosis with long run time. |

**Cnt_p**: the predicted count; **Cnt_r**: real count.
**D, S, T**: labels predicted by the three classifiers.
D = 0 (1): diagnosis fails (succeeds);
S = 0 (1): indicates (does not indicate) the existence of scan-chain defect;
T = 0 (1): the runtime is short (long).


Predicted labels
True labels
(a)
(b)

| True labels | weight ratio = 100:1, 0 | 1 | weight ratio = 1:1, 0 | 1 | threshold = 0.3, 0 | 1 | threshold = 0.7, 0 | 1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 127 | 13 | 44 | 96 | 100 | 40 | 127 | 13 |
| 1 | 745 | 3766 | 15 | 4496 | 107 | 4404 | 747 | 3764 |

**Figure 7**. The confusion matrices change with (a) different weight ratios for the two classes and (b) different classification thresholds. The results are from diagnosis success prediction of Chip 2. Label 0 means failure and Label 1 means success.

(70%) despite a 100% recall. ML, however, achieves a higher precision (77%) and an equal 100% recall.

As described in Section II.D, when dealing with imbalanced data, there are two ways to trade off between precision and recall of the minority class. The first strategy is to adjust the weight of the minority class, and the second one is to change the threshold of the classifier. Both methods are used to predict the diagnosis success of Chip 2. Fig. **7**(a) shows two confusion matrices using different weights. Compared to the case where weight ratio is simply 1:1, the classifier tends to predict more points as label 0 when the weight of the minority class is increased 100×. The recall of the minority class increases while the precision decreases. The effect of adjusting threshold is shown in Fig. **7**(b). In this case, setting a lower threshold tends to predict more samples as label 1, thus the recall of class labeled 0 becomes lower but the precision becomes higher. Raising the threshold has the opposite effect.

## D. Additional Observations

After obtaining the predicted results for all the testing samples, each fail log will have 8 possible outcomes of labels.

TABLE VII
TOP 5 RANKED FEATURES OF THE THREE CLASSIFIERS TRAINED ON CHIP 1 AND CHIP 2 DATA

| **Rank** | **Diagnosis success** | | **Failure type** | | **Runtime** | |
|---|---|---|---|---|---|---|
| | **Chip 1** | **Chip 2** | **Chip 1** | **Chip 2** | **Chip 1** | **Chip 2** |
| 1 | 1. num_fail_pattern | 6. mean_fo | 1. num_fail_pattern | 30. fail_flush_test | 24. pattern_single_sc | 1 .num_fail_pattern |
| 2 | 3. num_uniq_fo | 16. mean_fo_1 | 30. fail_flush_test | 3. num_uniq_fo | 29. diff_max_output_sc | 25. pattern_multi_sc |
| 3 | 25. pattern_multi_sc | 5. min_fo | 11. mean_fo_0 | 4. fo_only_1 | 2. num_fo | 28. max_output_sc |
| 4 | 26. mean_pattern_sc | 21. fail_pattern_0 | 6. mean_fo | 19. fo_only_1 | 1. num_fail_pattern | 26. mean_pattern_sc |
| 5 | 6. mean_fo | 11. mean_fo_0 | 9. max_fo_0 | 6. mean_fo | 17. num_fail_sc | 2. num_fo |

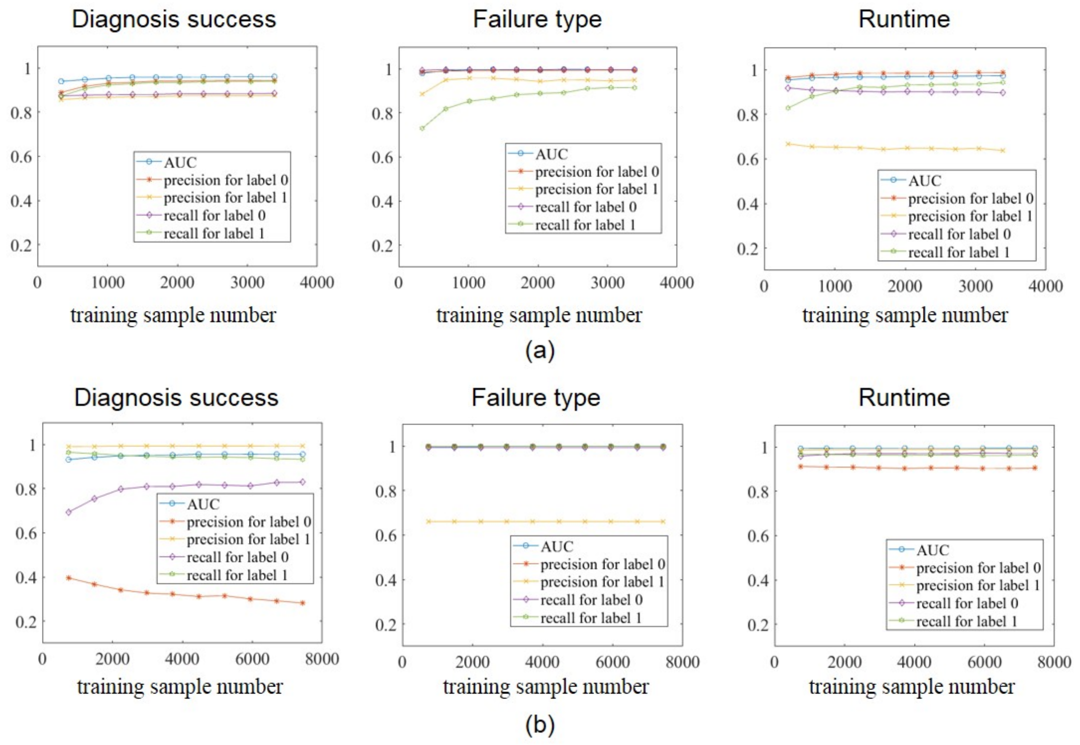


**Figure 8**. For the three classifiers, the change in precision, recall and AUC with respect to different training sizes are shown in (a) the result of Chip 1, and (b) the result of Chip 2. 20% of all the available samples are used as test set to calculate these metrics, while the size of training set varies from 10% ~ 100% of the remaining samples.

Table VI lists the predicted and real count, together with meaning of the 8 outcomes. Depending on the particular needs and resources within a particular manufacturing environment, some outcomes may be more advantageous than others. For example, the two rows shaded in dark blue (rows 5 and 7) correspond to cases where diagnosis succeeds and runtime is short. These two outcomes are likely the most desired since meaningful results can be obtained quickly. On the other hand, for cases predicted with conflicts (rows 1 and 2) and other cases predicted to fail (rows 3 and 4), these fail logs can be altogether skipped or scheduled to run last if resources are available. If the samples in the blue rows (rows 5-8) are not sufficient for yield learning, then the samples in rows 2 and 4 with short runtime can be diagnosed. Overall, the characterization results in Table VI aid the organization and scheduling of the diagnosis to achieve the most efficiency.

In addition to the good prediction performance shown in Tables IV and V, another benefit of a RF is that it provides feature importance. The top five important features for the six classifiers (three classifiers for each chip) are listed in Table VII. One can observe, despite some overlap, that the top-ranked features of the two chips are quite different. The differences are attributed to the fabrication technology and the makeup of the two chips. This result likely demonstrates there are no universal rules that can easily predict diagnosis success, failure type and runtime for all chips, thus validating the need to develop separate classifiers for each case by learning from the data. On the other hand, feature no. 30, which indicates whether the chip fails the flush test, has a very high rank in predicting failure type for both chips, which is consistent with our intuition.

Finally, we also conduct experiments to determine how many training samples are needed for good prediction results. We set 20% of the data as test set, and vary the training set size from 10% ∼ 100% of the remaining samples. For the same training size, we randomly select training samples and repeat the training and prediction process ten times to average out random fluctuation. The trends of AUC, precision and recall of Chip 1 are shown in Fig. **8**(a). It can be observed that the curves stabilize for training sets of 2,000 (59% of all the training samples) or larger. Thus, 2,000 training samples are sufficient for producing satisfying results.

More than half of survey respondents indicate that more than a thousand fail logs are diagnosed per week. Therefore, preparing a training dataset of the required size should not take much longer than a week or so. If the chip being diagnosed is large and only 100 can be diagnosed per week, it takes 20 weeks to collect the proper amount of training data, but recall can be compromised somewhat to reduce data preparation time. It should be also noted that as production proceeds, and more labeled data is generated due to completed diagnoses, re-training can take place using the larger data set.

Similar results for Chip 2 are given in Fig. **8**(b). Because the issue of imbalanced data is more severe for classifiers 1 and 2, the precision and recall of the minority class is always low. However, most curves stabilize for training sample sizes of 4,000 (57% of all the training samples) or larger. Again, if requirements are not too strict early on, then 2,000 samples is sufficient. It can be also observed that when the two classes are imbalanced (for failure type prediction), as the number of samples decreases, the precision of the minority class increases and the recall decreases. One possible reason is that, when the training size decreases, fewer samples of the minority class occur at the decision boundary while the majority class still has sufficient samples to guard the boundary. As a result, the boundary tends to retreat to the minority side, resulting in a lower recall but higher precision. One may also notice that for failure type and runtime, the prediction performance does not change much with more training data. One possible reason is that the performance becomes saturated (i.e., all the metrics except for one reaches 100%) with very few samples so the trained RF model seldom changes even with more training data.

## IV. Conclusion

In this work, a methodology is developed to provide a comprehensive preview of diagnostic outcomes beforehand. The predicted information includes three main aspects: diagnosis success, failure type and runtime. With such information, diagnosis resources can be prioritized in a more reasonable and effective way. Experiments show that the methodology can provide accurate prediction results (0.9+ precision, recall, F1-score and 0.96+ AUC) when the classification classes are balanced. Even when the classes are not balanced, our methodology still achieves satisfactory results and the users can effectively control the trade-off between the precision and recall for the minority class.

This work demonstrates that there is learnable correlation between fail-log data and the diagnostic information. Our future work will further exploit such correlation and extend both the feature space and prediction object space. One example is to predict diagnostic resolution with features extracted from test patterns.

## References

[1] K. Stanley, "High-accuracy flush-and-scan software diagnostic," *IEEE Design & Test of Computers*, vol. 18, no. 6, pp. 56–62, 2001.

[2] F. Yang *et al.*, "Detection of internal stuck-open faults in scan chains," in *International Test Conference*, 2008, pp. 1–10.

[3] D. Adolfsson *et al.*, "On scan chain diagnosis for intermittent faults," in *2009 Asian Test Symposium*, 2009, pp. 47–54.

[4] Y. Huang *et al.*, "Diagnose compound scan chain and system logic defects," in *International Test Conference*, 2007, pp. 1–10.

[5] J. E. Nelson, W. C. Tam, and R. D. Blanton, "Automatic Classification of Bridge Defects," in *International Test Conference*, 2010, pp. 1–10.

[6] H. Wang *et al.*, "Test-data Volume Optimization for Diagnosis," in *Design Automation Conference*, 2012, pp. 567–572.

[7] Y. Xue *et al.*, "PADRE: Physically-Aware Diagnostic Resolution Enhancement," in *International Test Conference*, 2013, pp. 1–10.

[8] Y. Xue, X. Li, and R. D. Blanton, "Improving Diagnostic Resolution of Failing ICs through Learning," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[9] C. Lim *et al.*, "Diagnostic Resolution Improvement through Learning-Guided Physical Failure Analysis," in *International Test Conference*, 2016, pp. 1–10.

[10] R. D. Blanton *et al.*, "DREAMS: DFM Rule Evaluation using Manufactured Silicon," in *International Conference on Computer-Aided Design*, 2013, pp. 99–106.

[11] R. D. Blanton *et al.*, "DFM Evaluation Using IC Diagnosis Data," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 463–474, 2017.

[12] W. C. Tam, O. Poku, and R. D. Blanton, "Systematic Defect Identification through Layout Snippet Clustering," in *International Test Conference*, 2010, pp. 1–10.

[13] W. C. Tam and R. D. Blanton, "LASIC: Layout Analysis for Systematic IC-Defect Identification Using Clustering," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1278–1290, 2015.

[14] L. R. Gómez and H.-J. Wunderlich, "A Neural-Network-Based Fault Classifier," in *Asian Test Symposium*, 2016, pp. 144–149.

[15] Y. Huang *et al.*, "Scan Chain Diagnosis Based on Unsupervised Machine Learning," in *Asian Test Symposium*, 2017, pp. 225–230.

[16] L. M. Huisman, M. Kassab, and L. Pastel, "Data Mining Integrated Circuit Fails with Fail Commonalities," in *International Test Conference*, 2004, pp. 661–668.

[17] Y. R. Ng *et al.*, "Scan shift debug using LVI phase mapping," in *International Symposium for Testing and Failure Analysis*, 2013, p. 322.

[18] I. H. Witten *et al.*, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2016.

[19] Wikipedia, "No free lunch in search and optimization," (https://en.m.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization).

[20] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[21] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *The fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[22] L. Breiman, "Random Forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[23] L. Breiman, *Classification and Regression Trees*. Routledge, 2017.

[24] Wikipedia, "Biasvariance tradeoff," (https://en.wikipedia.org/wiki/Bias-variance_tradeoff).

[25] Wikipedia, "Cross validation," (https://https://en.wikipedia.org/wiki/Cross-validation_(statistics)).

[26] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable Selection using Random Forests," *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2225–2236, 2010.

[27] B. Gregorutti, B. Michel, and P. Saint-Pierre, "Correlation and variable importance in random forests," *Statistics and Computing*, vol. 27, no. 3, pp. 659–678, 2017.

[28] T. Fawcett, "ROC graphs: Notes and Practical Considerations for Researchers," *Machine learning*, vol. 31, no. 1, pp. 1–38, 2004.

[29] N. V. Chawla *et al.*, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.