Towards Smarter Diagnosis: A Learning-based Diagnostic Outcome Previewer

QICHENG HUANG, CHENLEI FANG, SOUMYA MITTAL, and
R. D. (SHAWN) BLANTON, Carnegie Mellon University

Given the inherent perturbations during the fabrication process of integrated circuits that lead to yield loss, diagnosis of failing chips is a mitigating method employed during both yield ramping and high-volume manufacturing for yield learning. However, various uncertainties in the fabrication process bring a number of challenges, resulting in diagnosis with undesirable outcomes or low efficiency, including, for example, diagnosis failure, bad resolution, and extremely long runtime. It would therefore be very beneficial to have a comprehensive preview of diagnostic outcomes beforehand, which allows fail logs to be prioritized in a more reasonable way for smarter allocation of diagnosis resources. In this work, we propose a learning-based previewer, which is able to predict five aspects of diagnostic outcomes for a failing IC, including diagnosis success, defect count, failure type, resolution, and runtime magnitude. The previewer consists of three classification models and one regression model, where Random Forest classification and regression are used. Experiments on a 28 nm test chip and a high-volume 90 nm part demonstrate that the predictors can provide accurate prediction results, and in a virtual application scenario the overall previewer can bring up to 9× speed-up for the test chip and 6× for the high-volume part.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Hardware** → **Bug detection, localization and diagnosis**; **Online test and diagnostics**;

Additional Key Words and Phrases: Random forest, diagnosis economics, diagnosis preview

ACM Reference format:
Qicheng Huang, Chenlei Fang, Soumya Mittal, and R. D. (Shawn) Blanton. 2020. Towards Smarter Diagnosis: A Learning-based Diagnostic Outcome Previewer. *ACM Trans. Des. Autom. Electron. Syst.* 25, 5, Article 43 (August 2020), 20 pages.
https://doi.org/10.1145/3398267

1 INTRODUCTION

Given the fabrication perturbations inherent to integrated circuits (IC), yield learning is a crucial process to identify and mitigate sources of yield loss. Due to the virtually unlimited layout patterns that are created by new standard-cell libraries and the techniques employed by evolving place-and-route algorithms, yield learning for logic faces unique challenges for both process development and

Both authors contributed equally to this research.
This article is based on two earlier versions that appeared in 2018 IEEE International Test Conference (ITC) and 2019 IEEE VLSI Test Symposium (VTS).
Authors' addresses: Q. Huang, C. Fang, S. Mittal, and R. D. (Shawn) Blanton, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania, 15213; emails: {qichengh, chenleif, soumyami, rblanton}@andrew.cmu.edu.

1084-4309/2020/08-ART43 $15.00
https://doi.org/10.1145/3398267

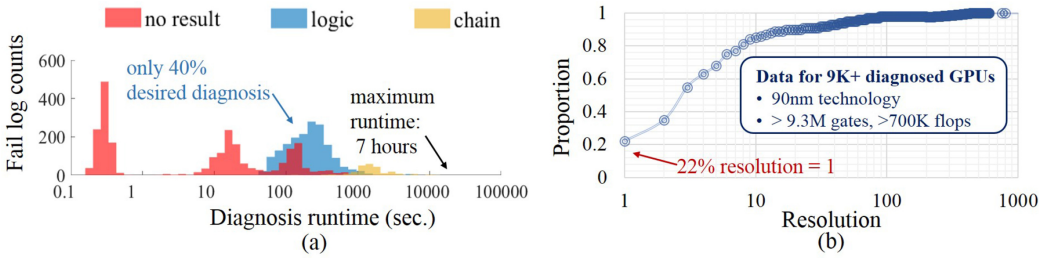

Fig. 1. Examples of challenges for diagnosis. (a) The histogram of diagnosis runtime for 4,235 28 nm test chips shows that only 40% fail logs result in successful logic diagnosis. The rest either fail to report any defect candidates or report candidates for chain failures. (b) Only 22% of 9K+ diagnosed GPUs have diagnostic reports of perfect resolution and more than 40% of them have a very poor resolution (larger than three).

high-volume manufacturing. Logic-circuit diagnosis, with the aim to both locate and characterize failure, serves as a key first step in yield learning under both scenarios.

Diagnosis is a software-based process that identifies possible locations and/or behaviors of defects in a failing chip. Diagnosis takes as input (i) design descriptions (i.e., a netlist and maybe its layout), (ii) test patterns, and (iii) tester response (also referred to as the fail log) of a failing chip. The quality of a diagnosis is usually characterized by *resolution*, which is defined as the number of candidates reported for possible defect(s),[1] and *accuracy*, defined as whether any reported candidates correspond to an actual defect. The output of a diagnosis tool often serves as a guide for physical failure analysis (PFA) to identify the root cause of the failure. Diagnosis results from a population of failing chips also serve as input to various volume-diagnosis techniques to reveal important statistics, including the defect distribution or the primary yield detractors [1, 2, 13, 15], and provide useful feedback for evaluating and improving the quality of manufacturing test and future diagnosis [9, 10, 17, 18].

While diagnosis plays an important role in yield learning, various uncertainties in the fabrication process bring a number of challenges, resulting in diagnosis with undesirable outcomes or low efficiency. For example, real defects may behave in an unexpected manner different from the fault models implicitly used by diagnosis tools. In addition, fault equivalence/dominance relationships and the existence of multiple defects further challenge the effectiveness of diagnosis tools. Such challenges may result in diagnosis outcomes characterized by low quality (e.g., poor resolution or inaccuracy), or even diagnosis failure (i.e., diagnosis exits without reporting any actionable information). Also, diagnosis computation time may be very long, which aggravates time-to-market pressure. Figure 1(a) shows the diagnosis runtime for 4,235 28 nm test chips using a state-of-the-art commercial diagnosis tool. The diagnosis runtime varies significantly—17% exceeds 1K seconds and the longest is about seven hours. Unfortunately, not all the diagnoses have desirable results: About 54% of the diagnoses (in red) are unsuccessful (i.e., fail to report any defect candidate), and 6% (in yellow) report chain failures. In other words, only 40% of the diagnoses (in blue) result in desired logic candidates. Figure 1(b), however, shows the distribution of resolution for over 9K diagnosed GPUs. Only 22% report a single candidate location (perfect resolution), which means a significant majority of the reports exhibit ambiguity. In particular, more than 40% of the diagnoses have poor resolution (i.e., number of candidates larger than three), which is usually too high for subsequent PFA.

[1]Please note that there may exist different definitions of resolution in IC test/diagnosis area. In this article, we define resolution as the number of candidates for a certain defect, so a higher resolution means a defect has more possible locations (candidates), therefore is less useful to pinpoint defect location for further investigation.

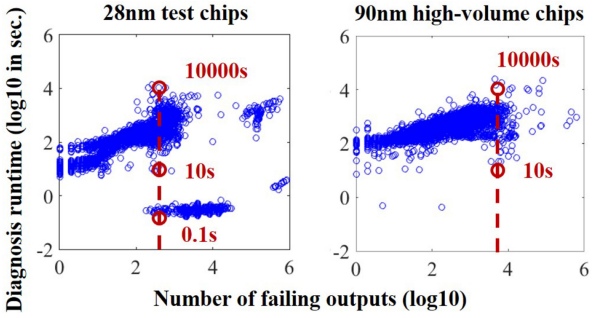

Fig. 2. Plots of diagnosis runtime versus a fail-log feature—the number of failing outputs—for 4,235 28 nm test chips (left) and 9,301 90 nm high-volume chips (right). There is no simple correlation between this feature and runtime.

According to one of our surveys from a dozen industrial practitioners from foundries, integrated device manufacturers, and fabless design houses, the number of diagnoses performed per week per design during technology development can be 10K, and one indicated that number exceeds 100K. Given the amount of diagnoses that can lead to no actionable information (e.g., 54% unsuccessful diagnoses in Figure 1(a) and 40+% diagnoses with very poor resolution in Figure 1(b)), it would be very beneficial to know beforehand which fail logs would result in actionable outcomes and which would result in either logic circuit or chain failure, since different yield learning techniques can be brought to bear. In a high-volume scenario, practitioners that are handling a large number of diagnoses indicate that sampling must be employed, implying that there are limited resources for diagnosis. Like technology development, resources can also be easily wasted if diagnosis results are not meaningful or not actionable (e.g., diagnosis failure, poor resolution, and/or large defect count). It would therefore again be very beneficial to know beforehand which fail logs would result in actionable information. Moreover, for a given set of fail logs, a constraint on diagnosis resources, and with all other things being equal, it is prudent to choose fail logs that have short runtime. The above observations motivate us to build a diagnosis previewer, which is able to predict the diagnostic outcomes (diagnosis success, resolution, runtime, etc.) before diagnosis even starts, so the practitioners can prioritize the fail logs for best allocation of diagnosis resources.

To predict diagnostic outcomes directly from fail-log information, however, is not trivial. Our preliminary analysis reveals that there is no **simple** correlation between fail-log features and diagnostic outcomes such as runtime or diagnosis success. For example, Figure 2 plots diagnosis runtime versus the number of failing outputs for a 28 nm test chip and a 90 nm high-volume part, respectively. For the same number of failing outputs, the runtime can vary by several orders of magnitude. The correlations of two chips are also different, meaning that there is no universal model for different chips. Fortunately, we found that there indeed exists **complex non-linear** correlations between fail-log features and diagnostic outcomes in a high-dimensional space. For example, after extracting 30 features from fail logs of 4,235 28 nm test chips, we use a dimension-reduction tool t-SNE[2] to map the 30 features into a three-dimensional space via non-linear transformation. The 4,235 samples with three mapped features are plotted in Figure 3 using different colors indicating different diagnosis outcomes with respect to three aspects: success, failure type, and runtime magnitude. The plots show that the diagnostic results are highly correlated to the

[2]t-SNE (t-Distributed Stochastic Neighbor Embedding) reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space [6].

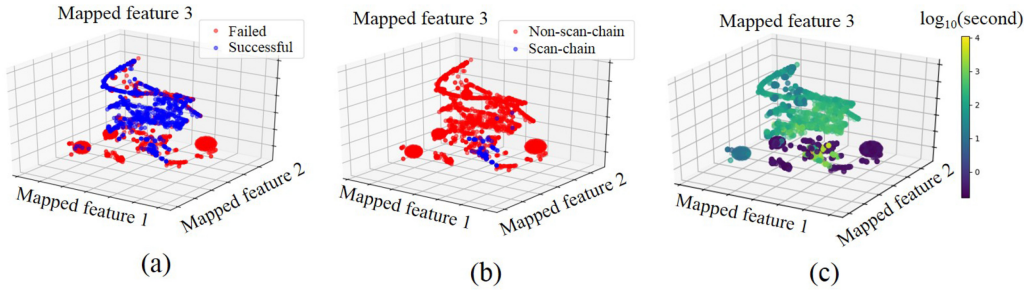

Fig. 3. Plots of samples corresponding to 4,235 28 nm test chips with respect to three aspects of diagnostic outcome: (a) success, (b) failure type, and (c) runtime magnitude. Each sample has three features mapped from 30 original features extracted from the fail log, and the mapping process is a non-linear transformation by t-SNE [6]. Different colors correspond to different outcomes, which are highly correlated to the three mapped features, indicating a non-linear complex correlation between diagnosis outcomes and the 30 original features.

three mapped features, which also implies a complex non-linear correlation to the 30 features in the original high-dimensional space. Due to the complexity of the correlation and the need to build models for different kinds of chips, we employ machine learning (ML) for its advantage in uncovering high-dimensional correlations to develop separate models for different IC designs.

ML has been broadly deployed in several prior works concerning diagnosis, which can be divided into three categories. The first category uses ML before diagnosis begins, with goal of preprocessing test data to assist diagnosis, such as to decide when to stop fail data collection while at the same time ensuring a high-quality diagnosis outcome [16]. The second category uses ML to improve diagnosis quality, including References [9, 11, 17, 18], where a support vector machine is learned from diagnostic results and physical information to predict whether a candidate is more likely to be the actual defect location. For the final category, ML is employed to interpret diagnosis results for better yield learning, such as to identify defect type using a random forest [12] or to discover the root cause of defects via statistical learning [1]. Our work belongs to the first category, but different from the assumption of Reference [16], we assume that the testing stage has ended and the available fail data is fixed. We believe that for the same type of design, if there exist correlations between the inputs (i.e., fail-log features) and outputs (diagnosis outcome) of the diagnosis tool, this correlation should be consistent through the diagnosed chips (for model training) and undiagnosed chips (wait to be predicted).

Specifically, the ML-based system in this work can provide a preview of diagnosis outcomes. We focus on several informative aspects indicating diagnosis quality and efficiency, including (i) diagnosis success, i.e., whether a diagnosis run successfully reports defect candidate(s); (ii) failure type, i.e., whether the reported candidate is located in a scan-chain or in the logic; (iii) existence of multiple defects (referred to as defect count); (iv) whether the diagnosis resolution is acceptable, and (v) the order of magnitude of diagnosis runtime, e.g., 1∼10 s, 10∼100 s. With such information in hand, we are able to evaluate the effectiveness of a diagnosis *a priori* and decide its priority to be executed accordingly, which helps allocate the diagnosis resources in a smarter way and expediates yield learning at minimal cost.

The five aspects of diagnosis outcomes are predicted by three classification models and one regression model, which are introduced in Section 2, together with details of the features used as input for the ML models. The performance of each model is demonstrated in Section 4, and we also demonstrate the efficacy of the overall previewer system by showing the diagnosis time reduction in a virtual application scenario. In addition, in case of imbalanced data, we provide details of how

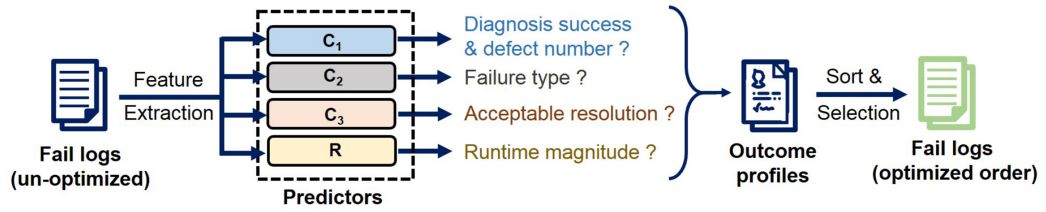

Fig. 4. An overview of the proposed preview system. The system takes in fail-log features and predicts five aspects of diagnostic outcome: diagnosis success, defect count, failure type, acceptable resolution, and runtime magnitude. The prediction forms a profile indicating diagnosis quality, according to which the fail logs can then be smartly organized to optimize diagnosis efficiency. The five aspects are predicted by three classification models ($C_1 \sim C_3$) and one regression model (R), where C_1 predicts both diagnosis success and defect number.

to properly evaluate the performance of classifiers and guidance to trade off between precision and recall in Section 3.

2 THE DIAGNOSIS PREVIEWER

The proposed diagnosis previewer takes in the features extracted from fail logs and predicts the diagnosis outcomes for each fail log. The prediction provides each fail log with a profile indicating the corresponding diagnosis quality and efficiency. Practitioners can prioritize the fail logs according to their profiles for optimal allocation of diagnosis resources (e.g., to diagnose the most promising fail logs first). Figure 4 illustrates the framework of the preview system. The prediction objectives and the problem formulation will be introduced in Sections 2.1 and 2.2, respectively. The details of the ML algorithms used and the extracted features from fail logs will be introduced in Sections 2.3 and 2.4, respectively.

2.1 Prediction Objectives

As shown in Figure 4, based on the features, the previewer predicts five aspects of diagnosis outcome by four separate predictors. These aspects are the key metrics indicating the quality and efficiency of a diagnosis run and provide a comprehensive preview of the diagnosis outcome.

- **Diagnosis success and defect count:** Based on the commercial survey we conducted, a majority of the practitioners indicated that they have encountered cases of unsuccessful diagnosis runs; specifically, three respondents indicated diagnosis fails to report any candidates more than 50% of the time. However, the existence of multiple defects within a failing IC is not desirable. The potential interaction among defects increases diagnosis difficulty and compromises accuracy. Also, the number of overall defect candidates also increases with defect number, which increases the complexity for PFA. With the capability to predict diagnosis success, fail logs that would result in reporting zero candidates can altogether be ignored or scheduled last. The fail logs predicted to have multiple defects can also be assigned a lower priority. Here, we put these two aspects together, because the success of diagnosis can actually be indicated by defect count—a diagnosis with zero defect count means the diagnosis tool fails to find any candidate for a failing circuit, and hence, is an unsuccessful diagnosis. Therefore, the two aspects are predicted by the same classifier C_1, with three possible labels: unsuccessful (indicated by "0"), single defect ("1"), and multiple defects ("2").
- **Failure type:** For a 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; thus, 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 Reference [14]. Failure type is predicted by classifier C_2 with a binary outcome: either logic ("0") or scan-chain ("1").

- **Resolution:** Since resolution implies the size of candidate space that practitioners need to explore for true defects, a very large resolution significantly reduces the likelihood of successful PFA. Thus, performing diagnosis on failing chips with a moderately high resolution is not a good use of resources. We therefore would ideally not run diagnosis on any fail log whose resolution is greater than some threshold. The third classifier C_3, hence, aims to predict whether a fail log would result in a diagnosis outcome with an acceptable resolution ("0" for yes and "1" for no). Typically, an acceptable resolution is not larger than three, because most practitioners from our survey indicated that they will not perform PFA on a failing chip with resolution larger than three.
- **Runtime magnitude:** Runtime information is an important aspect that indicates diagnosis efficiency, with which practitioners can easily avoid being burdened with a time-consuming or dead-end diagnosis run. In this work, we do not predict the exact runtime t, because the runtime depends on the hardware configuration of the computer and may vary even if we run the same job on the same computer. Instead, we predict the magnitude (defined as $\log_{10} t$ here) because, based on our survey results, users usually only need an approximation of the order of runtime magnitude. Different from the previous classification problems, runtime prediction is a regression problem and the regression model R outputs a real number.

The previewer is trained on a subset of fail logs, which are already diagnosed with all the information concerning diagnosis outcome available. The whole system is then applied to the remaining fail logs that have not been diagnosed yet. As outputs of the previewer, the different predicted aspects can be used as evaluation metrics to order diagnosis priority of the fail logs. These metrics can be freely selected and combined according to practitioners' needs. We show a practical usage example in Section 4.6.

2.2 Problem Formulation

In this section, we formally define the objective. Suppose d features are extracted from each fail log; they are then represented as a d-dimensional vector $\mathbf{x}$. The predictive result of classifier C_1 is encoded as a variable y_1 with three possible values 0, 1, and 2. The results of C_2 and C_3 are encoded as binary variables y_2 and y_3 (labeled 0 or 1), respectively. The variable y_4 concerning runtime is calculated as the logarithm with base 10 of the runtime.

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

To evaluate the performance of the four predictors C_1, C_2, C_3, and R of Figure 4, we need to define proper metrics. For the regression model R, we compare the predicted logarithm of runtime and the logarithm of real runtime and report the ratio of samples that have error within a certain

range. For classification models C_1, C_2, and C_3, the measuring metrics include *accuracy*, *precision*, *recall*, *F1-score,* and *area under curve* (AUC). Precision, recall, and F1-score measure the prediction performance of a specific class. Suppose a multi-class predictor gives prediction of K possible classes, then the precision, recall, and F1-score of a specific class k are calculated as Equation (1):

$$\begin{aligned} \text{Precision} &= \frac{M_{kk}}{\sum_{i=1}^{K} M_{ik}}, \\ \text{Recall} &= \frac{M_{kk}}{\sum_{i=1}^{K} M_{ki}}, \\ \text{F1-score} &= \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}, \end{aligned} \tag{1}$$

where M_{ij} is the number of samples that belong to class i and are predicted as class j. The precision of class k is the number of samples predicted to be in class k that are correct, and recall indicates how many true class-k samples are correctly identified. F1-score is the harmonic mean of precision and recall.

While the metrics in Equation (1) indicate the prediction performance for one specific class, accuracy shows the overall performance of the predictor with respect to all classes, which is defined as:

$$\text{Accuracy} = \frac{\sum_{i=1}^{K} M_{ii}}{\sum_{i=1}^{K} \sum_{j=1}^{K} M_{ij}} \quad . \tag{2}$$

Although accuracy is a convenient one-number metric, it cannot comprehensively capture the performance of the predictor when classes are highly imbalanced, which means the number of samples in one class dominates the other(s). For example, one can naively predict all the samples as the dominating class and still get a high accuracy. For significantly unbalanced scenarios it is also usually quite difficult to obtain a good precision and recall simultaneously for the minority class. We will discuss details of the imbalanced-data issue and why using another one-number metric AUC would be a better choice (Section 3).

2.3 ML Algorithms

For the three classification tasks and the regression task mentioned in Section 2.1, the *Random Forest* (RF) algorithm [3] is used due to its good performance in many ML applications. RFs are capable of both classification and regression tasks, so we use *Random Forest Classification* (RFC) for predictors $C_1 \sim C_3$ and *Random Forest Regression* (RFR) for the predictor R.

The final prediction of RFC and RFR comes from the averaging classification and regression results of decision trees (DTs). A DT recursively splits the data into two groups until a pre-specified stop condition is met. Before executing each split, DTs search over all binary splits of all variables for the one that minimizes a statistic indicating partition optimality. This optimality metric is exactly the main difference between using DTs for classification and regression.

While the commonly used optimality metric for RFC is Gini impurity [3], RFR uses metrics such as sum of squared errors (SSE) [6]. The SSE of a split result is defined as:

$$\text{SSE} = \sum_{i \in G_L} (y_i - \bar{y}_L)^2 + \sum_{j \in G_R} (y_j - \bar{y}_R)^2, \tag{3}$$

where G_L and G_R are the sample sets of the left and right children nodes, respectively; y_i and y_j are the values of samples in G_L and G_R, respectively; and $\bar{y}_L$ and $\bar{y}_R$ are the average values of the two sets.

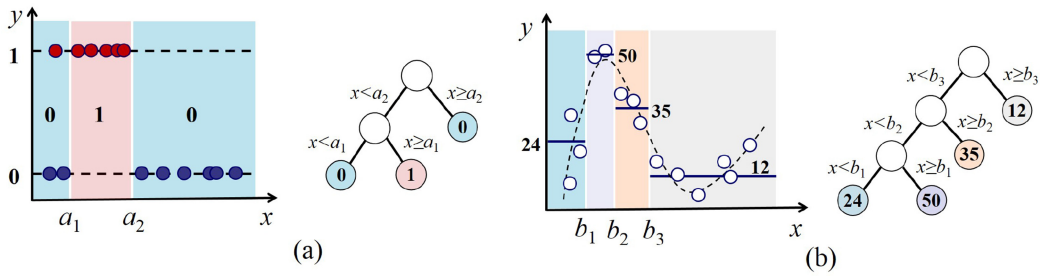

Fig. 5. Illustration of decision trees for (a) classification and (b) regression.

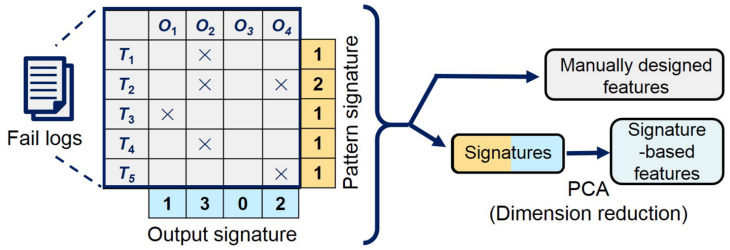

Fig. 6. An illustration of fail log feature extraction for ML. The features consist of two parts: fail-log signatures after dimension reduction and manually designed features. The table on the left shows a typical fail log, where × indicates a failure at the corresponding output and test pattern. The signature consists of two parts: output signature and pattern signature. PCA is applied to reduce feature dimensionality.

Figure 5 shows how a DT works for classification and regression problems with a single feature x. For the classification problem in Figure 5(a), the prediction objective is the label y with a binary value 0 or 1, while the regression objective y in Figure 5(b) is an arbitrary number. The partitioning methods of the two problems are similar with respect to the tree structures; the difference, however, is that for classification, the predicted y in each leaf node is the label of the majority class, but for regression, the predicted y value is the average of all the sample values in the node.

2.4 Feature Engineering

When practicing ML on real-world datasets, feature extraction can sometimes affect the prediction performance more than the algorithm. As illustrated in Figure 6, the features we use in this work come from two sources: manually designed features and signature-based features.

The manually designed features, which are shown in Table 1, incorporate domain knowledge and serve as a good abstraction of the fail logs. 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.

However, 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, if two scan chains A and B are connected to logic elements that are not in physical proximity, they do not share any

Table 1. Manually Designed 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	the average number of scan chains each failing pattern fails 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.

common element. 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.

The manually designed features incorporate domain knowledge and serve as a good starting point for feature designing. However, these features still miss important information that characterizes a failing chip. For example, many of the manually designed features include counts of failing outputs or failing patterns, which does not indicate the exact pattern or output that fails a certain chip. For example, assume we have two failing chips, each failing for one pattern at only one output, and both have the erroneous value 0. The failing output of the first chip has an input cone of 10K nodes, but the failing output of the second chip only has an input cone of 1K nodes. Even if some nodes can be ruled out by some intelligent diagnosis rules, we can expect the resolution and

runtime to be different for the two chips. However, the two chips have exactly the same values for the manually designed features. Using these features will not produce accurate results in this case.

To incorporate information of individual failing outputs and patterns into our framework, we use fail-log signatures, which are widely used for test compression. Authors of Reference [8] have used signatures as features to identify systematic defects. The table in the left of Figure 6 shows a typical fail log. For each failing pattern, i.e., $T_1 \sim T_5$, the failing outputs are recorded and marked with an "×." For each output, i.e., $O_1 \sim O_4$, the number of patterns it fails for is counted and recorded as an *output signature*. Similarly, the number of failing outputs for each test pattern is counted and recorded as a *pattern signature*. The two types of signatures are combined into a fail-log signature for each failing chip. Despite the rich information carried by signatures, we cannot directly use them for our case due to the high dimensional space. Specifically, modern ICs usually have hundreds of thousands of outputs and at least several hundreds of test patterns, which means the signature dimensional space will be enormous. Such a huge dimension will have several negative impacts: First, the computation cost will be too high to handle; second, the ML algorithm is much more likely to over-fit the dataset, especially when the number of data samples is much smaller than the feature dimension.

To address these dimensional issues, dimensionality reduction of the original features is necessary. Among a variety of dimension-reduction methods, principal component analysis (PCA) is widely used in the ML community, because it is both informative and efficient to compute. Intuitively, PCA projects the original features to another coordinate system. The projected directions are called principal components (PC). The first PC captures the direction where the dataset has the largest variance. The second PC is orthogonal to the first PC and is the direction with the second greatest variance. PCA is usually calculated from singular value decomposition (SVD) of the data matrix $\mathbf{X} \in \mathbb{R}^{p \times n}$, where p is the original feature dimension and n is the sample size:

$$\mathbf{X} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^T. \tag{4}$$

In the above equation, columns of $\mathbf{U}$ are the principal vectors, and columns of $\mathbf{V}$ are the coefficients for reconstructing the samples. Taking the first d components and the corresponding coefficients from $\mathbf{U}$ and $\mathbf{V}$ can reduce the feature dimension from p to d. By performing PCA on fail-log signatures, data dimension can be significantly reduced from over a hundred-thousand to a few hundred, which is a practical dimension for RF to give reasonable prediction. Manually designed features and signature-based features are combined into one feature vector, which provides information from both individual outputs and domain knowledge.

3 IMBALANCED DATA ISSUES

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. Accuracy in Equation (2) cannot properly represent the overall performance of the predictor, and it is usually very difficult to achieve good precision and recall at the same time for the minority class. In this section, we discuss the importance of using the metric AUC for imbalanced data and demonstrate the tradeoff between precision and recall for the minority class.

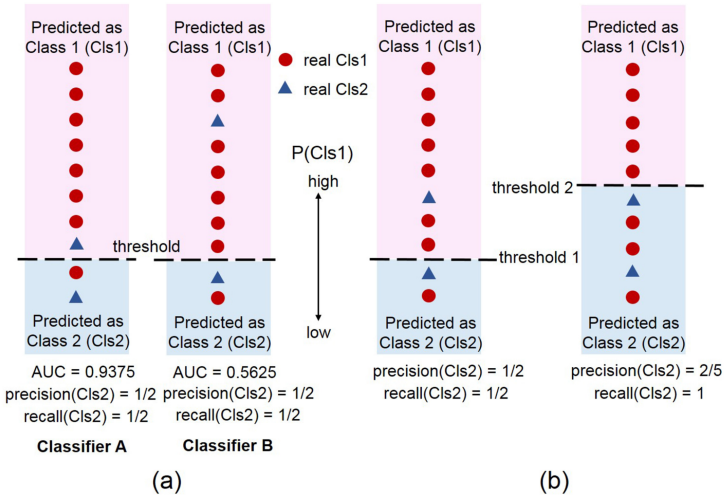

Fig. 7. Ten samples are sorted according to their probability score of being in class C1 as reported by different classifiers. (a) Precision and recall of the two classifiers are the same, however, Classifier A has higher AUC and better discriminating power. (b) With the same probability score ranking, different threshold values lead to different precision and recall.

3.1 Evaluation Metrics

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, 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.

To evaluate the classifier performance in a more meaningful manner, we use the metric AUC, which refers to the area under receiver operating characteristic (ROC) curve [5]. 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 an 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.

Figure 7(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 10 samples in Figure 7(a) include 8 samples of Class 1 (Cls1) and 2 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 Figure 7) 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.

Multi-class classification is also challenged by imbalance-data issues when one class dominates the others. Note that AUC is a metric only defined for binary classification, so when evaluating the three-class classifier C_1, we regard it as three "one-vs-all" classifiers. The AUC for C_1 is then defined as the average of the AUC values of these three classifiers:

$$AUC = \frac{1}{3}\left(AUC_{\text{0-vs-all}} + AUC_{\text{1-vs-all}} + AUC_{\text{2-vs-all}}\right). \tag{5}$$

3.2 Adaptive Tradeoff

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 tradeoff between recall and precision of the minority class. For example, to save diagnosis time, a practitioner might want to make sure no fail logs predicted with scan-chain failure are mistakenly included. At the same time, it is not very important to identify every fail log with respect to logic failure. In this case, a high precision for logic failure is preferred rather than recall. In contrast, recall is more important to identify all the possible logic defects when diagnosis resources are plentiful. Therefore, two strategies that effectively adjust the tradeoff are described for modulating the tradeoff between precision and recall.

The first strategy adjusts the class weights. An RF is sensitive to imbalanced data—it may ignore the minority 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 [4]. Here, class weight adjustment is employed. That is, all the samples in the minority class(es) are assigned a higher weight when calculating the Gini impurity for an RF. If we increase the weight of a minority class by N times, it is equivalent to every sample in the minority class being re-sampled N times. In this way, the classifier will be penalized more when each minority-sample is mistakenly predicted. As a result, the minority class will have a higher recall but lower precision. By adjusting the weights assigned to each class, tradeoff becomes straightforward.

Another strategy is usually only applicable to binary classification—to change the probability threshold for samples to be labeled positive. As shown in Figure 7(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.

4 EXPERIMENTS

In this section, we describe the details of experiments on two industrial chips. The results are based on test and diagnosis results of two types of industrial chips, with 4,235 and 9,301 instances, respectively.

We first evaluate the performance of the three classifiers and the regressor, respectively, on these chip instances. And then the performance of the whole system is evaluated in a virtual application

Table 2. Characteristics of Two Industrial Chips

Name	Chip 1	Chip 2
Chip type	Test chip	High volume chip
Technology	28 nm	90 nm
No. of standard cells	4.4M	9.3M
No. of scan chains	12	103
Test set size	500	1,000
No. of fail logs	4,235	9,301

scenario. For readers who want a quick preview of the results, we summarize the key points as follows:

- The defect count predictor and the failure-type predictor both achieve great performance—high AUC, accuracy, precision, and recall.
- The resolution predictor performs great on one of the two designs, with high AUC and accuracy.
- The runtime predictor gives accurate prediction of potential diagnosis runtime of a failed chip, with a small mean absolute error.
- In the virtual application scenario, the overall system achieves up to 9× and 16× speed-up for diagnosis of the two types of designs, respectively.

The trained RF models can be interpreted by analyzing the feature importance. We find the feature importance ranks for different types of chips are different, which supports the motivation of this work to learn different models for different types of chips. While we do not discuss model interpretation here due to space limitation, interested readers are encouraged to refer to Reference [7] for more details.

4.1 Setup

Since diagnosis is an essential step for both process development and high-volume manufacturing, two types of industrial chips (Chip 1 and Chip 2) are used in the experiments. Chip 1 is a logic test chip for process development of a 28 nm process. Chip 2 is manufactured in a mature 90 nm process in high volume. Their characteristics are listed in Table 2. Industrial partners executed circuit test on fabricated instances, and fail logs from 4,235 instances of Chip 1 and 9,301 instances of Chip 2 are available for analysis.

For each chip, half of the data are used for training and the rest are used for "testing," that is, for examining the quality of the predictors using various metrics. The features include manually designed features, pattern signatures after PCA, and output signatures after PCA, as described in Section 2.4. We keep the first 10 principal components for pattern signatures and the first 100 principal components for output signatures.

4.2 Defect Count Prediction

The defect-count prediction results for the two chips are shown in Table 3. If the diagnosis tool fails to report any candidate for some chips, the corresponding samples are labeled as 0. The samples with respect to single and multiple diagnosed defect(s) are labeled as 1 and 2, respectively.

Table 3 shows that the trained classifier C_1 achieves overall test accuracy of 0.85 for Chip 1 and 0.84 for Chip 2, and the AUC values for both chips are over 0.93. These results show that the

Table 3. Prediction of Defect Count for (a) Chip 1 and (b) Chip 2

Confusion matrix		Predicted 0	1	2	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	1028	101	10	1139	0.95	0.90	0.93	0.85	0.95
	1	50	575	61	686	0.74	0.84	0.79		
	2	4	97	192	293	0.73	0.66	0.69		

(a)

Confusion matrix		Predicted 0	1	2	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	86	53	1	140	0.88	0.61	0.72	0.84	0.93
	1	12	2653	311	2976	0.87	0.89	0.88		
	2	0	355	1180	1535	0.79	0.77	0.78		

(b)

Label 0: fail **Label 1:** single **Label 2:** multiple

trained classifier is quite effective in predicting whether the diagnosis of the chip will report zero, single, or multiple defects.

Since Chip 1 is a logic test chip manufactured in an immature process with many unknown defects, the diagnosis tool fails to report any candidates for 54% of chips. Table 3 shows that the classifier is able to pick out most of the chips that will result in an unsuccessful diagnosis, i.e., the samples with label 0. As running diagnoses on these chips is a waste of diagnosis resources, with such a prediction result, practitioners can save scarce resources by omitting these fail logs from analysis. However, Chip 2 is an actual product chip manufactured in high volume. 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 140 out of 4651 (3%) fail logs result in a reporting of zero defects, i.e., diagnosis failure. In this case, the predictor is still capable of identifying most of the fail logs that will lead to diagnosis failure.

For a successful diagnosis (i.e., diagnosis reports defect candidates), the more desirable outcome involves a single defect, because accuracy tends to be much higher. Specifically, a circuit with a single defect is much easier to diagnose than those with multiple defects—the former is more probable to be successfully diagnosed and typically requires shorter runtime. Therefore, accurately identifying fail logs with a single defect helps practitioners to obtain evidence concerning fabrication issues much more efficiently. The results in Table 3 show that the classifier achieves high precision and recall for single-defect prediction. This means that the classifier can not only identify most of the fail logs with a single defect, but also successfully exclude most multiple-defect circuits that are inefficient to diagnose. In this way, we guarantee very limited information loss from single-defect ICs and also enlighten allocation of diagnosis resources for multiple-defect cases.

4.3 Failure-type Prediction

Classifier C_2 predicts the failure type of each failed chip (i.e., scan-chain or logic failure). The results are shown in Table 4. In this experiment, we exclude chips for which diagnosis fails to report any candidate, as these chips can be identified by defect number predictor C_1. The ratio of scan-chain failure to all the samples is quite low (5% for Chip 1 and 1% for Chip 2), which results in significant data imbalance. However, it can be observed from the table that the classifier C_2 achieves high precision and recall for both classes of Chip 1 and Chip 2. The near-perfect AUC also indicates the excellent classification capability of C_2. With this predictor, practitioners can select strategies for each type of failure based on their goal for yield learning.

Table 4. Prediction of Failure Type for (a) Chip 1 and (b) Chip 2

Confusion matrix		Predicted 0	Predicted 1	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	854	4	858	1.00	1.00	1.00	0.99	1.00
	1	2	119	121	0.97	0.98	0.98		

(a)

Confusion matrix		Predicted 0	Predicted 1	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	4450	4	4454	1.00	1.00	1.00	0.99	1.00
	1	0	57	57	0.93	1.00	0.97		

(b)

Label 0: logic failure **Label 1:** scan chain failure

Table 5. Prediction of Resolution for (a) Chip 1 and (b) Chip 2

Confusion matrix		Predicted 0	Predicted 1	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	658	48	706	0.85	0.93	0.89	0.84	0.87
	1	112	167	279	0.78	0.60	0.68		

(a)

Confusion matrix		Predicted 0	Predicted 1	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	1983	828	2811	0.76	0.71	0.73	0.67	0.72
	1	633	1056	1689	0.56	0.63	0.59		

(b)

Label 0: acceptable **Label 1:** unacceptable

4.4 Resolution Prediction

The results for resolution prediction are shown in Table 5. Similar to the experiment for C_2, we exclude chips for which diagnosis fails to report any defect candidates. Chips with resolution ≤ 3 are defined to have acceptable resolution and are labeled 0. Table 5(a) shows that the accuracy for Chip 1 is 0.84 and the AUC value is also high at 0.87. Specifically, recall for class 0 is high (0.93), meaning that few chips with acceptable resolution are predicted incorrectly. We therefore do not lose much information from diagnosing such chips while saving significant computation resources from diagnosing chips with poor resolution.

However, resolution prediction for Chip 2 is not very successful, with accuracy being 0.67 and AUC being 0.72. A possible reason is that Chip 2 is a much more complex circuit compared to Chip 1. Chip 2 has over 700K outputs. Keeping only 100 principal components of output signatures is likely not sufficient. However, increasing the number of components not only increases computation time, but also increases the chance of over-fitting. The best way to enhance its performance is to acquire more training data; however, it is not possible in this case, since the chip is no longer being fabricated. Another possible solution is to use data augmentation methods to generate virtual fail logs as more training samples, which is a focus of our future work.

For all the three classifiers C_1, C_2, and C_3, we also conduct experiments to determine how many training samples are needed for good prediction results. We designate 20% of the data as test set and vary the training set size from 10% $\sim$ 100% for the remaining 80%. For the same training size, we randomly select training samples and repeat the training and prediction process 10 times to average out random fluctuation. We have observed that for all classifiers, most performance curves

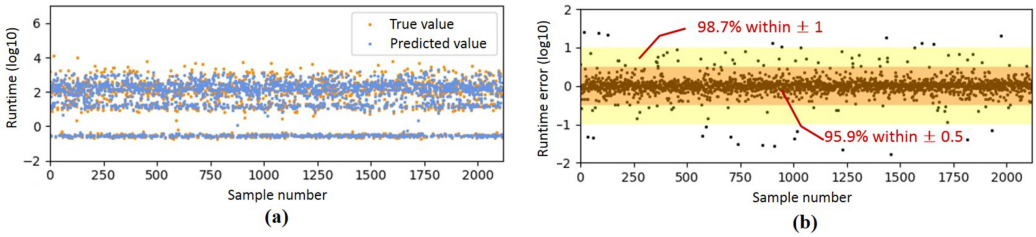

Fig. 8. Distribution of (a) runtime and (b) prediction error for Chip 1.

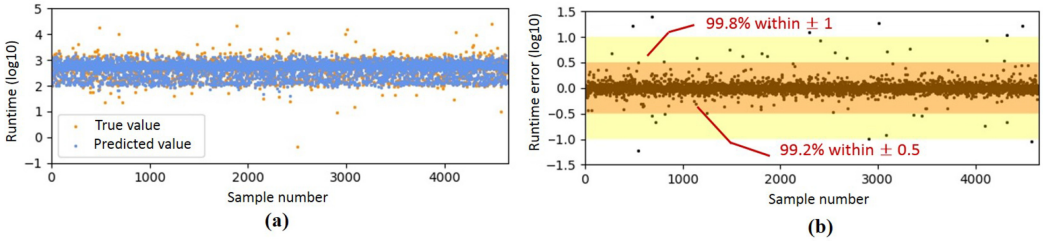

Fig. 9. Distribution of (a) runtime and (b) prediction error for Chip 2.

Table 6. Error Distribution of Runtime Magnitude Prediction

ε_{th}	Ratio of samples with |error| < ε_{th}	
	Chip 1	Chip 2
0.1	61.2%	86.3%
0.3	89.7%	98.3%
0.5	95.9%	99.2%
1.0	98.7%	99.8%

stabilize with training sets of 2K samples for Chip 1 and 4K for Chip 2. Thus, 2K and 4K training samples are sufficient for Chip 1 and Chip 2, respectively, to produce satisfying classification results.

4.5 Runtime Prediction

Figure 8 and Figure 9 show the prediction results of runtime for Chip 1 and Chip 2, respectively. Figure 8(a) and Figure 9(a) show the real and predicted order of magnitude of runtime for all the testing data, while Figure 8(b) and Figure 9(b) show the prediction errors. In both prediction error plots, most of the error values concentrate around zero. The orange band represents the range that has absolute error smaller than 0.5, while the yellow band represents 1. Over 95% of samples of Chip 1 and over 99% of samples of Chip 2 have absolute error below 0.5, which means that the real runtime lies within the range between one-third and 3× of the predicted runtime. It is not exact, but sufficient for an estimation of the order of magnitude of runtime. A more detailed summary of error distribution is shown in Table 6, which lists the ratio of samples with absolute error smaller than different thresholds. Specifically, 61.2% of the Chip 1 samples and 86.3% of the Chip 2 samples have absolute error smaller than 0.1, respectively, meaning the true runtime lies within the range of 0.8× to 1.3× of the predicted time. These results show that the learned predictor can provide a

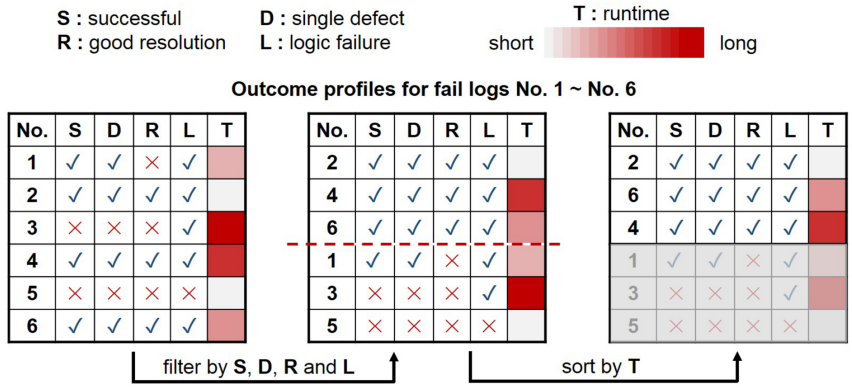

Fig. 10. A simple example of six fail logs for a virtual application of the proposed preview system. The objective is to find fail logs that will result in diagnoses with high quality (successful, reporting single logic defect, and with good resolution) and high efficiency (with short runtime). Based on the prediction results from the previewer, each fail log can obtain a profile indicating its diagnosis outcome, specified by whether it satisfies four conditions indicating the diagnosis quality (**S**, **D**, **R**, and **L**) and its runtime magnitude **T**. After filtered by **S**, **D**, **R**, and **L** and sorted by **T**, the fail logs with high quality are picked out and arranged by runtime.

very reasonable estimation of diagnosis runtime. Practitioners, therefore, can use such information for better allocation of diagnosis resources.

To further demonstrate the efficacy of the runtime predictor, we calculate the mean absolute error (MAE) of the order of magnitude of runtime with different training size. In this experiment, we keep 20% data as the test set and run regression using different training set sizes. The MAE for Chip 1 is as small as 0.18 even when the training size is only 340 samples, meaning that the real runtime range lies within 0.6× to 1.5× of the predicted runtime. The MAE for Chip 2 is even lower. It achieves 0.06 with 2K training samples, meaning the true runtime range is 0.9× to 1.2× of the predicted runtime, which is a very accurate estimation.

4.6 Fail Log Optimization

With the predicted outcome from the previewer, practitioners can create a profile for each fail log and organize the fail logs according to their specific need. In this subsection, we explore the efficacy of the whole preview system in a virtual application scenario where all the five aspects of predicted diagnostic outcome can be utilized.

Suppose the practitioner cares about all the five aspects and the goal is to run N "good" diagnoses for troubleshooting. A "good" diagnosis has both good quality and high efficiency, which is characterized by the following criteria:

(1) It is **successful**.
(2) It reports a **single** defect.
(3) It reports that the failure is due to a **logic** defect.
(4) It has a **good** resolution (no larger than three).
(5) It has a **short** runtime.

To this end, the five predicted aspects of diagnosis outcome for all the fail logs are summarized into a profile list, and all the fail logs that satisfy criteria (1)~(4) are selected out and then sorted by runtime. Figure 10 illustrates this process by a simple example of six fail logs. Each row in the table represents a profile for a certain fail log, showing whether the corresponding diagnosis is

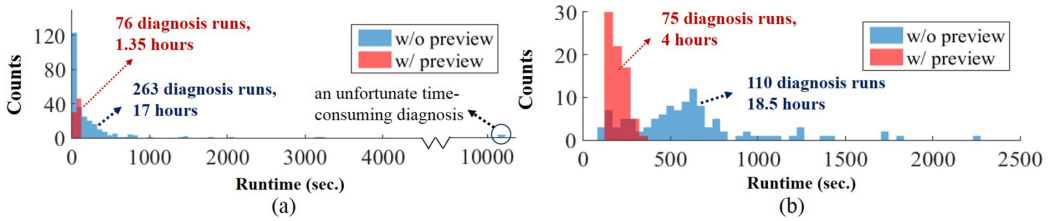

Fig. 11. Runtime histogram of all the diagnoses needed until executing 40 "good" runs, comparing two cases (with and without the previewer). The experiments are conducted for both (a) Chip 1 and (b) Chip 2. With the help of previewer, it takes far fewer diagnosis runs and less time to achieve the same diagnosis goal.

predicted to satisfy criteria (1)~(4) (**S**, **D**, **R**, and **L**) and the estimated runtime magnitude. First, we select all fail logs that satisfy the criteria (1)~(4) and leave the remaining aside. Note that, since the four aspects are predicted separately, there can be contradiction between classifiers. For example, the profile of fail log No. 3 indicates both diagnosis failure and reported logic failure (which implies diagnosis success). For such cases, we give more belief to the success prediction. In other words, only the fail logs with all check marks in the profile will pass the filter. After filtering, a set of potential "good" diagnoses is available, but we still want to run them in order of increasing runtime. Therefore, a sorting process is further executed for the selected fail logs with respect to the **T** column. Eventually, a list of fail logs is ordered by how well they are predicted to match the practitioner's expectation. Then, we run diagnosis according to this order until N "good" diagnosis runs are actually executed.

For comparison purposes, we also run diagnoses on fail logs based on a random order until N "good" diagnoses have resulted. The comparison is performed for both Chip 1 and Chip 2 with $N = 40$, and the results are shown in Figures 11(a) and (b), respectively. As demonstrated in the histogram of Figure 11(a), without the aid of the previewer, 263 diagnoses have to be executed to result in 40 "good" ones. Due to an extremely time-consuming diagnosis of more than 10K seconds, the total runtime is 17 hours. In contrast, with the previewer, we only need to execute 76 diagnoses and they are all very fast, which leads to a total runtime of only 1.35 hours. For Chip 2, illustrated in Figure 11(b), fortunately no diagnosis run with extremely long runtime is encountered in the randomly selected fail-log set, but the previewer still saves 35 diagnoses and 14.5 hours in total.

To further validate the performance of the previewer for more general cases, we repeat the previous experiment for more values of N varying from 20 to 180. For each N, we take the average value of five runs. In each run, different random seeds are used for RF, and in the case without previewer, different set of fail logs is randomly diagnosed. Figures 12(a) and (b) show the results for Chip 1 and Chip 2, respectively. For both the number of diagnosed fail logs and the overall diagnosis time, we observe significant resource savings brought by the previewer. For example, when $N = 80$, we achieve 9.25× and 4.59× speed-ups in diagnosis runtime for Chip 1 and Chip 2, respectively. For Chip 2, the maximum speed-up is 6× occurring when $N = 20$. Also, the curve for the "w/ preview" case has a much smaller slope than the curve for the "w/o preview" case, implying the difference between the two curves will further increase with larger N and the time savings resulting from the previewer will be even more significant.

This experiment only shows one possible application of the previewer, and practitioners can utilize the prediction results in different ways per their specific needs. For example, if time permits and the selected "good" fail logs are not sufficient to achieve the diagnosis goal, some predicted "sub-optimal" fail logs can also be diagnosed (e.g., No. 1 in Figure 10).

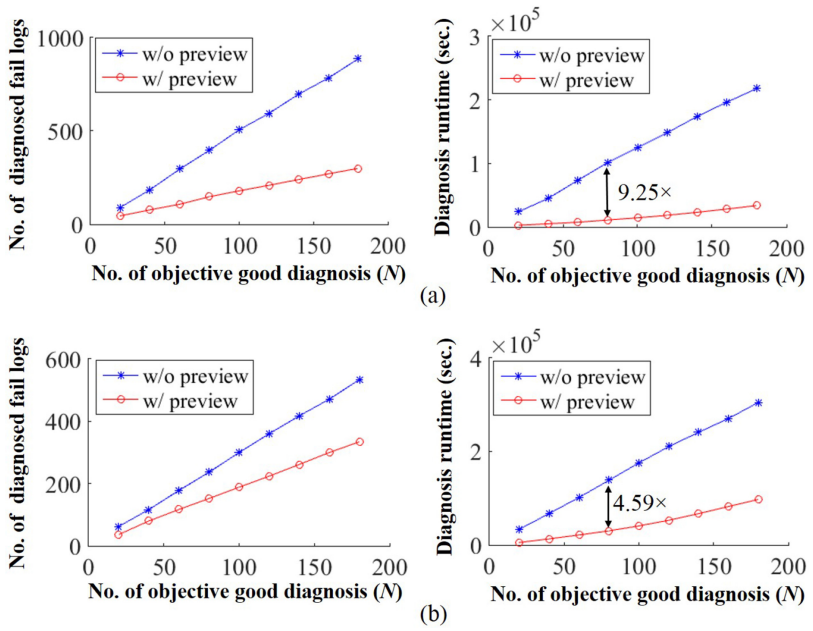

Fig. 12. Number of diagnosed fail logs and total diagnosis runtime for different number of "good" diagnoses we aim to execute (N) for (a) Chip 1 and (b) Chip 2. With the aid of the previewer, far fewer diagnosis runs and less total diagnosis runtime is needed for the same number (N) of objective good diagnoses.

5 CONCLUSION

In this work, we propose a learning-based diagnostic outcome previewer. With features extracted from fail logs, the previewer is able to predict five aspects of diagnostic outcome for each fail log, including diagnosis success, defect count, failure type, resolution, and runtime magnitude. Such predicted information indicates the quality and efficiency of the diagnosis with respect to a certain fail log, thus allowing practitioners to prioritize fail logs accordingly for best allocation of diagnosis resources. Random forests are employed for both classification and regression tasks. In experiments on a 28 nm test chip and a high-volume 90 nm part, the predictors can provide accurate prediction results for each aspect. In a virtual application scenario, the overall previewer can result in up to 9× speed-up for the test chip and 6× for the high-volume part.

This work demonstrates that there exist learnable correlations between fail-log features and diagnosis outcome and that random forest is an effective model to represent such correlations. Possible future research directions include considering those rare examples in defect count and failure-type prediction as "anomalies" and constructing an anomaly detector to better identify these samples. Furthermore, our work only shows a static scenario, where the training set is fixed. Diagnosis, however, is often considered as a continuous process. With more fail logs getting predicted and diagnosed, more training data can be collected, so how to update the model with more upcoming data and a possible shifted data distribution would also be an interesting topic to explore. In addition, instead of "passively" waiting for the samples to train on, active learning strategies could be adopted to select the samples to be labeled, which may lead to a better accuracy with fewer diagnosed samples for training.

REFERENCES

[1] B. Benware, C. Schuermyer, Manish Sharma, and Thomas Herrmann. 2012. Determining a failure root cause distribution from a population of layout-aware scan diagnosis results. *IEEE Des. Test Comput.* 29, 1 (2012), 8–18.

[2] Ronald D. Blanton, Wing Chiu Tam, Xiaochun Yu, Jeffrey E. Nelson, and Osei Poku. 2012. Yield learning through physically aware diagnosis of IC-failure populations. *IEEE Des. Test Comput.* 29, 1 (2012), 36–47.

[3] Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* 16 (2002), 321–357.

[5] Tom Fawcett. 2004. ROC graphs: Notes and practical considerations for researchers. *Mach. Learn.* 31, 1 (2004), 1–38.

[6] Aurélien Géron. 2017. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, Inc.

[7] Qicheng Huang, Fang Chenlei, Soumya Mittal, and Ronald D. Blanton. 2018. Improving diagnostic efficiency via machine learning. In *Proceedings of the International Test Conference.*

[8] Leendert M. Huisman, Maroun Kassab, and Leah Pastel. 2004. Data mining integrated circuit fails with fail commonalities. In *Proceedings of the International Test Conference.*

[9] Carlston Lim, Yang Xue, Xin Li, Ronald D. Blanton, and M. Enamul Amyeen. 2016. Diagnostic resolution improvement through learning-guided physical failure analysis. In *Proceedings of the International Test Conference.*

[10] Yen-Tzu Lin and Ronald D. Blanton. 2011. METER: Measuring test effectiveness regionally. *Trans. Comput.-Aided Des. Integ. Circ. Syst.* 30, 7 (2011), 1058–1071.

[11] Soumya Mittal and Ronald D. Blanton. 2019. LearnX: A hybrid deterministic-statistical defect diagnosis methodology. In *Proceedings of the IEEE European Test Symposium.*

[12] Jeffrey E. Nelson, Wing Chiu Tam, and Ronald D. Blanton. 2010. Automatic classification of bridge defects. In *Proceedings of the International Test Conference.*

[13] Jeffrey E. Nelson, Thomas Zanon, Jason G. Brown, Osei Poku, Ronald D. Blanton, Wojciech Maly, Brady Benware, and Chris Schuermyer. 2006. Extracting defect density and size distributions from product ICs. *IEEE Des. Test Comput.* 23, 5 (2006), 390–400.

[14] Yin Roy Ng, Howard Marks, Christopher Nemirov, Chun-Cheng Tsao, and Jim Vickers. 2013. Scan shift debug using LVI phase mapping. In *International Symposium for Testing and Failure Analysis.* 322.

[15] Wing Chiu Tam, Osei Poku, and Ronald D. Blanton. 2008. Precise failure localization using automated layout analysis of diagnosis candidates. In *Proceedings of the 45th Annual Design Automation Conference.* 367–372.

[16] Hongfei Wang, Osei Poku, Xiaochun Yu, Sizhe Liu, Ibrahima Komara, and Ronald D. Blanton. 2012. Test-data volume optimization for diagnosis. In *Proceedings of the Design Automation Conference.*

[17] Yang Xue, Xin Li, and Ronald D. Blanton. 2016. Improving diagnostic resolution of failing ICs through learning. *Trans. Comput.-Aided Des. Integ. Circ. Syst.* 37, 6 (2016), 1288–1297.

[18] Yang Xue, Osei Poku, Xin Li, and Ronald D. Blanton. 2013. PADRE: Physically-aware diagnostic resolution enhancement. In *Proceedings of the International Test Conference.*

Received May 2019; revised November 2019; accepted May 2020