Diagnosis Outcome Preview through Learning

Chenlei Fang, Qicheng Huang, 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
{chenleif, qichengh, soumyami, rblanton}@andrew.cmu.edu

***Abstract*—Logic diagnosis is a software-based methodology to identify the behavior and location of defects in failing integrated circuits, which is an essential step in yield learning. However, diagnosis can be time-consuming and produce unuseful information for further investigation of yield loss. It would therefore be desirable to have a preview of diagnosis outcomes beforehand, which helps engineers allocate diagnosis resources in a more efficient way. In this work, random forest classification and regression techniques are used to predict three aspects of potential diagnosis outcomes: existence of multiple defects, diagnosis resolution, and runtime magnitude. Experiments on a 28nm test chip and a 90nm high-volume manufactured chip prove the efficacy of the proposed methodology - high accuracy for multiple defect (up to 0.86 accuracy and 0.93 AUC) and resolution prediction (up to 0.84 accuracy and 0.87 AUC), and over 98% of the runtime magnitude prediction is within the error of one magnitude. These results prove that the method can provide helpful information to guide prudent allocation of diagnosis resources.**

I. Introduction

Given the inherent perturbations during the fabrication process of integrated circuits (IC), diagnosis of failing chips is a key step during yield ramping and high-volume manufacturing. Diagnosis is a software-based process that identifies possible locations and/or behavior 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 responses (also referred to as the fail log) of a failing chip. The quality of a diagnosis outcome is defined by resolution and accuracy. Resolution is defined as the number of candidates reported, and if the actual defect lies in the candidate set of locations, the diagnosis is deemed to be accurate. The output of a diagnosis tool has many applications. For example, it serves as a guide for physical failure analysis (PFA) to identify root causes of the failure, thus assisting yield learning, and is often used in many volume-diagnosis applications (e.g., [1-7]).

Diagnosis plays an important role in yield learning, but a lot of uncertainty in the fabrication process brings a number of challenges. First, because of the increasing complexity of IC technology and design complexity, real defects may behave in an unexpected manner, i.e., different from the fault models implicitly used by diagnosis tools. In addition, fault equivalence/dominance relationships and the existence of multiple defects further confuse diagnosis tools. These and other challenges may result in diagnosis outcomes characterized by poor resolution or inaccuracy. Another concern is that diagnosis computation time may be very long, and there is no simple rule to predict runtime as demonstrated in [8]. Given the various sources of uncertainty in diagnosis, machine learning (ML) can be a useful tool in improving diagnosis quality because of its advantage in dealing with uncertainty and uncovering high-dimensional correlations.

Prior work that applies ML to diagnosis can be broadly divided into two categories. The first category of work is employed after diagnosis, that is, it uses the outcomes of diagnosis tools to assist yield learning. For example, the authors of [9] use a random forest to identify whether the chip failing is due to a bridge defect. In [1-3], a support vector machine model is learned from diagnostic results and physical information to predict whether a candidate is more likely to be the actual defect location. The authors of [4] aim to discover the root cause of defects based on statistical learning on diagnosis results. The second category uses ML before diagnosis begins, and the goal is to preprocess test data in order to assist diagnosis. For example, in [10], ML is used to decide when to minimize fail data collection, while at the same time ensuring a high-quality diagnosis outcome. On the other hand, the work in [8] predicts various outcomes of diagnosis before diagnosis process begins for prudent diagnosis resource allocation. This work holds different assumptions as [10], that is, it assumes the testing stage has ended and the available fail data is fixed.

The proposed work in this paper has a similar context and goal as [8], which is to preprocess fail logs in order to select those chips that are most likely to produce meaningful diagnosis results. That work of [8] examines specific aspects of a diagnosis outcome, including whether a diagnosis run will succeed in providing some valid results, whether a chip fails due to a defective scan chain, and whether diagnosis runtime will exceed some time threshold. This work focuses on more informative aspects of a diagnosis. First, we focus on predicting the existence of multiple defects in a failing chip. During the yield ramp, the manufacturing process is not mature, meaning multiple defects are quite likely. Failing ICs with multiple defects may be an indicator of significant issues in the fabrication process, implying that it is beneficial to prioritize these failing ICs. Second, we focus on predicting the number of candidates that diagnosis reports for each defect, i.e., resolution. A very large resolution is not desirable because it greatly reduces the likelihood of successful PFA. To expand our understanding of the industrial landscape, we have conducted a survey of various industry practitioners who use diagnosis tools daily. Survey results indicated that most practitioners will not perform PFA on a failing chip with resolution larger than 3. 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 (i.e., 3). Third, we also predict the order of magnitude of diagnosis runtime, instead of a simple classification of “too long” or “short” as in [8], which is more informative and helpful for

978-1-7281-1170-4/19/$31.00 ©2019 IEEE

IEEE computer society

allocating diagnosis resources.

The rest of this paper is organized as follows. Section II reviews previous work on applying machine learning to improve diagnosis efficiency. Section III describes the details of the proposed methodology. Section IV demonstrates the efficacy of our method predicting the diagnosis outcomes of a test chip and a high-volume manufactured chip. Finally, Section V concludes the paper and provides directions for future work.

II. Background

ML has shown its effectiveness in predicting diagnosis outcomes [8]. That work uses three classifiers to predict different aspects of a diagnosis outcome. The first classifier targets an unwanted case in diagnosis, that is, diagnosis failure. For a given fail log, sometimes diagnosis fails to report any candidates, which is a common case especially during yield ramp. Diagnosis failure may result from complex defect behavior which does not match assumptions used by a diagnosis tool. The second classifier predicts the failure type, that is, whether failure is due to scan chain or logic. Because yield learning methods are different for the two types of failure and running diagnosis can be very time-consuming, it is also beneficial to know the failure type in advance. The third classifier predicts whether the runtime will be "short" or "long", based on a user pre-specified threshold. This classifier enables efficient allocation of computing resources for diagnosis. The experiment results described in [8] demonstrate that the three classifiers achieve superior prediction performance.

Although previous work has predicted useful information before diagnosis, the results are still preliminary and only cover limited aspects of diagnosis. First, chips fabricated in a mature process seldom have very complex defects, thus it is rare for diagnosis tools to report zero candidates when diagnosing failing chips manufactured in high volume. This makes the first classifier less effective for high-volume manufactured chips. In addition, only predicting the runtime as being "short" or "long" is not sufficiently informative. Diagnosis practitioners would benefit more from knowing diagnosis run time more precisely, allowing fail logs to be scheduled in a way that more aptly meet their objectives. Finally, prior work does not predict diagnostic resolution. It is not desirable to consume significant compute resources on a diagnosis run when resolution is very poor (e.g., >100).

III. Methodology

To overcome the limitations of [8], we develop a more advanced ML-based framework to help improve diagnosis efficiency. As illustrated in Fig. **1**, the whole flow starts from a fail log which records the failing outputs for each failing test pattern. For a fail log that is yet to be diagnosed, the first step is to extract features from the fail log. The features are composed of two classes: manually designed features and signature-based features developed based on domain knowledge. The features are then fed into three predictors and a preview of the diagnosis outcome is computed. These predictors are trained on previously diagnosed fail logs. The rest of the section is organized as follows. We first formally describe the problem we are working on in Section III-A, and the machine learning algorithm used is introduced in III-B. Another important aspect, feature extraction, is described in III-C.

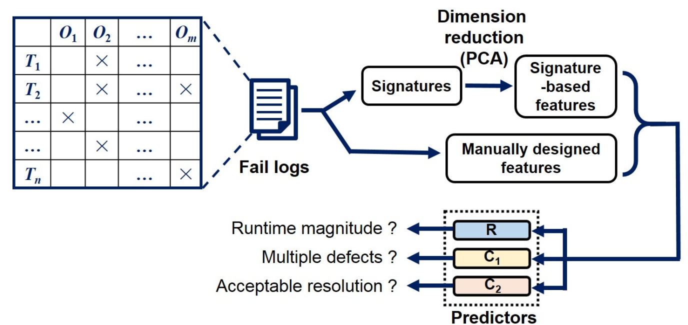

Fig. 1. Proposed framework for predicting diagnosis outcome.

A. Problem Formulation

We aim to predict three aspects of a diagnosis outcome: (i) the order of magnitude of runtime; (ii) existence of multiple defects; (iii) resolution. Specifically, the first task is a regression problem and the other two tasks are classification problems. The three tasks are accomplished by the three predictors illustrated in Fig. **1**. Each predictor takes as input d features that are extracted from fail logs, and outputs results y_1, y_2, y_3, respectively. The three predictors are succinctly defined as follows:

- Real number predictor R: $y_1 \in \mathbb{R}$. If the predicted runtime is t seconds, $y_1 = \log_{10} t$.
- Classifier C_1: $y_2 \in \{0, 1\}$. If the chip has only one defect, $y_2 = 0$; otherwise, $y_2 = 1$.
- Classifier C_2: $y_3 \in \{0, 1\}$. If the resolution is acceptable for further analysis, i.e., smaller than a certain threshold, $y_3 = 0$; otherwise, $y_3 = 1$.

We do not aim to predict the exact runtime because the runtime depends on the hardware configuration of the computer where diagnosis runs, 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 magnitude of runtime.

For the classifier C_2, the threshold of specifying an "acceptable" or "unacceptable" resolution can be determined by the user. In the experiments, we set threshold = 3 in order to coincide with the aforementioned survey results.

B. ML Algorithms

The *Random Forest* (RF) algorithm [11] 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 and C_2, 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 [11], RFR uses metrics such as *Sum of squared errors*

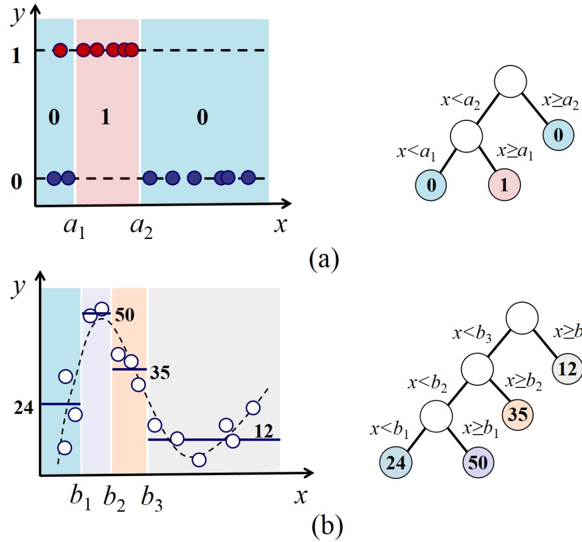

Fig. 2. Illustration of decision trees for (a) classification and (b) regression.

(SSE) [12]. The SSE of a split result is defined as:

$$SSE = \sum_{i \in G_l} (y_i - \bar{y}_l)^2 + \sum_{j \in G_r} (y_j - \bar{y}_r)^2, \tag{1}$$

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 sample values of in G_l and G_r, respectively; and $\bar{y}_l$ and $\bar{y}_r$ are the average values of the two sets.

Fig. **2** shows how a DT works for classification and regression problems with a single feature x. For the classification problem in Fig. **2**(a), the prediction objective is the label y with a binary value 0 or 1, while the regression objective y in Fig. **2**(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.

C. Features

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

The authors of [8] use a set of manually-designed features, which covers statistics derived from the test patterns, circuit outputs and the error values recorded in the fail log. Features include, for example, the number of failing patterns and failing outputs, which are related to computation cost. Other statistics, such as the average number of failing outputs having error value 0 may indicate the existence of a stuck-at-0 fault in a scan chain.

The manually-designed features incorporate domain knowledge and have shown effectiveness in predicting some preliminary diagnosis outcomes as discussed in [8]. We continue to use these features in this work. However, these features still miss important information that characterizes a failing chip. For example, many of the manually-designed features include

	O_1	O_2	...	...	O_m	Pattern Signature
T_1		×				1
T_2		×		×		2
...	×					1
...		×		×		2
T_n			×			1
Output signature	**1**	**3**	**1**	**2**	**0**	

Fig. 3. Illustration of fail-log signatures. × stands for a failure at the corresponding output and test pattern.

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 10,000 nodes, but the failing output of the second chip only has an input cone of 1,000 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 [13] have used signatures as features to identify systematic defects. The basic idea is described using Fig. **3**, where the table shows a typical fail log. For each failing pattern, i.e., $T_1 \sim T_n$, the failing outputs are recorded and marked with a "×". For each output, i.e., $O_1 \sim O_n$, 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 machine learning 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 data set 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:

$$\mathbf{X} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^T. \tag{2}$$

In the above equation, columns of $\mathbf{U}$ are the principal vectors,

TABLE I
CHARACTERISTICS OF TWO INDUSTRIAL CHIPS.

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	1,000
No. of fail logs	4,235	9,301
Fail-log data collection time	70 days	325 days

and columns of **V** are the coefficients for reconstructing the samples. Taking the first d components and the corresponding coefficients from **U** and **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. The effect of these new features is discussed in Section IV.

IV. Experiment

In this section, we describe the details of experiments on two industrial chips and show the efficacy of our proposed framework.

A. 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 28nm process. Chip 2 is manufactured in a mature 90nm process in high volume. Their characteristics are listed in Table I. Industrial partners executed circuit test on manufactured chips, and fail logs from 4,235 instances of Chip 1 and 9,301 instances of Chip 2 are collected for analysis.

For each chip, half of the data is used for training and the rest is used for testing. The features are a combination of three parts: manually-designed features, pattern signatures after PCA, and output signatures after PCA. We keep the first 10 principal components for pattern signatures, and 100 principal components for output signatures.

To evaluate the performance of the three predictors R, C_1, and C_2 of Fig. **1**, we need to define proper metrics. For the regression task 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 tasks C_1 (predicting the number of defects) and C_2 (predicting whether resolution is acceptable), the measuring metrics include *accuracy, precision, recall, F1-score* and *Area Under Curve (AUC)*. These metrics are calculated from four statistics: True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN). TP stands for the number of samples that are truly positive and predicted as positive. FP stands for samples with true label negative but predicted as positive. TN and FN are defined similarly. The metrics are:

$$
\begin{aligned}
Accuracy &= \frac{TP+TN}{TP+TN+FP+FN} \\
Precision &= \frac{TP}{TP+FP} \\
Recall &= \frac{TP}{TP+FN} \\
F1\text{-}score &= \frac{2}{\frac{1}{Recall}+\frac{1}{Precision}}.
\end{aligned} \tag{3}
$$

Precision and recall in Eq. 3 are defined for the positive class, those for the negative class can be defined similarly. Precision stands for the ratio that a sample predicted as positive is actually positive. Recall means the ratio that a truly positive sample is predicted correctly. AUC is a more reasonable metric when two classes are imbalanced [14].

TABLE II
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%

B. Runtime Prediction

Fig. **4** and Fig. **5** show the prediction results of runtime for Chip 1 and Chip 2, respectively. Fig. **4**(a) and Fig.**5**(a) shows the real and predicted order of magnitude of runtime for all the testing data, while Fig. **4**(b) and Fig.**5**(b) shows 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 three times 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 II 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, meaning the true runtime lies within the range of 0.8 times to 1.3 times of the predicted time. These results show that the learned predictor can provide a 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. In this experiment, we keep 20% data as the test set, and run regression using different training sizes. The plot of MAE with respect to varying training size is shown in Fig. **6**. 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 times to 1.5 times of the predicted runtime. The MAE for Chip 2 is even lower and achieves 0.06

with 2,000 training samples, meaning the true runtime range is 0.9 times to 1.2 times of the predicted runtime, which is a very accurate estimation.

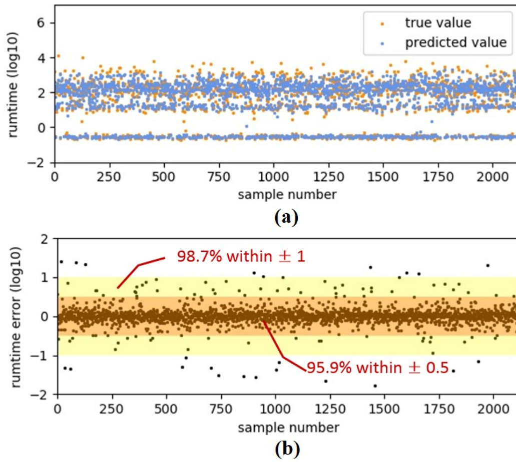

Fig. 4. Distribution of (a) runtime and (b) prediction error for Chip 1.

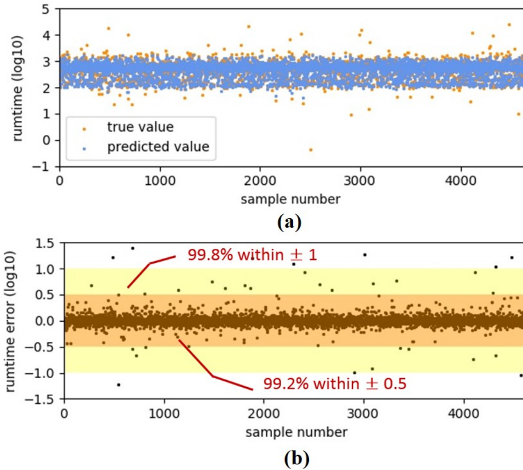

Fig. 5. Distribution of (a) runtime and (b) prediction error for Chip 2.

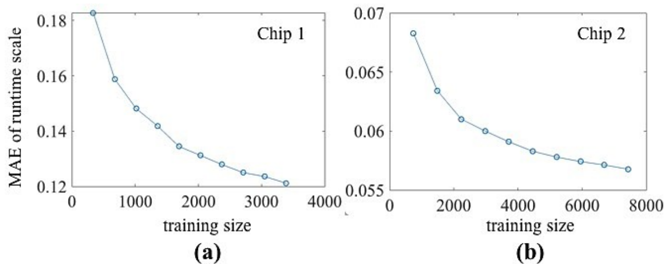

Fig. 6. The change in mean absolute value (MAE) of runtime magnitude prediction with different training sample size for (a) Chip 1 and (b) Chip 2.

C. Defect Number Prediction

The defect number prediction results for the two chips are shown in Table III. As explained before, half of the data are used for training and the rest for testing. 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. Such chips can be identified by the method described in [8], so these chips are not included for defect number prediction. Table III shows that the trained classifier C_1 achieves test accuracy of 0.83 for Chip 1 and 0.86 for Chip 2, and the AUC values for both chips are over 0.90. These results show that the trained classifier is quite effective in predicting whether the chip has a single or multiple defects.

TABLE III
PREDICTION OF DEFECT NUMBER FOR (A) CHIP 1 AND (B) CHIP 2.

Confusion matrix		Predicted 0	Predicted 1	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	627	57	684	0.85	0.92	0.88	0.83	0.90
	1	108	184	292	0.76	0.63	0.69		

(a)

Confusion matrix		Predicted 0	Predicted 1	Count	Precision	Recall	F1-score	Accuracy	AUC
True labels	0	2711	273	2984	0.88	0.91	0.90	0.86	0.93
	1	362	1164	1526	0.81	0.76	0.79		

(b)

Label 0: single **Label 1:** multiple

Note that for diagnosis purpose, a more accurate prediction result for single defect is preferred. 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 picking out fail logs with a single defect helps practitioners to obtain evidence concerning fabrication issues much more efficiently. The results in Table III 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 on multiple-defect cases.

To determine how many training samples are necessary for adequate classifying results, we set 20% of the data as test set, and run the prediction for different training-set sizes. The trends of AUC, accuracy, precision and recall for each of the classes are illustrated in Fig. **7** (a) for Chip 1. The prediction performance stabilizes after 2,000 training samples for Chip 1, meaning that 2,000 samples are sufficient for accurate prediction of multiple defects. The trends for Chip 2 are shown in Fig. **7** (b), showing that 4,000 samples are sufficient to produce satisfactory prediction.

D. Resolution Prediction

The results for resolution prediction are shown in Table IV. Similar to defect number prediction, we exclude the chips that diagnosis tools fail to report any defect candidates. Chips with resolution ≤ 3 are defined to have acceptable resolution. Table IV (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 (0.91), meaning that few chips with acceptable resolution are predicted incorrectly. We therefore do not lose much information from diagnosing such chips while saving significant compute resources from diagnosing chips with poor resolution.

TABLE IV
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

On the other hand, 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 nearly 0.8 million 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 practical here. 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.

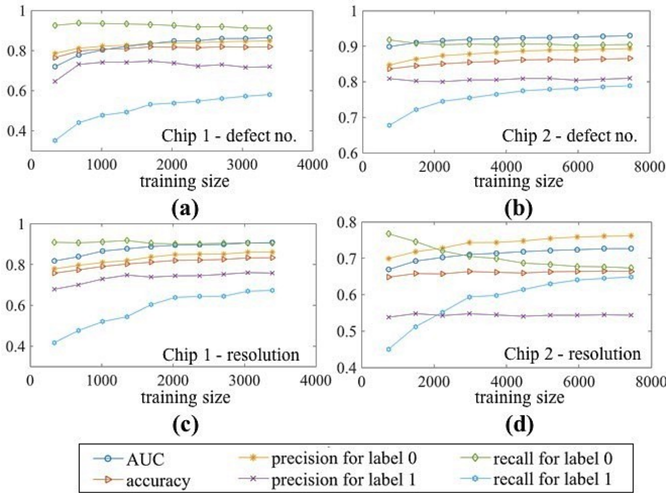

Fig. 7. The change in AUC, accuracy, precision and recall for (a-b) defect number prediction and (c-d) resolution prediction for different training sample sizes.

Similar to defect number prediction, the trends of AUC, accuracy, precision and recall for resolution prediction of two chips are shown in Fig. **7** (c) and (d). It is observed that 2,000 samples are sufficient for Chip 1 to have stable prediction performance. Although the prediction performance for Chip 2 is not satisfying, some metrics such as AUC keep increasing with larger training size, which means we may have better performance if more data is available.

We have compared the results using only the manually-designed features from [8] and the new features based on fail-log signatures. For runtime prediction, using these new features provides up to an 18% reduction of mean absolute error. For classification tasks, we observe up to a 6% improvement of AUC for resolution prediction and 1% for defect number prediction using the new features.

The three predicted aspects, together with other information such as other aspects predicted in [8], provide practitioners with a comprehensive preview of diagnosis outcome, which enables reasonable prioritization of fail logs and smart allocation of diagnosis resources. For example, the fail logs that are predicted to have a single defect, good resolution and short runtime are more desirable, and should be diagnosed first. In contrast, fail logs that are predicted to have many candidates and long run time can be skipped or scheduled later, only if resources are available.

V. CONCLUSION

In this work, we propose an ML-based framework for obtaining a preview of diagnosis outcomes. Compared to previous work, this work targets more informative aspects of a diagnosis outcome: runtime magnitude, existence of multiple defects, and resolution. This work also uses features from fail-log signatures for better prediction. Experiments on industrial chips show the effectiveness of the proposed method - 98% of the prediction error of the order of magnitude of runtime is within ±1, and the prediction of multiple defects has up to 0.86 accuracy and 0.93 AUC (Area Under Curve). The resolution classifier also achieves up to 0.84 accuracy and 0.87 AUC on one of the chips investigated.

This proposed method provides helpful information to help practitioners allocate diagnosis resources more prudently. Future work includes using data augmentation to generate more accurate predictors, especially for resolution prediction for complex circuits.

REFERENCES

[1] Y. Xue *et al.*, "PADRE: Physically-Aware Diagnostic Resolution Enhancement," *International Test Conference*, 2013.
[2] Y. Xue, X. Li, and R. D. Blanton, "Improving Diagnostic Resolution of Failing ICs through Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.
[3] C. Lim *et al.*, "Diagnostic Resolution Improvement through Learning-Guided Physical Failure Analysis," *International Test Conference*, 2016.
[4] B. Benware *et al.*, "Determining a Failure Root Cause Distribution from a Population of Layout-aware Scan Diagnosis Results," *IEEE Design & Test of Computers*, 2012.
[5] Y.-T. Lin and R. D. Blanton, "METER: Measuring Test Effectiveness Regionally," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2011.
[6] J. E. Nelson *et al.*, "Extracting Defect Density and Size Distributions from Product ICs," *IEEE Design & Test of Computers*, 2006.
[7] R. D. Blanton *et al.*, "Yield Learning through Physically Aware Diagnosis of IC-Failure Populations," *IEEE Design & Test of Computers*, 2012.
[8] Q. Huang *et al.*, "Improving Diagnostic Efficiency via Machine Learning," *International Test Conference*, 2018.
[9] J. E. Nelson, W. C. Tam, and R. D. Blanton, "Automatic Classification of Bridge Defects," *International Test Conference*, 2010.
[10] H. Wang *et al.*, "Test-data Volume Optimization for Diagnosis," *Design Automation Conference*, 2012.
[11] L. Breiman, "Random Forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[12] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.
[13] L. M. Huisman, M. Kassab, and L. Pastel, "Data Mining Integrated Circuit Fails with Fail Commonalities," *International Test Conference*, 2004.
[14] I. H. Witten *et al.*, "Data Mining: Practical Machine Learning Tools and Techniques," *Morgan Kaufmann*, 2016.