CNN-based Data-Model Co-Design for Efficient Test-termination Prediction

Hongfei Wang[1], Zhanfei Wu[2], Wei Liu[2] (Contact email: hongfei@hust.edu.cn)
[1]School of Cyber Science and Engineering, [2]School of Computer, Huazhong University of Science and Technology, China

2022 IEEE European Test Symposium (ETS) | 978-1-6654-6706-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ETS54262.2022.9810406

***Abstract*—Failure diagnosis is a software-based data-driven procedure. Collecting an excessive amount of fail data not only increases the overall test cost, but may also lead to degradation of diagnostic resolution. Test-termination prediction is thus proposed to dynamically determine which failing test pattern to terminate testing, producing an amount of test data that is sufficient for an accurate diagnosis analysis. In this work, we describe a novel data-model co-design method of using deep learning method for efficient test-termination prediction. In particular, images describing the failing test responses are constructed from failure-log files. A multi-layer convolutional neural network (CNN) embedding a residual block is then trained, based on the images and known diagnosis results. The learned CNN model is later deployed in a test flow to determine the optimal test-termination for an efficient and quality diagnosis. Experiments on actual failing chips and standard benchmarks demonstrate that the proposed method outperforms SOTA works. Our method creates opportunities to harness the power of deep learning for improving diagnostic efficiency and quality.**

I. Introduction

When a chip fails a test, the erroneous values on the primary outputs (or shifted out from scan flip-flops) are recorded in case diagnosis is later performed. The accumulated fail data thus constitutes the content on failure-log files collected from testers. Sufficient fail data must be collected to allow accurate diagnostic analytics. However, test cost can be significant. For wafer-sort test performed on a modern automatic test equipment (ATE), reducing test time by merely one second across 1M chips can save up to 60K $. Also, the limited memory on an ATE prevent storing too much data, especially during volume diagnosis. On the other hand, an excessive amount not only increases the overall cost, but also may lead to degradation of diagnostic resolution [1]–[3].

Terminating the test execution at an appropriate time meets the twin goals: achieving acceptable (or even better) diagnosis results, while saving the expensive ATE time and data-storage cost. The idea of test-termination prediction is first proposed in our earlier work [1]. The goal is to dynamically determine which failing test pattern to terminate testing, producing an amount of test data that is sufficient for an accurate diagnosis analysis. The work [1] has aroused significant research interests with the same or similar goals, among which some representative ones are [3]–[17].

The micro features and macro influences of real defects can be too arbitrary to be modeled deterministically. As a result, in addition to the continuous improvement of the commercial EDA tools, machine learning (ML) has been extensively studied to promote efficiency and accuracy in seeking solutions to diagnostic problems [1] [2] [16]–[21]. For example, [1] leverages ML to learn a statistical model that predicts the optimal test-termination points, by examing the cause-effect relationships between the amount of tests applied (by observable output responses) and diagnostic efficacy.

Novel and powerful methods are demanded to improve the efficiency and accuracy of the current ML-based approaches. In [1], a feature extraction scheme has been proposed to produce numerical data. Example per-test-pattern based features include: number of failing patterns applied thus far; cumulative failing outputs; number of unique erroneous output bits produced by the current test. Such transformation from failing responses to numbers in spreadsheet format inspired same or adjacent research from the perspective of test-data acquisition [2] [6] [16] [18], [22]–[24]. In the pursuit of better prediction power, existing works are always willing to derive heuristic- and statistic-based features, with the belief that lengthy feature vectors should bearing more useful information towards learning. Extracting those manually designed features can be time consuming. We need to automate the ad hoc processes that limit productivity. Moreover, many ML algorithms have already been carefully investigated by existing works, via explorations such as parameter tuning [1] [2] and model aggregation [6]. The representative algorithms include the least absolute shrinkage and selection (LASSO), support vector machines (SVM), *k*-nearest neighbor (*K*NN), decision trees, etc. They are termed as conventional (shallow) models in ML domain, as comparing with the current deep learning (DL). Although there is no firm conclusion, it is now generally believed in ML domain that DL are more expressive and thereby expected to be more powerful than shallow ML methods. Therefore, both the data representation (feature extraction) and model performance call for new idea to improve efficiency and quality.

In this work, a novel DL method that we call **CDCD** (**C**NN-based **D**ata-Model **C**o-**D**esign) is proposed to for efficient test-termination prediction. CDCD is a **data-model co-design** method that consists of two parts. (1) A data extraction scheme that constructs images using the test responses, collected either from actual testers or fault simulation. This part is mainly for increasing the expressiveness of fail data. (2) A prediction technique that determines the sufficient amount of test data for effective failure diagnosis. A convolutional neural network (CNN) embedding a residual block is trained for this aim. This part is for boosting the power of the prediction model.

To the best of our knowledge, CDCD is the first work

This work was supported by the National Natural Science Foundation of China (NSFC) under grant No. 62172173, 61702473.

explicitly transforms circuit failure information into images to enable efficient diagnosis. The availability of image data is an essential prerequisite for recent power DL algorithms that are excel at image processing and recognition [25]–[27]. Up to this point such particular capability of DL models are far from being fully exploited in test and diagnosis. Note artificial neural networks have been used in the test domain, but have not tackled the problem of failure diagnosis. Our data-model co-design method presents a new avenue of future research to investigate the ML/ DL implications in failure diagnosis.

II. Background and Related Work

A. Problem Formulation

Let X_{ij} denote the failure information regarding the j^{th} failing test results for the i^{th} chip, where $1 \leq i \leq N$, $1 \leq j \leq |T_i^F|$. N is the number of circuits under test (CUTs); T is the test set; T_i^F is the failing test set for the i^{th} CUT. $\{V_e\}^{1\times|T_i^F|}$ is a one dimensional vector representation for X_{ij}, such that $X_{ij} = \{V_e\}^{1\times|T_i^F|}$, $1 \leq e \leq E$.

Let Y_{ij} denote the diagnosis result for the i^{th} CUT, generated by using test results from the 1^{st} test pattern through the j^{th} failing test just occurred, one failing pattern per each diagnostic result. A *golden* diagnosis result for a failing chip refers to the one generated when using all the applied tests T (including both passing and failing tests). An *intermediate* result refers to any outcome generated without using the entire set of test patterns. A numerical value in the label vector is produced by measuring the similarity between an intermediate diagnosis result and a golden diagnosis, using the metrics developed in [1]. Obtained diagnosis results are further binarized for class labels, where 1's denote the diagnosis results the same as the golden ones, and 0's otherwise.

With feature vector and class label $\langle X_{ij}, Y_{ij}\rangle$ available, supervised ML can be performed to train a prediction model, which is later deployed in the test flow. It is desirable that test is terminated where its label is 1, since the test responses collected to this point is sufficient for an accurate diagnosis. Whenever a CUT fails a test pattern, the model is invoked to determine if the amount of test data accumulated so far is sufficient for an accurate diagnosis analysis. If yes, terminate testing; if no, continue. The termination point for each CUT is thereby dynamically determined, measured by data-volume reduction (DVR) ratio,

$$\text{DVR} = \frac{1}{N}\sum(1 - \frac{|V_i^{opt}|}{|V_i|}) \times 100\% \qquad (1)$$

where $|V_i|$ is the amount of test data collected for the i^{th} chip, by accumulating the number of failing output bits for all failing test patterns, from the first to the last. $|V_i^{opt}|$ is the optimized amount calculated in a similar way, from the first pattern to the test-termination point predicted for this CUT.

B. Existing Deep Learning Methods in Test and Diagnosis

Deep learning (DL) refers to the family of machine learning methods using deep artificial neural networks. It is probably the most exciting topic in both AI research and its industrial applications related to chips [28]. Depending on data types utilized in learning and inference, existing works using DL can be roughly grouped into three categories.

Vectors. Many existing works belong to this category [22]–[24], [29]. Similar to the feature extraction technique in [1], a data instance is represented by a class label and a vector. Each dimension in the vector describes an attribute of a data instance, also known as feature variable. If test-measurement data is not readily available as in [22], a feature extraction procedure is required to transform circuitry and test-outcome information into numerical numbers. For example, the work in [23] obtains 13 feature variables from test statistics and circuit netlists. Similarly, controllability and observability values from calculated from netlists, together with logic gate types, are used as input feature vectors to neural networks in [24].

Time Series. A time series is represented by a sequence of correlated variables. Strictly, time series are one special case of vector data, except that the number of variables for one instance can be considerably large. Such property fits into the scenarios where circuits are implemented with DFT structure [30] [18]. Intuitively, output bits from scan chains form time series. To process time series, one can choose an appropriate model architecture [30], or configure an adequate number of neurons in the input layer [18].

Images. Image data is generated in [31] [32] for using DL in chip test. Researches in this category view the test measurement(s) of every die/ device as one or multiple pixels in the produced image. The granularity is at die/ device level. As a result, although being able to identify defected devices or problematic dies, these methods are unable to examine the defects in more details for diagnostic analytics.

Remarks. Effective data acquisition for successful DL solutions in test and diagnosis are highly desirable. A data type must be devised properly, catering to the DL characteristics. DL is particularly good at handling signals representing images and speeches [25]–[27]. Most of the popular DL algorithms are proposed for handling images and signals. One peculiarity of these input data types is that, the values of single, smallest elements (such as pixels in an image, tokens in languages) do not affect the final class label. Instead, the correlation among multiple of them, as well as the pattern or trend they present altogether, determine the labels.

Given that DL is powerful at analyzing images, we give it full play in this work by adopting a data-model co-design approach: CDCD generates image data based on the failing outputs of the tested ICs; the images are used during the model training procedure and prediction stage.

III. Image Data Extraction

The proposed extraction scheme is illustrated in Fig. 1.

Images are extracted based on the failing outputs of CUTs. For each failing test of a CUT, a two dimensional image $\{P_{tm}\}^{|T|\times M}$ of size $|T| \times M$ is generated, where T is the entire test set applied to CUTs, and M is the number of circuit outputs (either primary outputs or outputs of scan registers). A row holds the responses of all the circuit outputs for a test,

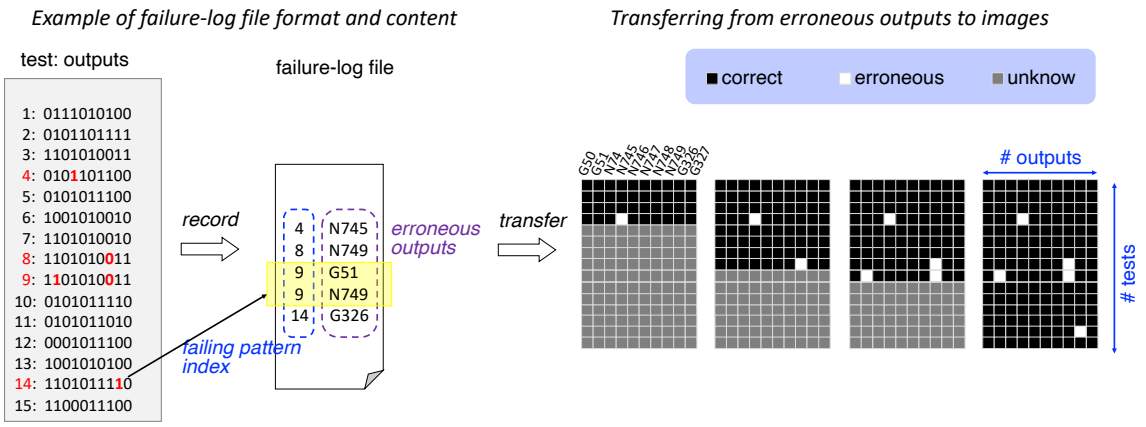


Fig. 1. Constructing images from a failure-log file. In this toy example, $|T| = 15, M = 10$. According to the failure-log file, the CUT failed four tests, thereby four two-dimensional images can be constructed reflecting the failure information originally recorded on plain texts.

while a column represents one output responses for all tests applied. In other words, the value of a pixel P_{tm} ($1 \leq t \leq |T|$, $1 \leq m \leq M$), which locates at the t-th row and the m-th column in the image, represents the observed values at the m-th output of the CUT for the t-th test applied. A pixel in the image is colored as follows: black for correct, white for erroneous, gray for unknown. We choose grayscale values, ranging from 0 to 255, as defined in Equation (2),

$$P_{tm} = \begin{cases} 0 & black \quad \text{correct ouptut} \\ 255 & white \quad \text{erroneous output} \\ 127 & gray \quad \text{unknown output} \end{cases} \tag{2}$$

Correct/ erroneous means the output polarity meets/ is opposite to the expected one, whereas unknown means it has not been tested yet. For a CUT, images are generated from the first failing test to the last one, sequentially. The number of images thus created equals to the number of failing tests occurred. Among all images obtained in this fashion for a failed CUT, except for the last one, there is at least one white pixel sitting in a row, which is precisely on top of the chunk of rows completely made up by gray pixels.

Recall from Section II-A that X_{ij} is the failure information regarding the j^{th} failing test results for the i^{th} chip. Previously in [1], X_{ij} is represented as a one dimensional, manually designed feature vector $\{V_e\}^{1\times|T_i^F|}$. Here CDCD transforms a X_{ij} into two dimensional image $\{P_{tm}\}^{|T|\times M}$, shown in Fig. 1, without effort and cost from feature extraction. The class label Y_{ij} for each image is obtained in the same way as described in Section II-A.

The above configuration of image pixels is not unique. First, color setting for pixels can be changed. To differentiate all three possible output cases (i.e., correct, erroneous, and unknown), three colors are needed to paint image pixels accordingly. Altogether there is $3! = 6$ matching solution for color-case pairs. Second, spatial arrangement of pixels is considered. In an image produced by CDCD, the order of rows in an image is consistent with the applied test patterns, and hence must be fixed. Such setting by no means limits CDCD to fixed-ordering of test sets. There is large body of research devoted to test pattern reordering to save test costs or for better results. The point here is that the order of rows in a generated image should be made consistent with the applied test patterns, irrespective of whether the tests have been rearranged or not. On the contrary, columns representing circuit output values can be swapped. There is no inherent ordering for neither the primary output ports or the shifted outputs from scan registers. What really matters is to follow the pre-determined agreement on output port ordering (such as from the port declaration part in RTL code), by which the test patterns are applied and failing-test responses are collected.

DL often suffers from computation burden. Image data instances occupy more memories than by vectors, especially when dealing with a large number of CUTs. Moreover, if the numbers of tests or circuit outputs increase, the image sizes are enlarged to require more resources. To tackle the problem, a sampling strategy is devised to select a small portion of images for training, significantly speeding up the learning process.

For a i^{th} failing CUT, a number of $|T_i^F|$ images are created in an order the same as the failing test-pattern sequence $j = 1, 2, ..., |T_i^F|$. Sampling is conducted on the entire image set $\{P_{tm}\}_{ij}^{|T|\times M}$ by the following rules.

1) The first image $\{P_{tm}\}_{i1}$ and the last image $\{P_{tm}\}_{i|T_i^F|}$ image are kept.
2) Given a bin size Δ, if $|T_i^F| > \Delta + 1$, proceed as follows. Starting from the second image $\{P_{tm}\}_{i2}$, for every Δ images, only the last one in the sequence is kept, denoted as $\{P_{tm}\}_{i(1+\Delta)}$, $\{P_{tm}\}_{i(1+2\Delta)}$, ... thus the number of bins is $(|T_i^F|-1)/\Delta$. If the last bin contains fewer images than Δ, keep $\{P_{tm}\}_{i|T_i^F|}$ and discard all the others. A total number of $1 + (|T_i^F| - 1)/\Delta$ images are selected in this case.
3) If $|T_i^F| \leq \Delta+1$, all $|T_i^F|$ images associated with this CUT are kept.

Fig. 2 shows an example from real fail data. The CUT fails a number of $|T_i^F| = 17$ tests (whose responses are recorded and placed on the left of Fig. 2), out of the entire test set of 65 ones. 17 images are created accordingly. By choosing $\Delta = 6$, the 1^{st}, 7^{th}, 13^{th}, and 17^{th} image are selected. Subsequent

DL only calculates these four images instead of the original 17 ones, thereby alleviating computation burdens.

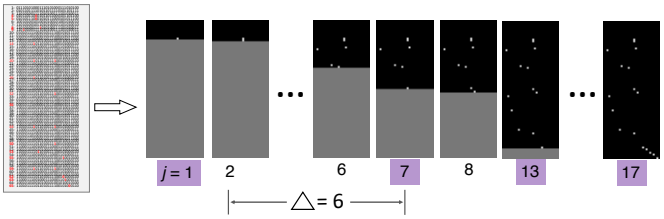


Fig. 2. Proposed sampling strategy to select training images. Indices of the selected ones are highlighted by purple.

The sampling strategy is feasible. At first glance, computation is saved at the cost of performance degradation, since much fewer samples are used. ML practitioners usually desire more data samples, especially training models via supervised learning. However, a closer examination reveals that the performance of learned models will not be compromised.The belief *"the more data the better model"* barely holds for our scenario. Consider two images $\{P_{tm}\}_{ip}$ and $\{P_{tm}\}_{iq}$, such that $1 \leq p < q \leq |T_i^F|$. $\{P_{tm}\}_{iq}$ contains all the failure information that $\{P_{tm}\}_{ip}$ has, because tests are applied sequentially, and test responses are collected incrementally. Consequently, failure information of the unselected images will not be completely discarded, but preserved by the selected ones. Moreover, for intermediate diagnosis results, the one using up to q^{th} test is generally better than that of p^{th} test, as more test failure information is available. Finally, from ML perspective, not every image contributes positively to training models, such as borderline images (that are created at or near the optimal test-termination point). Experiment results in Section V validate the sampling strategy.

IV. CNN for Test-Termination Prediction

With image data, the powerful DL methods are ready to take the spotlight. CDCD leverages convolutional neural network (CNN) to build test-termination prediction models. The designed CNN architecture is shown in Fig. 3. From the input to the output, an image is processed via three stages.

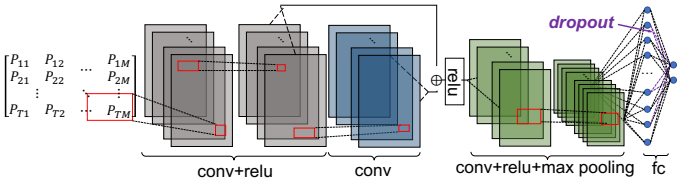


Fig. 3. The proposed CNN architecture for test-termination prediction. Layers with different functionalities are marked by colors.

Stage I uses a convolution layer to initialize the input images, and one residual block [27] to prevent the network from vanishing gradients. The residual network block consists of two convolution layers. Each of these three convolution layers is followed by a hidden layer using ReLU activation function [33] defined as $f(x) = max(x, 0)$. This means ReLU preserves convolution results ≥ 0 and discards anything else. Its piecewise linear and zero-thresholding functionality not only lead to the overall layer-by-layer nonlinearity for better expressivity, but also make the model parameters sparse.

Stage II contains two convolution layers, each followed by a ReLU layer and a max pooling layer. Kernel size is 3×3 for convolution to further extract and learn spatial features. Pool size is 2×2 for max pooling to reduce network parameters. Every time a convolution layer and a max-pooling layers alternates, the output data volume reduces by half. The design in this stage is partly inspired by the VGGNet [34].

Stage III has two fully connected (FC) layers, with ReLU as the activation function after the first FC layer. Dropout [35] is employed here to combat over-parameterization. Fig. 3 illustrates the functionality of dropout. Among all dashed lines connecting the final output nodes and the FC layer, the purple ones link the neurons that are selected and dropped out of the current training iteration. Softmax function produces the final class label probabilities.

CNN has three properties that make it very applicable in classifying the images that are extracted by CDCD: a pattern can be much smaller than a whole image and can appear in arbitrary regions of the image; downsampling the image pixels appropriately does not affect the prediction accuracy. These properties ensures the robustness of the CNN model. For example, column-wise image perturbation (shuffling the columns) has little affects towards the prediction results.

Stochastic gradient descent (SGD) [36] is combined with the backpropagation to train the CNN model. SGD optimization is an iterative method. A loss function using cross entropy as evaluation metric is used to measure the discrepancy between the predictions and the actual labels. It also helps determine appropriate number of training epochs to run SGD before it halts. The loss function is given by:

$$L = -\sum_{i=1}^{N_{tr}} \sum_{j=1}^{|T_i^F|} \sum_{c=1}^{C} y_{ij}(c)\ log(\hat{y_{ij}}(c)). \tag{3}$$

In Equation (3), N_{tr} is the number of training samples per batch ($N_{tr} \leq N$). C is the number of classes considered ($1 \leq c \leq C$), and is equal to two in this binary classification problem. $y_{ij} \in \{0, 1\}$ is the actual class label; $\hat{y_{ij}}(c)$ is the predicted probability values such that $\sum_{c=1}^{C} y_{ij}(c) = 1$.

Fig. 4 shows the initial dimension configuration of the CNN architecture through layer-by-layer transformation. K is the number of kernels used in convolution operation. The convolution kernel (also called filter) size is set to 2×16 in the first stage, instead of the commonly used small squares of size 3×3 or 5×5. The kernel pattern of such elongated rectangular shape is tailored as an adaptation to images created by CDCD. Increasing kernel width and reducing kernel length is to examine more circuit outputs at a time, while slightly reducing the number of tests considered.

V. Experiments

Experiments were performed on a workstation with three NVIDIA Quadro RTX 5000 16GB GPUs, two 3.30GHz CPUs. The CNN was implemented using TensorFlow (v2.3.1) [37].

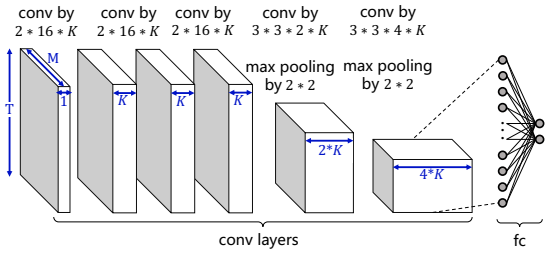


Fig. 4. Configuration of the dimensions in convolutional layers.

Altogether ten standard benchmark circuits are used to gauge the performance of CDCD. An in-house toolkit is used to perform defect simulation, which creates a population of failures from benchmark designs [1] [17]. On average, 1049 defects are created for each benchmark. The numbers of test patterns are 70∼150 for the circuit designs, generated by a commercial ATGP tool. Simulation responses from failing CUTs are recorded into failure-log files, which are then used by CDCD to generate images representing failing responses. In addition to the benchmark circuits, experiments were also performed on a system block on a fabricated chip (referred to as Chip A) from a mature node with volume production. A total population of 510 failing ones are analyzed. During testing, there is no hard limit (i.e., number of failing output pins) on the amount of data collected. The collection procedure does not encounter cases when fail data exceeds the ATE fail memory, either. This means the final list of failing outputs is complete to enable a diagnosis using all possible failing output information. A commercial diagnostic tool is used to produce diagnosis results, which are later analyzed for class labels Y_{ij} as described in Section II-A. Defected CUTs are partitioned into two disjoint sets: 90% for training, 10% for testing. 10-fold cross validation is performed to calculate averaged results.

Table I reports the results on the benchmarks. Prediction accuracy and savings are two main performance metrics considered. Savings are calculated according to Equation (1). The averaged test-termination prediction accuracy is $> 91.26\%$, while the saving is $> 30.11\%$. We compare the performance of CDCD to the work in [1]. CDCD significantly outperforms SOTA by both accuracy and savings. We find it advantageous to use the data-model co-design CDCD in test-termination prediction than the methods in [1] and similar works such as [2] [6] [16], where shallow ML (such as decision trees, SVM, and *K*NN) uses vector-based data for prediction tasks.

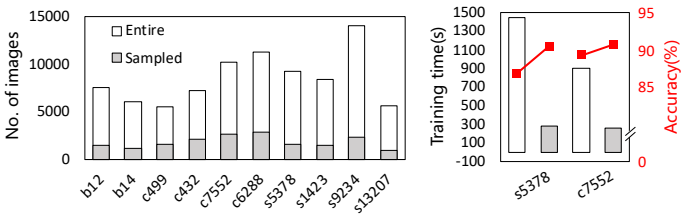
Tuning parameters is both a critical and an indispensable step in DL practice. Typical hyper-parameters include batch size, learning rate, number of kernels, dropout rate, number of training epochs, etc. A thorough design space exploration would be ideal were it not for the computation burden, and should be avoided [38]. Hence, each time we tune one hyper-parameter and keep all the others fixed. It should be noted that some of these parameters are correlated. For example, larger batch size is better for enhancing model stability, but often requires more epochs to maintain accuracy. Meanwhile, learning rate should also be increased moderately to cope with

TABLE I
PERFORMANCE COMPARISON ON EACH DESIGN

	CDCD		SOTA [1]	
Circuit	Accuracy(%)	Savings(%)	Accuracy(%)	Savings(%)
c432	91.25	20.26	90.74	13.15
c499	91.24	19.58	87.28	20.15
c6288	90.27	24.27	85.71	21.65
c7552	92.32	23.81	91.14	23.26
b12	91.40	17.39	89.29	10.91
b14	92.19	23.28	92.20	14.26
s1423	91.84	46.98	85.60	44.56
s5378	90.77	53.86	90.15	31.21
s9234	91.02	49.65	88.34	41.38
s13207	91.35	32.59	86.31	25.40
Chip A	90.15	24.91	83.51	23.20

larger batch sizes. DL usually requires running multiple epochs to optimize and update network parameters. However, an excessive number of training epochs leads to model overfitting, and consumes more computing time. While the loss function (defined in Equation (3)) value continues to drop, the model performance (i.e., accuracy and saving) is best at 60 $\sim$ 80 epochs. Generally, 60 epochs will suffice for CDCD to train models with acceptable performances.

Efficiency is critical in developing an EDA methodology, especially when considering incorporating it into a tool chain. Fig. 5 (a) shows that on average the numbers of images are decreased by $> 6X$ for benchmark circuits, using the proposed sampling strategy. Fewer training samples does not incur performance degradation, as evidenced by Fig. 5 (b). Experiments on other benchmark circuits resemble these two. This experiment indicates that choosing fewer but adequate training data samples not only maintains model performances, but also accelerates runtime considerably.



(a) (b)

Fig. 5. Sampling strategy for efficiency. (a) Volume comparison of the generated image volumes and sampled ones. (b) Performance comparison on training time and test-termination prediction accuracy.

Kernels play a significant role in CNN architecture. Unlike previous works [1] [2] [6] [23] where significant part of ML analytics is dedicated to feature extraction, kernels fulfill such equivalent role implicitly in DL. When a kernel moves over an image, it performs dot products between the kernel elements and the image pixels in a sub-region (of equal size as the kernel). Therefore, it would be plausible to suppose that kernel elements with large absolute values deserve more attention than those with small values, when it maps to a sub-region in an image representing output responses. This can be viewed as an implicit feature extraction procedure by analyzing output behavior.

Table II provides the chosen hyper-parameters for building the CNN architectures in CDCD. The last two columns give the actual runtime. Like the work in [1], model training is offline. Once a prediction model is constructed, it takes little time (a couple of milliseconds) to predict the test-termination point, therefore can easily be incorporated into a production test flow where prompt termination decisions are desired in the course of testing ICs. One more consideration for practical implementation is the hardware memory. We examined the trained mode sizes, and found that the largest one is s13207 for 106.6MB, and smallest one is c499 for 9.85MB. A hundred megabytes should not incur considerable computing or storing burden for a modern computer.

TABLE II
SUMMARY OF THE HYPER-PARAMETERS USED IN CNN

Circuit	Learning rate	Batch size	K	# epochs	Dropout rate	Δ	Training time(s)	Test time (ms)
$c432$	$1e^{-5}$	24	32	100	0.2	8	105.89	5.17
$c499$	$1e^{-5}$	64	32	120	0.5	6	81.45	3.97
$c6288$	$1e^{-3}$	16	48	80	0.5	7	292.08	4.17
$c7552$	$1e^{-4}$	64	32	60	0.4	6	265.53	3.79
$b12$	$1e^{-4}$	64	32	40	0.3	8	116.78	3.75
$b14$	$1e^{-5}$	16	32	60	0.3	6	93.42	3.01
$s1423$	$1e^{-4}$	48	32	60	0.5	8	241.22	3.53
$s5378$	$1e^{-4}$	64	48	40	0.2	9	268.02	3.92
$s9234$	$1e^{-4}$	96	16	60	0.3	10	343.54	3.95
$s13207$	$1e^{-4}$	32	32	40	0.3	9	137.42	3.58

K is the number of kernels used in convolution.
Δ is the bin size for image sampling.

The proposed CDCD is scalable for large circuits. As a data-driven DL method, the runtime and model sizes are determined by the number of images and their dimensions. The length and width of an image correspond to the number of applied test patterns and primary outputs of the CUTs, respectively, illustrated by Fig. 1. Complicated digital designs with more nets and gates inside have no direct correlation to affect the amount of image data used by CDCD.

VI. CONCLUSIONS

The data-model co-design CDCD makes two major contributions to promote diagnosis efficiency and quality. First, images representing test responses of failing circuits are derived as data source, paving the path towards exploring and utilizing powerful DL methods. Second, a CNN-based model is constructed to predict test-termination time, seeking an optimal solution to the intertwined challenges posed by test cost and diagnosis callouts.

REFERENCES

[1] H. Wang et al., "Test-data volume optimization for diagnosis," in *DAC*, 2012, pp. 567–572.
[2] Y. Xue, X. Li, and R. D. Blanton, "Improving diagnostic resolution of failing ICs through learning," *TCAD*, vol. 37, no. 6, pp. 1288–1297, 2018.
[3] I. Pomeranz, "Improving the accuracy of defect diagnosis by considering reduced diagnostic information," in *VTS*, 2015, pp. 1–6.
[4] S. Tanwir, S. Prabhu, M. Hsiao, and L. Lingappan, "Information-theoretic and statistical methods of failure log selection for improved diagnosis," in *ITC*, 2015, pp. 1–10.
[5] I. Pomeranz, "Improving the accuracy of defect diagnosis by considering fewer tests," *TCAD*, vol. 33, no. 12, pp. 2010–2014, 2014.
[6] Q. Huang, C. Fang, S. Mittal, and R. D. Blanton, "Improving diagnosis efficiency via machine learning," in *ITC*, 2018, pp. 1–10.
[7] Y. Xue, X. Li, R. D. Blanton, C. Lim, and M. E. Amyeen, "Diagnostic resolution improvement through learning-guided physical failure analysis," in *ITC*, 2016, pp. 1–10.
[8] S. Bodhe, M. E. Amyeen, I. Pomeranz, and S. Venkataraman, "Diagnostic fail data minimization using an N-cover algorithm," *TVLSI*, vol. 24, no. 3, pp. 1198–1202, 2016.
[9] N. Wang, I. Pomeranz, B. Benware, M. E. Amyeen, and S. Venkataraman, "Improving the resolution of multiple defect diagnosis by removing and selecting tests," in *DFT*, 2018, pp. 1–6.
[10] C. Bolchini and L. Cassano, "A novel approach to incremental functional diagnosis for complex electronic boards," *TC*, vol. 65, no. 1, pp. 42–52, 2016.
[11] I. Pomeranz, M. E. Amyeen, and S. Venkataraman, "Test modification for reduced volumes of fail data," *TODAES*, vol. 22, no. 4, Jun. 2017.
[12] I. Pomeranz, "Fail data reduction for diagnosis of scan chain faults under transparent-scan," in *VTS*, 2017, pp. 1–6.
[13] I. Pomeranz and M. E. Amyeen, "Hybrid pass/fail and full fail data for reduced fail data volume," *TCAD*, vol. 40, no. 8, pp. 1711–1720, 2021.
[14] Y. Huang, J. Janicki, and S. Urban, "Non-adaptive pattern reordering to improve scan chain diagnostic resolution," in *ETS*, 2019, pp. 1–6.
[15] I. Pomeranz and S. Venkataraman, "LFSR-based test generation for reduced fail data volume," *TCAD*, vol. 39, no. 12, pp. 5261–5266, 2020.
[16] C. Fang, Q. Huang, S. Mittal, and R. D. Blanton, "Diagnosis outcome preview through learning," in *VTS*, 2019, pp. 1–6.
[17] H. Wang et al., "Exploring graphical models with Bayesian learning and MCMC for failure diagnosis," in *ASP-DAC*, 2020, pp. 151–156.
[18] M. Chern et al., "Improving scan chain diagnostic accuracy using multi-stage artificial neural networks," in *ASP-DAC*, 2019, pp. 1–6.
[19] H. G. Stratigopoulos, "Machine learning applications in IC testing," in *ETS*, 2018, pp. 1–10.
[20] C.-K. Hsu et al., "Variation and failure characterization through pattern classification of test data from multiple test stages," in *ITC*, 2016.
[21] C. Shan, P. Babighian, Y. Pan, J. Carulli, and L.-C. Wang, "Systematic defect detection methodology for volume diagnosis: A data mining perspective," in *ITC*, 2017, pp. 1–10.
[22] F. Lin and K. Cheng, "An artificial neural network approach for screening test escapes," in *ASP-DAC*, 2017, pp. 414–419.
[23] L. R. Gmez and H. Wunderlich, "A neural-network-based fault classifier," in *ATS*, 2016, pp. 144–149.
[24] S. Millican, Y. Sun, S. Roy, and V. Agrawal, "Applying neural networks to delay fault testing: Test point insertion and random circuit training," in *ATS*, 2019, pp. 13–18.
[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
[26] C. Szegedy et al., "Going deeper with convolutions," in *CVPR*, 2015.
[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[28] J. Dean, "The deep learning revolution and its implications for computer architecture and chip design," in *ISSCC*, 2020, pp. 8–14.
[29] M. Shintani, M. Inoue, and Y. Nakamura, "Artificial neural network based test escape screening using generative model," in *ITC*, 2018.
[30] X. Wang, L. Jiang, and K. Chakrabarty, "LSTM-based analysis of temporally- and spatially-correlated signatures for intermittent fault detection," in *VTS*, 2020, pp. 1–6.
[31] C. H. Chuang et al., "A deep learning-based screening method for improving the quality and reliability of integrated passive devices," in *ITC-Asia*, 2020, pp. 13–18.
[32] C. Chen et al., "CNN-based stochastic regression for IDDQ outlier identification," in *VTS*, 2020, pp. 1–6.
[33] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *AISTATS*, 2011, pp. 315–323.
[34] K. Simonyan and A. Zisserman. (2015) Very deep convolutional networks for large-scale image recognition. [Online]. Available: arXiv:1409.1556
[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.
[36] S. Ruder. (2017) An overview of gradient descent optimization algorithms. [Online]. Available: arXiv:1609.04747
[37] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *OSDI*, 2016, pp. 265–283.
[38] A. B. Kahng, "New directions for learning-based IC design tools and methodologies," in *ASP-DAC*, 2018, pp. 405–410.