Cell-Aware Diagnosis of Customer Returns Using Bayesian Inference

Safa Mhamdi, Patrick Girard, Arnaud Virazel, Alberto Bosio, Aymen Ladhar

Cell-Aware Diagnosis of Customer Returns Using Bayesian Inference

S. Mhamdi P. Girard A. Virazel
LIRMM, Univ. of Montpellier / CNRS
Montpellier, France
<lastname>@lirmm.fr

A. Bosio
INL, École Centrale de Lyon
France
alberto.bosio@ec-lyon.fr

A. Ladhar
STMicroelectronics
Crolles, France
aymen.ladhar@st.com

***Abstract*—This paper presents a new cell-aware diagnosis flow that can be used to address a specific scenario (test protocol) one may encounter during diagnosis of customer returns. In this flow, we use a Bayesian classification method to precisely identify defect candidates. Experiments done on benchmark circuits as well as on a test chip from STMicroelectronics have proven the efficacy of our flow in terms of diagnosis accuracy and resolution.**

Keywords—Diagnosis, Customer Returns, Machine Learning

I. INTRODUCTION

Digital ICs are designed on the basis of a library of standard cells used to describe the logic behavior and physical layout of the lowest level component in a netlist. With the semiconductor industry that continuously pushes for higher density, it turns out that increasingly more defects occur within the cell structures. For the more advanced technology nodes, some estimates put the number of defects found within cells to represent almost half of all circuit defects [1].

Cell-aware test is a test generation approach that explicitly targets cell-internal defects [2]. Using cell-aware test today is the only way to achieve the required low defective-parts-per-million rates for critical applications. Similarly, Cell-Aware (CA) diagnosis is an approach used to pinpoint the possible defect candidates within the failing cell(s) of a defective IC [3-6]. Results of CA diagnosis are used to guide Physical Failure Analysis (PFA), which is a time-consuming and destructive process for physically exposing the defect in order to characterize the failure mechanism. The outcome of this PFA usually leads to modify the design or manufacturing process to ensure no similar occurrence in the future.

Customer returns are defective ICs that passed all functional and parametric tests after manufacturing but failed in the field. They are mostly due to latent defects [7]. This is especially true for, e.g., automotive products where a very comprehensive test flow has been applied to ensure zero test escape. So, the first step when a customer return occurs is to reproduce the failure mechanism with any original test and appropriate conditions. After that, a diagnosis program made of several routines is used to identify, step by step, the failing part and, finally, the suspected defects. Each routine corresponds to the application of a diagnosis algorithm at a given hierarchy level (system, core and cell levels) [8-11].

With the density level of today's ICs, a high resolution (very few or one defect candidate) is not always reachable by existing CA diagnosis tools based on conventional methods (effect-cause / cause-effect). As a result, many efforts have been dedicated recently for improving diagnosis resolution by using machine learning techniques [12]–[15]. Though efficient, these techniques address volume diagnosis for yield ramp-up, which is a different problem than fault diagnosis of customer returns. During volume diagnosis, a lot of data collected during manufacturing test and subsequent diagnosis phases are available, such as, e.g., hundreds of similar failed chips with candidates correctly labeled (good, bad). It is therefore possible to use these data for failure diagnosis of a new failed chip. Conversely, during fault diagnosis of a customer return, only one failed chip is investigated, with no information about the defective behavior of some other similar chips used in the same conditions (application, environment, workload). For this reason, learning-guided approaches existing for volume diagnosis cannot be reused for diagnosis of customer returns.

In [16], we proposed a learning-guided framework for CA diagnosis of mission mode failures in customer returns. Several supervised learning algorithms were considered in that work for predicting the nature (likelihood to be a good candidate) of each new data instance (defect) that has to be evaluated. Two distinct processes were developed to diagnose static and dynamic defects **separately**, each one assuming a dedicated testing scheme, i.e., basic scan and fast sequential, for static and dynamic CA test sequences application respectively. The effectiveness of the proposed framework was demonstrated through comparison with a commercial tool.

In an attempt to deal **concurrently** with all types of defects that may occur in customer returns, we proposed a new CA diagnosis flow in [17]. We assumed a test protocol in which **two** test sequences (static and dynamic) are used successively, each one assuming a dedicated testing scheme, i.e., basic scan and fast sequential. Constructing such a comprehensive flow imposed setting up a new framework with specific rules to achieve a high level of efficacy in terms of diagnosis accuracy and resolution. The proposed method was based on a Gaussian Naive Bayes trained model to predict good defect candidates. The flow was experimented and validated on industrial circuits.

In this paper, we propose a new version of the CA diagnosis flow in which **both** static and dynamic defects can be diagnosed owing to a **single** dynamic test sequence applied at-speed. Such a flow can be used to address a missing scenario in our former proposals [16-17]. As only dynamic instance tables are manipulated in this work, the representation of training and new data is simplified (a single type of feature vector is used) without loosing information and decreasing the quality of the training and inference phases. Experimental results have shown the benefit of using such a new CA diagnosis flow.

The rest of this paper is organized as follows. Section II discusses the test protocol considered in this work. Section III details the new CA diagnosis flow. Section IV shows results obtained on benchmark circuits and on a test chip developed by STMicroelectronics. Section VI concludes the paper.

II. Considered Test Protocol for Diagnosis

During diagnosis of customer returns in industry, several scenarios with different test protocols may occur. The most common is when the test sequences used during manufacturing test are available and can be used again for diagnosis purpose so as to mimic the process used initially during manufacturing test. This is the scenario assumed in our previous work [17], in which we considered two successive test sequences used to performed customer return diagnosis. First, a static CA test sequence generated by a commercial cell-aware ATPG tool is applied to the Circuit Under Diagnosis (CUD). This sequence targets all cell-level stuck-at faults plus cell-internal static defects, considering that these defects are not covered by a standard stuck-at fault ATPG. A standard (low speed) scan-based testing scheme is used to this purpose. Next, another option of the cell-aware ATPG is used to generate a dynamic CA test sequence that targets cell-level transition faults plus intra-cell dynamic defects not covered by a standard transition fault ATPG. In this case, an at-speed Launch-On-Capture (LOC) scheme (also called fast sequential) is used during test application. This scenario (test protocol) is sketched in Fig. 1.

Fault / Defect \ Test Sequence	Static	Static + Dynamic	Dynamic
Stuck-at & Static CA	[16]	[17]	[Proposed]
Transition & Dynamic CA		[17]	[16] [Proposed]

Figure 1: Test scenarios assumed in [16-17] and the proposed work

In [16], another scenario was assumed, in which two test sequences (static and dynamic) are also used to diagnose static and dynamic defects, but this time independently, and each one assuming a dedicated testing scheme, i.e., basic scan and fast sequential. This scenario is also sketched in Fig. 1.

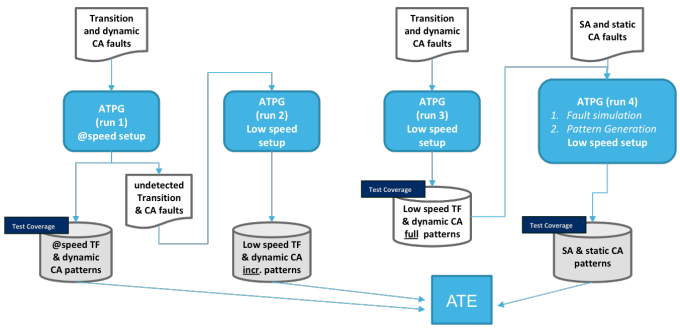

Figure 2: Example of a multi-run industrial ATPG flow

However, some other scenarios are possible, in which only one test sequence is available for customer return diagnosis. This test sequence can be composed of i) static test patterns applied at low speed, ii) dynamic test patterns applied at low speed, iii) or dynamic test patterns applied at high speed. Such a test sequence can be obtained from the various runs of an industrial ATPG flow as the one illustrated in Fig. 2.

In this paper, we consider the scenario in which **only one test sequence composed of dynamic cell-aware test patterns applied at high speed is available,** and has failed during test of the customer return. According to the flow depicted in Fig. 2, this scenario may happen when such a test sequence has been generated to target transition faults plus intra-cell dynamic defects, and appears to also cover the required percentage of stuck-at faults plus intra-cell static defects (or, more generally, satisfies the test coverage specifications) after the first run of ATPG in Fig. 2. In this case, note that only one (dynamic) datalog is generated after test application and can further be used for diagnosis purpose. Nevertheless, **both static and dynamic defects are taken into account** in this scenario. Again, the target of this scenario is sketched in Fig. 1.

Static defects are defects that require one-vector test patterns to be detected. Dynamic defects are defects that require two-vector test patterns to be detected. These defects can be non-resistive defects modeled by stuck-open faults. More generally, these defects are mainly due to resistive opens or shorts that prevent signals to propagate within a circuit at the normal speed, and hence lead to IC failure. In this case, they are modeled by (quantitative) delay faults or (qualitative) transition faults (also called gross delay faults). With the advent of deep submicron technologies, the occurrence of dynamic defects is constantly increasing, not only during the manufacturing process of ICs, but also during the lifetime of the circuit where latent or wear-out defects may appear due to various stress conditions (operational, environmental, etc.).

III. Proposed Cell-Aware Diagnosis Flow

Figure 3 is a generic view of the learning and prediction processes utilized in this work. Both approaches are based on supervised learning that takes a known set of input data and known responses (*labeled data*) used as training data, trains a model, and then implement a classifier based on this model to make predictions (*inferences*) for the response to new data.

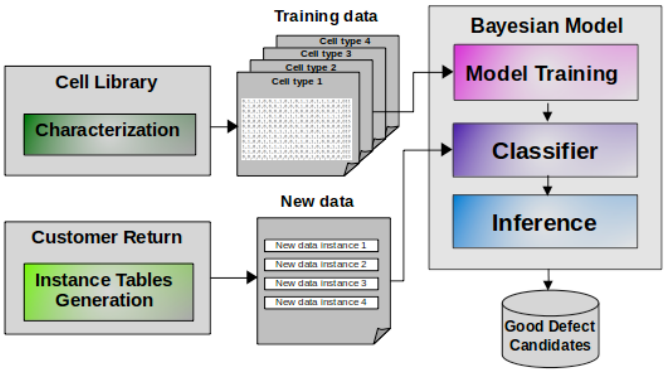

Figure 3: Generic view of the cell-aware diagnosis flow

In the next subsections, we detail the various steps of the proposed cell-aware diagnosis flow able to deal with all types of defects (i.e., static and dynamic) that may occur in customer returns. Again, only one test sequence composed of dynamic cell-aware test patterns applied at high speed is available, and has failed during test of the customer return.

A. Generation of Training Data

Training data are generated for each type of cell existing in the Circuit Under Diagnosis (CUD) during an off-line characterization process done only once for a given cell library. These data are extracted from cell-aware views provided by a

commercial CAD tool that contain all characterization results for a given cell type. These results are provided in the form of a fault dictionary containing, for each defect within a cell, the cell input patterns detecting (or not) this defect. An example of training data as used in [16-17] and containing four instances for an arbitrary two-input cell is shown in Fig. 4. Each instance is associated to a static defect (D_1, D_2) or a dynamic defect (D_{11}, D_{12}). A 1 (0) indicates that defect D_i is detectable (not detectable) at the output of the cell when the cell-level test pattern P_j is applied at the inputs of the cell. Cell-level test patterns (or *cell-patterns*) are static (one input vector - P_1 to P_4 in Fig. 4) or dynamic (two input vectors - P_5 to P_{16} in Fig. 4 in which R (F) indicates a rising (falling) transition at the cell input). For an n-input cell, there exists 2^n static cell-patterns and $2^n.(2^n–1)$ dynamic cell-patterns.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	Pattern
00	01	10	11	0R	0F	R0	RR	RF	R1	F0	FF	FR	F1	1R	1F	Defect
1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	D1
1	0	0	1	0	1	0	1	0	1	1	1	0	0	1	0	D2
0.5	0.5	0.5	0.5	0	0	0	0	0	0	1	0	0	0	0	0	D11
0.5	0.5	0.5	0.5	1	0	0	0	0	0	0	0	1	0	0	0	D12

Figure 4: Example of training data for all defect types in a two-input cell

Dynamic defects can be detected not only by dynamic patterns, but also by static patterns applied using a basic scan testing scheme, provided that i) at least one transition has been generated at the cell inputs between the next-to-last scan shift cycle and the launch cycle, and ii) the delay induced by the defect is large enough to be detected (*these are the detection conditions of a dynamic defect modeled by a stuck-open or a gross delay fault*). For this reason, the value '0.5' is assigned to each dynamic defect (D_{11}, D_{12}) for all related static cell-patterns, meaning that such a defect is detectable or not depending on whether or not the above conditions are satisfied.

P1'	P2'	P3'	P4'	P5'	P6'	P7'	P8'	P9'	P10'	P11'	P12'	Pattern
0R	0F	R0	RR	RF	R1	F0	FF	FR	F1	1R	1F	Defect
0	1	0	0	0	0	1	1	0	0	0	0	D1
0	1	0	1	0	1	1	1	0	0	1	0	D2
0	0	0	0	0	0	1	0	0	0	0	0	D11
1	0	0	0	0	0	0	0	1	0	0	0	D12

Figure 5: New format of training data for all defect types in a two-input cell

As only dynamic instance tables are manipulated in this work, the representation of training data as used in [16-17] can be simplified without loosing information and decreasing the quality of the training phase. This comes from the observation that a static defect is a particular case of dynamic defect (e.g., a full open is a resistive open with an infinite value of the resistance), and that all static cell-patterns for a given defect are embedded in its whole set of dynamic cell-patterns. Indeed, a dynamic defect requires a two-vector test pattern (V_1V_2) in which the values of V_1 and V_2 have to be properly defined for the defect to be detected. Conversely, only the value of V_2 is significant for a static defect to be detected by such pattern, irrespective of the value taken by V_1. When looking at Fig. 4, one can see that P_1={00} is embedded in P_6={0F}, P_{11}={F0} and P_{12}={FF}, and the same for P_2, P_3 and P_4. Similarly, we can see that static defect D_2 is detectable by P_1 and P_4, and hence by P_6, P_8, P_{10}, P_{11}, P_{12}, and P_{15}. So, by "compacting" a training dataset as shown in Fig. 5, in which only dynamic cell-patterns are considered, one can see that all meaningful information is still contained in this set, while redundant ('0' and '1' values in the first four columns of Fig. 4) or insignificant ('0.5' values in the same columns for dynamic defects) information is removed. More generally, such compact format for training data makes so that only one type of feature vector (dynamic) is used for both types of defect.

As the goal with training data is to provide a distinct feature vector for each data (defect), it is important to be able to distinguish between static and dynamic defects with such a new format of the training dataset. Let us consider two defects D_1 and D_{11} where D_1 is static and detectable by {00} and D_{11} is dynamic and detectable by {F0} (note that {00} is the second vector of {F0}). As can be seen in Fig. 5, these two defects can easily be distinguished since their training data instances (or *feature vectors*) are different. The consequence of using such a new format for training data (and hence for new data as will be shown later on) is not an improved accuracy or resolution, but rather a simplified manipulation of feature vectors.

B. Generation of Instance Tables

As illustrated in Fig. 3, new data are generated after post-processing of so-called *instance tables* describing the behaviour (pass / fail) of each suspected cell in presence of a real intra-cell defect (in one of the suspected cells) when a cell-pattern is applied to the cell. The format of a dynamic instance table looks like the one illustrated in Fig. 6 for a given two-input NAND cell and two dynamic cell-patterns [16]. In this example, the first part of the file gives information on how the cell is linked to other cells in the circuit, while the second part represents, respectively, the pattern number, the pattern status (failing, passing), and the cell output Z with the associated fault model for which exercising conditions are reported. These conditions shown right below each cell-pattern represent the stimulus arriving at the cell inputs during the shift phase (before '-') and applied during launch & capture cycles (after '-'). For example, cell-pattern 1 consists in applying a falling transition on input B, A being equal to static 1, and failing in detecting a rising transition on Z.

```
-----------------------------------------------------------------
                Nand Cell - ND2HVTX2
-----------------------------------------------------------------
        Z   Output    L412/C1381A
        A   Input     C2348/Z
        B   Input     C2415/Z
-----------------------------------------------------------------
    Pattern 1   FAILING   Transition on Z
        Z   000000000000011 - 01
        A   000000000000011 - 11
        B   000000000000001 - 10
    Pattern 2   PASSING   Transition on Z
        Z   000000000000011 - 10
        A   000000000000000 - 01
        B   000000000000001 - 11
-----------------------------------------------------------------
```

Figure 6: Example of a dynamic instance table for a NAND cell

The way to generate instance tables in our proposed cell-aware diagnosis flow is illustrated in Fig. 7. First, cell-aware dynamic test patterns are applied to the failing CUD. Remember that in our method, each test sequence is obtained from a commercial CA test pattern generation tool that targets intra-cell defects. Then, we obtain a datalog containing information on the failing test patterns and corresponding failing primary outputs. From this information and the circuit netlist, we perform a logic diagnosis (by using the same commercial tool used for test generation) that gives the list of suspected cells. By using datalog information, we can finally generate an instance table for each suspected cell.

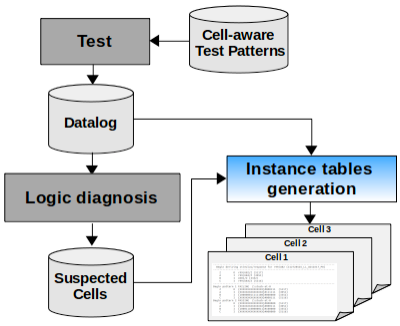

Figure 7: Generation flow of instance tables

Note that an instance table is generated for a given cell if and only if applying the dynamic test sequence to the CUD has led to at least one 'fail' at the circuit outputs. In other words, an instance table is generated for a given cell if the cell is a suspected cell after test application and logic diagnosis.

C. *Generation of New Data*

New data are composed of various instances, each of them being associated to one suspected cell in the CUD (customer return) and representing a feature vector that characterizes the real behavior of the cell during test application. From each feature vector (new data instance), we can further extract one or more defect candidates that have to be classified as good or bad candidate with a corresponding probability to be the root cause of failure. This classification is done by comparing the new data instance with the training data of the corresponding suspected cell, and identify those training data instances that match (or not) with the new data instance.

P1′	P2′	P3′	P4′	P5′	P6′	P7′	P8′	P9′	P10′	P11′	P12′	
0	1	0.5	0.5	0	0.5	0	1	0.5	0	1	0	Di

Figure 8: Format of a new data instance for a two-input cell

The format of a new data instance, illustrated in Fig. 8, is quite similar to that of the above training data instance, but has a different meaning. In each instance, the value '1' (resp. '0') is associated to a failing (resp. passing) cell-pattern P_i for a given defect candidate, meaning that the candidate is **indeed** detectable (resp. undetectable) by the cell-pattern P_i at the output of the cell during test of the CUD, and hence can (cannot) be the real defect. In such instance, the value '0.5' is associated to a cell-pattern for a given defect candidate when this pattern cannot appear at the inputs of a suspected cell during real test application with an ATE. The median value '0.5' was chosen to avoid missing information in new data instances while not biasing the features of these data.

D. *Cell-Aware Diagnosis Flow*

Figure 3 depicts the **two main steps** of the **supervised learning process** used for CA defect diagnosis. We use a Bayesian classification method for predicting the nature (likelihood to be a good candidate) of each new data instance. This choice comes from the results obtained in our previous study after experimenting several learning algorithms and observing their inference accuracies [16]. So, the **first main step** of our CA diagnosis flow consists in generating a Naive Bayes (NB) model and to train it by using the training dataset. Training a model is done based on labeled training data and then can be used to assign a pre-defined class label to new objects. In this step, training data are used to incrementally improve the model's ability to make inference. The training data is divided into mutually exclusive and equal subsets. For each subset, the model is trained on the union of all the other subsets. Once training is complete, the performance (accuracy) of the model is evaluated by using a part of the dataset initially set aside [18]. The **second main step** consists in implementing the NB classifier by using a Gaussian distribution to model the *likelihood* probability functions, and use this classifier to make probabilistic prediction (or inference) when a new data instance has to be evaluated.

An important preliminary step before the above two steps is training data preparation, which is carried out in three phases: *Data Selection*, *Data Preprocessing*, and *Data Transformation*. It first consists of selecting the subset of all training data that will be used to build the model and classify new data. Then, it consists in putting all data together and randomize the ordering. In our selection process, 70% to 90% of the available data were randomly selected and this operation was repeated several times to obtain training data with good randomness. Few other manipulations are also done, such as grouping data by considering equivalent defects or removing data instances of undetectable defects. Then, it consists in splitting the data in two parts. The first part is further used to train the model and is made of the majority of the dataset randomly selected. The second part is used for evaluating the trained model's performance. More details can be found in [16].

E. *Extended Utilization of the Proposed Diagnosis Flow*

Though the CA diagnosis flow has been proposed and experienced for a given test protocol discussed in Section II, it can also be used in many other scenarios existing in industry. Let us briefly discuss two of them in the following.

A first common scenario is when more than one clock scheme (typically basic scan and fast sequential) are used for a given test sequence (targeting static and dynamic defects), thus leading to a single instance table per suspected cells in which information about failing and passing cell-patterns are listed and used to create the corresponding single new data. In this case, despite the existence of different description formats for the exercising conditions of static and dynamic cell-patterns, the proposed flow can be used to easily extract information and create a "universal" new data for the suspected cell.

Another common scenario is when several test sequences (typically one static and one dynamic applied at low and high speed, respectively) are used for diagnosis, possibly generating several instance tables per suspected cells. In [17], in case the real defect is detected by both sequences, two new data are extracted from the two instance tables: a Static New Data (SND) and a Dynamic New Data (DND). From these data, we need to use a complex (though efficient) algorithm with a conflict identification protocol and specific rules to determine the final new data for the suspected cell. Conversely, in the flow proposed in this paper, such a final new data can be easily obtained by using simple and straightforward intersection rules between SND and DND. This is the consequence of using a compact format for new data (and training data as well).

Note that all other scenarios listed in Section II are also manageable by the proposed flow and data representation.

IV. EXPERIMENTAL RESULTS

Our CA diagnosis flow has been implemented in a Python program and was experimented in two different ways. First, we conducted experiments on a set of ITC'99 benchmark circuits. Then, we considered a test chip from ST and performed a simulated case study with a defect injection campaign to corroborate the results achieved on the ITC'99 circuits.

A. Experiments on ITC'99 Circuits

We first conducted experiments on ITC'99 benchmark circuits synthesized in a full scan manner using a 28 nm FDSOI technology from ST. A commercial CA ATPG tool was used to generate a dynamic CA test sequence targeting maximum fault coverage for each circuit. For each circuit and the corresponding test set, we simulated the behavior of the ATE by performing a defect injection campaign (about 2000 injections per circuit) into a number of randomly selected cells and collecting test information to build the tester data log. For the defect injection campaign, we considered each transistor of the selected cells and we targeted **all possible static and dynamic defects** affecting that transistor. As several defects have the same impact on the logic behavior of the cell, and hence are logical-equivalent defects, they were grouped in **defect classes**. We used a commercial logic diagnosis tool to determine the list(s) of Suspected Cells (SC) after dynamic test sequence application. The average number (#aSC) for each circuit is listed in Table I, together with information about each circuit: number of cells, scan flip-flops, dynamic test patterns, and transition (including dynamic CA) test coverage in %. The measured stuck-at test coverage of each dynamic CA test sequence is also given in Table I (sixth column).

TABLE I. CIRCUIT FEATURES AND RESULTS OF LOGIC DIAGNOSIS

Circuit	#Cells	#SFF	#dTP	TrTC	SaTC	#aSC
b15	2465	416	2665	88.38	90.05	1.84
b17	7960	1314	4423	91.10	89.94	2.38
b18	3238	215	4436	93.46	95.56	2.31
b19	6337	430	5583	93.79	96.52	1.61
b20	6733	430	5460	93.76	96.45	1.57
b22	3218	215	4670	93.78	95.39	1.75

For generating training data, we used characterization data provided by a commercial tool and ST technology libraries. For generating new data instances, we performed post-processing of instance tables obtained as seen in Fig. 7. From the training data and the Gaussian NB model, we make predictions on new data instances. Results obtained are a list of defect candidates with the highest probability to be the root cause of failure.

Table II summarizes the results obtained on the biggest ITC'99 benchmark circuits. The first part of the table is about accuracy and gives, for each circuit, the percentage of cases in which the injected defect was reported in the list of suspects provided by the proposed CA diagnosis and the commercial CA diagnosis tool, respectively. The commercial tool is non-probabilistic and provides the list of all suspects obtained after CA diagnosis with a ranking and a matching score. The **same characterization data and test protocol** have been used in both cases. These results show that the real (injected) defect is **always** identified by the proposed diagnosis flow. Sometimes, it is the only candidate and has a probability of 1 to be the best candidate. Sometimes, it is reported with some other candidates identified in one or more suspected cells. Conversely, we can observe that the commercial tool is **not always** able to report the injected defect as candidate. This proves the superiority of our proposed framework in terms of accuracy (always 100%), which is not the case of the commercial tool that sometimes provides inaccurate results (at least for 5 out of 6 circuits).

The reason of these misdiagnosis cases with the commercial tool can be explained as follows. In such a tool, the logic diagnosis phase is embedded in the whole CA diagnosis flow, and no intermediate result about logic diagnosis can be observed (conversely to what is done in our learning-based flow). In such configuration, it may happen that the cell in which the real defect has been injected is found with a much lower probability to be the source of failure compared to the other identified suspected cells. In such a case, the tool may possibly ignore this cell in the next phases of the process, and finally arrive at a situation where either wrong defects or no defect will be identified as suspected candidates. Being unable to go deeper inside the functioning of the commercial tool, this is the most likely explanation about such cases of misdiagnosis.

The second part of Table II is about resolution and gives, for each circuit and considering all injection campaigns, the average number of suspects reported by the proposed method and the commercial tool, respectively. As can be seen, the resolution achieved with our method is most of the time better. In only two cases (b18 and b22), the average resolution is slightly better with the commercial tool, but these results are biased, as the accuracy in both cases is less than 100%. So, overall, these results confirm the superiority of our approach.

TABLE II. OVERALL CELL-AWARE DIAGNOSIS RESULTS

Circuit	Accuracy		Resolution	
	Proposed	Com. Tool	Proposed	Com. Tool
b15	100 %	99.75 %	3.37	3.79
b17	100 %	100 %	5.96	11.14
b18	100 %	97.64 %	6.46	5.73
b19	100 %	97.96 %	2.23	2.85
b20	100 %	99.11 %	2.64	2.94
b22	100 %	99.12 %	5.40	4.82

For our experiments, we used a publicly available machine learning software package called Scikit-learn, which is an integrated development environment with a suite of ML tools [19]. The single defect assumption was considered, though the proposed framework is able to manage situations where multiple defects have occurred, provided that those defects are not in the same cell. This feature comes from the fact that our flow considers all suspected cells one at a time, and then incrementally constructs a list of suspected defects identified in each of these cells. Finally, in-field failure mechanisms related to premature aging, such as NBTI or HCI, essentially lead to resistive opens and shorts. These mechanisms, that need to be considered in the context of customer returns, can be appropriately taken into account in our CA diagnosis flow.

The CPU time taken by the proposed flow to provide a list of good defect candidates is always very low (few seconds) and does not depend on the circuit size. Only the number of suspected cells obtained after logic diagnosis may have an impact on the CPU time (for the generation of instances tables) but in a very slight manner (as this number is always very low). In fact, the most time-consuming part (few hours) of the flow is the cell library characterization phase, but it is done only once and is **not correlated** with the circuit size.

B. Comparison with [16]

In order to confirm the efficacy of the proposed diagnosis flow, we have compared the above results with those reported in Section VI.B of [16]. Experiments in [16] are different as only dynamic defect injections were done, while both static and dynamic defects were injected during experiments in the current work. Moreover, about 600 random injections per circuit were done in [16] whereas 2000 defect injections have been carried out in the new experiments. This explains the different values of accuracy and resolution between the two papers. However, despite these differences, we can observe the same overall tendency when compared to a commercial CA diagnosis tool, i.e., a diagnosis accuracy of always 100% and a resolution comparable or even better with our flows.

C. Simulated Test Case Study

We also conducted experiments on a silicon test chip developed by STMicroelectronics and designed with a 28 nm FDSOI technology. The test chip is only composed of digital and memory blocks, and one PLL. The digital blocks are made of 1.385.864 cells. Other features are given in Table III.

TABLE III. MAIN FEATURES OF THE SILICON TEST CHIP

#cells	#PIs	#POs	#SFF	#dTP	TrTC	SaTC
1.3M	91	33	88K	1238	99.93	95.73

We performed a simulated case study with an extensive defect injection campaign to corroborate the results achieved on the ITC'99 circuits. We randomly and successively injected 4800 defects (2300 static plus 2500 dynamic) in the circuit description. As only realistic defects were considered during the library characterization process, **only realistic defects were considered** during this defect injection campaign. All defects were injected in a single full scan digital block composed of 203K cells, and tested with a dynamic CA test sequence composed of 1238 test patterns and achieving a transition + dynamic CA test coverage of 99.93 %. The measured stuck-at test coverage of this test sequence was 95.73%.

Results obtained after executing our learning-based CA diagnosis flow and averaged over all defect injections have shown an accuracy of **100%** (the injected defect was always reported in the list of suspects). The average resolution obtained for the defect injection experiments is of **3.15**. The resolution ranges between 1 up to 7, and in most of the cases, the number of suspects was less than 4.

V. DISCUSSION AND FUTURE WORK

The above results show the appropriateness of a learning-based method to solve our problem, despite the small size of the training dataset used (only one sample for one defect class). This will be even truer when multiple defect sizes and test conditions will be used. In these cases, multiple samples (one for each defect size or defect size range, one for each PVT test condition) will be associated to a given defect class, simply because the behavior of the defect will differ when applying the same set of test patterns. In terms of timing and complexity, this will just slightly impact our method, since training dataset is extracted from characterized cell libraries that are generated anyway for test and diagnosis purpose. So, even with large cell libraries with a huge number of defects to be simulated (e.g., 631 cells in a library, each with 4 to 6 inputs, 951 shorts and 749 opens on average – example of an ST technology library), our flow will still be easily and time-efficiently applicable.

In our experiments, all injected defects for evaluation purpose were present in the training dataset. However, in most of real silicon cases, especially for customer returns, actual defect behavior may not perfectly match the fault models that are used to train the NB classifier. Further work will be dedicated to see how well the proposed flow works in this case. Another perspective consists in exploiting the ranking of suspected cells usually provided after logic diagnosis by commercial tools, which was not done in this work. By this way, our flow will provide a similar ranking among defect candidates, thus giving additional useful information for PFA.

ACKNOWLEDGEMENTS

This work has been funded by the French National Research Agency (ANR) under the framework of the ANR-17-CE24-0014-01 EDITSoC (Electrical Diagnosis for IoT SoCs in automotive) project.

REFERENCES

[1] S. Pateras, "IC Test Solutions for the Automotive Market," Mentor Graphics, White Paper, May 2017.

[2] F. Hapke, et. al., "Cell-Aware Test," *IEEE Transactions on Computer-Aided Design*, vol. 33, no. 9, pp. 1396 - 1409, 2014.

[3] A. Ladhar, M. Masmoudi, and L. Bouzaida, "Efficient and Accurate Method for Intra-Gate Defect Diagnoses in Nanometer Technology," in *Proc. IEEE/ACM Design Automation and Test in Europe*, 2009.

[4] Z. Sun, A. Bosio, L. Dilillo, P. Girard, A. Virazel, and E. Auvray, "Effect-Cause Intra-cell Diagnosis at Transistor Level," in *Proc. IEEE International Symposium on Quality Electronic* Design, 2013.

[5] T.P. Ho, E. Faehn, A. Virazel, A. Bosio, and P. Girard, "An Effective Intra-Cell Diagnosis Flow for Industrial SRAMs," in *Proc. IEEE International Test Conference*, 2018

[6] P. Maxwell, F. Hapke, and H. Tang, "Cell-Aware Diagnosis: Defective Inmates Exposed in their Cells," in *Proc. IEEE European Test Symposium,* 2016.

[7] J. Tikkanen, N. Sumikawa, Li-C. Wang, and M.S. Abadir, "Multivariate Outlier Modeling for Capturing Customer Returns – How Simple It Can Be," in *Proc. IEEE On-Line Test Symposium*, 2014.

[8] Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, A. Virazel, and O. Riewer, "A Comprehensive System-on-Chip Logic Diagnosis," in *Proc. IEEE Asian Test Symposium,* 2010.

[9] Li-C. Wang, "Data Learning Based Diagnosis," in *Proc. ACM/IEEE Asia and South Pacific Design Automation Conference,* 2010.

[10] L. M. Huisman, "Diagnosing Arbitrary Defects in Logic Designs Using the Single Location At a Time (SLAT)," *IEEE Transactions on Computer-Aided Design*, vol. 23, no. 1, pp. 91, 2004.

[11] S. Holst, H-J. Wunderlich, "Adaptative Debug and Diagnosis Without Fault Dictionaries," in *Proc. IEEE European Test Symposium,* 2007.

[12] Y. Xue, X. Li, R. D. Blanton, and C. Lim, "Diagnosis Resolution Improvement through Learning-Guided Physical Failure Analysis," in *Proc. IEEE International Test Conference*, 2016.

[13] Y. Huang, W. Yang, and W. Cheng, "Advancements in Diagnosis Driven Yield Analysis: A Survey of State-of-the-Art Scan Diagnosis and Yield Analysis Technologies," in *Proc. IEEE Euro. Test Symp.*, 2015.

[14] R.J. Tikkanen, S. Siatkowski, Li-C. Wang, and M.S. Abadir, "Yield Optimization Using Advanced Statistical Correlation Methods," in *Proc. IEEE International Test Conference*, 2014.

[15] Y. Xue, O. Poku, X. Li, and R. D. Blanton, "PADRE: Physically- Aware Diagnostic Resolution Enhancement," in *Proc. IEEE International Test Conference*, 2013.

[16] S. Mhamdi, P. Girard, A. Virazel, A. Bosio, E. Faehn, and A. Ladhar, "Cell-Aware Defect Diagnosis of Customer Returns Based on Supervised Learning," *IEEE Transactions on Device Material and Reliability*, 2020. DOI: 10.1109/TDMR.2020.2992482

[17] S. Mhamdi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "A Learning-Based Cell-Aware Diagnosis Flow for Industrial Customer Returns," in *Proc. IEEE International Test Conference*, 2020.

[18] S.B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica*, vol. 31, no. 3, 2007.

[19] *http://scikit-learn.org/stable/user_guide.html*