A Learning-Based Methodology for Accelerating Cell-Aware Model Generation

P. d'Hondt [1, 2] A. Ladhar [1] P. Girard [2] A. Virazel [2]

[1] STMicroelectronics
Crolles, France
pierre.dhondt@st.com, aymen.ladhar@st.com

[2] LIRMM – Univ. of Montpellier / CNRS
Montpellier, France
girard@lirmm.fr, virazel@lirmm.fr

Abstract— **Cell-aware model generation refers to the process of characterizing cell-internal defects, a key step to ensure high test and diagnosis quality. The main limitation of this process is the generation effort, which is costly in terms of run time, SPICE simulator license usage and flow complexity. In this work, a methodology that does not use any electrical defect simulation is developed to predict the response of a cell-internal defect once it is injected in a standard cell. More widely, the aim is to use existing cell-aware models from various standard cell libraries and technologies to predict cell-aware models for new standard cells independently of the technology. A Random Forest classification algorithm is used for prediction. Experiments on several cell libraries using different technologies demonstrate the accuracy and performance of the method. The paper concludes by the presentation of a new hybrid CA model generation flow.**

I. Introduction

The usage of Cell-Aware (CA) methodology becomes mandatory for semiconductor industry, especially for designs with the highest product quality. This is because fault models like stuck-at, transition, as well as layout-aware models are not accurate enough to achieve very low Defective Part Per Million (DPPM) rates and to resolve the underlying systematic yield detractors for a successful and fast yield ramp-up. Previous works on CA defect test and diagnosis can be classified into two categories. Techniques in the first category extend the application of logic algorithms to deal with transistor defects [1-2]. The main weakness of these techniques is the quality of the logic fault models that do not properly describe the behavior of potential transistor defects. These methods are limited to cell-level diagnosis and cannot be used during test pattern generation. The second category of cell-internal defect test and diagnosis techniques relies on the realistic assumption that the excitation of a defect inside a cell is highly correlated with the logic values at the input pins of the cell [3-4]. For this category, a cell-internal-fault dictionary or *CA model* (also referred to as *CA fault model* or *CA test model in the literature*), describing the detection conditions of each potential defect affecting a cell, is used [5-6]. These techniques are more efficient and can be used to guide the test pattern generation and CA diagnosis phases. However, the main limitation of these techniques is the generation effort needed to characterize all the standard cells per technology in terms of run time, SPICE simulator licenses, CPU requirements and disk usage.

Figure 1 represents a typical CA model generation flow. It starts with a SPICE netlist representation of a standard cell which is usually derived from a layout description, e.g. a GDSII file. This DSPF (Detailed Spice Parasitic Format) cell netlist is then used by an electrical simulator to simulate each potential defect against an exhaustive set of stimuli. Once the simulation is completed, all cell-internal defects are classified into defect equivalence classes with their detection information (required input values for each defect within each cell) and are synthetized into a CA model. As our standard cells may have up to eleven inputs, and thousands of cells with different complexities are used for a given technology, the generation time of CA models for complete standard cell libraries of a given technology may reach up to several months, thus drastically increasing the library characterization process cost.

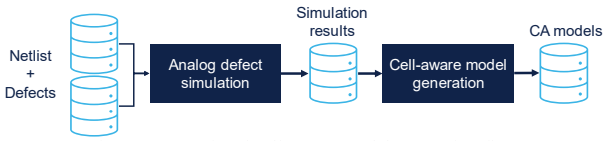


Fig. 1: Conventional cell-aware model generation flow

Based on the aforementioned, improving the generation run time of CA models and ease the characterization is required to faster deploy the CA methodology on industry designs and make it a standard in the qualification process of silicon products [7]. To this purpose, the objective of this work is to develop a methodology for predicting the behavior of cell-internal defects using Machine Learning (ML). More widely, the aim is to use existing CA models from various standard cell libraries developed using different technologies to predict CA models for new standard cells independently of the technology. To the best of our knowledge, **this is the first work to address this problem** since previous works on ML focused on cell library characterization without defect injection [8-10].

The rest of the paper is organized as follows. Section II motivates the work and presents the proposed ML-based CA defect characterization methodology. Sections III and IV describe how cell transistor netlists and cell-internal defects are represented and manipulated by the proposed methodology. Section V presents experimental results, comparison with a simulation-based approach, and a new hybrid generation flow.

II. Learning-Based CA Defect Characterization

A. Motivation

The motivation behind the use of ML for defective cell characterization is the result of several observations made while performing comparisons between several CA models coming from different standard cell libraries and technologies:

- Several cell internal defects, such as stuck-open defects, are independent of the technology and transistor size [11-12].

• For the same function, two cell structures are quite similar for two different technologies.
• Detection tables for static and dynamic defects, in the form of binary matrices describing the detection patterns for each cell-internal defect, are ML friendly.
• CA models may change with respect to test conditions and PVT corners. In fact, CA model generation for the same cell with different test conditions may exhibit slight differences. Few defects can be of different types (i.e. static or dynamic) or may have different detection patterns. Since CA models are generated for specific test conditions and can be used with different ones, it may lead to inaccurate characterization. This inaccuracy is usually allowed in industry since it is marginal. This indicates that we can also tolerate few error percentages in our ML-based prediction.
• Very simple CA models are used to emulate short and open defects, for which resistance values are often identical for all technologies.
• A large database of CA models is available and can be used to train a ML algorithm.

All these observations intuitively indicated that CA model generation through ML could be possible. However, the first challenging task was to be able to describe cell transistor netlist as well as corresponding cell-internal defects in a uniform (standardized) manner, so that a ML algorithm can learn and infer from data irrespective of their incoming library and technology. Indeed, similar cells (e.g. cells with same logic function, same number of inputs and same number of transistors) may be described differently in transistor-level (SPICE) netlists of various libraries (e.g. a transistor label does not always correspond to the same transistor in two similar cells coming from two different libraries), and it was therefore mandatory to standardize the description of cells and corresponding defects for our ML-based defect characterization methodology. Heuristic solutions developed to this purpose are described in Sections III and IV. The second challenging task was to find a way to represent all these information / input data so that they can be ML friendly. A matrix description of cells and corresponding defects was chosen to this purpose.

B. Proposed Defect Characterization Methodology

The proposed learning-based defect characterization methodology is used to predict the behavior of a cell affected by intra-cell defects, hence avoiding costly electrical defect simulations. The proposed flow is sketched in Fig. 2. It is based on supervised learning that takes a known set of input data and known responses (*labeled data*) used as training data, trains a model to classify those data, and then uses this model to predict (*infer*) the class of new data.

Training data are made of various and numerous CA models formerly generated by relying to brute-force electrical defect simulations. For each cell in a library, the CA model is transformed into a so-called *CA-matrix* and filled in with meaningful information. Cells with the same number of inputs and having the same number of transistors are grouped together to form the Training dataset.

The CA-matrix creation flow is depicted in Fig. 3. The flow starts by rewriting the CA model so that it can be ML friendly. Then, it identifies the activation conditions of each transistor with respect to input stimuli. Once the activation conditions for each transistor have been identified, transistor renaming is done. This is a critical step in this flow since it allows the usage of the training data across different libraries and technologies. Finally, the CA-matrix is created with the above information.

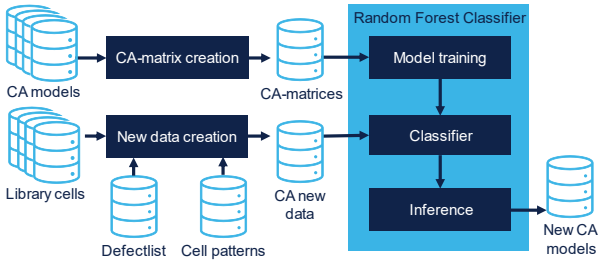


Fig. 2: Generic view of the ML-based defect characterization flow

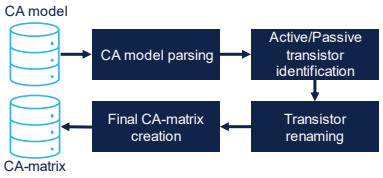


Fig. 3: Generic view of the CA-matrix creation flow

Table I shows an example of a training dataset for an NAND2 cell. It is composed of four types of information:

• *Cell patterns and responses*. This gives the values applied on input (A, B) as well as the cell response on output Z. As can be seen, the test pattern sequence provides all the possible input stimuli that can be applied to the cell. These stimuli must also be efficient to detect sequence depending defects like stuck-open defects. For this reason, a four-valued logic algebra made of 0, 1, R and F is used to represent input stimuli in the CA-matrix. R (resp. F) represents a Rising (resp. Falling) transition from 0 to 1 (resp. from 1 to 0).
• *Active / Passive Transistor Identification*. This indicates the activation conditions of each transistor in the cell schematic. Each transistor can be in the following state: active, passive, switching to active state, switching to passive state.
• *Defect description*. This gives information about defect locations in the cell transistor schematic.
• *Defect detection*. This is the class of the data sample (the output of ML classifier). A value '1' ('0') means that the defect is detected (undetected) by the input pattern.

The first three types of information constitute the inputs of the ML algorithm.

TABLE I. EXAMPLE OF TRAINING DATASET FOR A NAND2 CELL

Cell inputs & responses			**Transistor switching activity**				**Defect description**				**About defect**		**Defect detection**
A	**B**	**Z**	**N0**	**N1**	**P0**	**…**	**N1_D**	**N1_G**	**N1_S**	**…**	**name**	**type**	**fZ**
0	0	1	0	0	1	…	0	0	0	…	free	free	0
0	1	1	0	1	1	…	0	0	0	…	free	free	0
0	F	1	0	F	1	…	0	0	0	…	free	free	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	0	1	1	…	1	0	1	…	D15	short	1
1	1	0	1	1	0	…	1	0	1	…	D15	short	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

New data represent the cells to be characterized and are obtained for each standard cell from the cell description,

corresponding list of defects and cell patterns. The format of a new data instance is similar to that of the training data, except that the class (label) of the new data instance is missing. The ML classifier is used to predict that class. As for training data, new data are grouped together according to their number of cell inputs and transistors, so that inference can be done at the same time for cells with the same number of inputs and transistors.

Figure 2 depicts the **two main steps** of the **supervised learning process** used for ML-based CA model generation. In this work, we use a Random Forest Classifier for predicting the class of each new data instance. This choice comes from the results obtained after experimenting several learning algorithms (k-NN, Support Vector Machine, Random Forest, Linear, Ridge, etc.) and observing their inference accuracies. So, the **first main step** of our CA diagnosis flow consists in generating a Random Forest model and to train it by using the training dataset. A Random Forest Classifier is composed of several Decision Tree Classifiers, which are models predicting class of samples by applying simple decision rules. During training, a Decision Tree tries to classify data samples and its decision rules are modified until it reaches a given quality criterion. Then, the Forest averages the responses of all Trees and outputs the class of the data sample. The **second main step** consists in using the Random Forest Classifier to make prediction (or inference) when a new data instance has to be evaluated. Prediction for a new data instance amounts to answer to the question: "Does this stimulus detects this defect affecting this cell?". Answering to this question allows obtaining a new CA model for a given standard cell.

III. Cell Representation in the CA-Matrix

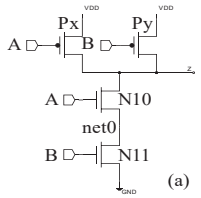
In this section, we detail the various steps required to take a cell from a transistor-level (SPICE) netlist and represent it in the CA-matrix in a standardized and ML friendly manner. This description must be accurate enough to clearly identify each transistor and each net of the cell transistor schematic. Moreover, this description must also be able to associate each transistor to its sensitization patterns and to report the output response for each pattern. For this reason, the cell description process requires a number of successive operations that are detailed below. Note that this process is applied to all cells in a library to be characterized.

A. Identification of Active and Passive Transistors

The first step consists in identifying active and passive transistors in the cell netlist with respect to an input stimulus. This information is used to represent the transistor netlist in the matrix format. To this end, we perform a single defect-free (golden) electrical simulation of each cell to be characterized, so as to identify active and passive transistors for each input stimulus (test pattern) and to measure the corresponding cell output value. An active NMOS (resp. PMOS) transistor is a transistor with a logic-1 (resp. logic-0) value measured on its gate terminal. A passive NMOS (resp. PMOS) transistor is a transistor with a logic-0 (resp. logic-1) value measured on its gate terminal. *Note that a Verilog simulation, with a CDL (Circuit Description Language) netlist that should be written using NMOS and PMOS primitives, can replace the single defect-free electrical simulation.* With this information, we can associate each cell pattern to the list of active transistors in the cell. After this step, the CA-matrix contains the columns:

- Cell inputs & responses columns. They contain all input stimuli (test patterns) that can be applied to the cell, and the corresponding responses.
- Transistor switching activity columns. They contain four possible values indicating if the transistor is active (1), passive (0), switching from an active state to a passive one (F) or switching from a passive state to an active one (R). Since PMOS and NMOS transistors are activated in opposite way, we use the '-' character before the PMOS values.

Let us consider the transistor schematic of an NAND2 cell as shown in Fig. 4.a. In the partial representation of the CA-matrix of the cell shown in Fig. 4.b, columns A and B list all the possible input stimuli for this cell. These columns also define the length of the CA-matrix, which is equal to $2^n + 2^n.(2^n-1)$, n being the number of cell input pins (2^n is the number of static stimuli, $2^n.(2^n-1)$ is the number of dynamic stimuli). For each stimulus, active and passive information about each transistor of the cell is entered in the CA-matrix. For example, AB=00 leads to two active PMOS transistors and two passive NMOS transistors.



A	B	Px	Py	N10	N11
0	0	-1	-1	0	0
0	1	-1	0	0	1
1	0	0	-1	1	0
...	...	...	...	...	...
R	R	-F	-F	R	R

(b)

Fig. 4: Example standard cell NAND2: (a) cell transistor schematic and (b) partial CA-matrix representation

This matrix representation of the cell schematic is strongly dependent on the transistor names and the order they are defined in the SPICE netlist. In fact, two similar cell schematics may have different transistor naming and the order of transistors in the SPICE netlist may differ from cell to cell and from one technology to another. Without an accurate naming convention of each cell transistor in the CA-matrix, any ML algorithm will fail to predict the behavior of the cell in presence of a defect. To avoid this problem, a second step in the cell description process is required. This step consists in renaming all cell transistors independently of their initial names and order in the input SPICE netlist. The algorithm developed to this purpose is detailed in the next subsection.

B. Renaming of Transistors

In the CA-matrix, two cells with the same *transistor structure* will have the same transistor names irrespective of their incoming library and technology. A *transistor structure* is a virtual SPICE netlist without specification of the connections between transistor gates, i.e. only source and drain connections between transistors are listed. The transistor-renaming algorithm consists of the following two steps:

Determination of branch equations. Since the transistor gate connections are not considered, the transistor structure is composed of one or more branches. A branch is a group of transistors connected by their drain and source terminals. The entry of each branch is the set of transistor gates and the exit is the connection net between the NMOS and PMOS transistors. A branch is connected to a power and/or a ground net. A branch equation is a Boolean-like equation describing how the transistors of the branch are connected, using Boolean-and (symbolized by '&') for serial transistors or serial groups of

transistors, and Boolean-or (symbolized by '|') for parallel transistors or parallel groups of transistors.

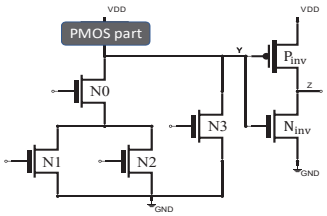


Fig. 5: Example schematic

Let us consider the example schematic in Fig. 5. The structure is composed of two branches. The two-transistors output-inverter is the simplest branch whose input is net Y and output is net Z. The inverter creates two paths between branch output and power nets, so its branch equation is (N_{inv}|P_{inv}). The equation of the second branch (NMOS branch driving net Y) is ((N0&(N1|N2))|N3). To do not rely on any name present in the SPICE netlist, the branch equations are anonymized, i.e. a NMOS is described by '1n' and a PMOS by '1p'. The anonymized equation of the NMOS branch driving net Y in Fig. 5 is therefore ((1n&(1n|1n))|1n).

Sorting of branch equations. Once all the branch equations for the considered cell have been determined, they are sorted by using deterministic criteria:

- *Level of each branch.* It is defined in ascending order with respect to the cell output (level-1 branches drive the cell output, level-2 branches drive the gates of transistors in level-1 branches, and so on and so forth),
- *Number of transistors in each branch* - in ascending order,
- *Anonymized branch equation defined in alphabetical order.*

C. Identification of Parallel Transistors

Identifying branch equations is not enough to rename all transistors of a given cell. The problem comes from parallel transistors. In fact, two or more parallel transistors share the same source and drain, which makes their identification quite difficult. For example, in Fig. 5, transistors N1 and N2 can be either presented as "N1|N2" or as "N2|N1". As the order of transistors in each branch will determine how transistors will be renamed, we cannot accept such confusing situation. A solution consists in sorting transistors inside their branch according to their activity with respect to the input stimuli. The algorithm developed to this purpose proceeds as follows. For each transistor, an *activity value* is computed. For a cell with n inputs, the activity value is a 2^n-bit integer that represents the accumulative activation state of a transistor for all possible stimuli applied to the cell. The input stimuli range from (0,0…,0) to (1,1…,1). For each of these stimuli, the transistor is either active (1) or passive (0). We define the activity value as a binary number, whose MSB is the activity of the transistor under input stimulus (0,0…,0) and LSB is the activity of the transistor under input stimulus (1,1…,1), with a bit significance decreasing for increasing binary value of input stimulus.

To compute the activity value, we need to know whether the transistor is active or passive for each input stimulus. We already have this information in the CA-matrix as described in Section III.A. To illustrate this process, activity values for the transistors of the NAND2 cell in Fig. 4.a are given in Table II.

TABLE II. ACTIVITY VALUES FOR THE NAND2 CELL IN FIG. 4.A

			old names			
A	B	Comment	Px	Py	N10	N11
0	0	MSB	1	1	0	0
0	1		1	0	0	1
1	0		0	1	1	0
1	1	LSB	0	0	1	1
Activity value			12	10	3	5
			↓ Renaming ↓			
			P1	P0	N0	N1

Finally, transistors of each branch are sorted by increasing activity value to give the final description of the cell in the CA-matrix. For our example in Fig. 4, this leads to: first NMOS transistor of the first branch is named N0, second NMOS transistor of the first branch is named N1, first PMOS transistor of the second branch is named P0, and second PMOS transistor of the second branch is named P1.

IV. DEFECT REPRESENTATION IN THE CA-MATRIX

In this section, we detail the cell-internal defects representation in the CA-matrix in a standardized and ML friendly manner. Cell internal defects are classified into:

- *Intra-transistor defects*. These defects affect transistor terminals (source, drain, gate and bulk) and can be either an open defect or a short. In order to describe these defects, all transistor terminals are listed as a column in the CA-matrix (cf. Table I). For an open defect, a value '1' indicates that this transistor terminal is affected by the defect, '0' otherwise. For a short, a value '1' on two transistor terminals indicates that a short exists between these two terminals, '0' otherwise.
- *Inter-transistor defects*. These defects affect a connection(s) between at least two different transistors. Though these defects are not considered in this work, the matrix representation is flexible enough to represent them. For these defects, we use the same representation mechanism as for intra-transistor defects

TABLE III. NAND2 CELL IN FIG. 4: EXAMPLE OF DEFECT COLUMNS

P0_S	P0_D	P1_S	P1_D	N0_S	…	N1_D	Comment
0	0	1	1	0	…	0	source-drain short on P1
1	0	0	0	1	…	1	net0 & P0-source short

Table III is an example of defect description in the CA-matrix of the example NAND2 cell in Fig. 4. Row with red cells describes the intra-transistor short defect between drain and source terminals of transistor P1 (formerly Px). Row with blue cells describes the inter-transistor short defect between P0-source and "net0" (net0 connects N0-source and N1-drain).

V. EXPERIMENTAL RESULTS

A. Results of Prediction Accuracy

We implemented our method in a python program. The ML algorithms were taken from the publicly available python module called scikit-learn. Our dataset was composed of 1712 standard cells coming from standard cell libraries developed using three technologies (C40 (446 cells), 28SOI (825 cells) and C28 (441 cells)). All these cells already had a CA model generated by a commercial tool. We generated the CA-matrix for each cell. The method was experimented in two different ways. First, the ML model was trained and evaluated using cells belonging to one technology. Second, we trained the model on one technology and evaluated it on another one.

1) Predicting defect behavior on the same technology

We first trained the ML model on cells of 28SOI standard cell libraries. As mentioned earlier, cells were grouped

according to their number of transistors and inputs. For *m* cells available in a given group, we trained the ML model over *m-1* cells and evaluate its **prediction accuracy** on the *m*-th cell. A loop ensured that each cell is used as the *m*-th cell. On average, a group contains 8.6 cells. In this work, we considered all possible open and short defects (static and dynamic) in each cell. Results presented below report the prediction for open defects. Results achieved for short defects are similar.

Table IV.a presents the prediction accuracy achieved for open defects. Due to lack of space, only results for cells with less than 7 inputs and 48 transistors are reported (we experimented on cells with up to 8 inputs and 112 transistors). Non-empty boxes report the **average** prediction accuracy obtained **for a group of cells**. Empty boxes mean that there is zero or one cell available and that the group cannot be evaluated. A green background indicates that the maximum prediction accuracy in this group is 100%, i.e. the ML model can perfectly predict the defective behavior of at least one cell. In contrast, white background indicates that no cell was perfectly predicted in that group (all prediction accuracies are less than 100%). For example, let us consider the circled box in Table IV.a. We have 24 cells having 4 inputs and 24 transistors: (i) 15 cells are perfectly predicted (100% accuracy), which leads to a green background, (ii) the prediction accuracy for the 9 remaining cells ranges from 99.82% to 99.99%, (iii) the average prediction accuracy over all 24 cells is 99.97%.

These results show that the ML model can accurately predict the behavior of a cell affected by a given defect and that our method could be used to generate CA models. The goal of the next subsection is to leverage on existing CA models to generate CA models for a new technology.

2) Predicting defect behavior on another technology

We also conducted experiments on cells belonging to two different technologies. Evaluation was slightly different compared to the previous one. Here, the ML model was trained over all available cells of a given technology and the evaluation was done on one cell of another technology. A loop was used to allow all cells of the second technology to be evaluated. Cells were grouped according to their number of inputs and transistors. Table IV.b shows the prediction accuracy achieved on open defects of the C28 cells after training on the 28SOI cells. Results are averaged over all cells in each group (same number of inputs and number of transistors). The average prediction accuracies are globally lower compared to those of Table IV.a. After investigating on this point, we noticed that the behavior of most of the cells (68% of cells) is accurately predicted (accuracy > 97%), while accuracy for few cells is quite low. This phenomenon is discussed in Section V.B.

To verify the efficiency of our method when different transistor sizes are considered, we trained the ML model over the 28SOI standard cells and used it to predict the behavior of C40 cells. Table IV.c shows the prediction accuracy achieved on open defects of the C40 cells after training on the 28SOI cells. Results are averaged over all cells in each group (same number of inputs and transistors). This time, 80% of cells are accurately predicted (accuracy > 97%), proving that our ML-based characterization methodology could be used to generate CA models for a (large) part of cells of a new technology.

B. Analysis - Discussion

We first analyzed cells for which the defect characterization methodology gives excellent prediction accuracy as well as those for which the prediction accuracy was quite low. Then, we investigated the limitation of the proposed method for CA model generation. After running several experiments on different configurations using one fault model at a time, we noticed the following behaviors:

- Accuracy for most of the cells is excellent, i.e. more than 97% prediction accuracy for 70% of cells. In this case, **the CA model generated by ML fit the real behavior achieved with electrical simulation**.
- Accuracy for few cells (30%) is quite low and the ML prediction is not accurate.

For the first cell category with good prediction score, cells have been analyzed manually to identify why they led to good results. The analysis showed that all these cells had at least one cell in the training dataset with the same transistor structure or a very similar one. The difference between very similar cells is always the same and is represented in Fig. 6. More precisely, cells giving good results are always composed of one of the configurations presented in Fig. 6 and at least one cell of the training dataset contains the other configuration. The difference between these two transistor configurations is the presence or absence of the red net. The logic function of these configurations is the same. These configurations are mostly found in high-drive cells.

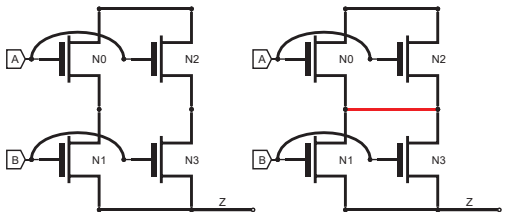


Fig. 6: Typical transistor configurations leading to good prediction

For the second cell category – cells leading to poor prediction accuracy – the manual analysis showed that they have (i) new logic functions that do not appear in the cells of the training dataset, or (ii) a transistor configuration which is completely new when compared to cells in the training dataset.

C. Hybrid Flow for CA Model Generation

Considering the above analysis, it appears that the ML-based CA model generation flow cannot be used for all cells in a standard cell library to be characterized. A mixed solution, which consists in combining ML-based CA model generation and conventional (simulation-based) CA model generation, should be preferably used. This is illustrated in the following.

We propose to use the flow sketched in Fig. 7 for accelerating the CA model generation. Typically, when the CA model for a new cell is needed, we first check if the ML-based generation will lead to high-quality CA models. This is done by analyzing the structure of the new cell and check whether the training dataset contains a cell with identical or similar structure (as presented in V.B). If the ML algorithm is expected to give good results, the new cell is prepared (representation in a CA-matrix) and submitted to the trained ML algorithm.

TABLE IV. AVERAGE PREDICTION ACCURACY FOR (A) CELLS IN THE SAME TECHNOLOGY; (B) CELLS IN DIFFERENT TECHNOLOGIES; (C) CELLS WITH DIFFERENT TRANSISTOR SIZES

(a)

Number of transistors	Prediction accuracy (%)	Number of inputs 2	3	4	5	6
	6	99.98	99.99			
	8	99.91	99.96	99.91		
	9		100.0			
	10	99.98	99.81	99.96		
	12	99.72	99.73	100.0	99.91	99.93
	14	99.70	99.56	99.83	99.92	99.96
	16	99.99	100.0	99.94		99.98
	18	99.99	99.94			
	20	100.0	99.98	100.0	99.73	
	22		99.84	99.98	99.62	
	24	100.0	99.84	99.97		99.85
	26	100.0	99.70	100.0		99.89
	28	99.49	99.98	100.0	99.88	99.81
	30	99.75	100.0	100.0		
	32	100.0	100.0			99.98
	42		100.0			
	44		100.0			
	46		99.81			
	47		99.98	99.95		

(b)

Number of transistors	Prediction accuracy (%)	Number of inputs 2	3	4	5	6
	6	98.21	99.47			
	8	94.56	96.86	99.00		
	9					
	10	94.69	96.01	99.27		
	12	87.73	98.05	99.10		99.76
	14	85.69	97.35	98.75		
	16	91.74		99.20		
	18	88.18	96.28			
	20	90.29	94.37			
	22	78.73		98.37		
	24	87.91	96.88	99.37		99.79
	26	87.24	98.92			
	28	88.18	98.68			
	30			97.52		
	32	88.73	95.6			
	42					
	44					
	46					
	47					

(c)

Number of transistors	Prediction accuracy (%)	Number of inputs 2	3	4	5	6
	6	100.0	99.80			
	8	87.39	99.14	99.03		
	9		97.19			
	10	92.07	95.49	99.32	98.46	
	12	91.71	98.07	99.24	98.47	99.46
	14	90.1	95.84	98.63	98.79	99.52
	16	91.17	93.59	99.23		99.59
	18	88.5	97.15	97.14	97.74	
	20	83.87	97.73	97.15	98.94	
	22	87.26		98.98	98.44	
	24	93.96	99.34	98.58	98.84	99.63
	26	87.52	97.55	99.04	99.02	99.92
	28	98.19		98.79	99.31	99.44
	30			99.13	99.37	99.58
	32	92.91			98.92	99.78
	42					
	44	92.03	98.82			
	46		99.23			
	47		98.29	99.76		

The output information is then parsed to the desired file format. Conversely, if the ML algorithm is expected to give poor prediction results, the standard generation flow presented in Fig. 1 is used to obtain the CA model. A feedback loop uses this new simulated CA model to supplement the training datasets and improve the ML algorithm for further prediction.

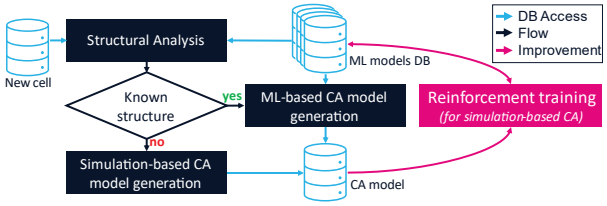


Fig. 7: Hybrid flow for CA model generation

To estimate the improvement in CA model generation time achieved with the flow in Fig. 7, we performed the following experiments. We trained the Random Forest model on 28SOI standard cells and generated CA models for a subgroup of the C40 standard cell libraries. A subgroup is composed of cells representing all the cell functions available in C40 libraries. In our experiments, this subgroup contained 409 cells: 118 (29%) have a cell with an identical structure in the training dataset, 87 (21%) have a cell with an equivalent structure (as explained in Section V.B) in the training dataset, and 204 (50%) have no identical or equivalent structure in the training dataset (a simulation-based generation is thus needed). For these 204 cells, the generation time was calculated and found to be equal to ~172 days (~ 5.7 months) considering a single SPICE license. Using the ML-based CA model generation for the 118 + 87 = 205 (50%) remaining cells requires 21947 seconds (~ 6 hours), again considering a single SPICE license. Considering that a simulation-based generation for these 205 cells would require ~78 days, we can estimate the reduction in generation time to **99.7%**. Now, if we consider the whole C40 subgroup composed of 409 cells, the hybrid generation flow would require ~172 days + ~6 hours, to be compared with ~172 days + ~78 days = ~250 days by using only the simulation-based generation. This represents a reduction in generation time of about **38%**. After investigating results of these experiments, we observed that the ML-based CA model generation works well for about 80% of cells of the C40 subgroup. Surprisingly, the structural analysis revealed that only 50% (205 cells) could be evaluated using the ML-based generation part of the flow. This shows that there is still room for further improvement of the structural analysis in our flow, and hence get better performance of the ML-based CA model generation process.

To conclude, experiments in sections IV.A and IV.B have been carried out on a reasonable size (1712) of standard cell population. Considering that more than 10000 cells have usually to be characterized for a given technology, the hybrid flow in Fig. 7 is expected to provide even better results, especially owing to the reinforcement training that uses simulation generated models for supplementing the training datasets, and hence reduce the number of electrical simulations.

REFERENCES

[1] 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.

[2] 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 Symp. on Quality Electronic Design*, 2013.

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

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

[5] F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, P. Weseloh, M. Wittke, M. Kassab, and C. W Schuermyer, "Cell-Aware Fault Model Creation And Pattern Generation," US Patent 12/718,799, 2010.

[6] 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 Conf.*, 2020.

[7] R. Guo, B. Archer, K. Chau, and X. Cai, "Efficient Cell-Aware Defect Characterization for Multi-bit Cells", in *Proc. IEEE International Test Conf. in Asia*, 2018.

[8] "Nanometer Library Characterization: Challenges and Solutions", *Webinar*, Silvaco, March 2019.

[9] "Improving Library Characterization with Machine Learning", *White Paper*, Mentor, A Siemens Business, 2018.

[10] "Unified Library Characterization Tool Leverages Machine Learning in the Cloud", *White Paper*, Cadence, 2018.

[11] S. Venkataraman and S.D. Drummonds, "A Technique for Logic Fault Diagnosis of Interconnect Open Defect", in *IEEE VLSI Test Symp*, 2000.

[12] C.-M. Li and E.J. McCluskey, "Diagnosis of Resistive-Open and Stuck-Open Defects in Digital CMOS ICs", *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 24, no 11, pp. 1748 – 1759, 2005.