Capacitance Extraction via Machine Learning with Application to Interconnect Geometry Exploration

Cheng-Yu Tsai
Georgia Institute of Technology
Atlanta, Georgia, USA
cytsai@gatech.edu

Suwan Kim
Samsung Electronics
Suwon, South Korea
sw245.kim@samsung.com

Sung-Kyu Lim
Georgia Institute of Technology
Atlanta, Georgia, USA
limsk@ece.gatech.edu

***Abstract*—As Moore's law slows down, foundries and design houses are resorting to design technology co-optimization (DTCO) to squeeze more performance out of a technology node. However, the efficiency of EDA tools plays a key role in driving DTCO. The existing commercial parasitic extraction tools are not able to respond to a process parameter change efficiently. For example, it takes 25 minutes to re-generate a parasitic netlist for a mere layer thickness change using existing commercial tools. This runtime becomes the bottleneck for standard cell DTCO. In this work, we demonstrate a machine-learning-based method targeted on standard cells that can efficiently extract parasitic capacitance within seconds while maintaining competitive error distribution compared to the state-of-the-art rule-based 2.5D extraction method, which suffers from pattern mismatch since no real layout is provided at the pre-characterization stage. We extract patterns from actual standard cell layouts as training data for the ML model, and the model can predict coupling capacitance with unseen layer thickness within milliseconds.**

***Index Terms*—parasitic extraction, capacitance extraction, machine learning, pattern extraction, 2.5-D extraction**

I. Introduction

As the process technology advances, the contribution of parasitic resistance and capacitance to the delay and power continuously increases due to the shrinking size of wires [1]. The role of parasitic extraction (PEX) is becoming increasingly critical for cell and chip design. Additionally, with the slowing pace of Moore's law, foundries and designers resort to design-technology co-optimization (DTCO) to squeeze even more performance, power, or area (PPA) gain out of a technology node. However, the efficiency of DTCO exploration relies heavily on the runtime of EDA tools. The existing EDA tools for parasitic extraction are still not efficient enough to drive iterative DTCO, particularly in process parameter exploration, such as metal and dielectric thickness, resistivity, via height, etc.

Fig. 1 illustrates the first motivation of our method. We use ML models as a storage medium to store information from multiple interconnect technology files (ITF) into a model to expedite process parameter exploration and enable interpolation. As a comparison, the existing tools are only good at handling layout changes. To evaluate process changes to PPA gain, none of the commercial tools can handle these changes efficiently. For example, for StarRC to handle a metal and dielectric layer thickness change, the pre-characterization must be executed again, which can take hours or even days for commercial nodes. This becomes a significant obstacle for process parameter exploration.

Secondly, we use ML to prevent the manual study of pattern generation and curve fitting. As will be introduced in Section II-A, state-of-the-art commercial tools, such as Synopsys StarRC, Cadence QRC, and Siemens xACT, embrace the idea of pre-characterization. These tools use field solvers to compute a finite number of patterns and store them in a pre-characterized library (Pre-K lib); thus, the extraction becomes querying a database or computing an analytical formula. However, given the diverse design rules, it is impractical to enumerate and store all possible patterns in a database. Thus, generating representative layout patterns to study and meticulous curve fitting is required, which requires not only manual time but also expertise. Nevertheless, the problem can be considered a regression problem and solved or approximated by machine learning (ML) models. With enough training datasets and careful design of the ML models to ensure generalization, the ML approach can be highly automated with minimum expertise. As illustrated in Fig. 2, in this work, we translate a list of shape coordinates into a feature matrix, and with the capacitance values computed from a field solver, we train an ML model that can predict the coupling capacitance between two metal shapes given a pattern.

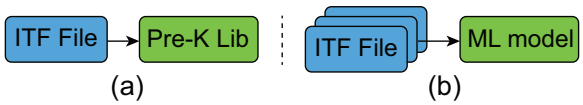

Fig. 1: Comparison of ITF processing workflows. (a) SOTA commercial tools convert each ITF file into a pre-characterized library (Pre-K Lib), requiring reprocessing upon changes. (b) The proposed method encodes multiple ITF files into a single set of ML models, eliminating repeated pre-characterization.

We summarize our contributions in this paper as follows:

- Introduced our custom adaptive cross-section pattern extraction algorithm for extracting patterns as training data.
- Introduced appropriate filtering strategies for the ML models to handle a reasonable range of number of shapes in a pattern.
- Demonstrated a way to encode layer thickness into feature vectors for ML models and the potential for interpolating and extrapolating unseen layer thickness.
- We compared deep neural network with LightGBM and concluded that LGB, which is 10x faster, suffices for most cases.

II. Background

A. Pattern-based Extraction

State-of-the-art parasitic extraction tools—such as Synopsys StarRC [2], Cadence QRC [3], and Siemens xACT [4]—employ pattern-based extraction, also referred to as 2.5-D extraction or rule-based extraction. As illustrated in Fig. 3, the pattern-based methodology involves two primary phases: pre-characterization and extraction.

In the pre-characterization stage, the tool generates a set of representative 2-D cross-sectional patterns. These patterns, together with process-specific information, are analyzed using a 2-D field solver to compute resistance and capacitance (R/C) per unit length. The resulting values are stored in a lookup database and are used for curve fitting to handle potential pattern mismatches during extraction.

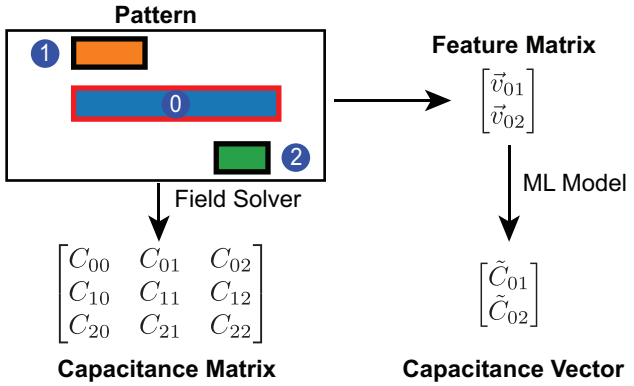

Fig. 2: Capacitance extraction from a given pattern (a list of shapes) using a traditional field solver vs. machine learning.

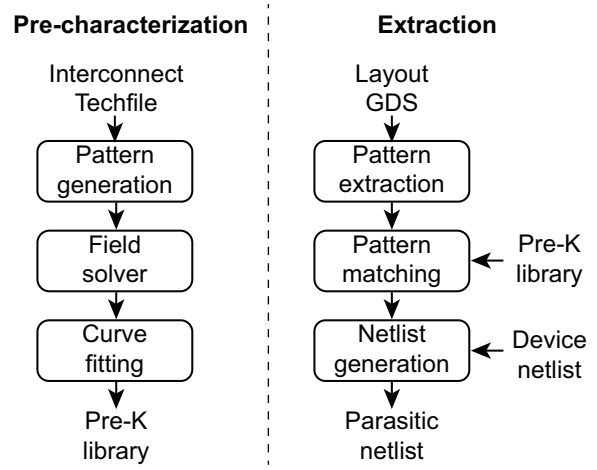

Fig. 3: Pattern-based extraction includes two stages: pre-characterization and extraction.

In commercial tools, the pre-characterization phase typically relies solely on process data, with pattern generation performed independently of the actual layout. Some even claimed they are generated empirically or manually [5]. In our experiments using an academic process design kit (PDK), this stage requires approximately 25 minutes when executed with 64 CPU threads. For advanced commercial technology nodes, the process may span several days due to increased complexity.

During the extraction phase, the layout is decomposed into potentially thousands of 2-D cross-sectional slices along both the x and y directions. For each extracted pattern, R/C values are retrieved from the pre-characterized database or estimated via interpolation when an exact pattern match is unavailable. Finally, the extracted values are assembled and translated into a parasitic netlist. This table-lookup-based approach significantly accelerates extraction, offering performance gains of several orders of magnitude compared to full 3-D field solver techniques, where numerical solutions are computed directly.

B. Related Works

The concept of 2.5-D parasitic extraction dates back to the 1990s. Cong et al. [6] laid the foundational work and successfully integrated 2.5-D extraction techniques into a commercial tool. The integration of machine learning into parasitic extraction emerged in the late 2010s. Kasai et al. [7] employed a multilayer perceptron (MLP) for capacitance extraction in 3D integrated circuits (ICs). Li et al. [5] utilized a deep neural network (DNN) as a classifier to dynamically select appropriate extraction formulas based on layout pattern geometries. Yang et al. [8] introduced a density map representation and demonstrated its effectiveness using convolutional neural networks (CNNs) for parasitic extraction. However, these initial studies were limited to synthetic pattern datasets.

A significant advancement was made by Abouelyazid et al. [9], who were the first to report extraction results based on real design layouts. They conducted a comprehensive analysis of pattern extraction setups and proposed a vertex-based feature representation to enable DNNs to effectively model process variations. In a subsequent study [10], the same authors developed a hybrid approach capable of automatically selecting the fastest method—traditional rule-based, DNN, or field solver—based on the user-defined accuracy requirements. More recently, Tsai et al. [11] proposed a residual learning framework combining a linear model with a DNN. In this architecture, the DNN is responsible only for learning the small residuals rather than the full capacitance values. Their method, based on a full 3-D extraction paradigm, achieves both high accuracy and computational efficiency. Despite these advances, existing approaches do not support exploration across varying layer thickness configurations, highlighting a key limitation in current methodologies. Additionally, none of them shared enough details of how they extracted their patterns from layouts.

TABLE I: Terminologies used in this paper.

Name	Description
Shape	A single piece of metal wire segment connecting two points in a layout, or a conductor polygon in a cross-section
Pattern	A 2-D cross-section, which includes a target shape and one or more neighbor shapes. We compute coupling capacitance between the target and each of its neighbors as depicted in Fig. 2
Target shape	Also called an aggressor. We allow every shape to be the target shape in turn and create cross-section patterns. Often drawn in red edges in our figures
Victim	A neighbor of the aggressor. We compute the capacitance between an aggressor and a victim
Pattern extraction	The process of scanning the layout and creating cross-section patterns. Will be introduced in Section IV
Target layer	The metal layer that the target shape is in
Secondary layer	A metal layer other than the target layer
Layer combination	A combination of a target layer and one or more secondary layers. We train one ML model for each layer combination
Feature vector	The numerical input for ML model to predict a capacitance value
Feature matrix	Concatenation of multiple feature vectors as ML model input to predict multiple capacitance values
ITF	Interconnect technology file for Synopsys tool, equivalent to ICT file in Cadence tool

III. Overview of Our Approach

In this section, we list the terminologies we will use throughout the paper in TABLE I and provide a high-level overview of our method.

As illustrated in Fig. 4, our proposed machine learning (ML)-based parasitic extraction flow comprises two main stages: *training* and *inference*. Each step depicted in Fig. 4 will be explained in detail in the following paragraphs, with supplementary figures referenced where helpful.

During the training stage, the inputs include a standard cell library and multiple interconnect technology files (ITFs). From the

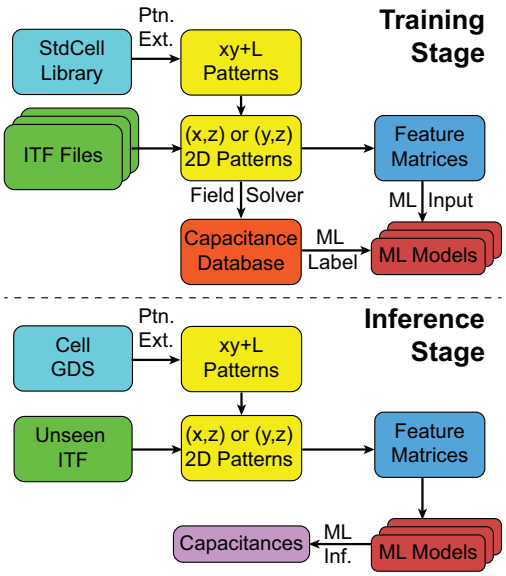

Fig. 4: Overview of our flow. During the training stage, we extract patterns from a stdcell library and use them to train the ML models. During the inference stage, our tool calculates capacitances from an input GDS and ITF file using the ML models.

cell library, our tool extracts cross-sectional patterns, which are represented by $(x, y,$ layer$)$ as denoted by "xy+L" in Fig. 4. The procedure for pattern extraction is described in Section IV. Using the other input, ITF files, we map each GDS layer name to a corresponding z-coordinate, thereby enabling 3-D spatial characterization. The resulting cross-sectional patterns, exemplified in Fig. 6(c)(e), are then calculated using a 2-D field solver—in our case, Synopsys Raphael [12]—to compute coupling capacitances between all relevant shape pairs within each pattern. Simultaneously, the extracted patterns are encoded into feature matrices, as illustrated in Fig. 2. The construction and semantics of these feature vectors are further discussed in Section V-B. With both the input features and target labels (capacitance values) available, we train the ML models to generalize across various layer thickness configurations.

In the inference stage, the workflow mirrors the training phase. Patterns extracted from a cell layout are transformed into feature matrices using the layer thickness information provided by a new ITF file. The pre-trained ML model then predicts the coupling capacitance directly from these feature matrices. Our ML-based methodology enables accurate parasitic capacitance estimation for unseen layer thicknesses without requiring a time-consuming pre-characterization process, which is typically necessary in existing commercial tools. Notably, the model is trained once and can generalize across varying technology configurations, offering substantial improvements in efficiency and adaptability.

IV. Pattern Extraction Algorithm

This section describes our algorithm to extract cross-section patterns from a cell layout, the first step in Fig. 4. These patterns will later become the training data of our ML models. We first introduce how our algorithm works, which iterates over every shape in the input cell layout. Then there are three sub-steps in the algorithm that we will introduce in more detail: sorting, filtering, and indexing.

A. Overview

As shown in Fig. 2, a *pattern* in this paper refers to a 2-D cross-section involving multiple metal *shapes*, or a list of shapes. A 2-D field solver is used to compute the capacitance between each pair of these shapes, yielding a capacitance matrix as shown in Fig. 2. The shape highlighted in red in the pattern is called a *target shape* or *aggressor*. These two terms will be used interchangeably throughout the paper. On the other hand, its neighbors are called *victims*. Our goal is to extract the coupling capacitance between the aggressor and every victim.

We describe our pattern extraction algorithm in Algorithm 1, and illustrate the key steps in the algorithm in Fig. 5, albeit the nested loop cannot be fully expressed in the flow chart. The algorithm has three nested for loops to extract cross-section patterns from a cell. The first for-loop iterates over all metal layers, and the iterator is *target layer*. The second for-loop iterates over all shapes in the target layer, with the iterator being *target shape*, or simply *target*. In summary, the first and second nested loops allow every shape in the layout to be the aggressor and add its neighbors to the pattern library.

Algorithm 1: Our 2.5D Pattern Extraction Algorithm. fax and lax stand for forward axis and lateral axis, respectively. bbox is bounding box.

```
   Input  : cell_layout, window_size
   Output: pattern_lib
1  pattern_lib ← {};
2  foreach target_layer ∈ all_layers do
3      foreach target ∈ target_layer do
4          foreach (fax, lax) ∈ {(x, y), (y, x)} do
5              window ← extend(target, lax, window_size);
6              neighbors ← all shapes within the window;
7              cutlines ← {};
8              foreach n ∈ neighbors ∪ {target} do
9                  cutlines.add(n.bbox[fax].lower);
10                 cutlines.add(n.bbox[fax].upper);
11             foreach c ∈ cutlines do
12                 if c ∉ n.bbox[fax] then continue;
13                 pattern ← {target};
14                 foreach n ∈ neighbors do
15                     if n overlaps (c, next(c)) then
16                         pattern.add(n);
17                 pattern.sort();
18                 pattern.filter(max_lateral = 2);
19                 pattern.index_assignment();
20                 pattern_lib.add(pattern);
21 return pattern_lib
```

The third loop is the scanning direction. Each shape is scanned along both the x and y axes. The target shape is first sliced into non-uniform segments (lines 7-10), where the cut lines are determined by the boundary of its neighbors in the search window. Then we traverse every segment (lines 11-16) to collect the neighbors overlapping the segment to create the cross-section pattern. We set the $window\_size$ to one CPP. In fact, with our preceding filtering in place, the window size has little impact on the accuracy as long as it is large enough.

In the example shown in Fig. 6b, the target shape (red-edged rectangle in the middle) is sliced into six segments in a horizontal scan by five cut lines, creating four unique cross-section patterns ① - ④ in Fig. 6c, while the leftmost segment has no neighbors and is omitted. Additionally, the vertical scan (Fig. 6d) creates the fifth pattern in Fig. 6e. No horizontal cut lines are drawn because there

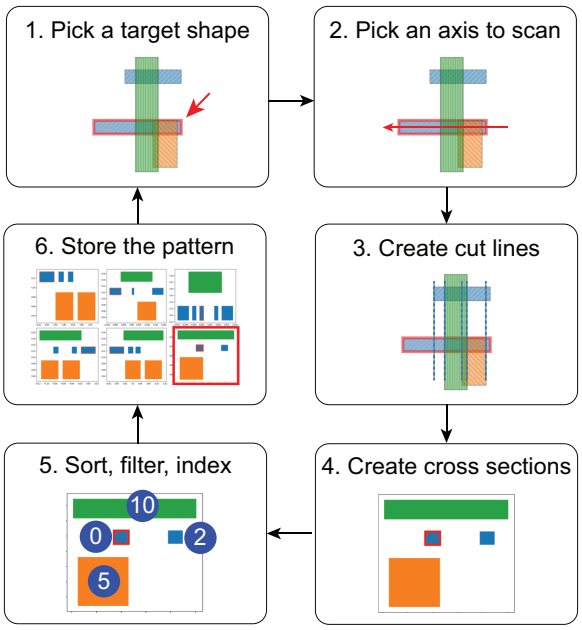

Fig. 5: Overview of our pattern extraction algorithm. Our tool first picks a target shape and scanning direction (1, 2), and draws cut lines based on its neighbors (3). In (4), another loop traverses every cut line and creates the cross sections. After some processing (5), the pattern is stored in the library (6), and the tool enters another loop.

are no neighbors to cut the target when scanning vertically. When storing the coordinates of the shapes, the target is centered on the origin, and the others are offset accordingly. The patterns are stored in a C++ `std::set` to prevent duplication.

Lastly, there is yet another outer loop to traverse all cells in the library. To make the description of scan direction independent of the shapes' routing direction, we define *long axis* and *short axis*. For a horizontal shape like the red-edged one in Fig. 6, the *long axis* is defined as the x-axis and *short axis* as the y-axis for this shape. Vice versa for a vertical shape like the green one in Fig. 6.

In addition to adding all surrounding shapes of the target into the pattern, we enforce sorting to ensure the victims are organized in a way that the ML model can easily capture hidden information and filtering to prevent extremely small values from disrupting the model's learning.

B. Shape Sorting in a Pattern

There are two goals for sorting the shapes in a pattern: a) to facilitate the translation from a graphical pattern to a feature matrix (Fig. 9), which we will discuss in Section V; and b) as a preliminary step for filtering.

First, we create $2L$ lists, where L is the number of unique layers in the pattern. L lists for shapes whose horizontal coordinates of the center are $\leq$ that of the target shape, i.e., on the left. The other L lists for otherwise, i.e., on the right. After every shape is placed in its corresponding list, the lists are sorted by the center-to-center distance of each shape to the target shape.

C. Shape Filtering Policy

In our work, we filter out some shapes in patterns. Such filtering is enforced for the following reasons: a) There is a hard upper bound for the number of shapes that our ML model can process. A high number of shapes sacrifices not only the efficiency but also the accuracy. b) Not all shapes are equally crucial to the aggressor. According to [9], two shapes for each direction suffice. Adding the third one has less than 1% effect on the aggressor. c) According to our experiments, adding extremely small values to the training data distracts the model from learning critical trends.

We first implement layer filtering, alternatively called vertical filtering. Suppose there are L_a layers above the target layer, L_l layers at the same level as the target layer (lateral layers), and L_b layers below the target layer. All the L_l layers are kept, and only the nearest layer from L_a and L_b layers, respectively, are kept. Note that the remaining above/below layer of target layer l is not necessarily $l \pm 1$. If a M0 shape has a segment not overlapping with M1 but with M2, a cross-section involving M0 and M2 is created.

Next, we filter the shapes within a layer, alternatively called horizontal filtering. Horizontal filtering is performed differently depending on the scan direction. For a long-axis scan as depicted in Fig. 7a, the target shape is thin, and the chance that the shape overlaps more than two shapes is slim. Thus, we filter the shapes according to their distance from the center. Based on Abouelyazid et al. [9] and our experiments, we only keep at most two shapes on the left and right, respectively. The third one has less than 1% capacitance contribution to the aggressor.

For a short-axis scan, the target shape can be long and overlap multiple shapes above or below, as shown in Fig. 7b. However, the capacitance of those overlapped shapes has been computed in the other scan and need not be extracted. We only need to take care of lateral and fringe capacitance in this scan. In Fig. 7b, despite crossing many shapes below, the gray shapes above and under the target shapes have been extracted during the other scan, and those outer gray shapes have negligible capacitance to the target. This scan aims to model the lateral (blue ones) and fringe (orange and green ones) capacitance. According to our experiment, removing gray shapes in Fig. 7b only causes less than 4% of increases in the lateral capacitance.

D. Shape Indexing in a Pattern

The last step is to assign an index for each shape in a given pattern. 0 is for the aggressor, $2i-1$ for shapes on the left, and $2i$ for those on the right. 1 to 4 are for lateral shapes, 5 to 8 for below shapes, and 9 to 12 for above shapes. These indices correspond to their assigned slots as we will show in Fig. 9. If fewer than three layers are involved, the maximum index, or number of inputs for the ML model, is decreased accordingly.

E. Categories of Capacitance

Based on empirical observations, we categorize the impact of layer thickness variation on capacitance into three types, as shown in Fig. 8. We define *sensitivity* as a numerical metric to explain the idea:

$$\text{Sensitivity} = \frac{\text{capacitance change (\%)}}{\text{thickness change (\%)}} \tag{1}$$

For example, doubling the layer thickness (+100%) yields a corresponding percentage change in capacitance, indicating its sensitivity.

Our results show that lateral capacitance (Fig. 8a) is highly sensitive, often exceeding 50% and occasionally 100%. Stretched capacitance (Fig. 8b) shows moderate sensitivity (-10% to -40%), while others, such as vertical capacitance (Fig. 8c), remain mostly within $\pm 5\%$.

This variability in sensitivity directly affects ML model performance: high-sensitivity regions create training samples with similar inputs but significantly different outputs, increasing modeling complexity. We select our ML method based on this observation.

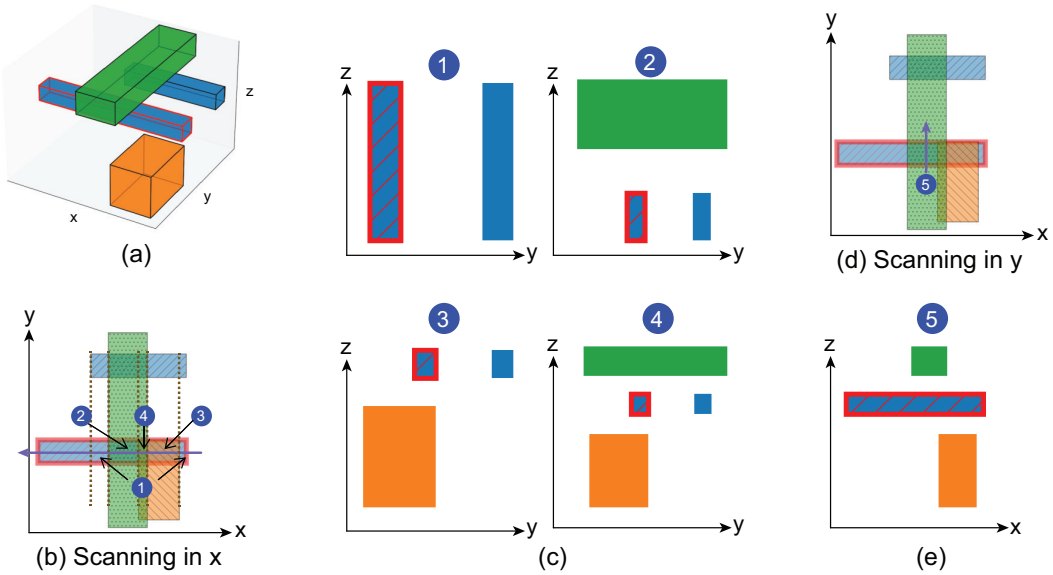

Fig. 6: A layout is scanned in both x and y directions, creating (x, z) (pattern #5) or (y, z) (patterns #1 to #4) cross-section patterns with a centered target shape and several surrounding shapes. (a) 3D view, (b) 2D top-down view, scanned horizontally, (c) cross-section patterns collected during horizontal scan, (d) 2D top-down view, scanned vertically, (e) cross-section pattern collected during vertical scan.

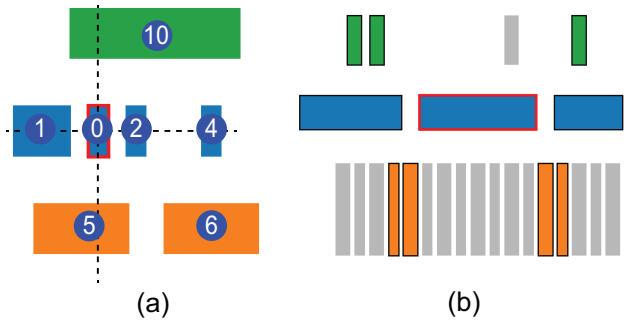

Fig. 7: Shape indices after sorting and filtering. These indices indicate their assigned slots in the feature vector. (a) Long-axis scan patterns. Two shapes on both sides suffice, (b) Short-axis scan patterns. Gray shapes can be removed with $< 4\%$ impact on the lateral capacitance.

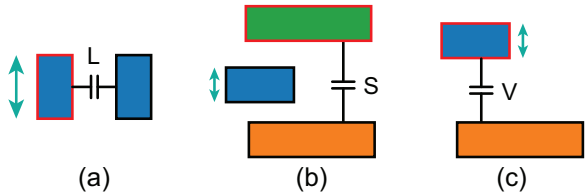

Fig. 8: Categories of cross-section coupling capacitance based on their sensitivity to a metal thickness change (indicated with arrows). Red-edged shapes are aggressors. (a) Lateral capacitance (L), (b) Stretched capacitance (S), (c) Vertical capacitance (V).

V. Capacitance Prediction with Machine Learning

This section describes our ML approach, including how we prepare our training data, translate them into numeric values for ML models, preprocess the data for better outcomes, the reason why we used multiple ML models, and finally, our training strategy.

A. Training Data Preparation

After we extracted patterns from the standard cell library, we fed each pattern to a 2-D field solver to solve the capacitance. As Fig. 2 shows, for a pattern with N shapes, the field solver will output an $N \times N$ symmetric matrix. C_{ii} is the total capacitance of conductor i, the sum of all its coupling capacitance to the others, while $C_{ij} (i \neq j)$ is the coupling capacitance between conductor i and j. On the other hand, our goal is to extract the capacitance between the aggressor and each of its victims, i.e., $N-1$ coupling capacitance values. Therefore, we formed $N-1$ feature vectors, $\vec{v}_{01}$ and $\vec{v}_{02}$ in Fig. 2's example. We will explain our feature vector in Section V-B. Then, C_{01} and C_{02}, which are from the field solver, are used as the training labels. The prediction is denoted as $\tilde{C}_{01}$ and $\tilde{C}_{02}$

B. Feature Vector Representation

Fig. 9 depicts how we organize a pattern into numeric values for the ML model. We apply the vertex-based representation from [9] for two reasons: a) This representation inherently includes layer thickness, which is one of the main targets in our flow. As a comparison, density-based representation—another representation adopted by [8], [10], [11]—focuses on a top-down view of shape geometries and cannot encode thickness into the feature vector. b) The large number of zeros in the feature vector acts as an efficient way to turn off irrelevant neurons.

In Fig. 9, we refer a *pattern library* to a collection of *patterns*. As Fig. 2 shows, a pattern with N shapes (1 aggressor, $N-1$ victims) includes $N-1$ capacitance values of interest, requiring $N-1$ *feature vectors*. Each feature vector consists of three equal-sized parts: *all shapes*, *aggressor*, and *victim*. The aggressor part is a filtered copy of the all-shapes part, leaving only the aggressor geometry description non-zero. Similarly, the victim part is also a copy of the all-shapes part, leaving only the victim geometry description non-zero. The *all-shapes vector* can be viewed as the concatenation of L *layer vectors*, where L stands for the number of metal layers involved in this pattern. In our later experiments, L ranges from 2 to 4. The first layer is always the target layer, and the first shape slot is always the aggressor, which is centered at the origin. A *layer vector* is the concatenation of 4 (secondary layers) or 5 (target layers) shapes. The shapes are sorted according to their relative position to the target aggressor or the origin, as we introduced in Section IV-B. Suppose, for example,

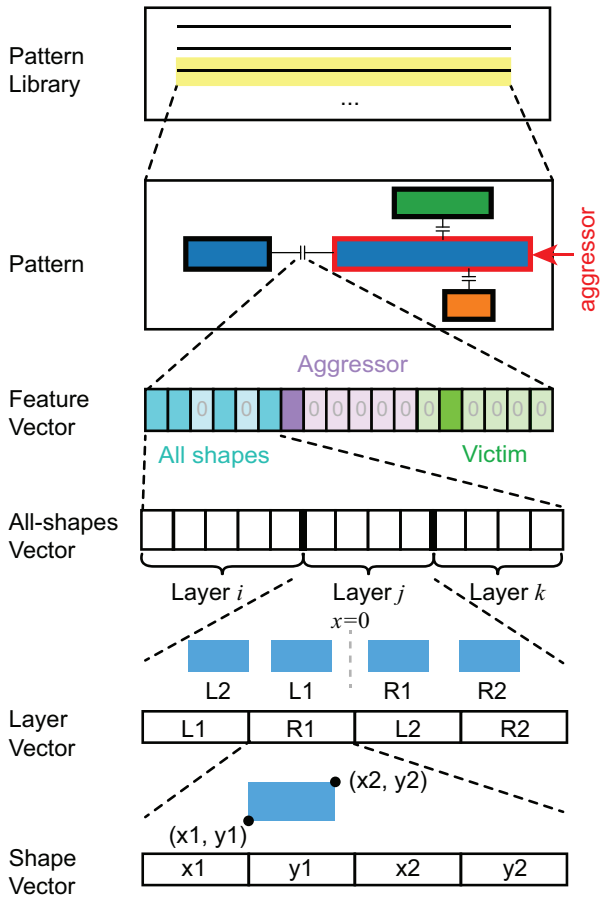

Fig. 9: Hierarchical breakdown of our feature vector representation. A pattern with N shapes yields $N-1$ feature vectors, each split into three parts: all shapes, aggressor, and victim. Feature vectors are built from layer vectors, which consist of 4–5 shapes per layer, with each shape represented by its (x, y) coordinates.

there is only one shape on the right; the R2 slot will be empty and filled with zeros to deactivate the corresponding neurons. Finally, each shape is represented by four numbers: its lower-left (x, y) and its upper-right (x, y).

Using the pattern in Fig. 2 as an example to summarize, which involves three layers and two victims. A feature row vector involving three metal layers can represent at most $5+4+4=13$ shapes. We enforce filtering during pattern extraction as described in Section IV-C to ensure each pattern has no more than 13 shapes. These shapes constitute $13 \times 4 = 52$-d all-shapes vectors. The dimension for a feature vector is $52 \times 3 = 156$. Thus, Fig. 2 will be translated into a matrix of size $(152, 2)$

C. Data Pre-processing

Our data preprocessing includes three parts: cleaning, scaling, and data augmentation.

1) Data Cleaning: Taking Fig. 7a as an example. The shape ④ is shielded by ② and has orders of magnitude less capacitance than ② or ①, which implies the need to model the coupling capacitance between ⓪ and ④, C_{04} is minimum. However, as [9]'s study, ④ still has a considerable electric field effect on ② and is essential to be included in the feature vector. To sum up, though ④ still appears in the feature vectors for predicting other capacitances, C_{04} will not be added to the training data since its value is too small, and even worse, this sort of shielded capacitance makes up a considerable portion of the training data if unfiltered. Both reasons distract the model's learning. To further generalize the idea, all the feature vectors describing the extraction of L2 and R2 shapes are removed from the training data.

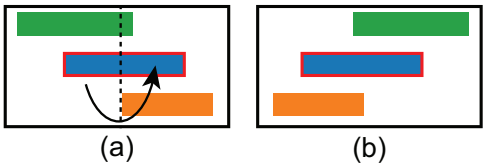

Fig. 10: Data augmentation. (a) Original pattern, (b) Flipped version for data augmentation.

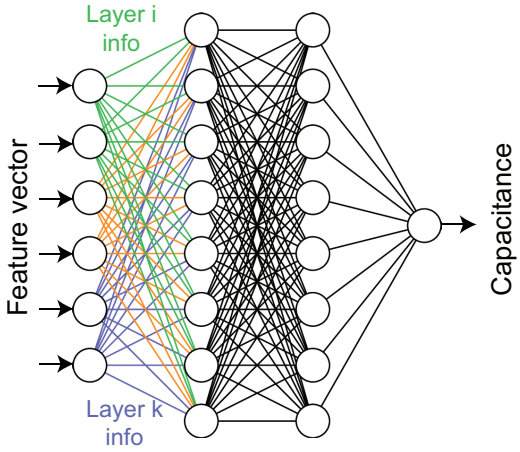

Fig. 11: A schematic of how layer information is stored in an ML model. The info is stored implicitly in the model weights.

2) Data Scaling: The input vectors are in the unit of µm. According to our experiments, there is no need for scaling for the input vectors. However, the capacitance values range from 3×10^{-16} to 1×10^{-18} F/µm. Without proper scaling, the model tends to predict all values to be zero. Thus, we normalized the values to the maximum capacitance in the dataset so that all values are between 0 and 1.

3) Data Augmentation: As shown in Fig. 10, we horizontally flip patterns to create new data to increase pattern coverage and improve the underrepresented patterns. After adding the flipped data to the dataset, we remove duplicates to prevent data contamination in the training and validation dataset. Specifically, all the data in the validation dataset differ from the training set and are not used for training.

D. Model Separation

As we introduced our input vector in Section V-B, only shape coordinates are provided to the model, not the dielectric or conductor information. Hence, as depicted in Fig. 11, we believe that the layer information is implicitly stored in the weights of the ML model. As a result, instead of training an almighty model for all layer combinations, we train one model for each. As will be shown in TABLE IV, there are ten ML models in our test case spanning from MD/Gate up to M2. Fig. 11 also explains why sorting and indexing in Section IV-B and Section IV-D are essential to the model.

E. Our Training Strategy

We tried two ML methods, deep neural network (DNN) and LightGBM (LGB) [13], a boosting tree algorithm. We concluded that DNN is more powerful, while LGB is much faster. Thus, our strategy

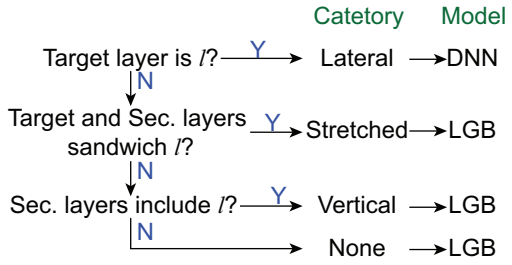

Fig. 12: The flow chart to determine the ML model we select by examining target and secondary layers, assuming we change the thickness of l

TABLE II: DNN hyperparameter search space.

Hyperparameter	Search range
# of layers	1, 2, 3, 4
hidden dim	[128, 512], step=32
learning rate	[2e-4, 1e-2], log-scale
batch size	256, 512, 1024, 2048, 4096
optimizer	Adam, SGD
activation function	ReLU, Tanh

TABLE III: LightGBM hyperparameter search space.

Hyperparameter	Search range	Hyperparameter	Search range
max_depth	[4, 12]	learning_rate	[1e-4, 0.3]†
subsample	[0.6, 1.0]	colsample_bytree	0.3, 1.0
min_child_weight	[1, 10]	lambda_l1	[1e-7, 1]†
lambda_l2	[1e-4, 10]†	boosting	gbdt, dart, rf
num_leaves	[64, 128]	path_smooth	[1e-3, 10]†
min_data_in_leaf	[5, 100]	bagging_fraction	[0.9, 1.0]
bagging_freq	[200, 1000]		

† Log-scale

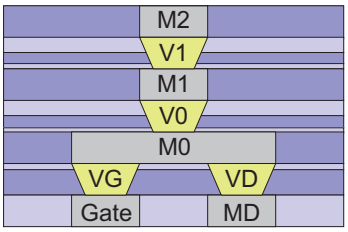

Fig. 13: Cross-sectional layer stack and multi-dielectric environment of the imec PDK [15] we use in this paper.

is to use LGB for simple models and DNN for complex ones. We will explain the determination of "simple" and "complex" in the following paragraphs.

Building upon the trichotomy introduced in Section IV-E, we extend it to classify each model's complexity or training difficulty. Fig. 12 shows the flow of how we apply the observation from Section IV-E to determine the ML method we choose. An enumeration of all the models we need will be presented in TABLE IV. Suppose the dataset includes multiple thicknesses of layer l. If the target layer is also l, the dataset of this layer combination includes a lot of data points whose input vectors are close but with very different capacitance. As we will later show in our experiments, we observe that such a situation is beyond the capability of LightGBM. Nevertheless, LGB suffices to handle the other cases whose target layer is not l.

We used Optuna [14] to tune the hyperparameters for both methods automatically. We first specify an aggressive range of hyperparameters for Optuna to search, for two purposes: a) to figure out influential hyperparameters b) To observe the baseline loss. Then, we narrow the search range and expect the mean squared error to be around 10^{-6} or less. Our hyperparameter search ranges for both models are reported in TABLE II and TABLE III, respectively. Some hyperparameters are used to adjust model complexity, while others are used to prevent overfitting. Finally, DNNs are trained for 2000 epochs with batch normalization, and LGBs are trained for 2000 iterations. Optuna runs 100 iterations for LGB and 30 iterations for DNN. Expressed another way, the training time we will report in TABLE IV is equivalent to training 100 LGB models or 30 DNN models for each row.

VI. Experimental Results

A. Experimental Setting

We use the imec N2 predictive nanosheet PDK [15] and its standard cell library as input, comprising 94 cells excluding filler cells. The largest cell spans 24 contacted poly pitch (CPP) across two rows and utilizes up to the M2 metal layer. The cross-section of the PDK is shown in Fig. 13. Since the standard cells use metals only up to M2, layers above M3 were omitted. On average, 682 cross-sectional patterns are extracted per cell. After deduplication, 17,178 unique patterns were identified across all metal stack combinations, resulting in 3k–27k data points per DNN model per ITF. Recall that a pattern with N shapes contributes $N-1$ data points.

We varied the thickness of the M0 layer to generate cross-section patterns. M0 was chosen due to: (a) its dense usage in standard cells, providing a rich dataset, and (b) the fact that adjusting gate or MD layer thickness would impact device performance beyond parasitics.

Two datasets were prepared. The first (training/validation) set spans M0 thicknesses from 22 nm to 58 nm in 6 nm increments (7 configurations), split into 80% training and 20% validation. The second (testing) set ranges from 18 nm to 62 nm in 2 nm steps. During hyperparameter tuning, we used the 38 nm thickness (near the center of the range) as the loss function. Training was conducted on a workstation equipped with dual Intel 6454S CPUs (64 cores, 128 threads total), utilizing 70% of available threads (89 threads). Additional ML-related configurations are detailed in Section V-E. Training runtimes are reported in TABLE IV. At inference, it took 0.082 seconds to extract patterns and 0.803 seconds for ML inference on average per cell.

In the remainder of this section, we first compare the prediction accuracy of DNN and LGB models in Section VI-B to support our ML model choice. We then analyze error trends across varying M0 thicknesses in Section VI-C, and conclude with a detailed error histogram at an unseen thickness in Section VI-D, a common visualization used by [9], [11]

B. Comparison of DNN and LightGBM

Our strategy is to use LightGBM whenever possible, since its training speed is more than 10x faster than that of DNN. However, Fig. 14 shows one of the two complex models in TABLE IV for which LGB is not good enough, on which we decide to use DNN despite the much longer training time. In Fig. 14, we show the scatter plots with M0 thickness 38 nm to compare the prediction accuracy of the two models. The x-axis marks the values the field solver calculates, and the y-axis is the ML model's prediction. All values are normalized by the maximum value in the dataset. All datapoints are expected to fall on the gray dotted line under an ideal situation. We separate the datapoints into two groups: lateral capacitance and all the others. As introduced in Section IV-E, since the sensitivity of lateral capacitance to changes in layer thickness is high, which is a vastly different property from other capacitances, LGB failed to capture the trend in

TABLE IV: ML models, dataset size, database build time (field solver runtime), and training time. We use DNN for models whose target layer is M0, and LightGBM for the rest.

Target layer	Secondary layer(s)	# of data points	Model	DB build time	Training time
Gate	MD, M0	79,870	LGB	12.82 mins	19.49 mins
Gate	MD, M1	27,650	LGB	4.17 mins	11.87 mins
MD	Gate, M0	91,980	LGB	15.46 mins	23.66 mins
MD	Gate, M1	23,464	LGB	3.91 mins	12.76 mins
M0	MD, Gate, M1	156,870	DNN	20.78 mins	1 day
M0	MD, Gate, M2	189,472	DNN	23.42 mins	1 day
M1	M0, M2	115,262	LGB	15.52 mins	27.30 mins
M1	MD, Gate, M2	82,586	LGB	10.09 mins	26.80 mins
M2	M1	2,968†	LGB	0.23 mins	4.42 mins
M2	M0	19,502	LGB	2.85 mins	14.62 mins
Total		789,614		1.8 hrs	2 days+2.3 hrs

† Less data since M0 change doesn't affect this layer combination.

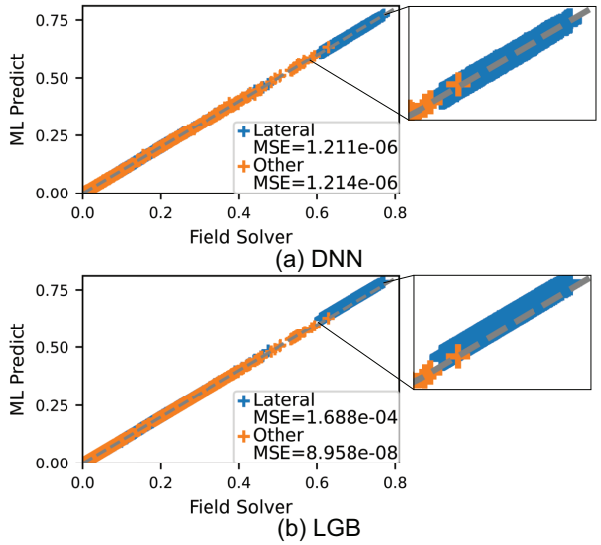

Fig. 14: Comparison of (a) DNN and (b) LightGBM for target layer = M0, secondary layers = {MD, Gate, M2}. We zoom in on large values and show them on the right. LGB cannot perform well in predicting lateral capacitances and misses the trend (gray dotted line).

this category accurately. Specifically, the upper right points in Fig. 14b significantly deviate from the $y = x$ line and introduce more error than the other group. On the other hand, DNN is more keen on the trend of all kinds. In conclusion, Fig. 14 helps to explain why we use DNN in some instances despite its much longer training time.

C. Prediction Accuracy Across Different Thicknesses

We report the errors for estimating unseen M0 thickness in Fig. 15. The labels on the x-axis are the thicknesses included in the training dataset. We extracted patterns from all 94 cells in the library without removing duplicated patterns for testing, so that the reported errors are weighted by the number of occurrences in the library. These patterns were translated into corresponding feature vectors as shown in Fig. 9 and fed to the corresponding ML model.

From Fig. 15 we can observe that: 1) Errors start to rise at margin points, even for interpolated points. That implies that, supposedly, we aim to explore thickness from 22nm to 58nm; we need to extend the training data coverage beyond this range to ensure accuracy. 2) The (b) figure proves that our strategy to use LGB on simple tasks is sufficient to achieve high accuracy. More than 85% of predictions are within $\pm 1\%$ errors except those around the margin.

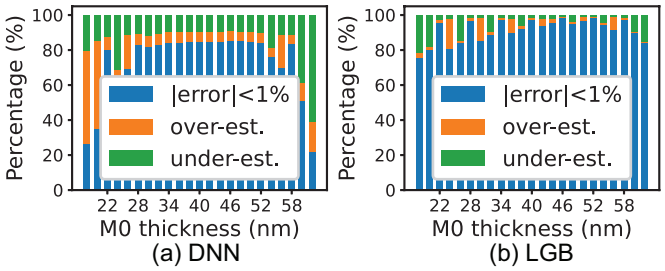

Fig. 15: Relative error across M0 layer thicknesses. Bars show error distribution for (a) DNN and (b) LightGBM models. Over 80% of predictions across most thicknesses fall within $\pm 1\%$ error. DNN is used for complex cases, while LGB performs well for simpler models.

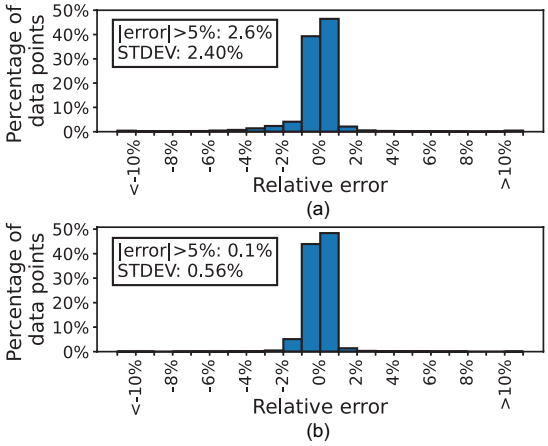

Fig. 16: Error distribution using M0=38n. (a) Aggregated result of DNN for the two complex models, (b) Aggregate result of LGB for the other eight models

D. Case Study: Impact of M0 Thickness Change

In Fig. 16, we pick M0 thickness 38nm, which is a point around the center in Fig. 15, to unfold the error distribution for an unseen thickness. For the two complex models using DNN, 85.75% of the values are within $\pm 1\%$ error, 1.75% have more than 5% of underestimate, and 0.82% have more than 5% of overestimate. For the other eight models using LGB, 92.34% of the values are within $\pm 1\%$ error, 0.037% have more than 5% of underestimate, and 0.079% have more than 5% of overestimate. All the $> 5\%$ outliers are with value less than 3×10^{-18} F/µm.

VII. Conclusion

We introduced our pattern extraction algorithm to extract cross-section patterns to prepare training data for our machine learning model, and we proposed our data preprocessing steps, including filtering and data augmentation. We assumed layer thickness as the main process parameter to explore, implemented a flow to extract capacitance from ITF and GDS files, and measured the runtime. Our experiment results demonstrated that the method has the potential to speed up the parasitic extraction flow by multiple orders of magnitude despite the longer training and data preparation time. We also demonstrated that LightGBM is an appealing ML method for its much faster speed, but DNN is more powerful for handling complicated tasks.

VIII. Acknowledgment

This research is funded by Samsung Electronics under the AI for Semiconductors Program.

REFERENCES

[1] J. H.-C. Chen, T. E. Standaert, E. Alptekin, T. A. Spooner, and V. Paruchuri, "Interconnect performance and scaling strategy at 7 nm node," in *IEEE International interconnect technology conference*. IEEE, 2014, pp. 93–96.

[2] Synopsys, Inc., "StarRC Parasitic Extraction," https://www.synopsys.com/implementation-and-signoff/signoff/starrc.html, 2025, accessed: 2025-08-07.

[3] Cadence Design Systems, Inc., "Quantus QRC Extraction Solution," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/silicon-signoff/quantus-extraction-solution.html, 2025, accessed: 2025-08-07.

[4] Siemens EDA, "xACT Parasitic Extraction," https://eda.sw.siemens.com/en-US/ic/calibre-design/circuit-verification/xact/, 2025, accessed: 2025-08-07.

[5] Z. Li and W. Shi, "Layout capacitance extraction using automatic pre-characterization and machine learning," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2020, pp. 457–464.

[6] J. Cong, L. He, A. B. Kahng, D. Noice, N. Shirali, and S. H.-C. Yen, "Analysis and justification of a simple, practical 2 1/2-d capacitance extraction methodology," in *Proceedings of the 34th annual Design Automation Conference*, 1997, pp. 627–632.

[7] R. Kasai, T. Kanamoto, M. Imai, A. Kurokawa, and K. Hachiya, "Neural network-based 3d ic interconnect capacitance extraction," in *2019 2nd International Conference on Communication Engineering and Technology (ICCET)*. IEEE, 2019, pp. 168–172.

[8] D. Yang, W. Yu, Y. Guo, and W. Liang, "Cnn-cap: Effective convolutional neural network based capacitance models for full-chip parasitic extraction," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[9] M. S. Abouelyazid, S. Hammouda, and Y. Ismail, "Fast and accurate machine learning compact models for interconnect parasitic capacitances considering systematic process variations," *IEEE Access*, vol. 10, pp. 7533–7553, 2022.

[10] M. S. Abouelyazid, S. Hammouda, and Y. Ismail, "Accuracy-based hybrid parasitic capacitance extraction using rule-based, neural-networks, and field-solver methods," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5681–5694, 2022.

[11] J.-C. Tsai, H.-M. Huang, W.-M. Hsu, P.-T. Lee, J.-H. Yang, H.-L. Huang, Y.-J. Su, and C. H.-P. Wen, "Rescap: Fast-yet-accurate capacitance extraction for standard cell design by physics-guided machine learning," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 1243–1250.

[12] Synopsys, Inc., "Raphael FX: 2D and 3D Resistance, Capacitance, and Inductance Extraction Tool," https://www.synopsys.com/manufacturing/tcad/interconnect-simulation/raphael.html, 2025, accessed: 2025-08-07.

[13] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[14] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2623–2631.

[15] A. Farokhnejad, S. S. Sahoo, D. Abdi, J. Cousins, A. Dutta, G. Mirabelli, V. Sankatali, L. Verschueren, S. Yang, O. Zografos *et al.*, "N2 nanosheet pathfinding-pdk (p-pdk tm) including back-side pdn," in *2024 IEEE European Solid-State Electronics Research Conference (ESSERC)*. IEEE, 2024, pp. 17–20.

[16] Y. Peng, D. Petranovic, and S. K. Lim, "Multi-tsv and e-field sharing aware full-chip extraction and mitigation of tsv-to-wire coupling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1964–1976, 2015.

[17] Y. Peng, T. Song, D. Petranovic, and S. K. Lim, "Full-chip inter-die parasitic extraction in face-to-face-bonded 3d ics," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 649–655.

[18] M. Basel, "Accurate and efficient extraction of interconnect circuits for full-chip timing analysis," in *Proceedings of WESCON'95*. IEEE, 1995, p. 118.

[19] J.-H. Chern, J. Huang, L. Arledge, P.-C. Li, and P. Yang, "Multilevel metal capacitance models for cad design synthesis systems," *IEEE Electron Device Letters*, vol. 13, no. 1, pp. 32–34, 1992.

[20] N. D. Arora, K. V. Raol, R. Schumann, and L. M. Richardson, "Modeling and extraction of interconnect capacitances for multilayer vlsi circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 1, pp. 58–67, 1996.