Developing a Robust Timing Verification and Signoff Framework Using Machine Learning

Report on the Framework for Integrating ML-Based Timing Models for More Accurate Static Timing Analysis (STA)

Submitted by:

Pooja Beniwal, Vaibhav Singh, Tarush Garg, Jatin Kumar

March, 2025

Supervised by:

Dr. Sneh Saurabh

Contents

1 Introduction

In traditional Static Timing Analysis (STA), technology libraries are essential for accurately representing standard cells and their attributes, specifically focusing on timing, which is crucial for circuit functionality and performance. One commonly used timing model is the Non-Linear Delay Model (NLDM), where a delay and transition time lookup tables are created during cell characterization to model cell delay and output transition time. These tables specify timing values for input waveform transition times (t_t) and output capacitance (C_L) [1]. However, these tables have several limitations. Interpolation and extrapolation techniques are required to extract timing values for input transition, and output capacitance values are absent in the tables. Foundries release libraries for multiple process, voltage, and temperature (PVT) corners to accommodate process and environmental variations. However, these corners may not cover all scenarios, such as operating at voltages beyond the provided range, leading to only estimated circuit performance. Moreover, the actual environmental conditions may deviate from foundry assumptions, hindering the designer's ability to fully realize the circuit's potential or potentially causing problems due to incorrect assumptions [2]. Multiple-input logic gates are typically characterized using single-input switching (SIS), assuming constant side inputs. However, in real designs, transitions at different input pins of the same multiple-input gate occurring simultaneously can result in overestimation or underestimation of delays, known as Multiple Input Switching (MIS) [3].

Recently, machine learning (ML)-based methods have been emerging for the application of Electronic Design Automation (EDA), especially in cell library characterization. In [4], a ML-based approach for modeling MIS in timing analysis is introduced. This work emphasizes the complexity of accurately predicting the timing behavior when multiple inputs of a logic cell switch simultaneously. By employing ML techniques, models were developed that significantly improve the prediction accuracy over traditional methods. In [5], deep-learning techniques for cell-delay modeling are explored. This research presents a deep-learning model that can predict cell delays with high accuracy, outperforming conventional analytical models. This method not only accelerates the STA process but also adapts better to the variability in modern semiconductor processes. The intersection of design technology co-optimization (DTCO) and ML is investigated in [6]. ML models for cell library characterization that facilitate DTCO by optimizing both design and technology parameters simultaneously are developed, allowing for more efficient exploration of the design space and leading to better performance and reduced power consumption in integrated circuits. In [7], a ML-based aging-aware cell library delay model, ADAM, is introduced. This feed-forward neural network model accounts for aging effects in semiconductor devices. A learning-based methodology to accelerate the generation of cell models is presented in [8]. This approach uses ML to significantly reduce the time required to generate an accurate cell model. The use of deep learning for generating cell libraries that

account for On-Chip Variation (OCV) modeling is demonstrated in [9]. This deep learning framework provides fast and accurate OCV models, addressing one of the key challenges in modern VLSI design, where process variations can significantly impact circuit performance.

In summary, the integration of ML in cell library characterization has introduced significant improvements in the accuracy, efficiency, and adaptability of the characterization process. These advancements are crucial for addressing the challenges posed by modern VLSI design, such as delay variations due to MIS, process variations, aging effects, and the need for fast design cycles. As the field continues to evolve, further innovations in ML-based methods are expected to drive even more profound enhancements in EDA. With all these advancements, an automatic ML library characterization tool is required to fasten the ML-based characterization process. Hence, this report addresses these issues and makes the following key contributions:

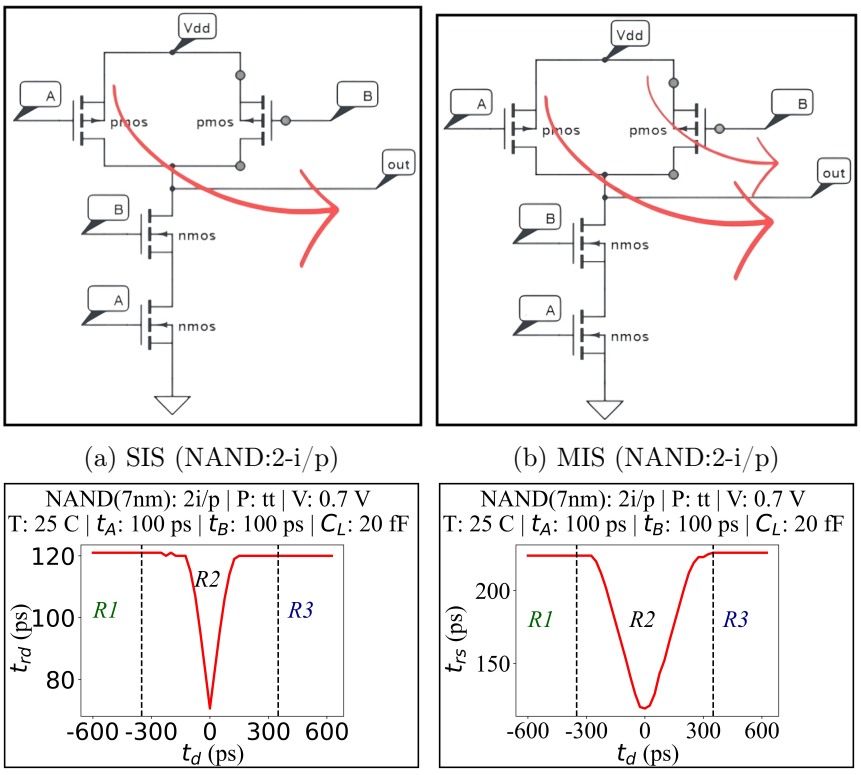
- **Enhanced Timing Models**: Developed an ML model to improve traditional timing attributes transition time (t_t) and load Capacitance (C_L) by incorporating temporal distance (t_d) to tackle delay variation due to MIS. A
- **Efficient Characterization**: Reduced characterization effort (data collection) using the proposed dataset optimization method.
- **LiMo Framework**: Introduced Intelligent Library Model (LiMo), a user-friendly framework that automates dataset generation, optimization, visualization, multiprocessing, model training, and evaluation, boosting timing model development efficiency.
- **Benchmark Validation**: Validated the ANN model's effectiveness with the ISCAS'85 C17 benchmark circuit, showing consistency with SPICE results.
- **OpenSTA Integration with the developed ML Delay Model**: Modified the OpenSTA source code to bypass traditional lookup tables and instead utilize a trained ML model for delay calculation. This integration enables direct inference of delays under MIS, allowing dynamic and more accurate timing analysis within the STA flow

2 Methodologies

The methodologies for developing ML-based timing models encompass dataset generation, preprocessing, training, testing, and model generation. Generating comprehensive datasets through SPICE simulations, which capture critical features. These datasets undergo preprocessing to clean and normalize the data. ML models are then trained using this preprocessed data, with key steps involving model selection, training, and hyperparameter tuning. Post-training, the models are evaluated and validated using metrics.

2.1 Features: Key Elements Driving Model Performance

For a 2-input NAND gate, the features used in developing ML model are explained in this section.



(a) SIS (NAND:2-i/p)

(b) MIS (NAND:2-i/p)

(c) Output Rise Delay

(d) Output Rise Slew

Fig. 1: Impact of t_d

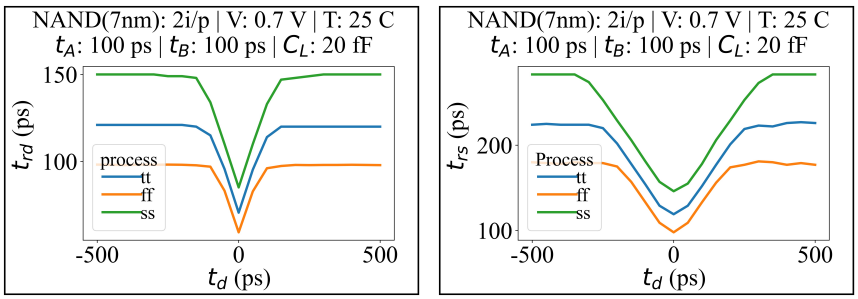
2.1.1 Temporal Distance

With the increase in temporal distance (t_d) between two input signals, delay increases or decreases based on the topology of transistors. One such example is shown in Fig. 1a when SIS

is occurring rise delay (t_{rd}) path is from a single PMOS. Fig. 1b shows the case of MIS t_{rd} path from parallel PMOS. As illustrated in Fig. 1c and Fig. 1d, when the t_d between two input signals is significant (SIS), causing them to be distant, both t_{rd} and t_{rs} remain constant. Conversely, in the case of MIS, where the t_d between two input signals is minimal, resulting in their proximity, t_{rd} and t_{rs} exhibit a decrease.

2.1.2 Process (P)

From Fig. 2a and Fig. 2b, we observe that t_{rd} and t_{rs} are typically more at the slow-slow (ss) corner and shorter at the fast-fast (ff) corner. This is because the ss corner conditions are pessimistic, representing the worst-case scenario for circuit operation, while the ff corner conditions are optimistic, representing the best-case scenario. Additionally, the width of the V-shaped dip due to MIS is greater at the ss corner and lowest at the ff corner. This indicates that the impact of MIS on t_{rd} and t_{rs} is more pronounced in the SS corner.



(a) (b)

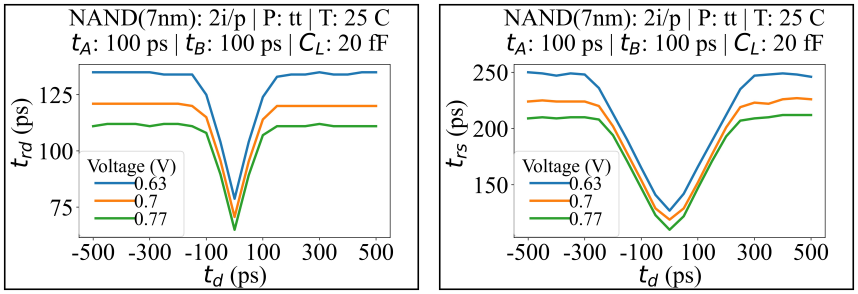
Fig. 2: Correlations between t_{rd}, t_{rs} and P.

2.1.3 Voltage (V)

From Fig. 3a and Fig. 3b, we observe that t_{rd} and t_{rs} are more at lower voltages than at higher voltages. A larger gap between the gate and source voltages in a MOSFET allows more current to flow. When VDD is reduced, gate drive voltages decrease, reducing the current flow. Since CMOS loads are largely capacitive, lower current results in slower capacitor charging or discharging, thus increasing delay. The greater width of the V-shaped dip at lower voltages signifies the heightened sensitivity of the circuit to MIS.

2.1.4 Temperature (T)

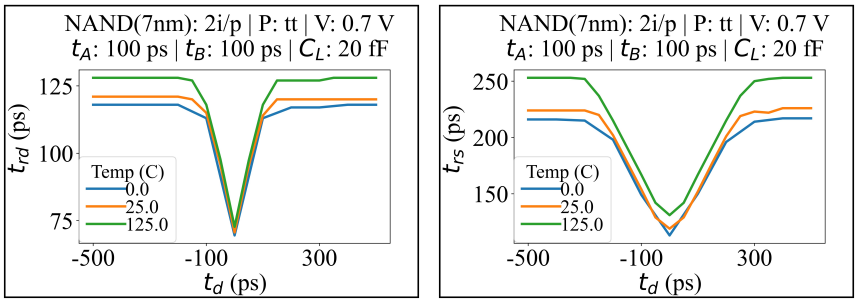
From Fig. 4a and Fig. 4b, we observe that t_{rd} and t_{rs} are more at higher temperatures than at lower temperatures. As temperature rises, carrier mobility decreases, reducing current flow



(a) (b)

Fig. 3: Correlations between t_{rd}, t_{rs} and V.

through the MOSFET. Consequently, the transistors switch more slowly, leading to higher t_{rd} and t_{rs}. The greater width of the V-shaped dip due to MIS indicates at higher temperatures, the circuit experiences a more pronounced impact of MIS.



(a) (b)

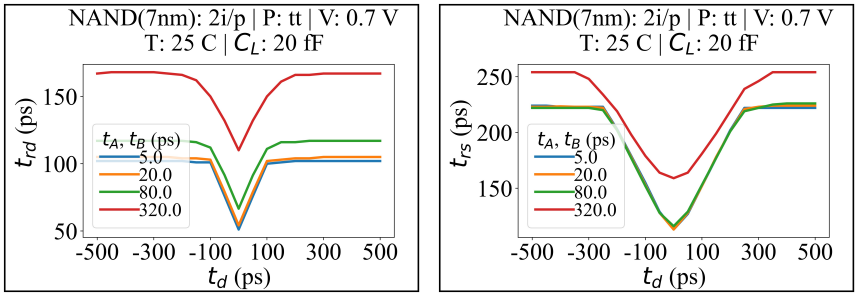
Fig. 4: Correlations between t_{rd}, t_{rs} and T.

2.1.5 Transition Time (t_A and t_B)

From Fig. 5a and Fig. 5b, we observe that t_{rd} and t_{rs} are more at higher t_A and t_B. A slower t_A and t_B increases the time it takes for the input signal to reach the switching threshold, resulting in slower output transitions. Consequently, the transistors switch more slowly, leading to higher t_{rd} and t_{rs}.

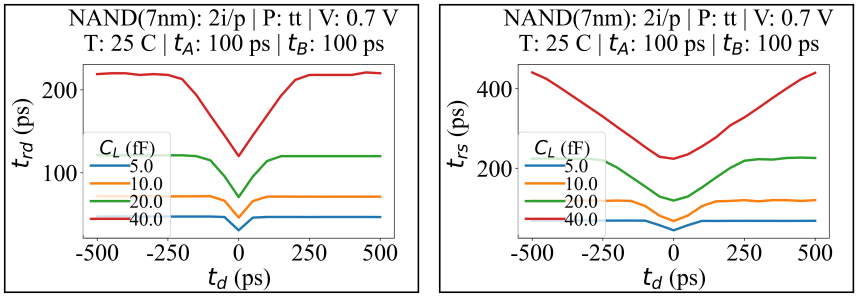
2.1.6 Output Capacitance (C_L)

From Fig. 6a and Fig. 6b, we observe that t_{rd} and t_{rs} are more at higher C_L . When we increase the C_L, the output current takes longer to charge the capacitor, resulting in larger delays. Consequently, the transistors switch more slowly, leading to higher t_{rd} and t_{rs}. Here, at higher C_L, we can see a significant impact on the width of the V-shaped dip due to MIS. It highlights the importance of C_L in modeling MIS.



(a) (b)

Fig. 5: Correlations between t_{rd}, t_{rs} with T_A , t_B.



(a) (b)

Fig. 6: Correlations between t_{rd}, t_{rs} and C_L.

2.2 Dataset Generation and Optimization

The dataset is generated with features (P,V, T, t_A, t_B, C_L, and t_d) with labels (t_{rd}, t_{fd}, t_{rd}, t_{rs}). An algorithm is developed to systematically extract extensive data via SPICE simulation for each combinational cell. It iterates over every possible combination of P, V, T, t_A, t_B, C_L, and t_d within specified ranges. The resulting simulated data, including input parameters and corresponding timing attributes, are stored in an output file.

To reduce characterization efforts, this algorithm is extended to generate an optimized dataset. As illustraed in Fig. 1c and 1d sweeping t_d across the entire range reveals three distinct regions, and this algorithm identifies their boundaries. It systematically explores the t_d range, from the minimum negative to the maximum negative value of t_d. At each t_d value, simulations are conducted to observe t_{rd} and t_{rs}. If t_{rd} and t_{rs} remain constant for four consecutive simulations, the boundary between region $R1$ and $R2$ is marked. Similarly, the algorithm examines the positive range of t_d, from the minimum positive to the maximum positive value. At each t_d, it performs simulations to observe t_{rd} and t_{rs}. If t_{rd} and t_{rs} remain constant for four consecutive simulations, the boundary between region $R2$ and $R3$ is identified. This systematic approach helps differentiate the regions of interest and optimize the dataset.

Regions $R1$ and $R3$ require no further simulation. Region $R3$ captures all points of t_d show-

ing MIS and at least four points showing SIS. By omitting simulations for Regions $R1$ and $R3$ and concentrating on $R2$, the characterization process is optimized. This ensures comprehensive coverage while reducing the effort required for characterization and model training.

2.3 Model-Training and Testing

The overall flow of model training and testing is shown in Fig. 7. In the training and testing phases, the process begins with data pre-processing, which involves several crucial steps. Initially, dependent and independent variables are extracted to define the target and features for the model. Categorical variables (P) are then encoded to convert them into a format suitable for ML algorithms. Subsequently, variable scaling (min-max scaling) is performed to standardize the range of features ($0 \rightarrow 1$), ensuring that all variables contribute equally to the model's performance. After pre-processing, the data is split into training and testing sets. During the

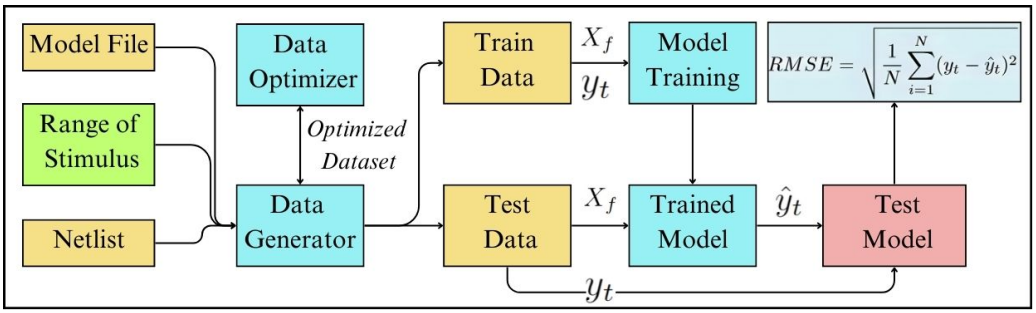


Fig. 7: Training and Testing of model

training phase, the appropriate learning algorithm is selected based on the problem's requirements. The model is trained using the training dataset, followed by hyperparameter tuning to optimize model performance.The models used in this study include linear regression, Lasso regression, decision trees, and artificial neural networks (ANN), each varying in complexity and data representation needs. Typically, more complex models require greater computational resources for training. However, they can provide higher accuracy if overfitting is controlled. Therefore, selecting a predictive model involves balancing complexity, resource demands, and desired accuracy. Once training is complete, the model is tested by applying it to unseen data from the test set, and performance scores are computed to assess its accuracy and generalization capabilities. To further evaluate the model's robustness, k-fold cross-validation is employed, which involves dividing the dataset into multiple subsets and iteratively training and validating the model on different combinations of these subsets. This process helps in detecting issues of underfitting and overfitting, ensuring that the model performs well not just on the training data but also generalizes effectively to new, unseen data.

To evaluate the model's performance, the Root Mean Square Error (RMSE) metric is utilized. This metric compares the predicted target variables, denoted as $\hat{y}_t$, with the actual target

variables, y_t. The prediction error is quantified using RMSE, which provides a measure of the average deviation of the predicted values from the actual values.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_t - \hat{y}_t)^2} \tag{1}$$

where N is the number of samples used to compute the error. Lower RMSE values indicate better predictive accuracy. Once the best-performing model is identified, it is deployed to make predictions on new, unseen data.

3 LiMo's Overview: Previous Work

The proposed tool, LiMo, facilitates the creation of ML-based library models for digital applications. LiMo's structure includes scripts for execution, gate-specific folders containing simulation files, and various Python scripts for dataset generation, optimization, multiprocessing, preprocessing, training, testing, and visualization. It supports the following commands:

- *LiMo:* Invoke the LiMo tool, initiating its main interface.
- *help:* List all available commands within the LiMo tool, along with brief descriptions of their functionality.
- *getLibCells:* Display a list of all available digital logic gates that can be used for dataset generation. This helps users identify which gates are supported by the tool.
- *getVar VAR_NAME:* Report the current path of a specified tool variable.There are two variables "lm_gate_dir" and "lm_out_dir".
- *setVar VAR_NAME Value:* Set or update the value of a specified tool variable.
- *makeOutput:* Create empty folders in the output directory corresponding to each gate present in the gates directory. This prepares the directory structure for storing simulation results and datasets.
- *setInput -cell CELLNAME:* Open the *user_input.tcl* file for a specific gate in the default text editor. This file allows users to define input parameters necessary for dataset generation.
- *genDataset –gate CELLNAMES -optimize OPTIMIZATION_METHOD -num_processes NUM:* This command initiates the dataset generation process for selected gates. It allows users to specify various options, including multiprocessing and optimization methods, to enhance the dataset generation process.
  - *gate:* This parameter specifies the name of the gate for which the user wants to generate datasets.
  - *num_processes:* This parameter allows users to specify the number of CPU cores used for multiprocessing during dataset generation. By setting the value of this parameter, users can control the degree of parallelism and optimize the dataset generation process based on the available computational resources.
  - *optimize:* Users can specify the optimization method for dataset generation using this parameter. The available options include:

    - *none:* This option indicates that no optimization method will be applied during dataset generation. The dataset will be generated without any additional optimization techniques.
    - *skew_ opt:* Choosing this option applies skew optimization during dataset generation/
    - *skew_ slew_ opt:* This option enables both skew and slew optimization during dataset generation.

- *viewDataset -cell CELLNAME –file_ name FILENAME:* Open a specified dataset file for a specific gate using the default CSV viewer. This allows users to inspect the generated dataset.

- *loadData -cell CELLNAME –file_ name FILENAME:* Load data for a specific gate from a file.

- *plotData -cell CELLNAME –file_ name FILENAME:* Generate visual plots of the dataset for a specific gate. This helps in understanding the data distribution and characteristics.

- *infoData -cell CELLNAME –file_ name FILENAME:* Generate and save detailed information about a specific dataset file. This includes statistics about the dataset.

- *splitData -cell CELLNAME –file_ name FILENAME -test_ size TESTSIZE:* Split the dataset into training and testing sets based on a specified test size ratio.

- *preProcessData_ train -cell CELLNAME:* Preprocess the training data for a specific gate. This includes cleaning, normalization, and encoding steps required before training a model.

- *preProcessData_ test -cell CELLNAME:* Preprocess the testing data for a specific gate. This includes cleaning, normalization, and encoding steps required before training a model.

- *trainModel -cell CELLNAME -output MODELNAME:* Train a ML model using the preprocessed training data and save the trained model with the specified name.

- *testModel -cell CELLNAME –load MODELNAME –report REPORTNAME:* Test a previously trained model on the preprocessed testing data and generate a performance report with the specified name.

- *exit:* Exit the LiMo tool, ending the current session.

4 LiMo-OpenSTA: OpenSTA Integration with the Developed ML Delay Model

OpenSTA is an open source Static Timing Analyzer (STA) tool used in digital design. It is utilized to analyze and verify the timing performance of digital circuits at the gate level.

4.1 Installation of LiMo-OpenSTA

LiMo-OpenSTA builds directly on top of the original OpenSTA project. This section provides a step-by-step guide to install OpenSTA, followed by the additional steps required to integrate the LiMo framework.

4.1.1 Install Prerequisites

OpenSTA relies on various dependencies and prerequisites. Please ensure that you have the necessary build dependencies mentioned below installed before proceeding with the installation. Other versions may work, but these are the versions used for development.
cmake 3.10.2 3.24.2 3.16.2
clang 9.1.0 14.0.3
gcc 3.3.2 11.3.0
tcl 8.4 8.6 8.6.6
swig 1.3.28 4.1.0 4.0.1
bison 1.35 3.0.2 3.8.2
flex 2.5.4 2.6.4 2.6.4

4.1.2 To install OpenSTA, follow these steps

- Clone the OpenSTA repository by executing the following command:
  git clone https://github.com/The-OpenROAD-Project/OpenSTA.git
- Move into the OpenSTA directory that was created during the cloning process using the following command:
  cd OpenSTA
- Create a build directory using the following command:
  mkdir build

- Move into the build directory using the following command:
  cd build
- Configure the build by executing the following command:
  cmake .. This command configures the build process for OpenSTA, generating the necessary build files based on the project's configuration.
- Build OpenSTA by running the following command:
  make This command initiates the build process for OpenSTA. It compiles the source code and generates the executable files.
- Install OpenSTA by executing the following command:
  sudo make install

For more information:
Github:
https://github.com/The-OpenROAD-Project/OpenSTA
Manual:
https://github.com/The-OpenROAD-Project/OpenSTA/blob/master/doc/OpenSTA.pdf

4.1.3 Integrate LiMo with OpenSTA

Once OpenSTA is successfully built, no changes to the build system are required to support LiMo-based ML delay modeling. LiMo-OpenSTA reuses the original build flow and simply adds new features.All dependencies remain the same as the original OpenSTA project.

4.1.4 Invoking OpenSTA

The "sta" command is used to invoke OpenSTA and launch the tool. By running this command, you can check if OpenSTA is properly installed. If the installation was successful, the OpenSTA tool interface should launch without any errors or issues source test.tcl This command executes the OpenSTA tool and runs the commands specified in the
test.tcl" script file.

4.2 LiMo-OpenSTA: Modified OpenSTA with ML Integration

To support machine learning-based delay modeling for timing analysis, a modified version of OpenSTA—called **LiMo-OpenSTA**—has been developed. This version incorporates all

necessary changes to replace traditional Liberty-based delay lookups with dynamic predictions from trained ML models.

The complete source code for LiMo-OpenSTA, along with configuration files and integration logic, is available on GitHub:

https://github.com/ndcliiitd/LiMo-OpenSTA

4.3 OpenSTA Modifications for ML-Driven Delay Modeling

4.3.1 Configuration via `models_config.json`

To support flexible ML model integration, a new configuration file *models_config.json* is introduced. This file defines the mapping between gate names and their corresponding ML model files, along with critical metadata required for inference.

Sample Configuration: Sample Configuration:

```
{
  "lazyLoad": true,
  "models": [
    {
      "name": "nand2x1_sc",
      "path": "nand2x1_model.json",
      "modify_annotation": true,
      "inputFormat": ["load", "slew_a", "slew_b", "skew_ab"],
      "outputFormat": ["rise_delay", "rise_slew"],
      "scaleInput": [
        7e-16, 4.65e-14,
        5e-12, 3.2e-10,
        5e-12, 3.2e-10,
        -1e-9, 1e-9
      ],
      "scaleOutput": [
        7.59e-13, 2.96e-10,
        4.56e-12, 5.34e-10
      ]
    }
  ]
}
```

Key Points:

- name: Specifies the gate for which the model applies.

- path: Path to the ML model.
- modify_annotation: true: Enables Ml-based delay modelling instead of LUT based delay modelling.
- inputFormat and outputFormat: Ensure correct feature ordering and dimensions.
- scaleInput and scaleOutput: Provide min-max normalization ranges for features and predictions.
- Can add models for additional gates by simply modifying the JSON — no recompilation needed..

4.3.2 Newly Introduced Classes in the `modification/` Directory for ML Integration

To integrate the ML-based delay model into OpenSTA's timing engine, three new classes are introduced under the modification/ directory:

StaInterface: Acts as a bridge between OpenSTA's delay engine and the ML pipeline. It is responsible for intercepting delay queries, extracting gate attributes (e.g., input slews and output load).

DataToModel: Receives raw data from StaInterface, formats it according to the ML model's input expectations, and applies required scaling using the values defined in models_config.json. After inference, it also processes and stores the predicted outputs (delay and output slew).

MlModel: Loads and manages the ML models for each gate (as defined in the configuration file). When queried, it performs inference based on the formatted inputs and returns predictions.

4.3.3 Functional Flow of ML-Based Delay Modeling Integration

The integration of ML-based delay modeling into OpenSTA is achieved through the coordinated interaction of three newly introduced classes: `StaInterface`, `DataToModel`, and `MlModel`. These components work together to replace traditional Liberty-based delay lookups with dynamic ML-based predictions. The complete flow operates as follows:

- **StaInterface** intercepts the delay calculation request from OpenSTA for a specific gate instance.
- It extracts the relevant gate context, including:
  - Input slews

  - Output load
  - Temporal Distance

- This data is passed to **DataToModel**, which performs the following:
  - Identifies the corresponding ML model for the gate by referencing `models_config.json`
  - Applies Min-Max scaling to normalize the input features
  - Prepares and formats the input vector according to the model's expectations

- The formatted input is then passed to **MlModel**, which:
  - Loads the specified ML model (e.g., JSONformat)
  - Performs inference to predict:
    * **Delay**
    * **Output slew**

- The predicted values are returned to **DataToModel**, which:
  - Applies inverse scaling to convert predictions back to physical units
  - Formats the results for annotation

- Finally, **StaInterface** receives the modified delay and output slew and:
  - Overrides the default delay values from the Liberty library

As a result, OpenSTA uses the MIS-aware, ML-predicted delay and slew values in its timing engine. All downstream timing calculations—including arrival times, required times, and slack analysis—are now based on these enhanced delay metrics, providing improved accuracy under realistic switching conditions.

4.4 Benefits of ML-Based OpenSTA

- **High Accuracy**: Captures Multi-Input Switching (MIS) effects that are difficult to model using traditional Lookup Table (LUT) approaches.
- **Speed**: Enables fast inference of delay and slew during STA runtime, avoiding the overhead of complex analytical models or SPICE simulations.
- **Modularity**: Allows new gates or ML models to be added dynamically via the $models\_config.json$ file without requiring any modification to the OpenSTA source code.

- **Backward Compatibility**: When `modify_annotation` is set to `false`, OpenSTA automatically reverts to its standard Liberty-based delay model, ensuring compatibility with traditional flows.

5 Results

In this paper, we have employed the ISCAS85 benchmarks [13] utilizing the ASAP 7nm PDK [14] to obtain various results.

5.1 Gate-Level Model Evaluation

Table 1 provides the Root Mean Squared Error (RMSE) values for different ML models applied to various combinational cells. The RMSE metric is used to measure the accuracy of the models, with lower values indicating better predictive performance.

Table 1: Evaluation of different ML models at the gate level

	NAND-2	AND-2	NOR-2	OR-2
Linear	0.092	0.073	0.093	0.073
Lasso	0.085	0.062	0.085	0.064
Decision Tree	0.046	0.040	0.049	0.049
Random Forest	0.039	0.038	0.042	0.042
ANN	0.037	0.038	0.027	0.035

Linear and Lasso regression models yield relatively high RMSE values across all combinational cells. These models assume a linear relationship between the features and the target variable, which may not adequately capture the complex, nonlinear interactions inherent in the data. As a result, their predictive performance is limited, reflected in the higher RMSE values. Decision trees and random forests show improved performance compared to linear models, as evidenced by their lower RMSE values. Decision trees partition the data into subsets based on feature values, which allows them to capture nonlinear relationships more effectively. Random forests, an ensemble of decision trees, further enhance this capability by averaging predictions from multiple trees, which reduces overfitting and improves generalization. The lower RMSE values for these models indicate their superior ability to model the complexity of the dataset. The ANN model consistently achieves the lowest RMSE values across all gate types. ANNs are capable of modeling intricate patterns due to their multi-layered architecture, which includes numerous interconnected neurons and non-linear activation functions. This flexibility enables ANNs to effectively capture complex relationships and variations in the data, leading to superior predictive accuracy.

Based on the RMSE values, the ANN model outperforms all other models, demonstrating its suitability for modeling the complexities associated with combinational logic gates. Due to its exceptional performance, we have selected the ANN model for further analysis.

5.1.1 Hyperparameter Tuning for ANN

Hyperparameter tuning was performed using grid search to optimize the performance of the ANN model. This process involves systematically exploring different combinations of hyperparameters to find the set that minimizes the RMSE. The results of this tuning are presented in Table 2.

Table 2: Comparison of RMSE for various ANN layer configurations

Layers	**RMSE**	**Layers**	**RMSE**
10, 10	0.0875	150, 150	0.0460
10, 10, 10	0.0846	150, 100, 50	0.0493
50, 50	0.0691	150, 150, 150	0.0388
50, 50, 50	0.0670	500, 500, 500	0.0529
100, 100	0.0612	1000, 1000	0.0518
100, 100, 100	0.0572	1000, 1000, 100	0.0529

The configuration of three hidden layers, each with 150 neurons (150-150-150), yielded the lowest RMSE of 0.0388. This suggests that a deeper network with a moderate number of neurons per layer effectively captures the complex patterns in the data. Other configurations, such as 50-50-50 and 150-100-50, also performed well but achieved a different level of accuracy. Configurations with fewer layers or neurons (e.g., 10-10-10) and excessive layers (e.g., 1000-1000) exhibited higher RMSE values, indicating insufficient overfitting issues. Using the ReLU activation function and the Adam optimizer, combined with L2 regularization, contributed to the ANN model's effectiveness. These choices help prevent overfitting and improve convergence during training.

The optimal hyperparameters identified for the ANN model include a learning rate of 0.001, a batch size of 500, and a network architecture of 150-150-150 neurons across three hidden layers. These settings provide the best balance between model complexity and performance.

5.1.2 Optimized Gate Level Evaluation

Table 3: Evaluation of ML models with and without optimized datasets

Gates	Without Optimization				With Optimization			
	#Total	#Train	#Test	RMSE	#Total	#Train	#Test	RMSE
NAND	8064	6451	1613	0.037	4857	4857	1613	0.039
NOR	8064	6451	1613	0.038	4521	4521	1613	0.038
AND	8064	6451	1613	0.027	4027	4027	1613	0.030
OR	8064	6451	1613	0.035	3633	3633	1613	0.037

Table 3 provides a detailed comparison of ML model performance when trained with optimized versus non-optimized datasets across various logic gates. For models trained with the

optimized datasets, the RMSE values are generally comparable to those obtained from models trained with the larger, non-optimized datasets. These findings indicate that the optimized dataset approach maintains an accuracy similar to the larger, non-optimized datasets. The reduction in RMSE values is minimal, demonstrating that dataset optimization does not significantly compromise model performance. Our optimization strategy effectively reduces the time required for characterization and training while preserving the integrity of the model's predictive capabilities. Importantly, the test dataset remains consistent across both optimized and non-optimized models, ensuring a fair comparison and validating that the observed efficiency gains are a direct result of the dataset optimization. This approach highlights the effectiveness of dataset optimization in enhancing computational efficiency without sacrificing accuracy.

5.2 Circuit-Level Model Evaluation

To further validate the ANN model, we applied it to the ISCAS'85 C17 benchmark circuit. This evaluation involves analyzing nodes Y1 and Y2, as depicted in Fig. 8 and 9. This circuit comprises five inputs and two outputs, and our analysis focuses on two nodes labeled Y1 and Y2. The evaluation framework is depicted in Figure 9, where we perform Static Timing Analysis (STA) and extract key parameters such as P, V, T, t_A, t_B, C_L, and the temporal difference between arrival times at the two inputs (t_D). These inputs are fed into both a SPICE simulation and the trained model. From the SPICE simulation, we obtain actual values for t_{rd} and t_{rs} under MIS. It is important to note that, as we are experimenting on a NAND gate, MIS effects will predominantly manifest as variations in t_{rd} and t_{rs}, with slow-down effects being minimal (negligible); therefore, we focus on these parameters. The variables t_{rd} and t_{rs} under MIS conditions are also extracted from the timing reports.

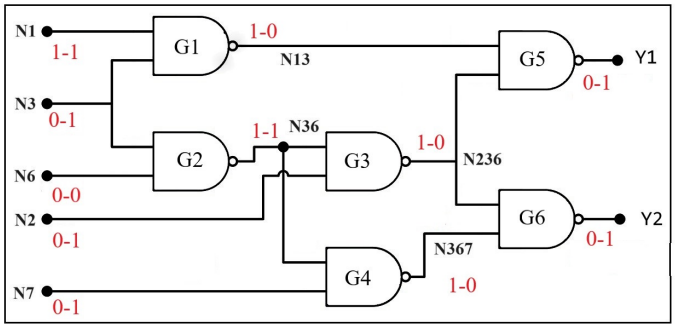


Fig. 8: ISCAS'85 C17 Benchmark Circuit

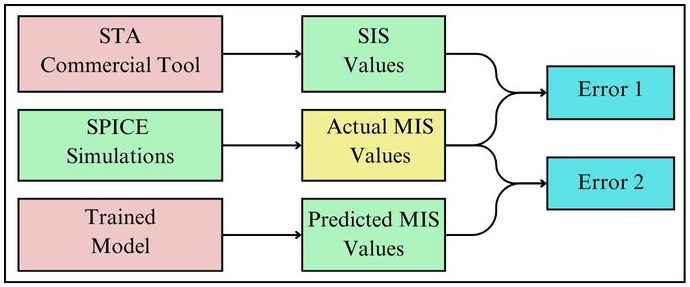


Fig. 9: Model Evaluation Framework

Table 4: Evaluation of Delays in C17 ISCAS Benchmark

(a) Timing attributes at Y1

(t_{N13} : 16ps, t_{N236} : 21ps, $C_L(Y1)$: 20fF)

AT_{N13}–AT_{N236}	R	SIS t_{rd}	SIS t_{rs}	Actual MIS t_{rd}	Actual MIS t_{rs}	Pred. MIS t_{rd}	Pred. MIS t_{rs}	Error 1	Error 2
-1000	R1	105	222	104	223	106	221	0.70	1.41
-200	R2	105	222	104	198	100	198	6.54	1.92
-100	R2	105	222	100	148	98	157	27.50	4.04
0	R2	105	222	57	112	54	128	91.21	9.77
100	R2	105	222	96	154	99	163	26.77	4.48
200	R2	105	222	103	203	102	208	5.65	1.72
1000	R3	105	222	103	222	104	221	0.97	0.71
Average								**22.76**	**3.44**

(b) Timing attributes at Y2

(t_{N236} : 21ps, t_{N367} : 16ps, $C_L(Y2)$: 20fF)

AT_{N236}–AT_{N367}	R	SIS t_{rd}	SIS t_{rs}	Actual MIS t_{rd}	Actual MIS t_{rs}	Pred. MIS t_{rd}	Pred. MIS t_{rs}	Error 1	Error 2
1000	R1	105	222	104	222	108	223	0.48	2.15
200	R2	105	222	104	198	104	193	6.54	1.26
100	R2	105	222	100	148	103	153	27.50	3.19
0	R2	105	222	57	112	52	127	91.21	11.08
-100	R2	105	222	96	154	100	161	26.77	4.36
-200	R2	105	222	103	203	104	197	5.65	1.96
-1000	R3	105	222	103	222	105	223	0.97	1.20
Average								**22.73**	**3.60**

As illustrated in Fig. 9, the SIS timing values (t_{rd} and t_{rs}) are obtained from STA reports. The actual MIS timing values (Actual t_{rd} and Actual t_{rs}) are derived from SPICE simulations, while the predicted MIS timing values (Pred. t_{rd} and Pred. t_{rs}) are generated through our trained model. To assess the accuracy of these predictions, error percentages are calculated between the SIS values, actual values, and predicted values using the following equations:

$$Error\ 1 = \left(\frac{\text{SIS } t_{rd} - \text{Actual MIS } t_{rd}}{\text{Actual MIS } t_{rd}} \right) \times 100\% \tag{2}$$

$$Error\ 2 = \left(\frac{\text{Pred. MIS } t_{rd} - \text{Actual MIS } t_{rd}}{\text{Actual MIS } t_{rd}} \right) \times 100\% \tag{3}$$

The evaluation results for node Y1 are summarized in Table 4(a). The column labeled AT_{N13}-AT_{N236} represents the difference in arrival times at the two input nodes, which is analogous to t_D in the proposed ML model.

The results for node Y1, presented in Table 4(a), indicate that the prediction model aligns closely with SPICE results. Specifically, the average error for the first metric (Error 1) is 22.76% across all test cases. In contrast, the average error for the second metric (Error 2) is considerably lower at 3.44%. For node Y2, as shown in Table 4(b), similar trends are observed. The average error for the first metric (Error 1) is 22.73%, while the average error for the second metric (Error 2) is slightly higher at 3.60%. These results underscore the model's capability to produce predictions that are consistently in close agreement with actual results, even under MIS conditions.

5.3 Evaluation of LiMo-OpenSTA

5.3.1 Evaluation of LiMo-OpenSTA Without Considering MIS

To validate the correctness and accuracy of the ML-based delay model integrated into OpenSTA (referred to as **LiMo-OpenSTA**), a comparative study was performed against the traditional timing model derived from the Liberty (.lib) file. The evaluation was carried out using a simple Verilog module with a 2-input NAND gate (`nand2x1_sc`) connected to two input signals `IN1`, `IN2` and a single output `OUT`.

```
nand2x1_sc i1(.Y(OUT), .A(IN1), .B(IN2));
```

Timing analysis was performed using both:

- **Traditional STA based on the .lib NLDM model**
- **Modified STA using the ML-based delay prediction (ANN model)**

As shown in the STA report snapshots Fig 10, both approaches yield correct timing propagation results.

STA based on the traditional library model (NLDM)

```
This program comes with ABSOLUTELY NO WARRANTY; for detai
% source sta0.tcl
Startpoint: IN1 (input port clocked by clock)
Endpoint: OUT (output port clocked by clock)
Path Group: clock
Path Type: max

  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock clock (rise edge)
   0.00    0.00   clock network delay (ideal)
   1.25    1.25 ^ input external delay
   0.00    1.25 ^ IN1 (in)
   0.34    1.59 v i1/Y (nand2x1_sc)
   0.00    1.59 v OUT (out)
           1.59   data arrival time

  10.00   10.00   clock clock (rise edge)
   0.00   10.00   clock network delay (ideal)
   0.00   10.00   clock reconvergence pessimism
  -1.00    9.00   output external delay
           9.00   data required time
---------------------------------------------------------
           9.00   data required time
          -1.59   data arrival time
---------------------------------------------------------
           7.41   slack (MET)
```

STA based on the ML model

```
% source sta1.tcl
Warning: example_typ.lib line 1, library NangateOpenCellL
Startpoint: IN1 (input port clocked by clock)
Endpoint: OUT (output port clocked by clock)
Path Group: clock
Path Type: max

  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock clock (rise edge)
   0.00    0.00   clock network delay (ideal)
   1.25    1.25 v input external delay
   0.00    1.25 v IN1 (in)
   0.33    1.58 ^ i1/Y (nand2x1_sc)
   0.00    1.58 ^ OUT (out)
           1.58   data arrival time

  10.00   10.00   clock clock (rise edge)
   0.00   10.00   clock network delay (ideal)
   0.00   10.00   clock reconvergence pessimism
  -1.00    9.00   output external delay
           9.00   data required time
---------------------------------------------------------
           9.00   data required time
          -1.58   data arrival time
---------------------------------------------------------
           7.42   slack (MET)
```

Fig. 10: Evaluation of LiMo-OpenSTA Without Considering MIS

- **NLDM-based delay**: 0.34 ns
- **ML-based delay**: 0.33 ns

The overall timing remains consistent, and the slack remains met in both cases:

- **Slack using .lib model**: 7.41 ns
- **Slack using ML model**: 7.42 ns

Additionally, a Table 5 highlights the delay values produced under varying input slew conditions:

Table 5: Delay Comparison Between .lib and ML Models

IN1 (Slew)	IN2 (Slew)	Delay (.lib)	Delay (ANN)
300p	300p	340p	330p
400p	400p	370p	370p
500p	500p	390p	400p

The similar traditional and ML-based delay values confirm the robustness of the LiMo-OpenSTA approach in standard timing scenarios where MIS effects are not considered.

5.3.2 Evaluation of LiMo-OpenSTA While Considering MIS

To validate the correctness and accuracy of the ML-based delay model integrated into OpenSTA while considering MIS, a comparative study was performed against the traditional timing model derived from the Liberty (.lib) file. The evaluation was carried out using a simple Verilog module with a 2-input NAND gate (`nand2x1_sc`) connected to two input signals `A1`, `A2` and a single output `out`.

```
nand2x1_sc i1(.Y(out), .A(A1), .B(A2));
```

Timing analysis was performed using both:

- **Traditional STA based on the .lib NLDM model**
- **Modified STA using the ML-based delay prediction (ANN model) which also take into account temporal distance**

STA based on the traditional library model (NLDM)

```
Startpoint: A2 (input port clocked by virtual_clk)
Endpoint: ZN (output port clocked by virtual_clk)
Path Group: virtual_clk
Path Type: max

     Delay       Time   Description
---------------------------------------------------------
   0.00000    0.00000   clock virtual_clk (rise edge)
   0.00000    0.00000   clock network delay (ideal)
   0.00000    0.00000 v input external delay
   0.00000    0.00000 v A2 (in)
   0.18160    0.18160 ^ NAND1/Y (nand2x1_sc)
   0.00000    0.18160 ^ ZN (out)
              0.18160   data arrival time

   1.00000    1.00000   clock virtual_clk (rise edge)
   0.00000    1.00000   clock network delay (ideal)
   0.00000    1.00000   clock reconvergence pessimism
   0.00000    1.00000   output external delay
              1.00000   data required time
---------------------------------------------------------
              1.00000   data required time
             -0.18160   data arrival time
---------------------------------------------------------
              0.81840   slack (MET)
```

STA based on the ML model

```
Startpoint: A2 (input port clocked by virtual_clk)
Endpoint: ZN (output port clocked by virtual_clk)
Path Group: virtual_clk
Path Type: max

     Delay       Time   Description
---------------------------------------------------------
   0.00000    0.00000   clock virtual_clk (rise edge)
   0.00000    0.00000   clock network delay (ideal)
   1.00000    1.00000 v input external delay
   0.00000    1.00000 v A2 (in)
  -0.79498    0.20502 ^ NAND1/Y (nand2x1_sc)
   0.00000    0.20502 ^ ZN (out)
              0.20502   data arrival time

   1.00000    1.00000   clock virtual_clk (rise edge)
   0.00000    1.00000   clock network delay (ideal)
   0.00000    1.00000   clock reconvergence pessimism
   0.00000    1.00000   output external delay
              1.00000   data required time
---------------------------------------------------------
              1.00000   data required time
             -0.20502   data arrival time
---------------------------------------------------------
              0.79498   slack (MET)
```

Fig. 11: Evaluation of LiMo-OpenSTA While Considering MIS

The STA report snapshots Fig 11. Input slews (A1:21ps, A2:16ps), output load (20fF), and arrival times at each pin were extracted during analysis. From this information, temporal distance (1000ps) was calculated and passed to the ML model. The predicted delay was then annotated back into the STA flow.

As illustrated in Fig. 11, the integration of the ML-based delay prediction model with the temporal distance into the OpenSTA framework and the subsequent annotation of predicted delays were functionally successful. The modified STA flow executed without errors. However, we are currently observing significant deviations between the delays predicted by the ML model and those from the NLDM-based analysis. The issue is under active investigation and debugging is ongoing to identify and resolve the discrepancy.

6 Conclusion

In conclusion, LiMo significantly improves over traditional lookup table-based methods for timing verification in VLSI design. By automating the creation of ML based timing models and considering factors such as MIS and PVT variations, LiMo provides a more accurate and efficient solution. LiMo's automation of dataset generation, optimization, training, and evaluation streamlines the timing verification process. Testing showed that LiMo's ANN model has an impressive accuracy, with a Root Mean Square RMSE of around 4% compared to SPICE simulations. Finally, by modifying OpenSTA to support ML-based delay modelling, LiMo enables direct inference of delay under MIS conditions, offering more dynamic and adaptive timing analysis. Furthermore, a customized version of OpenSTA, named LiMo-OpenSTA, has been developed to replace traditional Liberty (.lib) delay lookups with real-time predictions from trained ML models. This integration enables accurate delay inference under MIS conditions and supports a more intelligent and dynamic timing analysis within the STA flow. The complete source code, configuration files, and integration logic for LiMo-OpenSTA are publicly available at:
https://github.com/ndcliiitd/LiMo-OpenSTA
However, during evaluation under MIS conditions, we observed notable discrepancies between the ANN-based predictions and NLDM-based values. This deviation is currently under investigation. Future work will focus on deeper integration with OpenSTA, improved ML model architectures, and advanced dataset optimization techniques to further enhance accuracy and scalability.

References

[1] S. Saurabh, "Introduction to VLSI Design Flow," *Cambridge University Press*, 2023.

[2] A. B. Kahng, U. Mallappa, L. Saul and S. Tong, " "Unobserved Corner" Prediction: Reducing Timing Analysis Effort for Faster Design Convergence in Advanced-Node Design," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019, pp. 168-173, doi: 10.23919/DATE.2019.8715102.

[3] D. Sinha, V. Rao, C. Peddawad, M. Wood, J. Hemmett, S. Skariah, and P. Williams, "Statistical Timing Analysis considering Multiple-Input Switching," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2020, pp. 1-6. DOI: 10.1109/DAC18072.2020.9218601.

[4] O. V. S. Shashank Ram and S. Saurabh, "Modeling Multiple-Input Switching in Timing Analysis Using Machine Learning," in *IEEE Transactions on Computer-Aided Design of*

Integrated Circuits and Systems, vol. 40, no. 4, pp. 723-734, April 2021, doi: 10.1109/T-CAD.2020.3009624.

[5] W. Raslan and Y. Ismail, "Deep-learning cell-delay modeling for static timing analysis," *Ain Shams Engineering Journal*, vol. 14, no. 1, pp. 101828, 2023. [Online]. Available: https://doi.org/10.1016/j.asej.2022.101828. ISSN: 2090-4479.

[6] F. Klemme, Y. Chauhan, J. Henkel, and H. Amrouch, "Cell library characterization using machine learning for design technology co-optimization," in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD '20)*, November 2020, pp. 1–9. DOI: 10.1145/3400302.3415713.

[7] S. M. Ebrahimipour, B. Ghavami, H. Mousavi, M. Raji, Z. Fang and L. Shannon, "Aadam: A Fast, Accurate, and Versatile Aging-Aware Cell Library Delay Model using Feed-Forward Neural Network," *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, San Diego, CA, USA, 2020, pp. 1-9.

[8] P. d'Hondt, A. Ladhar, P. Girard and A. Virazel, "A Learning-Based Methodology for Accelerating Cell-Aware Model Generation," *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2021, pp. 1580-1585, doi: 10.23919/DATE51398.2021.9474227

[9] E. Naswali, N. Kim, and P. Chandran, "Fast and Accurate Library Generation Leveraging Deep Learning for OCV Modelling," *IEEE International Symposium on Quality Electronic Design (ISQED)*, pp. 349-354, 2021. doi: 10.1109/ISQED51717.2021.9424316.

[10] D. Sinha et al., "Statistical Timing Analysis considering Multiple-Input Switching," *2020 57th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2020, pp. 1-6, doi: 10.1109/DAC18072.2020.9218601.

[11] Cadence Design Systems, Inc., "Tempus Timing Signoff Solution," [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/silicon-signoff/tempus-timing-signoff-solution.html. [Accessed: May 24, 2024].

[12] Cadence Design Systems, Inc., "Virtuoso ADE Suite," [Online]. Available: https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-design/virtuoso-ade-suite.html. [Accessed: May 24, 2024].

[13] ISCAS Circuits. Available: http://www.pld.ttu.ee/ maksim/benchmarks/

[14] L.T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthya, and G. Yeric,"ASAP7: A 7-nm FinFET Predictive Process Design Kit," *Microelectronics Journal*, vol. 53, pp. 105-115, July 2016.