Digital VLSI Design

Leakage Prediction of C499 Circuit Using ML

Project 2 Report

Sankalp S Bhat - 2020112018

Sri Anvith Dosapati - 2020102015

Shreeya Singh - 2020102011

Srujana Vanka - 2020102005

1 Introduction

In the domain of integrated circuit design, the importance of understanding and forecasting leakage power has surged. The continuous drive towards achieving greater integration density and performance in CMOS devices has led to a significant increase in leakage current, which now stands as a major contributor to total power consumption.

In VLSI circuit design, thorough testing under various conditions is imperative before production, necessitating exhaustive evaluations of MOSFETs across diverse parameters outlined in the technology files. However, conventional testing methods for individual inputs can be time-consuming, potentially impeding the pace of circuit development and production. This project aims to address this challenge by leveraging machine learning techniques to accurately predict leakage power in CMOS circuits.

2 Objectives

1. Generate datasets for standard cells on all tech nodes, with a particular focus on the gates involved in the ISCAS-85 C499 complex circuit. This involves sampling Process, Voltage, and Temperature (PVT) combinations from predefined distributions and using these values to simulate circuits.

2. Conduct exploratory data analysis to gain insights on different variables. Analyze how leakage and delay vary with tech node by conducting exploratory data analytics on the generated dataset

3. Train separate models for each standard cell type (5 models) at the 22nm MGK/HP technology node. Use these models to predict leakage power for each standard cell in the C499 circuit and calculate the predicted leakage of each cell to estimate the total leakage power of C499.

3 Dataset Generation

3.1 Methodology

The methodology for generating the PVT varied datasets on a 45nm-HP node comprises the following steps:

1. **Spice Netlist Construction:**

- Spice netlists for all the standard gates were constructed based on the provided circuit schematics.
- The netlists are stored in the "netlists" folder, with gate delay and leakage netlists named as <gate name>_delay/leakage.net.

2. **PVT Distribution Generation:**

   - PVT distributions were generated within predetermined bounds for critical process parameters, voltage, and temperature.
   - Monte Carlo distribution principles were applied to ensure realistic variations reflecting semiconductor manufacturing variability.

   Temperature Parameters:

   ```
   temp_min = -55
   temp_max = 125
   ```

   Voltage Parameters:

   ```
   nominal_voltage = 1.0
   voltage_variation = 0.1
   ```

   Process Parameters:

   ```
   toxe_n = 9e-010
   toxm_n = 9e-010
   toxref_n = 9e-010
   xj_n = 1.4e-008
   ndep_n = 6.5e+018

   toxe_p = 9.2e-010
   toxm_p = 9.2e-010
   toxref_p = 9.2e-010
   xj_p = 1.4e-008
   ndep_p = 2.8e+018

   toxp_par = 6.5e-010
   ```

   Delay Parameters:

```
cqload_min = 0.01e-15
cqload_max = 5e-15
```

3. **Sampling Methodology:**
    - A systematic sampling methodology was implemented to generate diverse PVT combinations for each standard cell.
    - PVT values were sampled from the generated distributions, ensuring adequate coverage of the parameter space.
4. **Simulation Setup:**
    - NGSPICE was configured to simulate circuits using the constructed spice netlists and sampled PVT values.
    - Each simulation run considered a specific PVT combination, facilitating the characterization of circuit behavior under varied conditions.
5. **Circuit Simulation and Data Collection:**
    - Circuits were simulated for each PVT combination, and static leakage power and propagation delay measurements were recorded.
    - DC analysis was performed for leakage power estimation, while transient analysis with PWL signals was utilized for delay estimation.
6. **Dataset Generation Script:**
    - The final script for each gate to generate the dataset includes functions for running NGSPICE simulations for leakage and delay.
    - For each gate, the script constructs a spice netlist based on the provided circuit schematics and PVT parameters.
    - The script then runs NGSPICE simulations using the constructed netlists and sampled PVT values, collecting leakage and delay measurements for each PVT combination.
    - The results are stored in a data frame and saved to a CSV file for further analysis.

We now conduct data analytics on the generated dataset to extract useful insights and observations. The data points collected from the NGSpice code are fed to the machine learning algorithm to predict leakage in the complex cell C499.

4 Exploratory Data Analysis

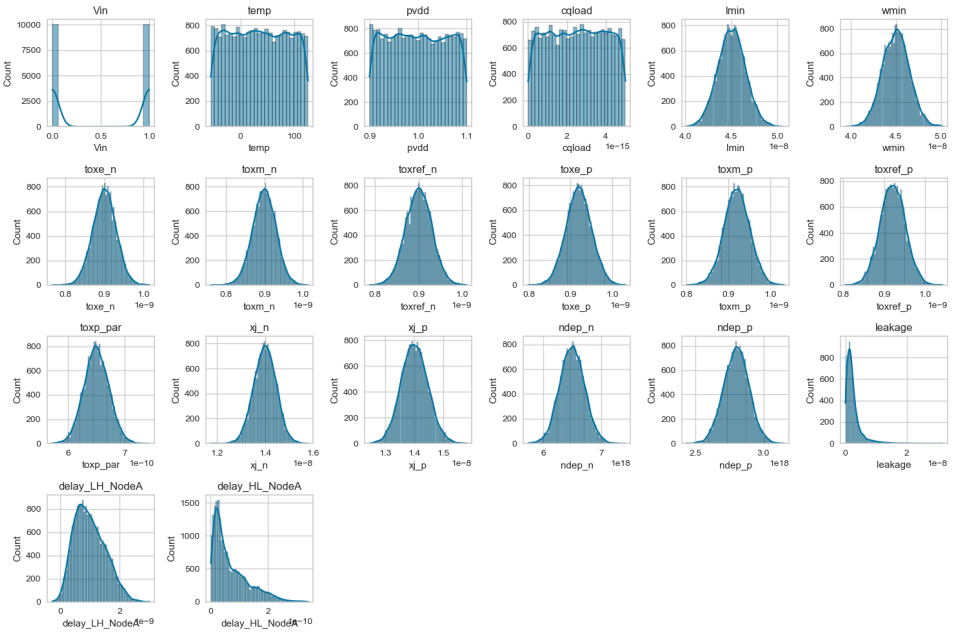

Figure 1: Pairplot

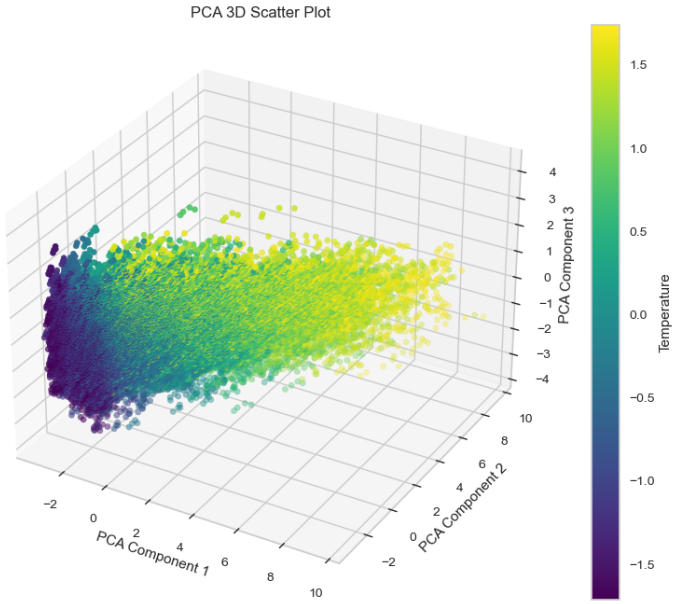

Figure 2: PCA

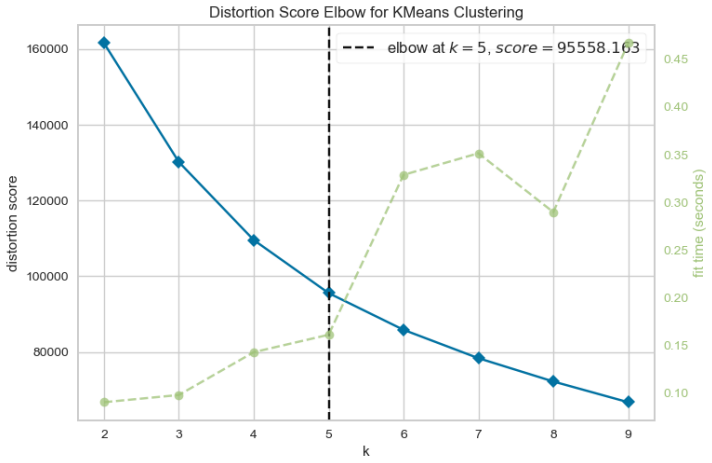

Figure 3: Elbow Plot

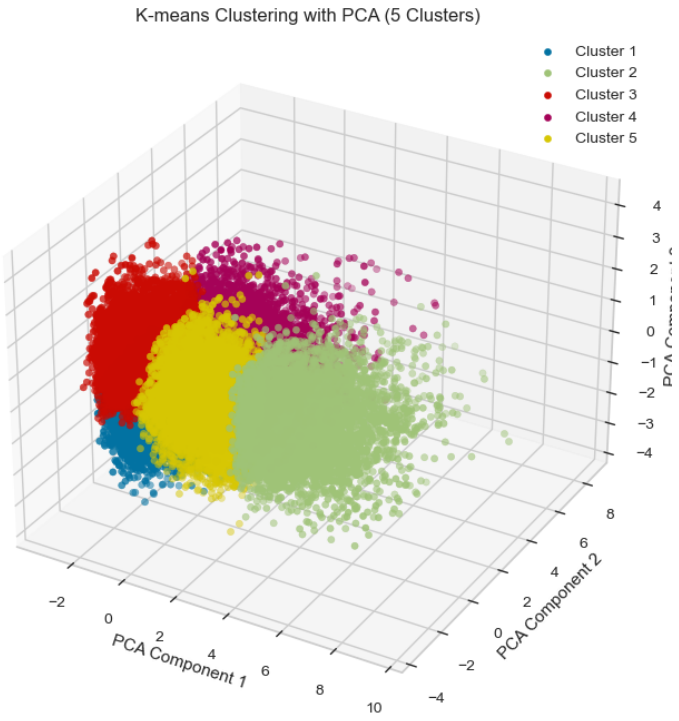

Figure 4: K-means Clustering

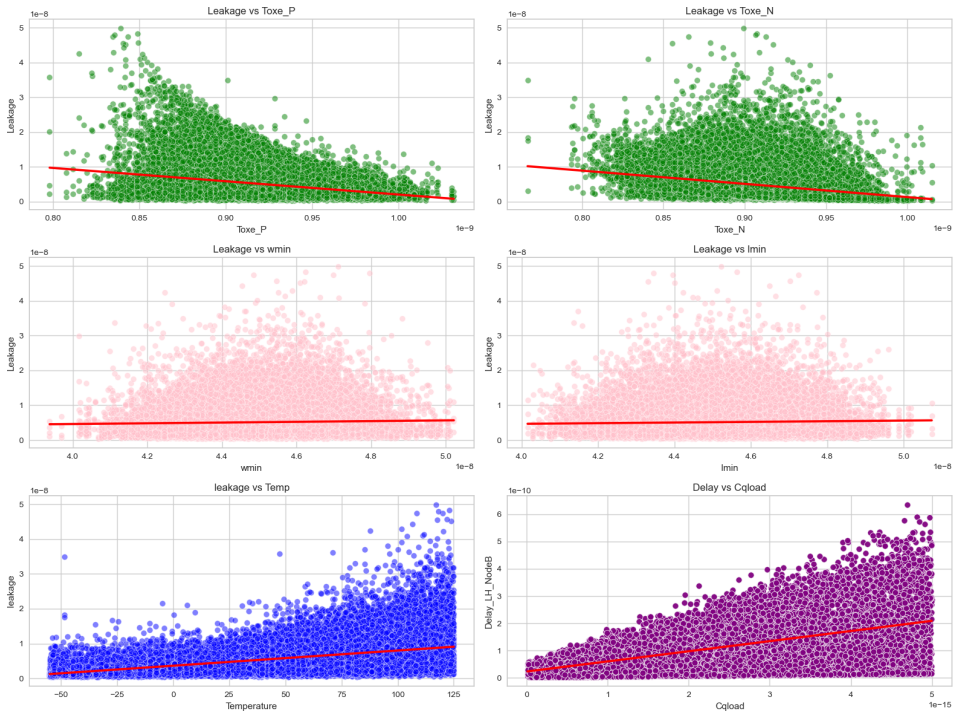

Figure 5: Various Trends

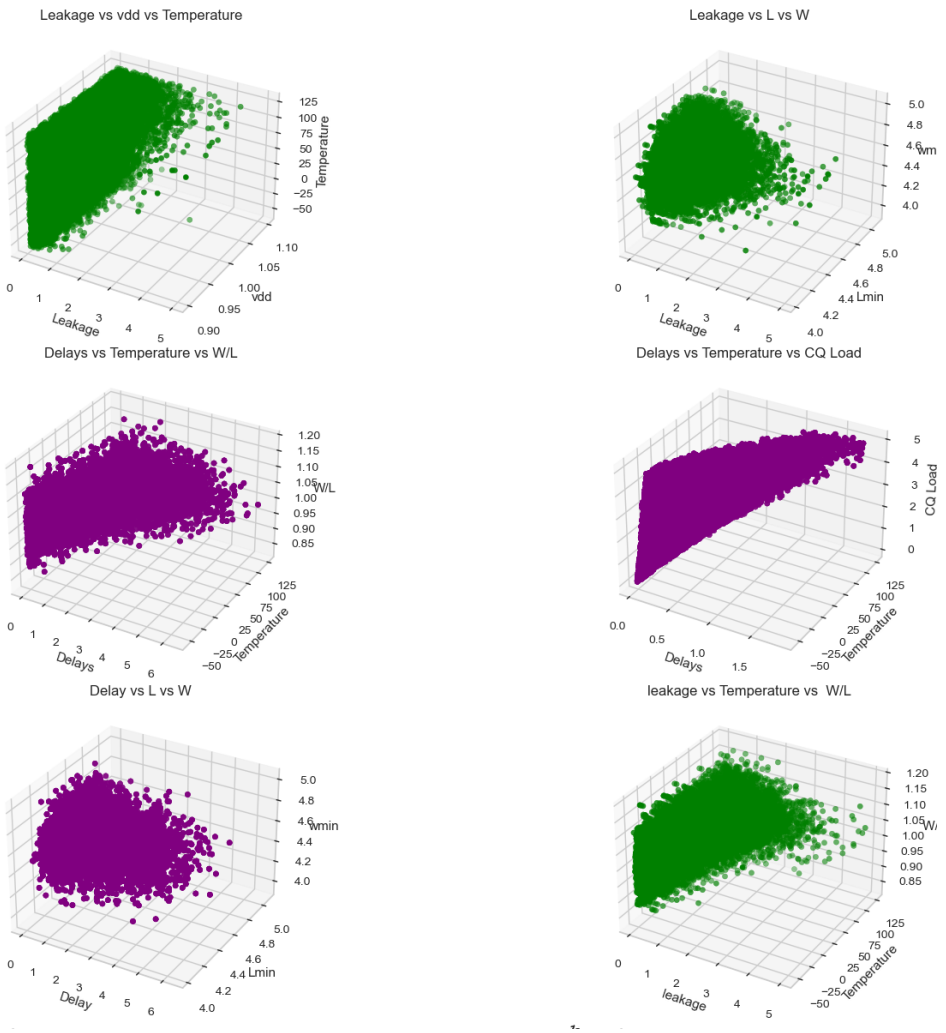

Figure 6: 3D Plots - Various Trends

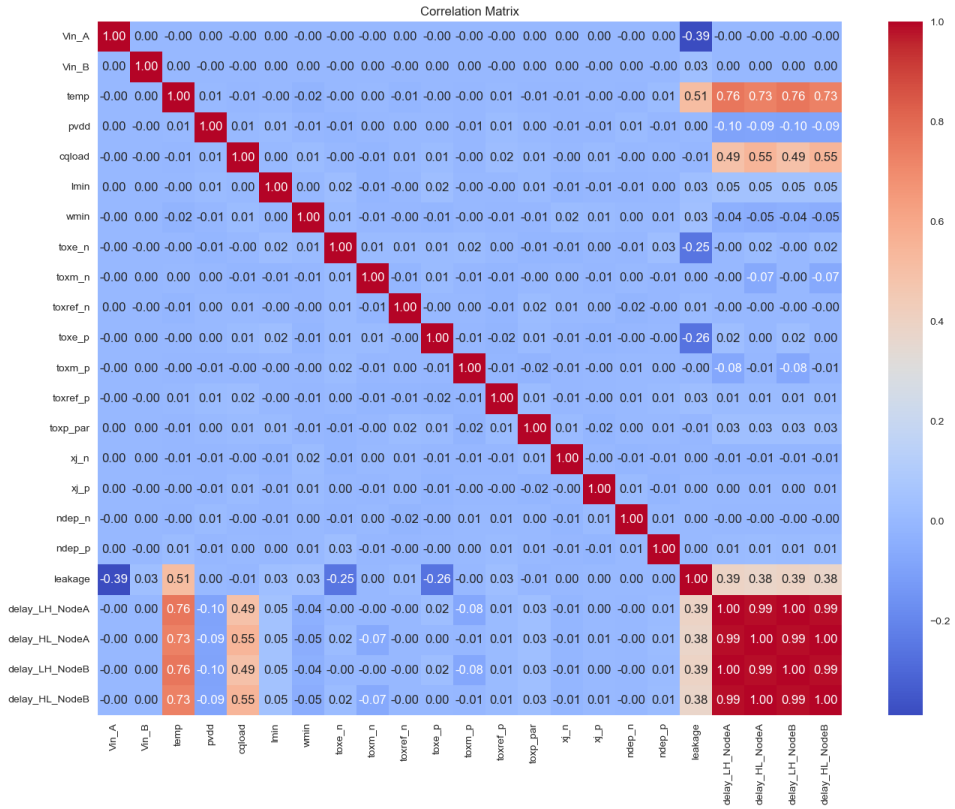

Figure 7: Correlation Matrix

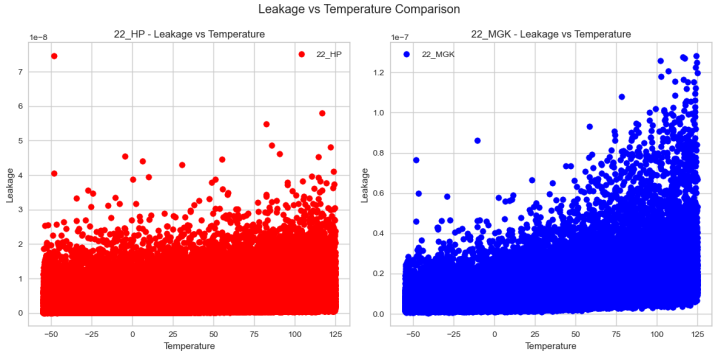

Figure 8: MGK vs HP

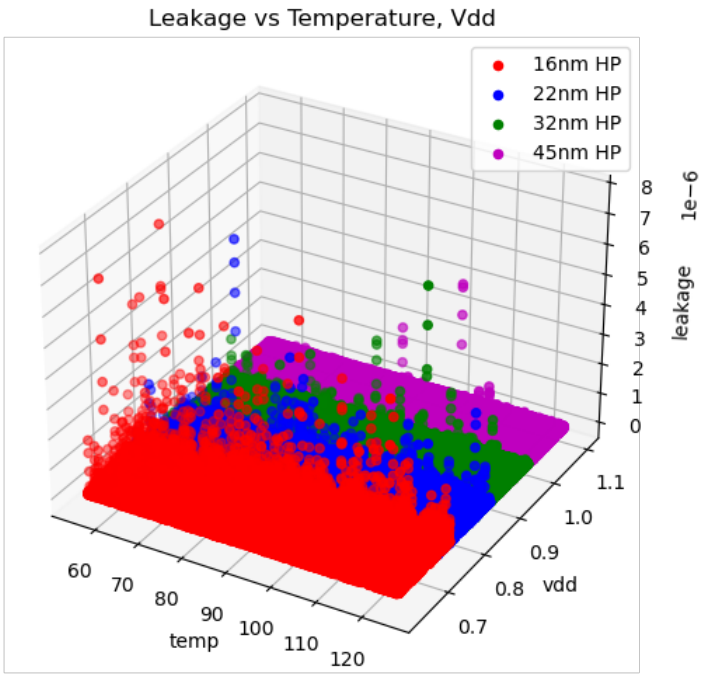

Figure 9: Across nodes

5 Meta Learning Approach

Meta-learning involves training a model on multiple tasks or datasets, with the goal of enabling the model to generalize well to new tasks or datasets. In the context of our leakage power prediction task, meta-learning allows the model to learn from various scenarios and adapt quickly to new instances.

5.1 Reptile ALgorithm

The Reptile algorithm is a meta-learning technique that focuses on rapid adaptation to new tasks. It operates through the following steps:

1. **Initialization:** Randomly initialize model parameters (weights and biases) for each task.

2. **Inner Loop (Task-Specific Optimization):**

- Sample a batch of data for a specific task.
- Perform several iterations of gradient descent on this task-specific data to update the model parameters

3. **Meta Update:** Update the model's initial parameters based on the changes observed in the task-specific parameters. This update is done using a factor called epsilon, which controls the extent of adaptation.

The meta-update step allows the model to adjust its parameters in a way that enhances its ability to generalize across tasks. Reptile focuses on finding a common initialization point that works well for a wide range of tasks.

5.2 Justification

- **Fast Adaptation:** Reptile's rapid adaptation capability is well-suited for scenarios where we encounter new instances or variations in leakage power prediction tasks.
- **Generalization**: By learning from multiple tasks, the model becomes more robust and generalizes better to unseen instances, potentially improving prediction accuracy.
- **Efficient Training:** The iterative nature of Reptile allows for efficient training, especially in scenarios with limited data or computational resources.

5.3 Training Results

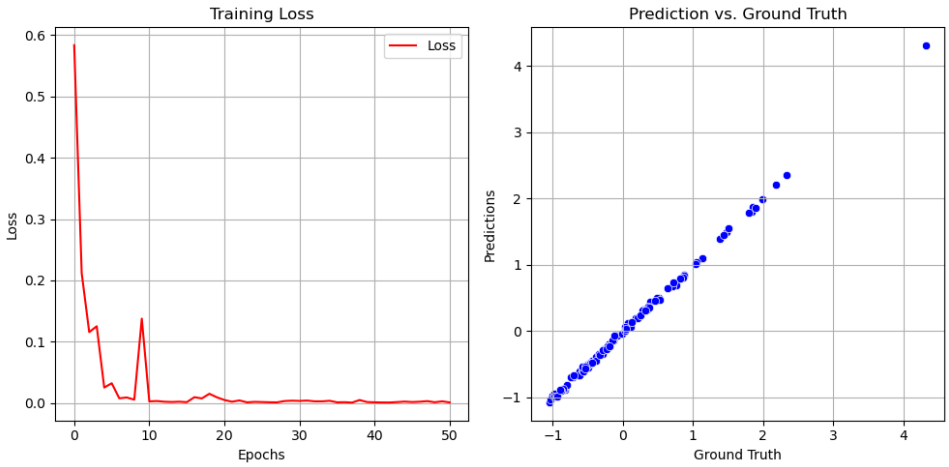

Figure 10: Training Loop

Metric	Value
Loss	0.024841774255037308
R2 Score	0.9716715174654722
MAE	0.1306450707632746
EV	0.9749384640126713
MAPE	0.42481212010346

Table 1: Metrics at the 50^{th} Epoch

Based on the training results, we observed significant improvements in performance metrics over the training epochs. These improvements indicate that the Reptile algorithm effectively learned and adapted to the leakage power prediction task.

5.4 Algorithm

The following equations showcase the key components of the Reptile algorithm and its adaptation process:

1. **Initialization**: w_i and b_i are initialized randomly for each layer i. - w_i represents the weights of layer i. - b_i represents the biases of layer i.

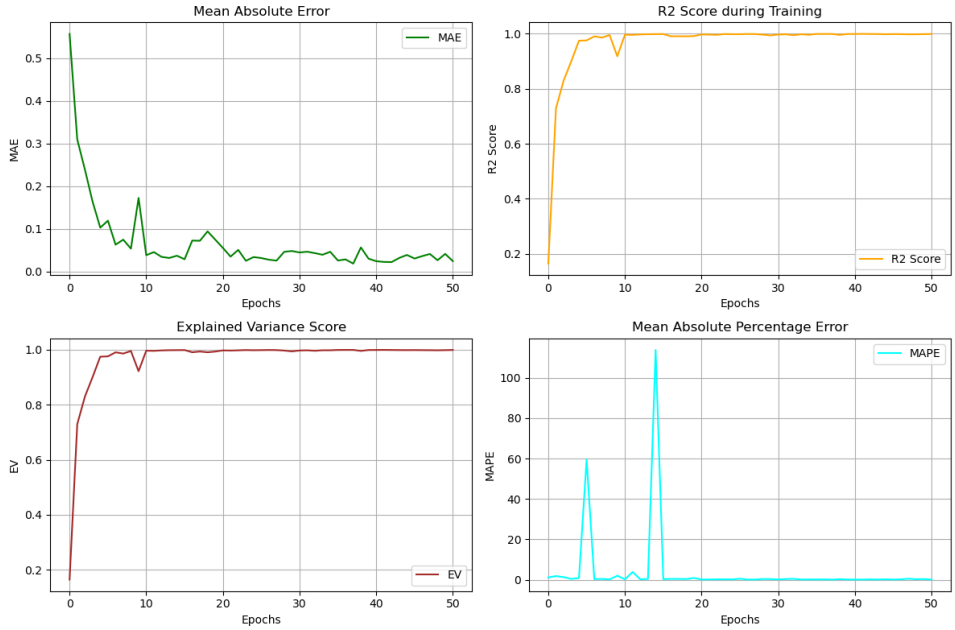

Figure 11: Training Results

$$z_i = X \times w_i + b_i$$

$$a_i = \text{Activation Function}(z_i)$$

$$\hat{Y} = z_n$$

2. **Loss Function**: Mean Squared Error (MSE) is commonly used for regression tasks.

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

3. **Meta Update**:

$$\text{Updated } w_i = \text{Old } w_i + \epsilon \times (\text{Task-Specific } w_i - \text{Old } w_i)$$

$$\text{Updated } b_i = \text{Old } b_i + \epsilon \times (\text{Task-Specific } b_i - \text{Old } b_i)$$

By leveraging meta-learning and the Reptile algorithm, we can enhance our leakage power prediction model's adaptability and accuracy across diverse scenarios.

Algorithm 1 Reptile, serial version

Initialize ϕ, the vector of initial parameters
for iteration $= 1, 2, \ldots$ **do**
  Sample task τ, corresponding to loss L_τ on weight vectors W
  Compute $W = \mathrm{SGD}(L_\tau, \phi, k)$
  Update $\phi \leftarrow \phi + \epsilon(W - \phi)$
end for

Figure 12: Pseudo Code of the Reptile Algorithm

6 Mathematical Transformations Approach

6.1 Models Explored

Gradient boosting models (LightGBM, CatBoost, XGBoost), AdaBoost, and Random Forests are being explored initially for their ability to predict leakage power in a circuit design, considering their strength in capturing complex relationships within the data.

The chosen models (LightGBM, CatBoost, XGBoost, Random Forest Regressor, AdaBoost Regressor) are all ideal for regression tasks, but each brings specific strengths to predicting leakage power:

1. **XGBoost Regressor**: Another ensemble method using gradient boosting for decision trees. It can effectively capture complex patterns and handle high dimensionality with proper hyperparameter tuning.

2. **LightGBM Regressor**: A high-performance gradient boosting framework known for efficiency and accuracy. It excels at capturing non-linear relationships and feature interactions.

3. **CatBoost Regressor**: A gradient boosting framework similar to LightGBM, it offers good accuracy and handles categorical features well. It also focuses on interpretability through feature importance scores.

4. **Random Forest Regressor**: An ensemble method that averages predictions from multiple decision trees. A robust option for handling high dimensionality and non-linear relationships, especially when

interpretability is a concern (to some extent through feature importance).

5. **AdaBoost Regressor**: This ensemble boosting method combines weak decision trees for improved prediction accuracy. While simpler than the others, it can still learn non-linear relationships by combining weak learners (decision trees). It can be particularly suitable if interpretability is a high priority due to its simpler structure.

6. **Linear Regression**: Though not chosen due to its limitations with non-linearity, can be a good baseline model. It provides a simple and interpretable model, but its accuracy might be lower compared to the ensemble methods on this dataset.

6.2 Model Evaluation and Selection

After preprocessing the data, the data was divided into input features (X) representing the circuit design and the target variable (y) representing leakage power. The models were trained on a portion of the data and evaluated on a separate hold-out test set (test size = 0.3). R-squared (R^2) scores were used to assess the proportion of variance in leakage power explained by each model. Mean Squared Error (MSE) was calculated to measure the average squared difference between predicted and actual leakage power values. The results are as follows:

Model	**Mean Squared Error**	**R2 Score**
Linear Regression	1.015×10^{-16}	0.1747
AdaBoost	1.364×10^{-16}	-0.1087
LightGBM	2.237×10^{-18}	0.9818
CatBoost	5.269×10^{-19}	0.9957
Random Forest	1.230×10^{-16}	-1.76×10^{-6}
XGBoost	1.230×10^{-16}	-1.842×10^{-6}

Table 2: Model Performance Before Transformation

Based on the R^2 scores, LightGBM, CatBoost, AdaBoost, and Linear Regression were chosen for further processing to predict leakage power in circuit design.

6.3 Transformations on Data

6.3.1 PCA Transformation

Principal Component Analysis (PCA) identifies a smaller set of features (principal components) that capture most of the variance in the data. It is benefitial when there's a high degree of correlation between features. However, for leakage power prediction, non-linear relationships between features are important. PCA can discard such information, potentially harming the performance of models like LightGBM and CatBoost that excel at capturing these non-linearities. Linear Regression and AdaBoost, which struggle with non-linearity, might see some improvement from PCA as it focuses on capturing linear relationships.

Results

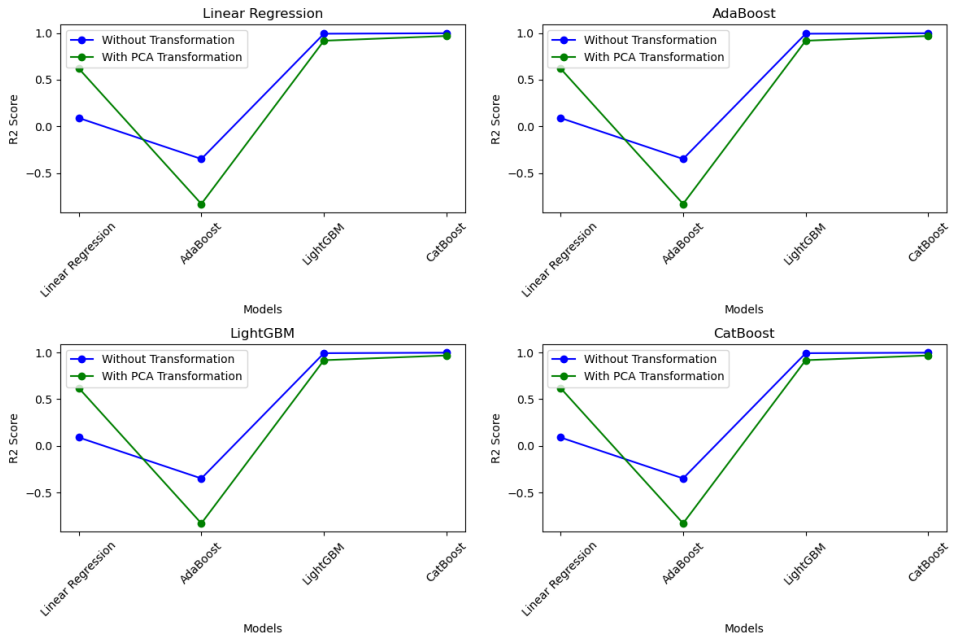

Figure 13: Model Performance after PCA Transformation

Model	Mean Squared Error	R2 Score
Linear Regression	5.759×10^{-17}	0.5318
AdaBoost	1.527×10^{-16}	-0.24096
LightGBM	1.761×10^{-17}	0.8569
CatBoost	1.125×10^{-17}	0.9086

Table 3: Model Performance after PCA Transformation

Inferences

- PCA excels at capturing linear relationships in the data. Leakage power has a primarily linear relationship with the original features, and PCA has improved only linear regression's performance because it works under the assumption of linearity in the data.
- Other models like AdaBoost, LightGBM, and CatBoost are more flexible and can capture non-linear relationships. PCA might discard some of this valuable non-linear information, leading to a decrease in R-squared score for these models.

6.3.2 Standard Scaling

This technique aims to normalize the distribution of features in the data. Scaling brings them to a common range (usually around -1 or 1) allowing the model to focus on the relative importance of each feature rather than their absolute values.

The features have significantly different scales. So scaling ensures that models like LightGBM and CatBoost that are less sensitive to scaling don't get biased towards features with larger values. Linear Regression and AdaBoost, which are more sensitive to scaling, can benefit from this normalization.

Results

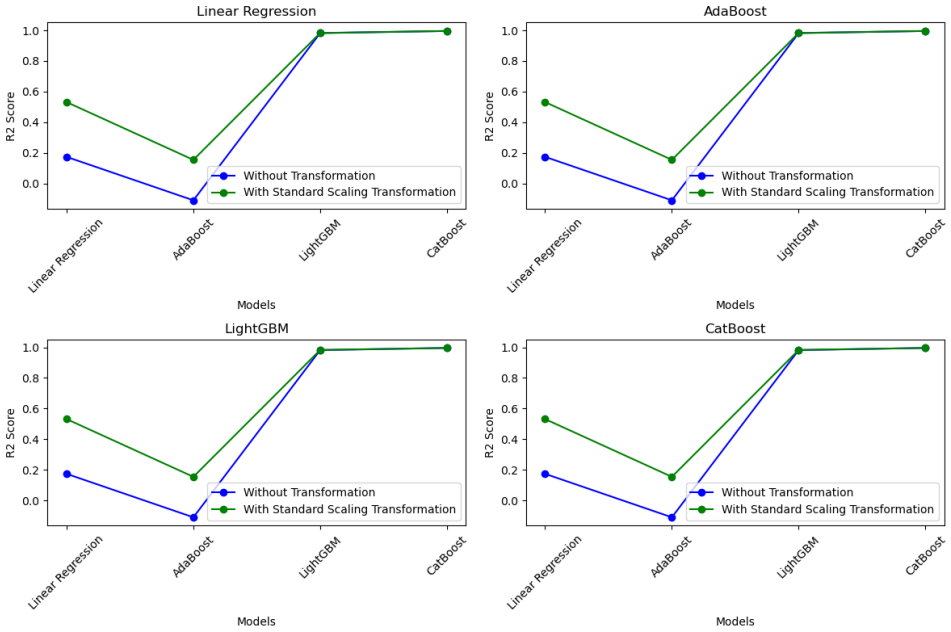

Figure 14: Model analysis after Standard Scaling

Model	**Mean Squared Error**	**R2 Score**
Linear Regression	5.759×10^{-17}	0.5318
AdaBoost	1.039×10^{-16}	0.1552
LightGBM	2.142×10^{-18}	0.9826
CatBoost	5.201×10^{-19}	0.9958

Table 4: Model Performance After Standard Scaling

Inferences

- Scaling has improved R^2 score of all models. This means the original features had very different scales. This makes the models focus on the relationships between features, not just their size, leading to overall better performance.

6.3.3 Feature Engineering

Feature engineering involves creating new features from existing ones based on domain knowledge. This helps models learn complex patterns without

relying solely on the original features.

We identify important features (e.g., temperature, voltage) and their relationships (linear, non-linear, logarithmic, etc) through techniques like importance scores. By applying different mathematical transformations to these features, we create new features.

Temperature, toxeref_n, VinA an VinB are the top 4 important features. By transforming features (like temperature with log) and creating interaction terms (voltage x temperature), we make complex relationships more learnable, boosting the accuracy of all models, especially those less suited for non-linearity.

Results

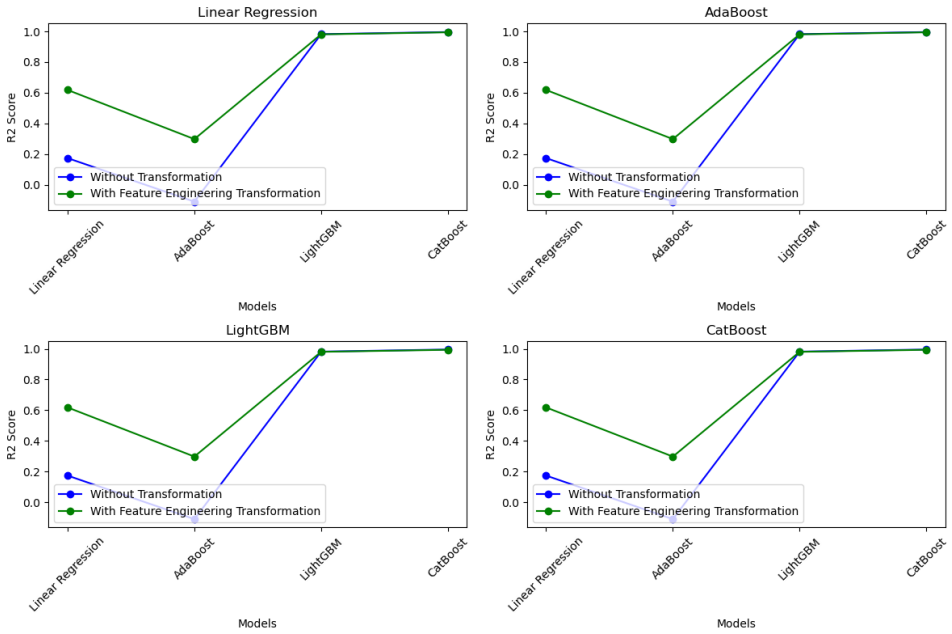

Figure 15: Model analysis after Feature Engineering

Model	Mean Squared Error	R2 Score
Linear Regression	4.407×10^{-17}	0.6184
AdaBoost	8.101×10^{-17}	0.2985
LightGBM	2.238×10^{-18}	0.9806
CatBoost	7.013×10^{-19}	0.9939

Table 5: Model Performance

We apply further transformations and create new features. Now we consider oxide layer thickness as well (2nd highest importance score) and different mathematical transformations like powers, and reciprocal.

Results

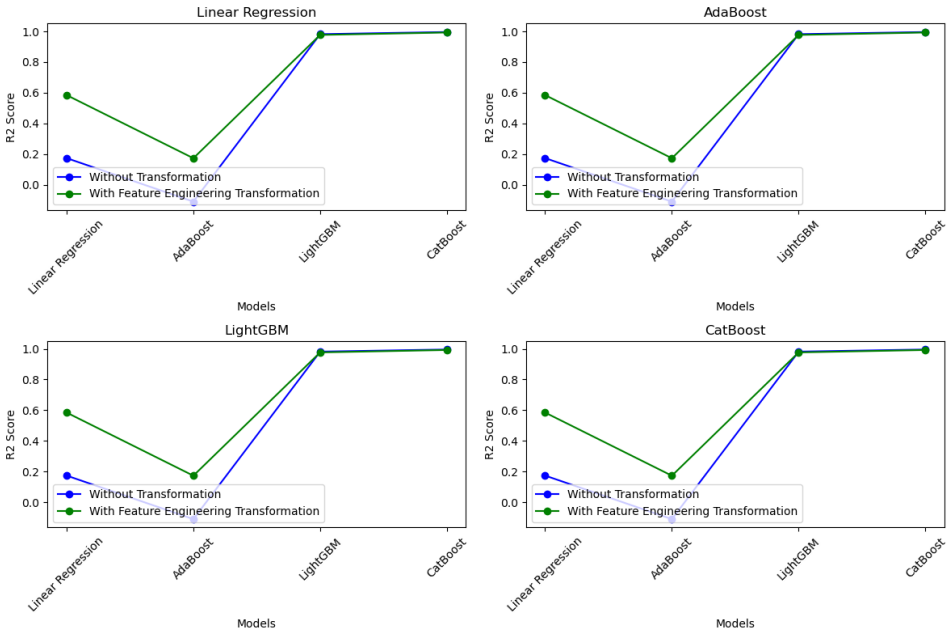

Figure 16: Model analysis after Feature Engineering 2

Model	Mean Squared Error	R2 Score
Linear Regression	4.792×10^{-17}	0.5850
AdaBoost	9.539×10^{-17}	0.1740
LightGBM	2.671×10^{-18}	0.9769
CatBoost	8.449×10^{-19}	0.9927

Table 6: Model Performance Feature Engineering 2

Inferences

- LightGBM and CatBoost are decision tree-based algorithms that can inherently handle non-linear relationships between features and the target variable. The transformations applied might have already been captured to some extent by their internal decision tree structures.
- The transformation have helped linear regression and AdaBoost learn the non-linearity.

6.3.4 Quantile Transform

This transformation scales features to a specific range (often 0-1) based on quantiles. It preserves the distribution of the data more closely compared to scaling techniques like StandardScaler.

Results

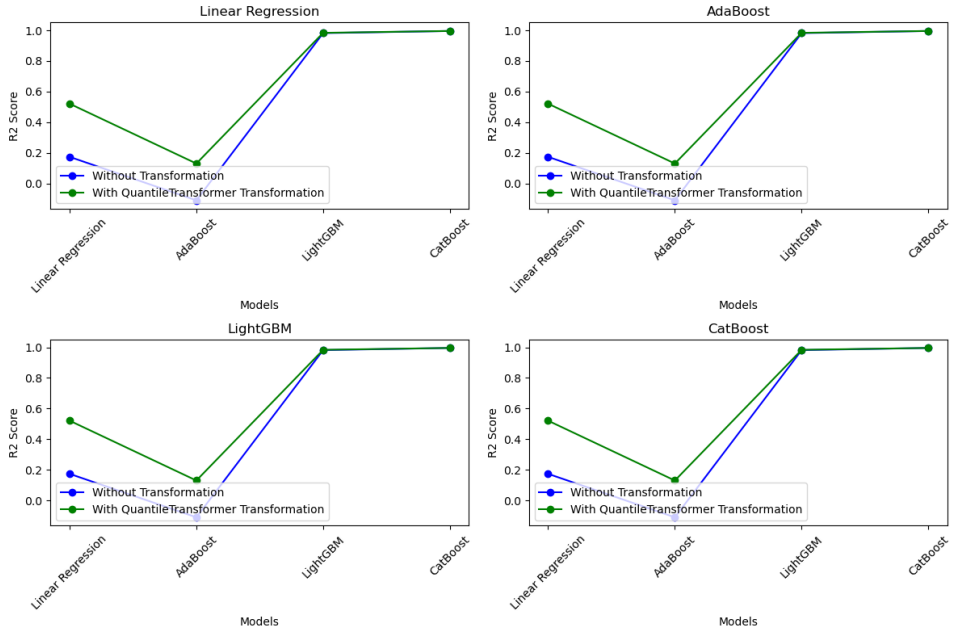

Figure 17: Model analysis after Quantile Transform

Model	**Mean Squared Error**	**R2 Score**
Linear Regression	5.888×10^{-17}	0.5214
AdaBoost	1.069×10^{-16}	0.1311
LightGBM	2.025×10^{-18}	0.9835
CatBoost	5.201×10^{-19}	0.9958

Table 7: Model Performance after Quantile Transform

Inferences

- THe transformation has improved R^2 score of the models. This means the original features had different scales. This makes the models focus on the relationships between features, not just their size, leading to overall better performance.

6.3.5 Yeo Johnson's Transformation

Yeo Johnson's transformation technique can be applied to zero and negative values as well. So we apply this to scaled data. An advanced form of Box

Cox transformation technique - boxcox cannot be applied because 0 can also be one of the values in the dataset, but only possible values are allowed for this transformation.

Results

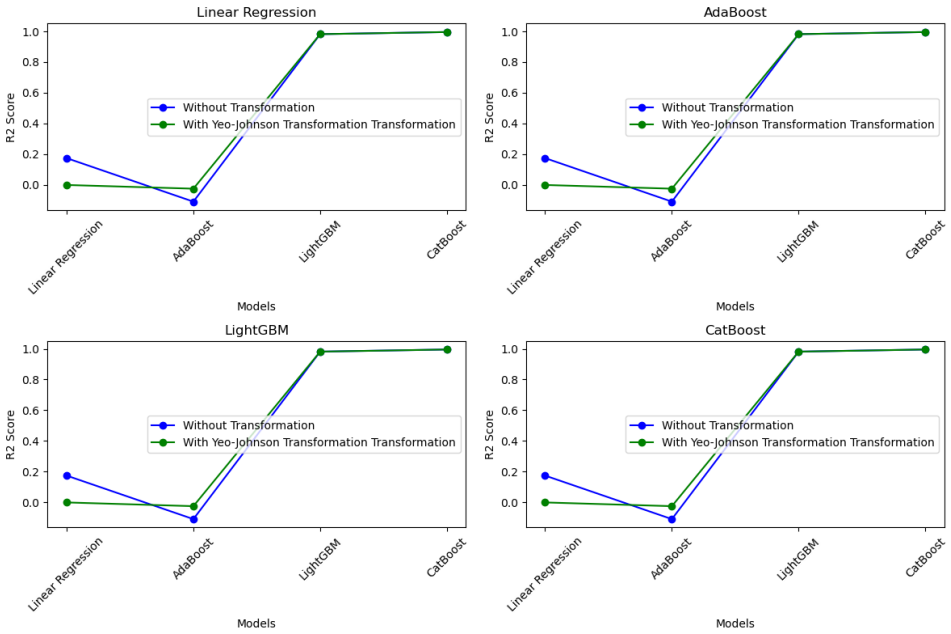

Figure 18: Model analysis after Yeo Johnson's Transformation

Model	**Mean Squared Error**	**R2 Score**
Linear Regression	1.230×10^{-16}	-0.00019
AdaBoost	1.260×10^{-16}	-0.0244
LightGBM	2.237×10^{-18}	0.9818
CatBoost	5.301×10^{-19}	0.9957

Table 8: Model Performance after Yeo Johnson's Transformation

Inferences

- Transformations might not significantly improve R^2 scores when the feature-target relationships are already close to linear. In some cases,

transformations can even introduce non-linearity that the model can't handle, leading to lower R^2.

- The Yeo-Johnson transformation might not have a large impact if AdaBoost learned the relevant patterns from the original features. However, if it uncovered subtle non-linearity, it could have benefitted AdaBoost.

6.4 Final Results and Inferences

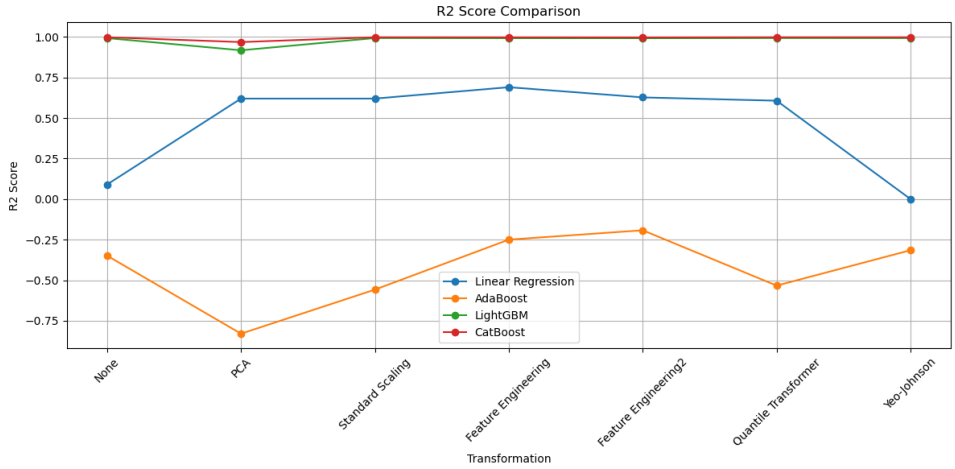

Figure 19: R2 Score Comparision between Tranformations

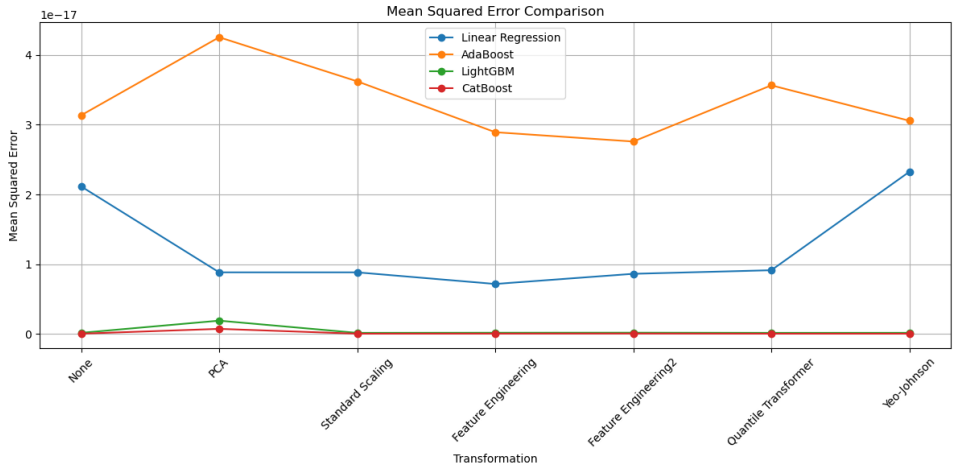

Figure 20: Enter Caption

Model	**Best R2 Score**	**Best Transformation**
Linear Regression	0.6900	Feature Engineering
AdaBoost	-0.1919	Feature Engineering2
LightGBM	0.9929	PCA
CatBoost	0.9979	Yeo-Johnson

Table 9: Best R2 Scores and Transformations

- LightGBM and CatBoost outperformed Linear Regression and AdaBoost significantly. Their R^2 scores (0.993 and 0.998, respectively) indicate their effectiveness in capturing complex non-linear relationships between features and leakage power.
- Feature engineering benefited both Linear Regression and AdaBoost the most. By transforming features (e.g., logarithm) and creating interaction terms, feature engineering improved their R^2 scores (Linear Regression: 0.690, AdaBoost: recovered from negative value). This suggests the original features lacked some information crucial for these models.
- PCA yielded the best results for LightGBM, suggesting it benefited from dimensionality reduction while still capturing relevant information. In contrast, Yeo-Johnson transformation boosted CatBoost, potentially improving the distribution of its features for better learning.

Overall, the results highlight the importance of both model selection and data transformation for achieving optimal machine learning performance.

7 Conclusion

In conclusion, this project has successfully addressed the critical challenge of predicting leakage power in CMOS circuits through the application of machine learning techniques. By leveraging comprehensive datasets and conducting in-depth exploratory data analytics, we have gained valuable insights into the intricate variations of leakage and delay across different technology nodes. The trained machine learning models have been used in predicting leakage power.

8 Acknowledgements

We would like to express our sincere gratitude towards our course instructor, Prof. Zia Abbas, for his valuable guidance, encouragement, and support throughout the project. We would also like to extend my heartfelt thanks to our TAs, who provided guidance, and helped us whenever we faced any difficulties. Their presence and support have been instrumental in shaping our project.

9 Contributions

- **Sankalp S. Bhat** - Netlist design, scripting, Dataset generation for all nodes, C499 design, Report
- **Sri Anvith Dosapati** - Netlist design, scripting, Dataset generation for all nodes, C499 design, Report
- **Shreeya Singh** - Meta Learning - Reptile Approach, EDA, Data Analysis, Report
- **Srujana Vanka** - Mathematical Transformations Approach, EDA, Data Analysis, Report