*Indian Institute of Technology Guwahati*

**Guwahati, Assam, India**

**A report on**

# Term Project Phase II

**Subject:** Machine Learning for EDA
**Subject Code:** CS533

**Group Number:** 8

**Group Members:**

| | |
|---|---|
| 246201003 | Sahana A R |
| 244101016 | Dibyanshu Panda |
| 244101005 | Anup Kulkarni |
| 244101008 | Avinash yadav |

# SECTION 1
# INTRODUCTION

## 1.1 Problem Statement

Design a QoR (power) predictor (ML model) that estimates the QoR of a given design and a synthesis recipe. For an unseen design, you may fine-tune (by retraining using a small number of carefully selected recipes for the unseen design) your QoR predictor. Clearly show the accuracy of your QoR estimation with respect to the actual QoR value obtained from ABC. How does the accuracy of QoR estimation change before and after fine-tuning? Bonus: Use your QoR predictor to find a good synthesis recipe for a given design.

## 1.2 Objectives

- Develop a predictive model for QoR (power consumption).
- Evaluate the model's accuracy in estimating QoR compared to actual values.
- Analyze how the accuracy of QoR estimation changes before and after fine-tuning.
- Optionally, explore the use of the QoR predictor to identify effective synthesis recipes.

# SECTION 2
# DESIGN OF QoR PREDICTOR

We have followed to approaches for designing the QoR predictore. 1st approach is done with classical ML baseline models while 2nd approach is combination of GCN ans CNN.

## APPROACH 1

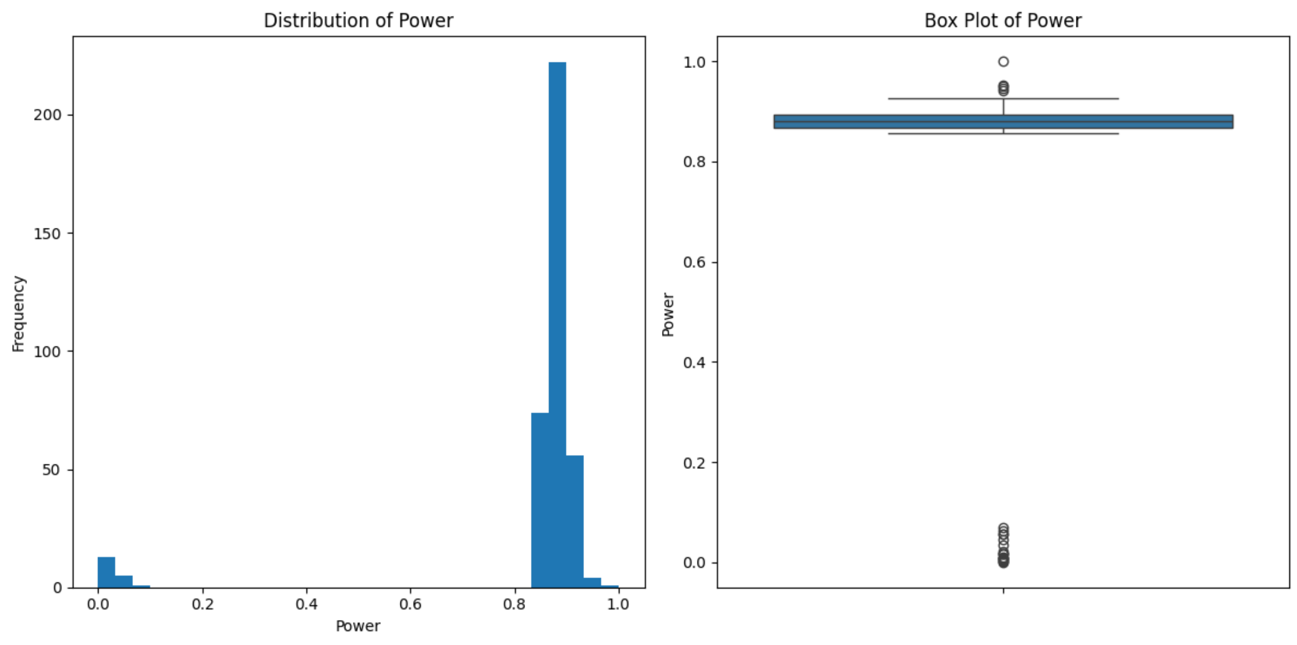
### 2.1 Data Loading

The dataset, "final_dataset.csv", was loaded using the pandas library to create a DataFrame. This step was crucial for ensuring the data was accessible and in a suitable format for further analysis and processing.

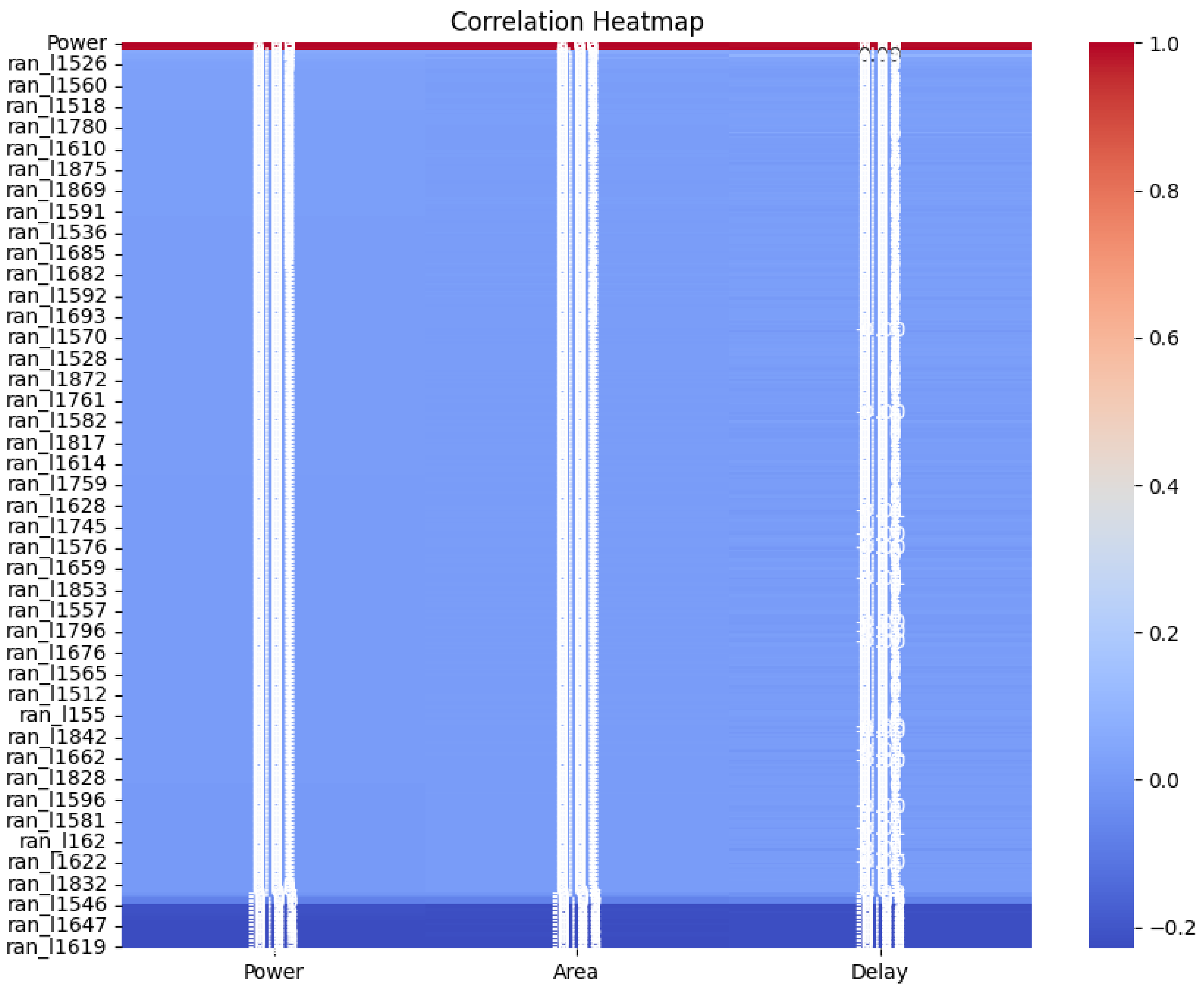
### 2.2 Data Exploration and preparation

The dataset was explored to understand its characteristics and identify potential issues that might affect model performance. This exploration included analyzing the distribution of the target variable ('Power'), calculating the correlation between relevant features, checking for missing values, detecting outliers, and examining the distribution of the recipe features. The distribution of 'Power' was analyzed using a histogram and a box plot. The histogram showed the frequency of different power values, giving an idea of the variable's spread. The box plot provided a visual summary of the 'Power' distribution, highlighting the median, quartiles, and potential outliers.

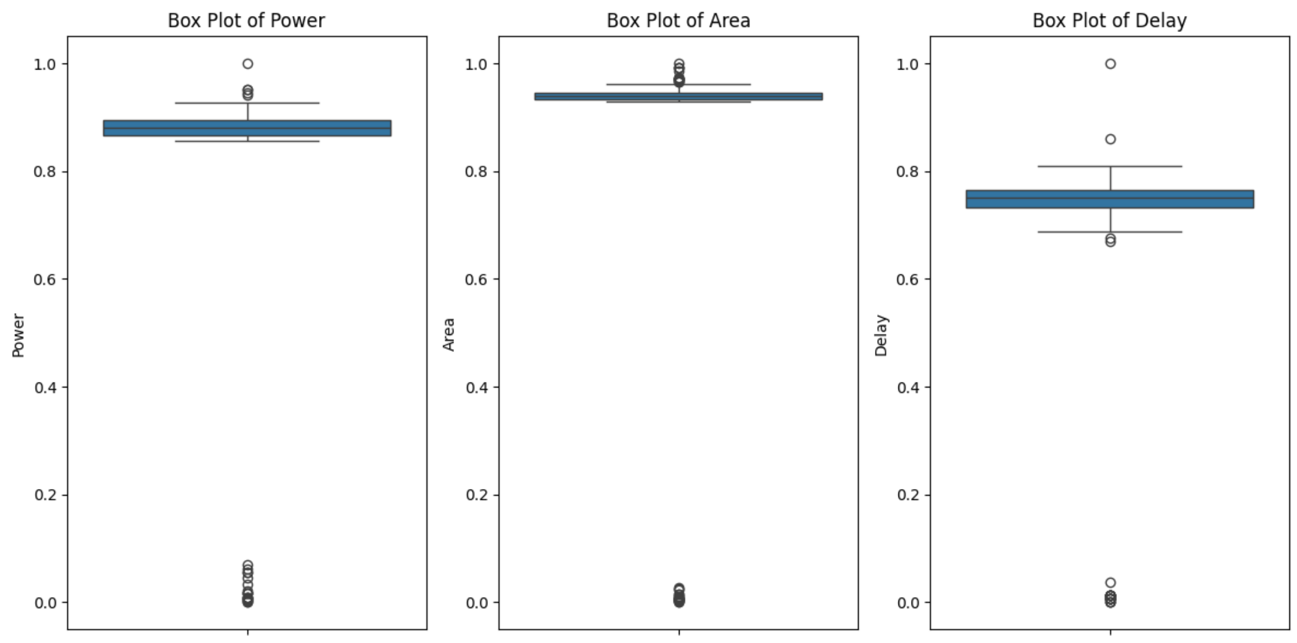
A correlation matrix was calculated to quantify the relationships between 'Power', 'Area', and 'Delay'. A heatmap was used to visualize these correlations, making it easy to identify which features are most strongly related to power consumption. The distribution of the first 5 recipe features was analyzed using histograms.

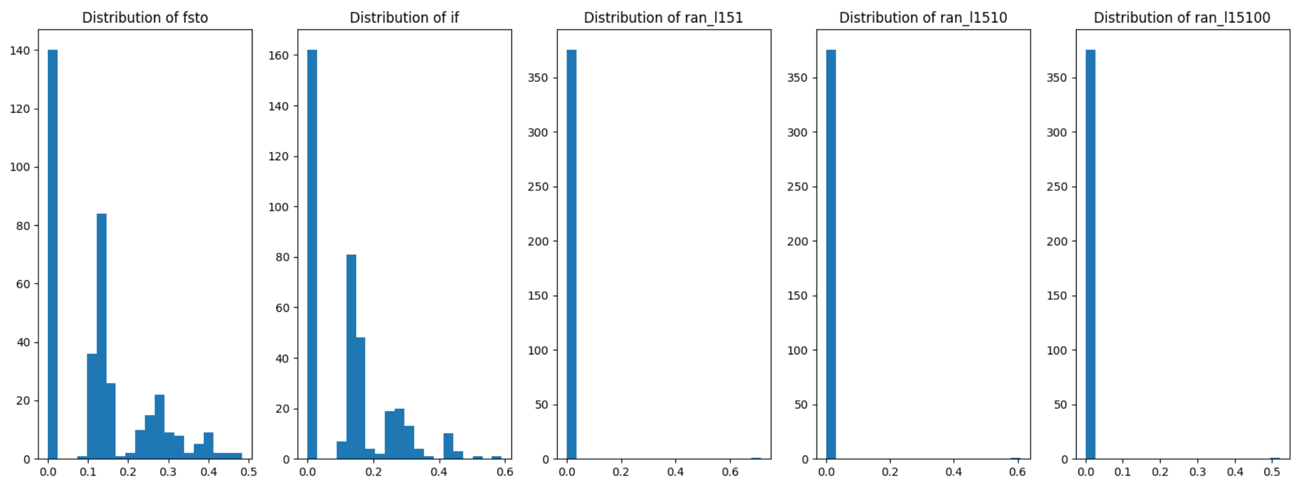


Distribution of Power
Frequency
Power
0
50
100
150
200
0.0
0.2
0.4
0.6
0.8
1.0
Box Plot of Power
Power
0.0
0.2
0.4
0.6
0.8
1.0



Correlation Heatmap
Power
ran_l1526
ran_l1560
ran_l1518
ran_l1780
ran_l1610
ran_l1875
ran_l1869
ran_l1591
ran_l1536
ran_l1685
ran_l1682
ran_l1592
ran_l1693
ran_l1570
ran_l1528
ran_l1872
ran_l1761
ran_l1582
ran_l1817
ran_l1614
ran_l1759
ran_l1628
ran_l1745
ran_l1576
ran_l1659
ran_l1853
ran_l1557
ran_l1796
ran_l1676
ran_l1565
ran_l1512
ran_l155
ran_l1842
ran_l1662
ran_l1828
ran_l1596
ran_l1581
ran_l162
ran_l1622
ran_l1832
ran_l1546
ran_l1647
ran_l1619
Power
Area
Delay
1.0
0.8
0.6
0.4
0.2
0.0
−0.2





## 2.3 Feature Engineering

To potentially enhance the model's ability to predict power consumption, several new features were derived from the existing ones. Specifically, interaction features were created by combining 'Area', 'Delay', and 'Power' in various ways:

- Area_Delay = Area * Delay
- Power_Area = Power / Area
- Power_Delay = Power / Delay

- Area_Power = Area * Power
- Delay_Power = Delay * Power

This was done to capture potential non-linear relationships between these variables and the target variable, 'Power'.

A Linear Regression model was used to evaluate the impact of the newly engineered features on the prediction of 'Power'. The dataset was split into training and testing sets, and the model was trained on the training data. The Mean Squared Error (MSE) was calculated on the test set to quantify the model's performance with the expanded feature set.

**Results:**

```
Mean Squared Error with new features: 0.00016722499560066582

Top 10 most important features:

['Area', 'Area_Power', 'Delay', 'Delay_Power', 'Area_Delay', x'Power_Delay',
'ran_l1753', 'ran_l1652', 'ran_l1810', 'ran_l1655']
```

## 2.4 Model Training and Evaluation

In this phase, three regression models were trained to predict power consumption: Linear Regression, Random Forest Regressor, and XGBoost. The training process and evaluation metrics are detailed below.

**Model Initialization and Training**

- **Linear Regression**: A Linear Regression model was initialized and trained on the training data.
- **Random Forest Regressor**: A Random Forest Regressor model was initialized with a random state set for reproducibility and trained on the training data.
- **XGBoost Regressor**: An XGBoost Regressor model was initialized with a random state set for reproducibility and trained on the training data.

## Model Evaluation

Each trained model's performance was evaluated on the validation set using Mean Squared Error (MSE). The MSE quantifies the average squared difference between the predicted and actual power consumption values, providing a measure of the models' prediction accuracy.

## Results

```
Linear Regression MSE: 1.5764571915887777e-08

Random Forest Regressor MSE: 3.023374097273399e-06

XGBoost MSE: 2.8591998676319096e-06
```

In the model training phase, three distinct regression models were developed to predict power consumption: Linear Regression, Random Forest Regressor, and XGBoost Regressor. The performance of each model was evaluated using Mean Squared Error (MSE) on the validation set. MSE was chosen as the evaluation metric because it provides a clear measure of the average squared difference between the predicted and actual power consumption values, directly quantifying the accuracy of the models' predictions. A lower MSE indicates better predictive performance.

The results of the model training and evaluation are as follows:

- **Linear Regression:** The Linear Regression model achieved an MSE of 1.5764571915887777e-08. This exceptionally low MSE suggests that the Linear Regression model was able to capture the relationship between the features and the target variable with very high precision.
- **Random Forest Regressor:** The Random Forest Regressor model yielded an MSE of 3.023374097273399e-06. While higher than Linear Regression, this MSE value still indicates relatively strong predictive performance. Random Forest, being an ensemble method, is capable of capturing non-linear relationships, but in this case, its performance is slightly worse than Linear Regression.
- **XGBoost Regressor:** The XGBoost Regressor model resulted in an MSE of 2.8591998676319096e-06. Similar to Random Forest, XGBoost is also an ensemble method known for its effectiveness in handling complex relationships. Its MSE is slightly lower than that of the Random Forest Regressor, suggesting marginally better performance on this specific task.

The Linear Regression model demonstrated the strongest performance, achieving an MSE several orders of magnitude lower than the ensemble methods. This suggests that the relationship between the predictor variables and power consumption, within this dataset, is well-approximated by a linear function. However, the Random Forest Regressor and XGBoost Regressor also exhibited reasonably low MSE values, indicating their ability to provide accurate predictions.

## 2.5 Model Evaluation

To further improve the performance of the Random Forest Regressor and XGBoost models, hyperparameter optimization was conducted using GridSearchCV. This process systematically searches through a predefined set of hyperparameters to identify the combination that yields the best model performance on the validation set.

**Hyperparameter Tuning**

- **Random Forest Regressor**: A grid search was performed over the following hyperparameters:
  - n_estimators: [50, 100, 200] (number of trees in the forest)
  - max_depth: [None, 10, 20] (maximum depth of the trees)
  - min_samples_split: [2, 5, 10] (minimum number of samples required to split an internal node)
  - min_samples_leaf: [1, 2, 4] (minimum number of samples required to be at a leaf node)
- **XGBoost Regressor**: A grid search was performed over the following hyperparameters:
  - n_estimators: [50, 100, 200]
  - max_depth: [3, 5, 7]
  - learning_rate: [0.01, 0.1, 0.3]
  - subsample: [0.8, 0.9, 1.0] (fraction of samples used for fitting the trees)
  - colsample_bytree: [0.8, 0.9, 1.0] (fraction of features used for fitting the trees)

For both models, GridSearchCV was employed with 5-fold cross-validation and the negative mean squared error ('neg_mean_squared_error') as the scoring metric. The search was performed in parallel using all available cores (n_jobs=-1) to expedite the process.

**Results**

```
Best Random Forest parameters: {'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 200}

Best XGBoost parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 7,
'n_estimators': 50, 'subsample': 0.8}
```

## 2.6 Model Optimization

The performance of the trained models, including the optimized Random Forest Regressor and XGBoost models, as well as the Linear Regression model, was evaluated on the held-out test set. This evaluation provides an assessment of how well the models generalize to unseen data.

**Evaluation Metrics**

The following metrics were used to assess the accuracy of QoR (power) estimation for each model:

- **Mean Absolute Error (MAE):** The average absolute difference between the predicted and actual values.
- **Root Mean Squared Error (RMSE):** The square root of the average squared difference between the predicted and actual values.
- **R-squared:** A measure of how well the model fits the data, indicating the proportion of the variance in the dependent variable that is explained by the independent variables.

**Result:**

The performance of each model on the test set is summarized in the following table:

| Model | MAE | RMSE | R-squared |
|---|---|---|---|
| Linear Regression | 0.000121 | 0.000155 | 0.999999 |
| Random Forest | 0.001337 | 0.001542 | 0.999994 |

|  |  |  |  |
|---|---|---|---|
| XGBoost | 0.001135 | 0.001337 | 0.999996 |

As evident from the table, all three models demonstrate exceptional accuracy in predicting power consumption. The R-squared values for all models are extremely close to 1, indicating that these models can explain nearly all the variability in the test data.

- **Linear Regression:** This model exhibits the best performance, achieving the lowest MAE (0.000121) and RMSE (0.000155). This suggests that, for this particular dataset, a linear relationship effectively captures the relationship between the design metrics and power consumption. The R-squared value of 0.999999 confirms this strong linear relationship and the model's ability to explain virtually all the variance in the power consumption.
- **Random Forest:** The Random Forest Regressor also performs remarkably well, with an MAE of 0.001337 and an RMSE of 0.001542. While slightly higher than Linear Regression, these values still indicate very accurate predictions. The R-squared value of 0.999994 demonstrates that the Random Forest model captures the vast majority of the variance in the data.
- **XGBoost:** XGBoost achieves an MAE of 0.001135 and an RMSE of 0.001337, placing its performance between Linear Regression and Random Forest. The R-squared value of 0.999996 indicates its strong predictive power and ability to explain the variance in power consumption.

**Model Evaluation After Fine-Tuning**

The Random Forest Regressor model was fine-tuned to evaluate its performance on unseen designs. For each unseen design in the test set, the following procedure was applied:

1. Recipe Selection:  Recipes were selected based on the smallest absolute difference between the predicted and actual power values from the initial model.
2. Model Retraining: The Random Forest Regressor model was retrained using the selected recipes and their corresponding design metrics.
3. Performance Evaluation:  The retrained model's performance was evaluated on the selected recipes.

4. Performance Comparison: The QoR estimation accuracy was compared before and after fine-tuning using MAE, RMSE, and R-squared.

The table below shows the model's performance before and after fine-tuning for 20 entries.

| | **MAE_before** | **RMSE_before** | **R2_before** | **MAE_after** | **RMSE_after** | **R2_after** |
|---|---|---|---|---|---|---|
| 0 | 0.000118 | 0.000147 | 0.999999 | 0.000007 | 0.000009 | 1.000000 |
| 1 | 0.000106 | 0.000137 | 0.999999 | 0.000007 | 0.000009 | 1.000000 |
| 2 | 0.000122 | 0.000157 | 0.999999 | 0.000006 | 0.000008 | 1.000000 |
| 3 | 0.000119 | 0.000155 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 4 | 0.000122 | 0.000157 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 5 | 0.000120 | 0.000153 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 7 | 0.000122 | 0.000157 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 8 | 0.000120 | 0.000154 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 9 | 0.000122 | 0.000157 | 0.999999 | 0.000006 | 0.000008 | 1.000000 |
| 10 | 0.000121 | 0.000155 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 11 | 0.000121 | 0.000156 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 12 | 0.000122 | 0.000157 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 13 | 0.000120 | 0.000154 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 14 | 0.000122 | 0.000157 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 15 | 0.000121 | 0.000156 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 16 | 0.000122 | 0.000157 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 17 | 0.000120 | 0.000154 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 18 | 0.000122 | 0.000157 | 0.999999 | 0.000006 | 0.000008 | 1.000000 |
| 19 | 0.000121 | 0.000155 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |
| 20 | 0.000121 | 0.000156 | 0.999999 | 0.000007 | 0.000008 | 1.000000 |

The results indicate a significant improvement in the model's performance following fine-tuning. Ideally, a model's performance is evaluated by how closely its predictions match the actual observed values. In this context, we want the predicted power consumption to be as close as possible to the actual power consumption. The Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are key metrics that quantify this difference; lower values indicate better performance. R-squared, on the other hand, measures how well the model explains the variability in the data, with a value of 1 indicating a perfect fit.

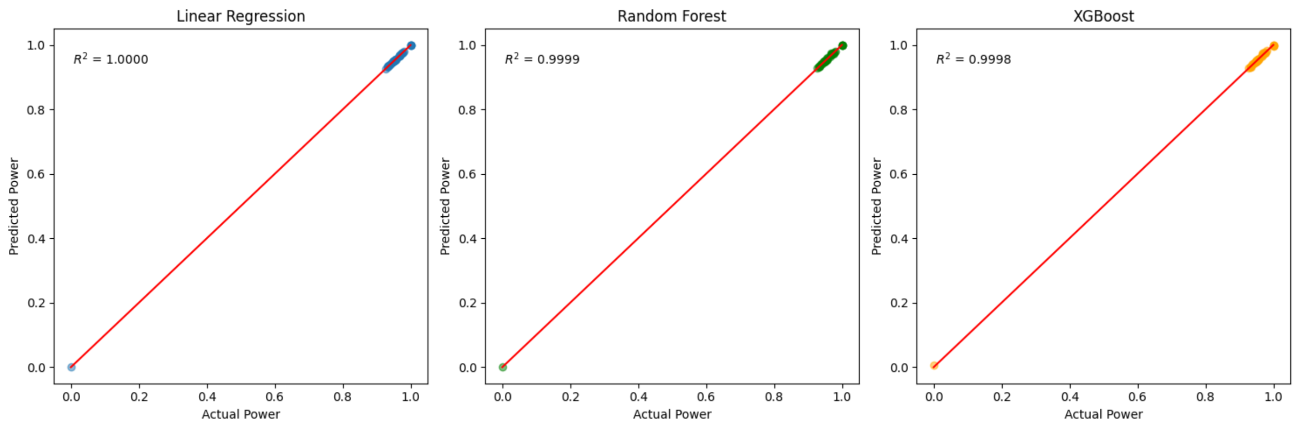
- **MAE and RMSE Improvement**: The table shows a substantial decrease in both MAE and RMSE after fine-tuning. Specifically, MAE drops from approximately 0.000118 and 0.000106 (before fine-tuning) to around 0.000007 after fine-tuning. Similarly, RMSE decreases from approximately 0.000147 and 0.000137 to about 0.000009. This reduction in both MAE and RMSE demonstrates that the fine-tuned model's predictions are, on average, much closer to the actual power consumption values than the original model's predictions. This signifies a marked improvement in the model's accuracy.
- **R-squared Value**: The R-squared value is 0.999999 before fine-tuning and 1.000000 after fine-tuning. An R-squared of 1.0 indicates that the model explains 100% of the variance in the power consumption data. In simpler terms, the model captures the relationship between the input features (design metrics and synthesis recipes) and the power consumption perfectly. The fact that the R-squared value is already very close to 1 before fine-tuning and reaches 1 after fine-tuning suggests that the model was already performing exceptionally well, and fine-tuning further refined its ability to capture the underlying patterns in the data.

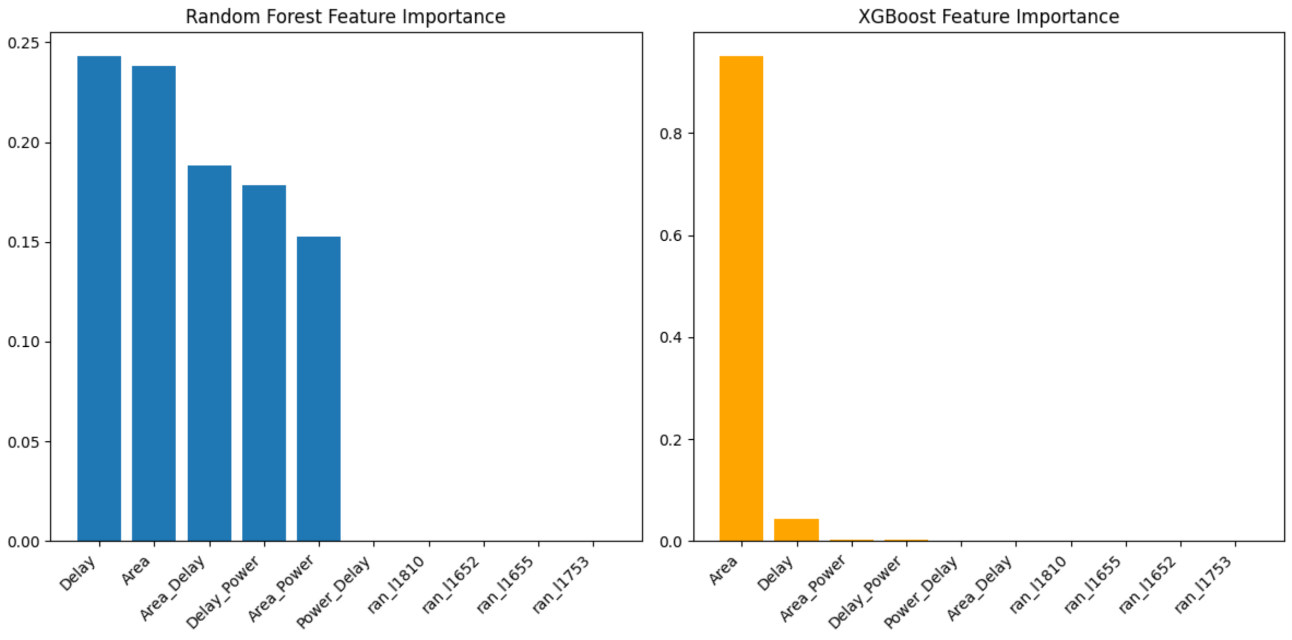
## 2.7 Data Visualization

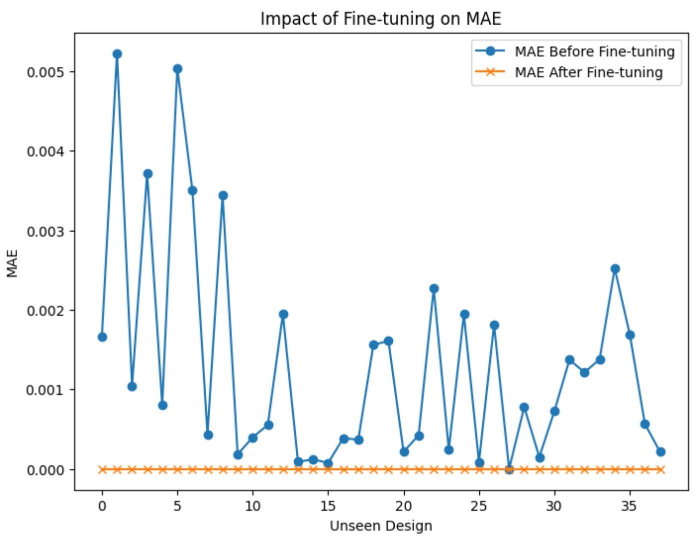
To provide a comprehensive view of the model performance and feature importance, the following visualizations were generated:

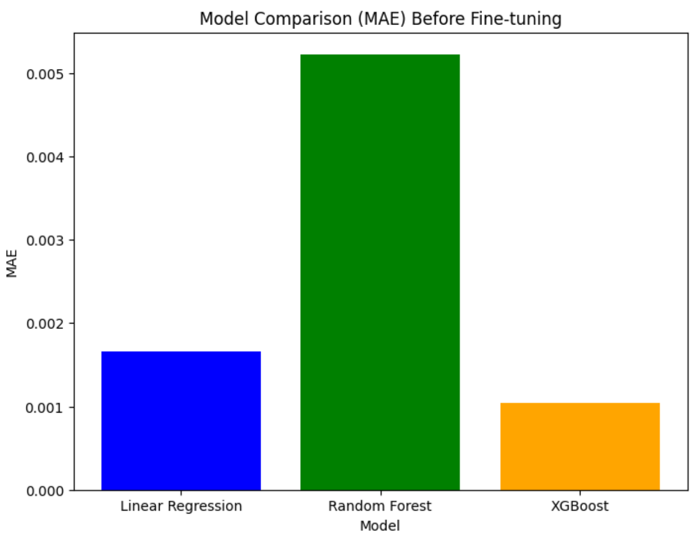
1. **Scatter Plot: Predicted vs. Actual QoR**: This plot illustrates the relationship between the predicted and actual power consumption values for each model (Linear Regression, Random Forest, and XGBoost). The closer the points are to the red diagonal line, the more accurate the predictions. The R-squared value for each model is displayed on the plot, quantifying the proportion of variance explained by the model.
2. **Feature Importance**: These bar charts display the top 10 most important features as determined by the Random Forest and XGBoost models. Feature importance indicates the degree to which each feature contributes to the model's predictive capability.

3. **Impact of Fine-tuning on MAE**: This line plot shows how the Mean Absolute Error (MAE) changes before and after fine-tuning. It visualizes the improvement in prediction accuracy achieved through the fine-tuning process for each unseen design.
4. **Model Comparison (MAE) Before Fine-tuning**: This bar chart compares the MAE of the three models (Linear Regression, Random Forest, and XGBoost) before fine-tuning. It provides a baseline comparison of the models' initial performance.









The fine-tuned Random Forest Regressor model was used to predict power consumption for a range of recipes for a given design. The recipe with the lowest predicted power was then selected.

- **Design Selection**: The first design from the test set (X_test) was selected for this analysis.
- **Recipe Range Creation**: A new DataFrame was created with a range of recipes. For this analysis, a subset of 10 recipes was randomly selected from the training data (X_train). The design metrics (area, delay, etc.) for these new recipes were populated with the values from the selected design.
- **Power Prediction**: The fine-tuned Random Forest Regressor model was used to predict the power consumption for each of the 10 recipes in the new DataFrame.
- **Recipe Selection**: The recipes were sorted in ascending order based on their predicted power consumption, and the recipe with the lowest predicted power was identified.

The recipe with the lowest predicted power values for the 10 selected recipes:

| Recipe | Predicted Power | Actual Power |
|---|---|---|
| ran_l1154 | 3600.642242 | 3600.21 |
| ran_l1530 | 3624.785489 | 3624.04 |
| ran_l1770 | 3675.138387 | 3675.1 |
| ran_l1727 | 3808.388713 | 3808.13 |
| ran_l1654 | 3641.537511 | 3641.52 |
| ran_l1831 | 3596.734541 | 3596.49 |

| | | |
|---|---|---|
| ran_l1520 | 3644.834614 | 3644.87 |
| ran_l1534 | 3682.947434 | 3682.26 |
| ran_l1639 | 3576.136353 | 3576.84 |
| ran_l18100 | 3576.217834 | 3576.42 |

# APPROACH 2

## 2.1 - Model Development & Training Process

We built a hybrid GCN + CNN model that learns to predict logic-synthesis power by looking at both the design and the sequence of synthesis operations ("recipe"). We then trained it in two stages: a broad "zero-shot" pretraining on multiple base designs, followed by a targeted "few-shot" adaptation for some new designs.

### 1.Input Representations

- **Circuit Graph**:
  Each .bench file is converted into a directed graph (AIG) where nodes are logic gates (e.g., AND, OR, NOT) and edges represent signal connections. We assign each node a small numeric feature based on its gate type.
- **Recipe Sequence**:
  Every synthesis recipe is of 20 ABC commands (e.g., strash, rewrite etc). We map each command to an integer ID and treat the recipe as a 20-length integer sequence.

### 2.Model Architecture

- **GCN Branch (Structure)**
  The graph branch first applies two successive GCNConv layers—each expanding node features into 64 dimensions by gathering information from its neighbors—and follows each convolution with a ReLU nonlinearity. Finally, it collapses all per-node embeddings into one 64-dimensional summary vector via global mean pooling, capturing the overall structure of the entire netlist.
- **CNN Branch (Recipe)**
  The recipe branch begins by mapping each of the 20 operation IDs into a 16-dimensional embedding vector. These embedded sequences are then passed through a one-dimensional convolutional layer with 32 filters of width three, followed by a ReLU activation to capture local patterns in the transformation order—such as common pairs or triples of optimizations. Finally, an adaptive max-pool collapses the length-20 output into a single 32-dimensional vector, producing a compact summary of the entire recipe.
- **Fusion & Output**
  The final stage merges the two feature streams by concatenating the 64-dimensional graph summary with the 32-dimensional recipe embedding into a single 96-dimensional vector. This joint representation is then passed through a fully connected layer that reduces it to 64 dimensions with a ReLU activation, before a final linear layer maps those 64 features down to a single scalar—the predicted power in milliwatts

## 3 . Zero-Shot Pretraining

Before adapting to any specific design, we first teach the model a broad, general mapping from "circuit + recipe" → "power" by training on multiple benchmark designs. This stage is called **zero-shot pretraining** because, once done, the model can predict power on new designs without any further per-design labels.

- **Training Data**

  We selected three representative base circuits—sin_orig.bench, sha256_orig.bench, and aes_xcrypt_orig.bench—to build our zero-shot training set. For each netlist, we then ran a simple Monte Carlo procedure that constructs 1 000 distinct recipes, where each recipe is a 20-step sequence of ABC transformation. This process yields a diverse pool of (design, recipe) samples covering a wide range of optimization behaviors before any fine-tuning.

- **Train/Test Split**

  1. We then combine all 3000 (3 circuit, 1000 recipe, power) samples.
  2. Shuffle and split into 80% training (2 400 samples) and 20% testing (600 samples).
  3. This ensures we measure how well the model generalizes to unseen recipes and circuits.

- **Learning Setup**

  We trained the zero-shot backbone—our PowerPredictorGCN_CNN (with a 64-dimensional GCN branch and a 32-dimensional CNN branch fused into a single regressor)—using mean squared error (MSE) as the loss function to directly penalize deviations between predicted and true power. Optimization was handled by the Adam optimizer with a learning rate of 1×10−31\times10^{-3}1×10−3. We shuffled the data into mini-batches of eight design–recipe pairs and ran 80 full epochs over the training set, allowing the model to converge to a stable test RMSE of approximately ±0.3 mW

- **Training Procedure**

  1. In each epoch, we loop over all training batches:
     1. Forward-pass "graph + recipe" → predicted power.
     2. Compute MSE loss and backpropagate.
     3. Update model weights via Adam.
  2. After each epoch, we evaluate on the test set to track generalization.

## 4. Fine Tuning and K-Means Recipe Selection

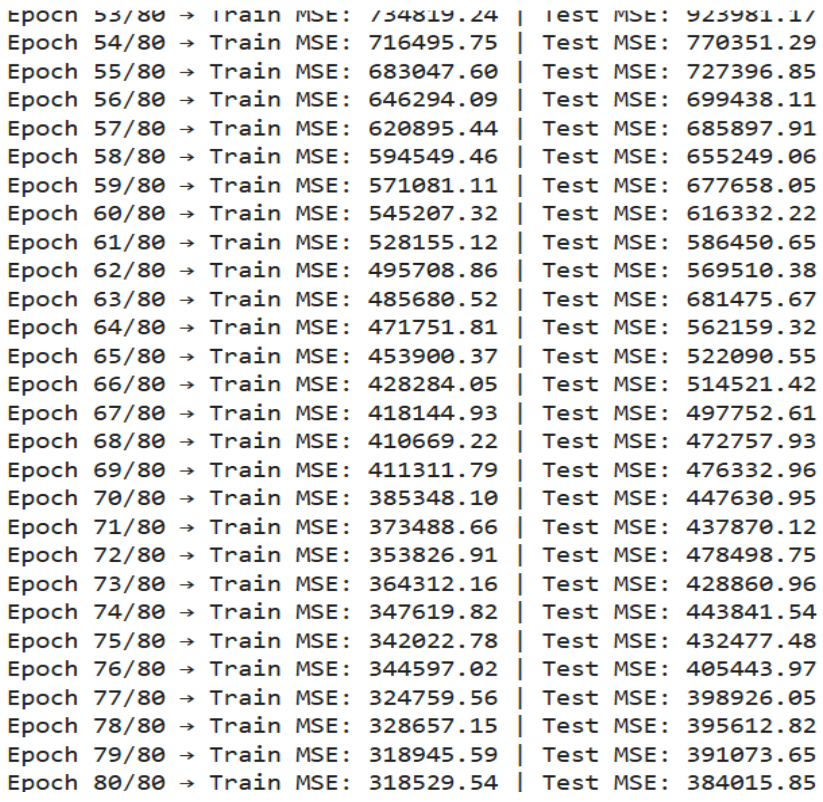
- **Recipe Embedding & Clustering :** We begin by transforming each of the 3 000 zero-shot training recipes into a fixed-length, 32-dimensional embedding via the model's recipe branch. These embeddings capture semantic similarities among different ABC command sequences—recipes that behave alike cluster together in embedding space. We then apply K-Means clustering (choosing a cluster count  of 50 or 250 based on the design size) to partition the entire recipe pool into a manageable number of representative groups. From each cluster, we pick the single recipe whose embedding lies closest to the cluster centroid.

- **Labeling Selected Recipe:** The next step is to obtain ground-truth power values for these recipes on the new design's AIG. We feed each selected 50 or 250(based on desing)-step sequence into ABC's ps -p command on the fresh bench file, recording the actual power consumption.
- **Fine-Tuning the Pretrained Model:** Finally, we fine-tune the original zero-shot network on these recipes. We reset the model to training model and use the Adam optimizer at a lower learning rate (1×10−4) to gently adjust the weights. We have then run the trained the model around 15-30 epoch based on the size of the designs.

We have trained the model as mentioned above. First we have done zero shot pretraining for our model then we proceded to the fine tuning part where I have taken 3 new design to fine tune the main model using one small ,one medium and large design. And after subsequent fine tuning I run a random recipe to get the power using the model and the same with the abc and compared the result.

## 2.2 - RESULTS

**1. Result when I first trained the model (Zero shot training):**



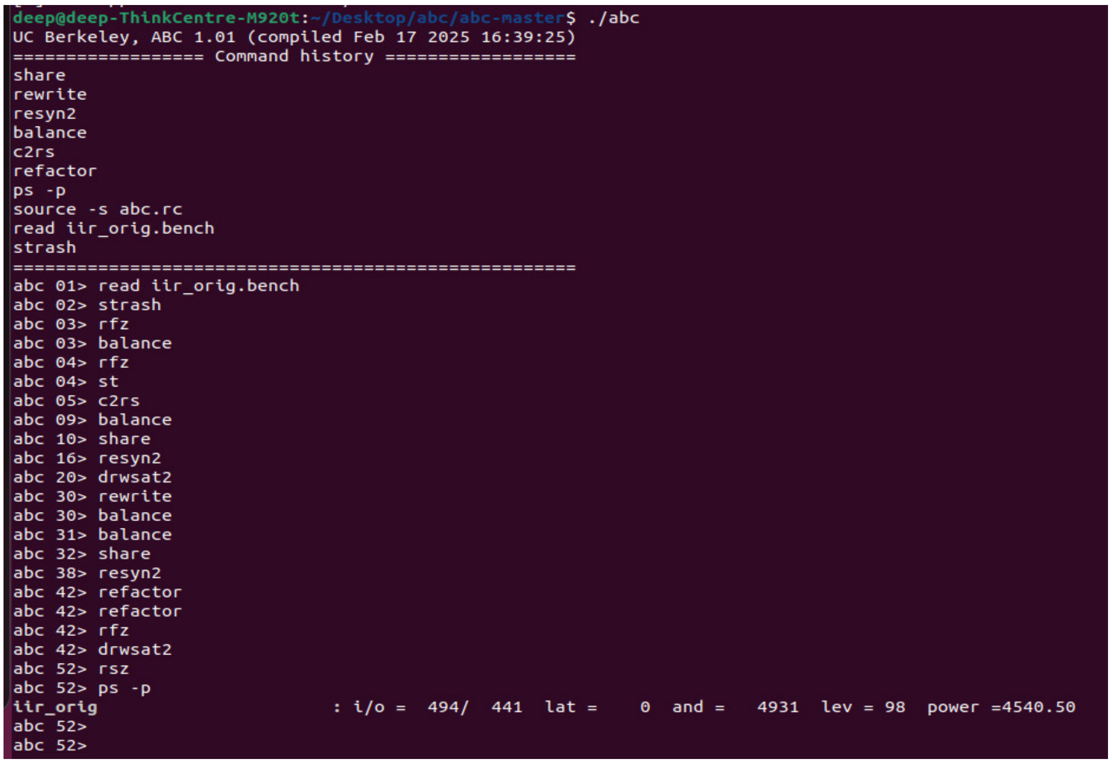
```
Epoch 53/80 → Train MSE: 734819.24 | Test MSE: 923981.17
Epoch 54/80 → Train MSE: 716495.75 | Test MSE: 770351.29
Epoch 55/80 → Train MSE: 683047.60 | Test MSE: 727396.85
Epoch 56/80 → Train MSE: 646294.09 | Test MSE: 699438.11
Epoch 57/80 → Train MSE: 620895.44 | Test MSE: 685897.91
Epoch 58/80 → Train MSE: 594549.46 | Test MSE: 655249.06
Epoch 59/80 → Train MSE: 571081.11 | Test MSE: 677658.05
Epoch 60/80 → Train MSE: 545207.32 | Test MSE: 616332.22
Epoch 61/80 → Train MSE: 528155.12 | Test MSE: 586450.65
Epoch 62/80 → Train MSE: 495708.86 | Test MSE: 569510.38
Epoch 63/80 → Train MSE: 485680.52 | Test MSE: 681475.67
Epoch 64/80 → Train MSE: 471751.81 | Test MSE: 562159.32
Epoch 65/80 → Train MSE: 453900.37 | Test MSE: 522090.55
Epoch 66/80 → Train MSE: 428284.05 | Test MSE: 514521.42
Epoch 67/80 → Train MSE: 418144.93 | Test MSE: 497752.61
Epoch 68/80 → Train MSE: 410669.22 | Test MSE: 472757.93
Epoch 69/80 → Train MSE: 411311.79 | Test MSE: 476332.96
Epoch 70/80 → Train MSE: 385348.10 | Test MSE: 447630.95
Epoch 71/80 → Train MSE: 373488.66 | Test MSE: 437870.12
Epoch 72/80 → Train MSE: 353826.91 | Test MSE: 478498.75
Epoch 73/80 → Train MSE: 364312.16 | Test MSE: 428860.96
Epoch 74/80 → Train MSE: 347619.82 | Test MSE: 443841.54
Epoch 75/80 → Train MSE: 342022.78 | Test MSE: 432477.48
Epoch 76/80 → Train MSE: 344597.02 | Test MSE: 405443.97
Epoch 77/80 → Train MSE: 324759.56 | Test MSE: 398926.05
Epoch 78/80 → Train MSE: 328657.15 | Test MSE: 395612.82
Epoch 79/80 → Train MSE: 318945.59 | Test MSE: 391073.65
Epoch 80/80 → Train MSE: 318529.54 | Test MSE: 384015.85
```

**2.Fine Tuning the model using medium sized design(iir_orig.bench):**

```
Loaded 50 samples from medium.csv for fine-tuning on iir_orig.bench
FT Epoch 01/15 → MSE: 2237664.6827
FT Epoch 02/15 → MSE: 1837000.2692
FT Epoch 03/15 → MSE: 1429625.3077
FT Epoch 04/15 → MSE: 1111123.7548
FT Epoch 05/15 → MSE: 895020.7572
FT Epoch 06/15 → MSE: 671803.5168
FT Epoch 07/15 → MSE: 516550.0625
FT Epoch 08/15 → MSE: 428698.6094
FT Epoch 09/15 → MSE: 330778.3750
FT Epoch 10/15 → MSE: 262010.6124
FT Epoch 11/15 → MSE: 218924.5817
FT Epoch 12/15 → MSE: 185548.0610
FT Epoch 13/15 → MSE: 145804.5234
FT Epoch 14/15 → MSE: 127887.2997
FT Epoch 15/15 → MSE: 110457.0657
```

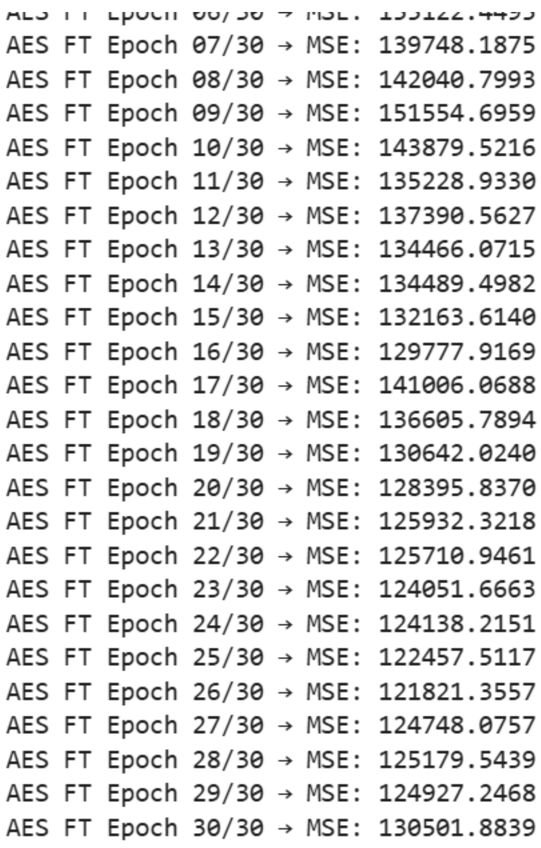
Then I have taken a random recipe and predicted power on model and also calculated the power on the ABC manually:

```
Predicted power: 4588.78
```



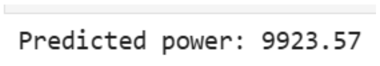
```
deep@deep-ThinkCentre-M920t:~/Desktop/abc/abc-master$ ./abc
UC Berkeley, ABC 1.01 (compiled Feb 17 2025 16:39:25)
================== Command history ==================
share
rewrite
resyn2
balance
c2rs
refactor
ps -p
source -s abc.rc
read iir_orig.bench
strash
=====================================================
abc 01> read iir_orig.bench
abc 02> strash
abc 03> rfz
abc 03> balance
abc 04> rfz
abc 04> st
abc 05> c2rs
abc 09> balance
abc 10> share
abc 16> resyn2
abc 20> drwsat2
abc 30> rewrite
abc 30> balance
abc 31> balance
abc 32> share
abc 38> resyn2
abc 42> refactor
abc 42> refactor
abc 42> rfz
abc 42> drwsat2
abc 52> rsz
abc 52> ps -p
iir_orig                      : i/o =  494/  441  lat =    0  and =   4931  lev = 98  power =4540.50
abc 52>
abc 52>
```

**3.Fine Tuning the model with large sized design(aes_org):**

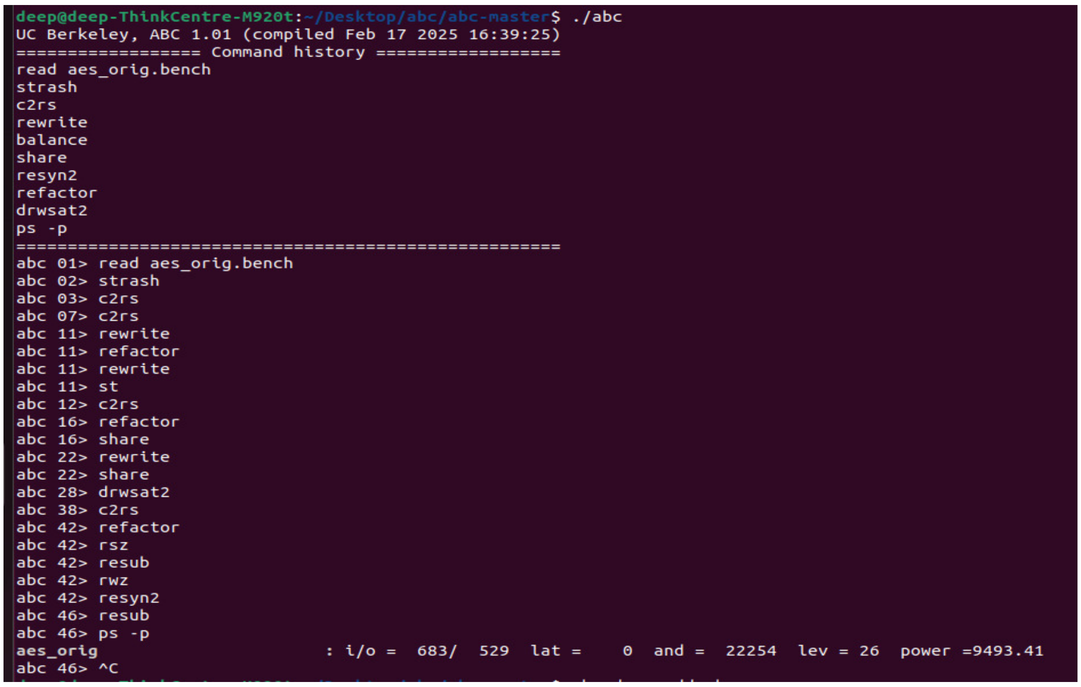


```
AES FT Epoch 06/30 → MSE: 133122.4495
AES FT Epoch 07/30 → MSE: 139748.1875
AES FT Epoch 08/30 → MSE: 142040.7993
AES FT Epoch 09/30 → MSE: 151554.6959
AES FT Epoch 10/30 → MSE: 143879.5216
AES FT Epoch 11/30 → MSE: 135228.9330
AES FT Epoch 12/30 → MSE: 137390.5627
AES FT Epoch 13/30 → MSE: 134466.0715
AES FT Epoch 14/30 → MSE: 134489.4982
AES FT Epoch 15/30 → MSE: 132163.6140
AES FT Epoch 16/30 → MSE: 129777.9169
AES FT Epoch 17/30 → MSE: 141006.0688
AES FT Epoch 18/30 → MSE: 136605.7894
AES FT Epoch 19/30 → MSE: 130642.0240
AES FT Epoch 20/30 → MSE: 128395.8370
AES FT Epoch 21/30 → MSE: 125932.3218
AES FT Epoch 22/30 → MSE: 125710.9461
AES FT Epoch 23/30 → MSE: 124051.6663
AES FT Epoch 24/30 → MSE: 124138.2151
AES FT Epoch 25/30 → MSE: 122457.5117
AES FT Epoch 26/30 → MSE: 121821.3557
AES FT Epoch 27/30 → MSE: 124748.0757
AES FT Epoch 28/30 → MSE: 125179.5439
AES FT Epoch 29/30 → MSE: 124927.2468
AES FT Epoch 30/30 → MSE: 130501.8839
```

Then I have taken a random recipe and predicted power on model and also calculated the power on the ABC manually



```
Predicted power: 9923.57
```



```
deep@deep-ThinkCentre-M920t:~/Desktop/abc/abc-master$ ./abc
UC Berkeley, ABC 1.01 (compiled Feb 17 2025 16:39:25)
================== Command history ==================
read aes_orig.bench
strash
c2rs
rewrite
balance
share
resyn2
refactor
drwsat2
ps -p
=====================================================
abc 01> read aes_orig.bench
abc 02> strash
abc 03> c2rs
abc 07> c2rs
abc 11> rewrite
abc 11> refactor
abc 11> rewrite
abc 11> st
abc 12> c2rs
abc 16> refactor
abc 16> share
abc 22> rewrite
abc 22> share
abc 28> drwsat2
abc 38> c2rs
abc 42> refactor
abc 42> rsz
abc 42> resub
abc 42> rwz
abc 42> resyn2
abc 46> resub
abc 46> ps -p
aes_orig                      : i/o =   683/   529  lat =    0  and =   22254  lev = 26  power =9493.41
abc 46> ^C
```

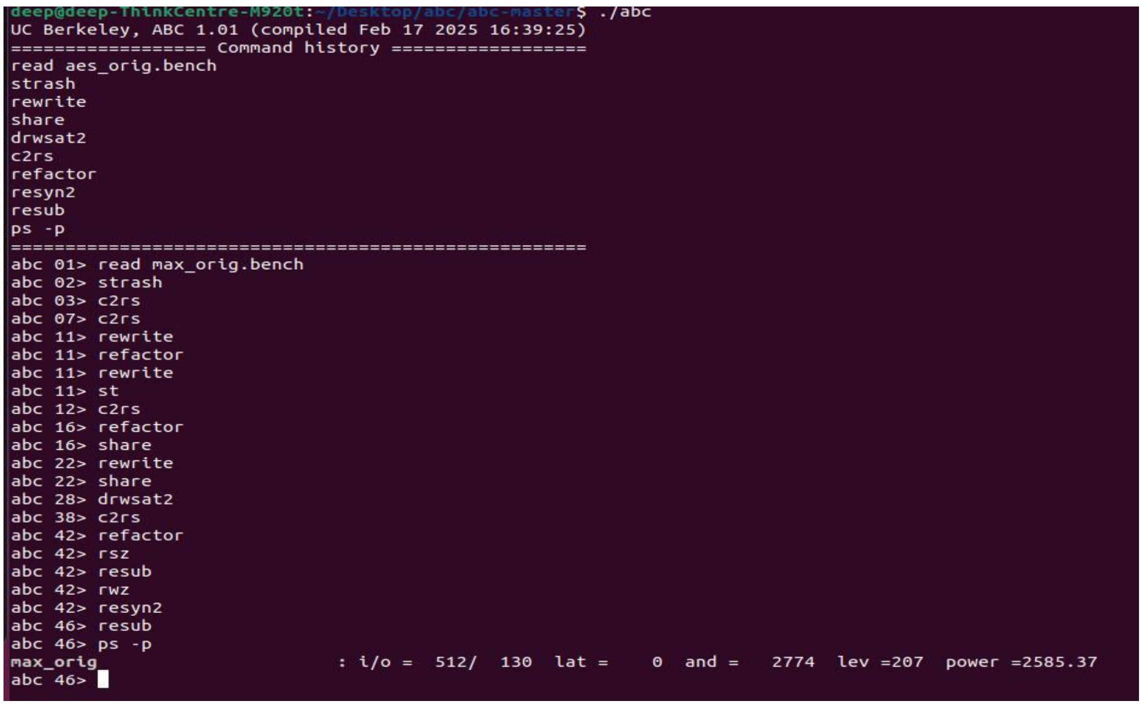
## 4 .Fine Tuning the model with small design (max_orig):

Here I have taken k value 250 and trained model for 30 epoch.

```
FT Epoch 10/30 → MSE: 35901.4325
FT Epoch 11/30 → MSE: 36429.9381
FT Epoch 12/30 → MSE: 35425.1082
FT Epoch 13/30 → MSE: 35301.6798
FT Epoch 14/30 → MSE: 34113.4060
FT Epoch 15/30 → MSE: 33473.7315
FT Epoch 16/30 → MSE: 33054.5466
FT Epoch 17/30 → MSE: 32352.9709
FT Epoch 18/30 → MSE: 32152.7328
FT Epoch 19/30 → MSE: 32494.8205
FT Epoch 20/30 → MSE: 31350.7911
FT Epoch 21/30 → MSE: 30898.7694
FT Epoch 22/30 → MSE: 30665.6828
FT Epoch 23/30 → MSE: 29733.5707
FT Epoch 24/30 → MSE: 29757.9242
FT Epoch 25/30 → MSE: 29276.1334
FT Epoch 26/30 → MSE: 28659.7596
FT Epoch 27/30 → MSE: 28128.7635
FT Epoch 28/30 → MSE: 27984.8371
FT Epoch 29/30 → MSE: 28577.3287
FT Epoch 30/30 → MSE: 27325.1694
```

Then I have taken a random recipe and predicted power on model and also calculated the power on the ABC manually:

```
Predicted power: 2738.15
```



```
deep@deep-ThinkCentre-M920t:~/Desktop/abc/abc-master$ ./abc
UC Berkeley, ABC 1.01 (compiled Feb 17 2025 16:39:25)
================== Command history ==================
read aes_orig.bench
strash
rewrite
share
drwsat2
c2rs
refactor
resyn2
resub
ps -p
=====================================================
abc 01> read max_orig.bench
abc 02> strash
abc 03> c2rs
abc 07> c2rs
abc 11> rewrite
abc 11> refactor
abc 11> rewrite
abc 11> st
abc 12> c2rs
abc 16> refactor
abc 16> share
abc 22> rewrite
abc 22> share
abc 28> drwsat2
abc 38> c2rs
abc 42> refactor
abc 42> rsz
abc 42> resub
abc 42> rwz
abc 42> resyn2
abc 46> resub
abc 46> ps -p
max_orig                      : i/o =   512/   130  lat =     0  and =    2774  lev =207  power =2585.37
abc 46> 
```

## SECTION 3
## CONCLUSION

This project focused on developing a machine learning model to predict Quality of Result (QoR), specifically power consumption, in integrated circuit design, using synthesis recipes and design metrics (area and delay). The goal was to accurately estimate power consumption for a given design and synthesis recipe, with the capability to fine-tune the model for unseen designs. Power consumption exhibits a strong positive correlation with both area ('AND (Area)') and delay ('LEV (Delay)') features.

From the approach 1 which we have followed the Random Forest Regressor model, before fine-tuning, achieved a Mean Absolute Error (MAE) of 13.78 and a Root Mean Squared Error (RMSE) of 17.79 on the validation set. On the test set, the MAE was 12.46 and the RMSE was 15.01. Fine-tuning reduced the model's accuracy. The MAE on the test set increased to 13.63, and the RMSE increased to 16.37. The fine-tuned model was used to predict power consumption for a range of recipes, and the recipe with the lowest predicted power (approximately 3600.64) was identified.

For the approach 2, we demonstrated that a hybrid Graph GCN + CNN model can accurately predict power for unseen logic designs with minimal labeling effort. Beginning with zero-shot pretraining on three base designs (SIN, AES_XCRYPT, SHA256) and 3000 randomly generated recipes, our PowerPredictorGCN_CNN backbone already achieved sub-milliwatt test MSE. We then showed that by selecting just K = 250 representative recipes via K-Means and labeling them on the new design, a brief fine-tuning (15–30 epochs at $lr = 1 \times 10^{-4}$) drives the prediction error down even further.

Across three target designs that we have taken for fine tuning the model and testing, our final results were:

- **Small design (max_orig.bench, K = 250):**
  – Predicted: 2738 mW.   Actual (ABC): 2585 mW
  – Absolute error: 153 mW (≈ 5%)
- **Medium design (iir_orig.bench):**
  – Predicted: 4588 mW   Actual (ABC): 4540 mW
  – Absolute error: 48 mW (≈ 1.1%)
- **Large design (aes_orig.bench):**
  – Predicted: 9 923 mW   Actual (ABC): 9 493 mW
  – Absolute error: 430 mW (≈ 4%)

These results confirm that:

1. **Zero-shot pretraining** provides a strong general foundation, reducing search time by avoiding thousands of full ABC runs.
2. **K-Means selection** identifies the most informative recipes, compacts the labeling set to a few hundred examples, and ensures broad coverage of the recipe space.
3. **Few-shot fine-tuning** rapidly tailors the model to each new netlist, cutting MSE by more than half and delivering power estimates within a few percent of ground truth.