Machine Learning-Based Load Estimation Model for A Research Office

MOK JIA CHENG

UNIVERSITI TUNKU ABDUL RAHMAN

Machine Learning-Based Load Estimation Model for A Research Office

MOK JIA CHENG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Electrical and Electronic Engineering with Honours

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

September 2024

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : MOK JIA CHENG

ID No. : 1906647

Date : 18 September 2024

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"Machine Learning-Based Load Estimation Model for A Research Office"** was prepared by **MOK JIA CHENG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Electrical & Electronic Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Ir. Dr. Wong JianHui

Date : 18 September 2024

Signature :

Co-Supervisor :

Date :

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to everyone who contributed to the successful completion of this project. First and foremost, I am deeply thankful to my supervisor, Ir. Dr. Wong JianHui, for her invaluable advice, guidance, and constant motivation throughout the development of this project. Her support and expertise have been instrumental in shaping both the direction and outcome of my work.

I would also like to extend my sincere thanks to my loving parents and friends, whose unwavering support and encouragement helped me persevere through the most challenging moments. The process was, at times, exhausting, especially when personal and academic pressures took their toll on my health. As someone who spent at least an hour commuting to and from university every day, the journey was not easy. In those difficult times, my parents and friends were always there to uplift me and remind me that challenges are part of the path to success.

Additionally, I want to acknowledge Ir. Dr. Wong for her advice beyond this project, particularly in shaping my future career prospects. Her insights have been incredibly valuable, and I have learned a great deal under her mentorship. I am also grateful to my friend Mr. Thomas Mok Shao Chung for providing thoughtful advice on managing stress and challenges. Lastly, I would like to thank Ms. Goh Yong Yee for her diligent teaching, particularly in research methodologies and presentation skills, which have been essential throughout this journey.

ABSTRACT

This project addresses the challenge of load forecasting in research offices, where uneven energy usage leads to inefficient load management. Most effective models require extensive data inputs, such as weather and economic variables, which are not always available. Additionally, most models are static and cannot adapt to changing load patterns, limiting the effectiveness in dynamic environments. This study aims to develop a load forecasting model that uses limited data length (1- 2 years), limited variables—only time and power consumption—to achieve compatible accuracy. The selected algorithms include Catboost, LSTM, GRU, and CNN-BiLSTM, with a focus on incorporating a self-updating feature to improve adaptability. The results show that while deep learning models achieve reasonable accuracy, Catboost outperformed with an RMSE of 0.0819 kW, MAE of 0.0474kW, and MAPE of 0.5127%. The self-updating Catboost model further enhanced performance compared to its static counterpart to capture future dynamic load, with MAPE below 5% across every month. The developed models are ready for deployment and require only power consumption data for training, making them both robust and adaptable for predicting future load profiles.

TABLE OF CONTENTS

DECLARATION **i**
APPROVAL FOR SUBMISSION **ii**
ACKNOWLEDGEMENTS **iv**
ABSTRACT **v**
TABLE OF CONTENTS **vi**
LIST OF TABLES **ix**
LIST OF FIGURES **xi**
LIST OF SYMBOLS / ABBREVIATIONS **xiii**
LIST OF APPENDICES **xvi**

CHAPTER

1 INTRODUCTION **1**
1.1 General Introduction 1
1.2 Importance of the Study 2
1.3 Problem Statement 3
1.4 Aim and Objectives 3
1.5 Scope and Limitation of the Study 4
2 LITERATURE REVIEW **5**
2.1 Introduction 5
2.2 Traditional Machine Learning Model 5
2.3 Deep Learning Model 7
2.4 Hybrid Model 8
2.5 Self-updating Model 10
2.6 Dataset 11
2.7 Research Gap 12
2.8 Summary 12
2.8.1 Summarised Finding 13
3 STATIC MODEL DEVELOPMENT & EVALUATION **17**

3.1	Introduction	17
3.2	Flowchart and Work Plan	18
3.3	Data Collection and Analysis	20
	3.3.1 Dataset	20
	3.3.2 Statistical analysis	22
	3.3.3 Box plots analysis	23
	3.3.4 Autocorrelation Analysis	25
3.4	Machine Learning Algorithm	27
	3.4.1 Categorical Boosting (CatBoost)	27
	3.4.2 Long Short-Term Memory (LSTM)	28
	3.4.3 Gated Recurrent Unit (GRU)	29
	3.4.4 Hybrid Model	30
3.5	Evaluation Metrics	31
	3.5.1 Mean Absolute Error (MAE)	31
	3.5.2 Root Mean Squared Error (RMSE)	32
	3.5.3 Mean Absolute Percentage Error (MAPE)	32
3.6	Feature Selection	34
	3.6.1 LSTM	36
	3.6.2 GRU	37
	3.6.3 CatBoost	38
	3.6.4 CNN-LSTM	39
3.7	Data Preprocessing	40
	3.7.1 Sliding Window	40
	3.7.2 Min-Max Technique	41
3.8	Hyperparameter Tuning	42
	3.8.1 Grid Search Method	42
	3.8.2 LSTM Tunning	43
	3.8.3 GRU Tunning	48
	3.8.4 CatBoost Tunning	53
	3.8.5 CNN-LSTM Tunning	59
3.9	Convergence Analysis	64
	3.9.1 LSTM	64
	3.9.2 GRU	66
	3.9.3 CatBoost	67

3.9.4 CNN-LSTM 68
3.10 Performance Measure 69
3.11 Limitations of Static models 75
4 ADAPTIVE CATBOOST DEVELOPMENT & EVALUATION 78
4.1 Flowchart & Work Plan 78
4.2 Dataset 79
4.3 Preprocessing 80
4.4 Drift Detection 81
4.5 Optuna Optimization 82
4.5.1 Updates Starting in 2024 (Scenario 1) 83
4.5.2 Updates Starting in 2023 (Scenario 2) 84
4.6 Updating Size 84
4.7 Updating Model Evaluation 85
4.8 Performance Measure Result 87
4.8.1 Updates Starting in 2024 (Scenario 1) 87
4.8.2 Updates Starting in 2023 (Scenario 2) 91
4.9 Summary 95
5 CONCLUSION 96
5.1 Conclusion 96
5.2 Recommendations for Future Work 96
REFERENCES 98
APPENDICES 103

LIST OF TABLES

Table 2.1:	Input Parameters and Self-Update Capabilities of Studied Papers	13
Table 2.2:	Comparison of traditional machine learning and deep learning	15
Table 2.3:	Comparison of deep learning algorithms	16
Table 3.1:	Sample of Dataset Used in this Project.	20
Table 3.2:	Dataset Characteristics.	21
Table 3.3:	Augmented Dickey-Fuller Test	22
Table 3.4:	Statistical Analysis Data	23
Table 3.5:	Feature Categorization by Correlation Strength	36
Table 3.6:	Feature Experiment of LSTM model	37
Table 3.7:	Feature Experiment of GRU model	38
Table 3.8:	Feature Experiment of CatBoost model	39
Table 3.9:	Feature Experiment of CNN-LSTM model	40
Table 3.10:	Hyperparameter Tunning of LSTM model (Phase 1)	44
Table 3.11:	Hyperparameter Tunning of LSTM model (Phase 2)	45
Table 3.12:	Optimizer Tunning of LSTM model	46
Table 3.13:	Hyperparameters Tunning of GRU model (Phase 1)	49
Table 3.14:	Hyperparameters Tunning of GRU model (Phase 2)	50
Table 3.15:	Optimizer Tunning of GRU model	51
Table 3.16:	Hyperparameters Tunning of CatBoost model (Phase 1)	54
Table 3.17:	Hyperparameters Tunning of CatBoost model (Phase 2)	56
Table 3.18:	Hyperparameters Tunning of CatBoost model (Phase 3)	57
Table 3.19:	Hyperparameters Tunning of CNN-LSTM model (Phase 1)	60

Table 3.20:	Hyperparameters Tunning of CNN-LSTM model (Phase 2)	61
Table 3.21:	Optimizer Tunning of CNN-LSTM model	62
Table 3.22:	Overall Comparison between LSTM, GRU, CNN-LSTM, CatBoost on Test Data	75
Table 4.1:	Dataset Characteristics (January 2024 – June 2024)	80
Table 4.2:	Adaptive Catboost Optimum Parameter Tunned by Optuna Optimization for Each Updating (Update from Jan 2024 Onwards)	83
Table 4.3:	Adaptive Catboost Optimum Parameter Tunned by Optuna Optimization for Each Updating (Update from Jan 2023 Onwards)	84
Table 4.4:	Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2024 Onwards)	90
Table 4.5:	Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2023 Onwards)	94

LIST OF FIGURES

Figure 3.1:	Iterative Development in Software Development	17
Figure 3.2:	Flowchart of Static Model Development	19
Figure 3.3:	Box plot of hourly electric load	23
Figure 3.4:	Box plot of yearly electric load	24
Figure 3.5:	Box plot of monthly electric load	24
Figure 3.6:	Box plot of daily electric load	25
Figure 3.7:	Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) Plot	26
Figure 3.8:	Correlation matrix of potential feature	35
Figure 3.9:	Illustration of sliding window technique.	41
Figure 3.10:	Flowchart of LSTM Tunning Process	47
Figure 3.11:	Flowchart of GRU Tunning Process	52
Figure 3.12:	Flowchart of CatBoost Tunning Process	58
Figure 3.13:	Flowchart of CNN-LSTM Tunning Process	63
Figure 3.14:	Training and validation loss for LSTM	65
Figure 3.15:	Training and validation loss for GRU	67
Figure 3.16:	Training and validation loss for CatBoost	68
Figure 3.17:	Training and validation loss for CNN-LSTM	69
Figure 3.18:	Prediction curves on 21/10/2023	70
Figure 3.19:	Prediction curves on 11/11/2023	71
Figure 3.20:	Prediction curves on 22/11/2023	71
Figure 3.21:	Prediction curves on 31/12/2023	72
Figure 3.22:	Prediction curves on 21/10/2023 to 27/10/2023 (one week)	72

Figure 3.23: Prediction curves on 20/10/2023 to 31/12/2023 (full test data) 74

Figure 3.24: Prediction curves on 31/12/2023 74

Figure 3.25: Prediction curves on 21/06/2024 – 27/06/2024 76

Figure 3.26: Plotting of Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2024 Onwards) 76

Figure 3.27: Load Power Consumption of Whole Dataset (Dec 2021 to June 2024) 77

Figure 4.1: Flowchart of Adaptive CatBoost 79

Figure 4.2: Prediction curves on 11/02/2024 to 17/02/2024 (one week) 87

Figure 4.3: Prediction curves on 06/05/2024 to 12/04/2024 (one week) 87

Figure 4.4: Prediction curves on 06/05/2024 to 12/05/2024 (one week) 88

Figure 4.5: Prediction curves on 23/06/2024 to 29/06/2024 (one week) 88

Figure 4.6: Plotting of Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2024 Onwards) 90

Figure 4.7: Prediction curves on 01/08/2023 91

Figure 4.8: Prediction curves on 01/08/2023 to 07/08/2023 (one week) 91

Figure 4.9: Prediction curves on 15/10/2023 to 22/10/2023 (one week) 92

Figure 4.10: Prediction curves on 11/05/2024 to 17/05/2024 (one week) 92

Figure 4.11: Plotting of Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2023 Onwards) 94

LIST OF SYMBOLS / ABBREVIATIONS

V_{PhA}	*Phase A voltage*
V_{PhB}	*Phase B voltage*
V_{Phc}	*Phase C voltage*
I_{PhA}	*Phase A current*
I_{PhB}	*Phase B current*
I_{PhC}	*Phase C current*
PF	*Power Factor*
X	*The original value*
X_{min}	*The minimum values of the feature*
X_{max}	*Maximum values of the feature*
x'	*Normalized value*
$\emptyset_i$	*Non-seasonal autoregressive (AR)*
Φ_i	*Seasonal autoregressive (AR)*
L	*The lag operator*
p	*Order of the non-seasonal AR terms*
P	*Order of the seasonal AR terms*
S	*Length of the seasonal period*
d	*Order of non-seasonal differencing*
D	*Order of seasonal differencing*
Y_t	*Time series at time t*
θ_i	*Coefficients of the non-seasonal moving average (MA)*
Θ_i	*Coefficients of the seasonal moving average (MA)*
q	*Order of the non-seasonal MA*
Q	*Order of the seasonal MA terms*
f_t	*Forget gate at time t*
i_t	*Input gate at time t*
x_t	*Input at timestep t*
h_t	*Output state of the LSTM at timestep t*
C_t	*Cell state at timestep t*
W	*Weights corresponding to each gate*

b	*Biases corresponding to each gate*
σ	*Sigmoid activation function*
σ_o	*SoftMax activation function (multiple classes)*
$Tanh$	*Hyperbolic tangent function,*
U	*The time insensitive hidden state matrix*
z_t	*Update gate at time t*
r_t	*Reset gate at time t*
h'_t	*Candidate hidden state at time t*
$h_{t(GRU)}$	*Final hidden state at time t*
σ	*sigmoid activation function, used to regulate the gates*
n	*Total number of observations in the dataset*
y_i	*The actual observed value for the i^{th} observation*
y'_i	*The predicted value for the i^{th} observation*
ACF	*Autocorrelation Function*
ADAM	*Adaptive moment estimation*
AI	*Artificial Intelligence*
ANN	*Artificial Neural Network*
AR	*Autoregressive*
ARIMA	*Auto Regressive Integrated Moving Average*
BiLSTMS	*Bidirectional Long Short-Term Memory Networks*
CNN	*Convolutional neural network*
DL	*Deep learning*
GRU	*Gated recurrent unit*
GTO	*Golden Tortoise Optimization*
GWO	*Grey Wolf Optimizer*
LSTM	*Long short-term memory*
MAE	*Mean Absolute Error*
MAPE	*Mean Absolute Percentage Error*
ML	*Machine learning*
PACF	*Partial Autocorrelation Function*
RMSE	*Root Mean Squared Error*
RNN	*Recurrent neural network*
SAMF	*Self-Adaptive Momentum Factor*

SARIMA	*Seasonal Autoregressive Integrated Moving Average*
SVR	*Vector Regression*
TCN	*Temporal Convolutional Networks*
VMD	*Variational Mode Decomposition*
WNN	*Wavelet Neural Network*

LIST OF APPENDICES

Appendix A: Codes 103

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In today's age where the sustainability of the energy systems concerns us more than ever, it is not just a technical challenge but a necessity to be able to forecast the electrical load within commercial buildings precisely. This project serves to be a substantial step forward in the field of energy management by creating a state-of-the-art machine learning model for a two-storey research office. As such, this technology has the potential not only to improve the ways in which people consume energy but also to make it easier for us to optimize this consumption on the go, reducing operational costs and increasing sustainability for many businesses. As more and more businesses are focusing on reducing the impact on environment and becoming more sustainable and operationally efficient, the role of such intelligent automation in managing energy will only end up increasing. In research offices, where the usage of this form is uneve, this technology will also prove its utmost utility. Being put to use in a country like Malaysia, which is now focusing on acquiring the benefits of technology and sustainability, and using complex machine learning techniques, this project aims to make load forecasts more accurate.

This is not just the next step in the rapid evolution of technology. In fact, the emergence of artificial intelligence in this situation serves to be a massive force that changes the way of energy management and conservation. As such, this model, along with other AI technologies, will bring us a step closer to a world in which the energy systems know what do and more than that, do it in a manner that is inherently sustainable. The computational capabilities of artificial intelligence that will allow businesses to compete on a global level now equipped with the intellectual capabilities needed to approach this competition in a manner that is both profitable and beneficial for the environment. As such, this project not only serves the immediate operational needs of load forecasting for businesses but also advances the universal discussion on how artificial intelligence and other technological advancements will define the future in terms of sustainability and energy efficiency.

1.2 Importance of the Study

As the world's attention shifts towards the sustainability of the energy systems, the ability to reliably forecast the electrical load within commercial buildings becomes not just a technical difficulty but a mandate upon which the efficiency and environmental responsibility of energy use hinge. This capacity is especially critical amid the rapid advancements in energy management and digital technology, whose potential to revolutionize the way people conserve and consume energy remains largely untapped. Recognizing this imperative, this project seeks to develop an advanced machine learning model designed for a two-storey research office. By doing this, the goal of the project is inspired by the need not only to revolutionize how the system consume energy but also to develop a tool that reliably taps this technology to elevate operational sustainability, reduce operational costs.

Increasingly, the worldwide shift of businesses toward greener, more efficient operational models elevates the role of intelligent automation in energy management. Both the needs and opportunities of doing this emerge with special depth in settings like research offices, where energy usage is volatile and unpredictable in nature. In the context of Malaysia's technological and sustainability growth, this project's prospects and contributions can be wholly attributed to its use of advanced machine learning methodologies for load forecasting. Therefore, Artificial Intelligence's dawn and increasing footprint can be characterized as an epochal shift in on two concomitant levels: by infinitely expanding the capacity of energy management and conservation, and fundamentally repositioning the domains of work and its role in the greater world.

Adding the immense analytical capacities of AI, the potential of the model created in this project is limitless in all practical senses. Meanwhile, the contributions made by this project can be equally instrumental not only to the goals and broader strategies that the connected industry would develop, but also to the crucially important discussion on how AI can be developed and employed to create a sustainable future. In the context of this encompassing ambition, this project is characterized by both prospect and closure, representing a collection

of insights on how the world can change, and a foundational step making this transformation a reality.

1.3 Problem Statement

According to Ahmad, et al. (2022), most conventional energy forecasting models struggle at one point to capture the dynamic and often non-linear features of the energy electrical load. This shortcoming creates flaws in energy planning, leading to higher operational cost for the research office. In most studies, it is always advised to improve the accuracy of the model, the input data used to train the model should include as much detailed information like weather, and temperature and economies data. However, in many cases, the detailed information might not be always available or easily accessible due to funding and accessibility issues. The question posed in the case of a research office, which lacks all the detailed information is how to achieve maximum accuracy of energy electrical load forecasting with the least detailed data. There is a critical need for a robust forecasting model that can effectively capture the dynamic and non-linear characteristics with limited data (time and power consumption) to predict future load profiles, ensuring energy efficiency and operational continuity in research facilities.

1.4 Aim and Objectives

This project aims to evaluate the feasibility of machine learning-based load estimation for a research office. The major objectives are presented as follows:

- **(i)** To investigate the current state-of-the-art algorithms applied on load forecasting and select suitable algorithms.
- **(ii)** To develop a system that could estimate the next day load using multiple machine learning methods.
- **(iii)** Evaluate the efficiency of the developed models in predicting the next day load profile.

1.5 Scope and Limitation of the Study

To begin with, it is essential to outline the study's scope, being limited to the construction of a load forecasting model designed for a two-storey research office. As already mentioned, the model is developed using power and time variables to predict energy consumption and identify specific patterns that would support future consumption trends. Therefore, the contribution of this research is a predictive mechanism that allows for better energy management by analysing historical electricity usage within particular periods. Thus, the established model facilitates making better decisions regarding consumption strategies. Simultaneously, the scope involves a detailed analysis of the existing power usage data, the imposition of an algorithm based on time series analysis, and a detailed analysis of the results to forecast the load profile within desired short-term periods.

However, the study is limited to the office context and may require adaptation when applied to other types of buildings or larger scales. Another limitation is that the access to high-computational resources is limited, and thus, the training and refinement of the models and feature tunings differentiate from the theoretical perspective. The limitations of the study in terms of the resources available are acknowledged, as the objective of the research is to aim for the best possible outcomes within the given constraints.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Load forecasting plays a crucial role in energy management, allowing for efficient energy distribution, capacity planning, and reducing operational costs. Traditionally, statistical methods have been employed, but recent years have seen a shift towards machine learning (ML) and deep learning (DL) approaches due to the superior predictive performance and flexibility in handling complex nonlinear relationships in energy consumption data.

2.2 Traditional Machine Learning Model

Accurate load forecasting models help reduce operational costs, enhance grid reliability, and support demand-side management initiatives. Various traditional machine learning and statistical models have been employed for this task, each offering unique strengths and limitations. Linear Regression (LR) is one of the most basic for forecasting due to its simple structure. It attempts to model the relationship between the dependent variable and one or more independent variables by fitting a linear equation to the observed data. However, the simplicity of LR often limits its effectiveness in capturing the complex, non-linear relationships present in load forecasting data. For example, Zhnag et al. (2019) found that LR had a relatively low R^2 value, indicating that a significant proportion of the variance in the load data could not be explained by the model. This suggests that while LR can provide some basic insights, it is prone to significant prediction errors when applied to short-term load forecasting, especially in the presence of non-linear patterns.

Decision Trees (DT) are another popular method in load forecasting. Decision Trees split the data into subsets based on the value of input features, creating a model that is easy to visualize and understand. However, DTs are prone to overfitting, particularly when the trees become too deep, leading to models that perform well on training data but poorly on unseen data. Huang and Huang (2020) observed that the R^2 value for DT models was significantly lower

than for other methods, such as Random Forests, particularly as the complexity of the data increased.

Random Forests (RF), a method that builds multiple decision trees and aggregates the results, has been shown to improve upon the predictive accuracy and generalization ability of single decision trees. By averaging the predictions of many trees, RF reduces the risk of overfitting and typically performs better on a variety of datasets. Zhou et al. (2021) found that RF achieved R^2 more than 0.90 in load forecasting tasks, which was higher than both DT and LR models, indicating that RF provides a more accurate and reliable prediction. SARIMA (Seasonal AutoRegressive Integrated Moving Average) is a traditional statistical model widely used in time series forecasting, including load forecasting. SARIMA extends the ARIMA model by explicitly modeling seasonal effects, making it particularly useful for datasets with strong seasonal patterns. Elamin and Fukushige (2018) has introduce a SARIMA model for hourly electricity demand forecasting that considering external factors like weather and calendar events. The model's performance is enhanced, reducing the MAPE by 22.2% and achieving significant reductions in MAE and RMSE by 21.3% and 21.8%, respectively. However, it performance drop when due to non-linear load pattern.

Gradient Boosting Machines (GBMs), such as XGBoost, have gained popularity due to the ability to achieve high accuracy in a wide range of prediction tasks. GBMs work by sequentially adding models (typically decision trees) to correct the errors made by previous models, gradually improving the overall predictive performance. Deng et al. (2022) presented XGboost load forecasting model with average MAPE of daily load is reduced by 1% -1.5%, and the average MAPE of peak load is reduced by 3%, outperforming other traditional models like RF and SVR. However, GBMs require careful tuning of hyperparameters to avoid overfitting and ensure optimal performance.

In recent years, CatBoost has gained recognition as a powerful gradient boosting algorithm, particularly in applications that involve complex datasets with categorical variables. However, CatBoost's utility extends beyond such scenarios. Even in cases where the dataset is limited to time and load power, CatBoost has demonstrated robust performance. For instance, its ability to handle noise and reduce overfitting through ordered boosting is particularly

beneficial. While earlier studies like those by Bian et al. (2024) explored CatBoost's efficacy in HVAC power consumption predictions, achieving an R^2 value of 0.91, these findings suggest that CatBoost's strengths are not solely dependent on the presence of multiple categorical features. The model's inherent capabilities make it well-suited for various forecasting tasks, including those with simpler datasets, making it a compelling choice for load forecasting based solely on time and load power data.

2.3 Deep Learning Model

Despite traditional model having advantages in its simple structure and training speed, but it have limitations in capturing the nonlinear characteristics of load series. Deep learning is a subset of machine learning, which in turn is a subset of artificial intelligence (AI) that employs neural networks with many layers.

Duan (2020) proven that deep learning models are able to capture complex functions and non-linear pattern of the data. These models have shown significant improvements in the accuracy of electricity load forecasting when using large dataset. Additionally, the ability of deep learning techniques to capture long and non-linear patterns in building power consumption is better than traditional forecasting methods. Therefore, the application of more advanced machine learning models has enhanced the electricity load forecasting.

According to Eren and Kucukdemiral (2024), the common deep learning models applied in load forecasting are long short-term memory (LSTM), convolutional neural network (CNN), recurrent neural network (RNN), gated recurrent unit (GRU), and autoencoders. These deep learning were used to overcome the limitations of traditional forecasting methods. Lin, et al. (2022) presents a load forecasting model for short-term electricity load using LSTM based model. It is using a novel attention mechanism to improves forecasting by considering feature correlations and temporal dynamics, and refined selection of relevant weather data for more precise regional load predictions. The research employed the Global Energy Forecasting Competition 2014 datasets with power consumption and temperature data as input to train the model. It demonstrated superior performance over existing methods in both point and probabilistic forecasting.

Xu and Baldick (2019) investigated single and deep-stacked LSTM neural networks with different activation functions to forecast power load one hour ahead using temperature and load data. The results showed that the two-stacked LSTM network achieved highest accuracy with a MAPE of 1.53%. According to the research of Bouktif, et al. (2019), LSTM and GRU are outperformed machine learning and single-sequence models using RTE power consumption dataset with power consumption and weather data. Both LSTM and GRU are effective, but GRU is better in term of training speed due to it algorithm's structure is simpler than LSTM. Massaoudi, et al. (2020) demonstrated that temporal CNN can provide effective forecasting model with benchmarks of SVM and LSTM, its architecture can contribute to the training time reduction by decreasing the model complexity with an improvement of the model accuracy when integrate with another model such as CNN-LSTM.

According to Tong, et al. (2022), the autoencoder part of the model has been created to accomplish two tasks: dimensionality reduction in the encoding phase and reconstruction in the decoding phase. As a result, it allows for effective compression of time series characteristics with the possibility of reconstructing the input sequences to enable the latent vectors to be indicative of the original data. Autoencoders can be educated without a teacher. It implies that it can strive to identify critical features in the input data without necessitating the use of explicit tags. When learning is completed, the encoder network can be employed to gather characteristics from new data samples, which can be used in an LF problem. However, it may not perform well compared to other deep learning models on specific supervised learning tasks, especially those involving sequential data, where RNNs, GRUs, LSTMs, and CNNs have shown superior performance.

2.4 Hybrid Model

Each of the individual algorithm have its advantages and shortage, so in order to further improve the performance of the model, people started to proposed

hybrid model. The advantage of hybrid model is that if one component of the hybrid model fails to produce accurate predictions under certain conditions, other components can help to cover for its shortcomings to increase the accuracy.

Sajjad, et al. (2020) proposed CNN-GRU hybrid model achieved the smallest error rate in MSE, RMSE, and MAE which are 0.09%, 0.31%, and 0.24% respectively using same datasets when compared with other popular models, showcased high precision and efficiency in short-term residential load forecasting. Kim and Cho (2019) then combined Convolutional Neural Networks and Long Short-Term Memory networks to build a CNN-LSTM hybrid model and predict the housing energy consumption. This model is designed to extract both spatial and temporal features affecting energy consumption, allowing it to account for complex patterns including irregular trends in time series data. The performance of the proposed CNN-LSTM network is highlighted as achieving almost perfect prediction accuracy and recording the smallest value of RMSE when compared to conventional forecasting methods.

Fan, et al. (2023) introduces an innovative hybrid model GWO-VMD-GTO-CNN-BiLSTMS, which uses Variational mode decomposition (VMD) optimized by Gray Wolf optimization (GWO), for feature extraction along with Convolutional Neural Network and Bidirectional Short-Term Memory neural network for very accurate short-term load forecasting. This model provides a reasonable way of dealing with uncertainty and complexity in power systems, thus, representing superior predictive performance. However, the dataset chosen in the analysis is too small and may bias the results. In addition, the model may be seen as relatively complex, thus, leading to an increase in the training time. In addition, the study by Hadjout, et al. (2022) presents an innovative hybrid deep learning model combining Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Temporal Convolutional Networks (TCN) to forecast monthly electricity consumption for the economic sector in Algeria. The model is trained using data from various economic sectors consumers, and it outperforms other single forecasting methods.

2.5 Self-updating Model

Self-updating model are models that are able to automatically adjust its parameters in response to new data or changes in its environment, without need to retrain the underlying models from scratch or updating the internal parameters manually. This type of method is aimed to solve the problem of decrease in accuracy when model expose to change of trend due to season or change of environment in new input data. In fact, it modifies the effect or weight of each model in the ensemble depending on the performance with the new data (Kolter & Maloof, 2004). The method enhances the ensemble's general prediction by enhancing the use of more accurate models and lowering the importance of less accurate models, depending on the recent performance of the model. This is a faster adaption method that does not need the computational resources and time that full retraining does.

Taleb, et al. (2022) has developed a model the hybrid model, which can be applied in different region and changes over time. The latter feature allows the developed model to increase the forecasting accuracy at every forecasting range. Despite seasons, the model has shown an MAPE of 1.71% for 30-minute predictions, 3.5 % for 24-hour predictions, and 5.1 % for one-week prediction. The model adjusts itself through calculating the mean and standard deviations of the past forecasting errors and changes weights according to them. In addition, Zulfiqar, et al. (2022) also achieve dynamic weight adjustment by incorporating a Self-Adaptive Momentum Factor (SAMF) into the load forecasting model, result in MAPE of 1.71% for 30-minute predictions, 3.5% for 24-hour predictions, and 5.1% for one-week predictions, indicating a strong performance across different prediction intervals.

The self-updating can also be achieved in online-model, Li, et al. (2023) has integrated both online and offline methods to improve the accuracy and robustness of residential load forecasting. The offline component uses historical data to build a foundational predictive model and the online component dynamically updates the forecasts based on real-time data. Fekri, et al. (2021) has also proposed a load forecasting model that able to continuous learning and adapting to new patterns using online learning RNN algorithm.

2.6 Dataset

The dataset used in load prediction influences the models generated by it, therefore selection of the data is crucial. It is important to choose the dataset according to its impact on the accuracy of forecasts and its capacity to generate meaningful insights. The accuracy of the predictions will generally depend on how well the models are able to learn from the samples of real consumption patterns and develop into meaningful forecasting systems. The quality of the data used in the modelling can have effect on every stage, from the training and testing to the application of the models. Quality data should be error free, do not have missing values and or outliers (Qin et al., 2019). It is collected on the same level of detail as the patterns of load. The data also undergoes preprocessing such as data cleaning, normalization, and sometimes transformation to ensure the consistency of the model.

Table 2.1 in summarised finding shows that many load prediction models incorporate features to capture the dynamics of energy usage. These features typically encompass power consumption and time along with factors, like weather conditions (temperature, humidity) time related indicators (daily weekly) and occasionally socio-economic factors (holidays, events). These features may help to increase the accuracy in prediction, but at the same time may overfit the models or increase the complexity of training process.

However, in many cases, especially in regions where the technological infrastructure is limited or in certain applications where data collection is limited, it might be unfeasible to collect a comprehensive dataset that includes all potential features. In these instances, focusing on a simpler dataset that mainly includes power consumption and time could offer specific benefits. By doing so, the researchers and analysts can focus on the bare minimum of temporal patterns that determine electricity usage while avoiding additional complexity that other variables can introduce. In these cases, this course of action is not just convenient but also required.

2.7 Research Gap

After doing a thorough study of the current literature on load forecasting models, it is found that most studies achieve robust model using dataset with a wide range of criteria other than just power usage and time, such as weather conditions, consumer habits, and socioeconomic data. This strategy, while effective, frequently demands complicated data collecting and processing. There is a huge gap in the investigation of models that rely exclusively on limited inputs which are power consumption and time. Models like this are not only important in those scenarios with limited data variable, but it also can achieve simpler implementation while maintaining forecast accuracy.

Furthermore, the models that is capable of self-updating and self-correction is still not being study much. Most existing forecasting models are static, meaning they do not evolve after training and deployment. This static nature restricts the application in dynamic situations where load patterns may keep changing due to new technologies or changes in usage patterns.

2.8 Summary

In early, traditional statistical models were the most frequently utilized solutions in load forecasting. It included such popular ones as ARIMA and SARIMAX, which are highly applicable to small datasets and relatively simple prediction conditions. Nonetheless, traditional models are limited in its ability to process nonlinear and dynamic patterns, which are typical for energy consumption data. Eventually, with the increase in computational power and the amounts of data, machine learning models such as Support Vector Regression became widely applied. These solutions are more suitable for high-dimensional data, as well as complex relationships and are able to overcome the drawbacks of traditional methods. However, it still face difficulties in capturing long-term dependencies.

With the emergence of deep learning technologies, it contributed to the development of load forecasting. LSTM models, CNN, GRU, and autoencoders have demonstrated unique opportunities for monitoring intricate and non-linear dependencies in vast data sets. Time-dependent models have shown the best results in terms of temporal dependencies, constantly surpassing all the rest of the traditional and earlier machine learning models in terms of accuracy and stability. That is why hybrid models began to be developed, which consist of a

tandem of multiple forecasting models for increased prediction data quality. Hybrid models simultaneously utilize several models using ensemble methods, such as bagging and boosting, for further improved generalization and decreased prediction variance by aggregating forecasts from the individual base models.

Moreover, to solve the problem of decrease in accuracy when model expose to change of trend due to season or change of environment in new input data, the self-updating model seems to be a good solution for it. These models dynamically adjust the parameters in response to new data, enabling them to continuously learn and adapt to changing patterns without manual intervention or extensive retraining. However, this type of model is still not widely utilized and requires further research and development to optimize its performance and applicability in load forecasting.

2.8.1 Summarised Finding

Table 2.1: Input Parameters and Self-Update Capabilities of Studied Papers

Author	**Load Type**	**Input Parameters**	**Self-Update Capability**
Duan (2020)	Residential Load	Time, weather, and power	✕
Eren and Kuçukdemiral (2019)	Residential Load	Temperature, time, and power	✕
Fan et al. (2023)	Power Grid	Time, weather temperature, and load data.	✕
Fekri, et al. (2021)	Residential	Time, temperature, wind speed and direction, pressure, and humidity	✓
Hadjout, et al. (2022)	Commercial	Time, weather temperature, economic data, and load data.	✕

Kim and Cho (2019)	Residential Load	Time, and load data	✕
Sajjad, et al. (2020)	Residential load	Time, Temperature, humidity, pressure, power etc..	✕
Skomski, et al. (2020)	Office	Time, Temperature, and power.	✕
Taleb, et al. (2022)	Power Grid	Time, temperature, load data	✓
Xu and Baldick (2019)	Building cooling load	Time, weather temperature, and load data.	✕
Zulfiqar, et al. (2022)	Power Grid	Time, temperature, load data	✓

Table 2.2: Comparison of traditional machine learning and deep learning

Approach	**Characteristics**	**Forecasting Applications**	**Core Advantages**	**Primary Limitations**
Traditional Machine Learning	- Models based on statistical methods and algorithmic approaches	- Typically used for short to medium-term forecasting - Load forecasting - Energy price prediction	- Simple and interpretable - Requires less computational power - Effective with well-structured data	- Often requires significant manual feature engineering - Limited in capturing large and complex datasets
Deep Learning	- ML subset with layered neural architecture - Processes large datasets efficiently	- All forecasting timeframes (short to long-term) - Load demand forecasting -Renewable energy forecasting	- Good in selecting features and data classification -Robust computational capabilities - Wide applicability for forecasting needs	- Lengthy model training durations - Higher complexity in model development - Limited performance with insufficient in feature variety & dataset size.

Table 2.3: Comparison of deep learning algorithms

Key Consideration	**RNN**	**LSTM**	**GRU**	**CNN**	**Autoencoder**
Sequential Data Handling	Suitable for sequence processing.	Excellently mitigates gradient issues for time-series data.	Gate mechanism effectively processing time-series data.	Designed for multi-dimensional spatial data, not primarily for sequences.	Typically used for spatial data arrays, not sequences.
Model Complexity	Less complex with a straightforward architecture.	Contains three gate structures, leading to higher complexity.	Simplify the architecture by combining input and forget gates.	Architecture varies greatly; may require strategies against overfitting.	Encoder-decoder framework can vary in complexity according to the number of latent features.
Computational Efficiency	Efficiency depends on the specific use case and model design.	Usually requires more computational power due to complexity.	More efficient than LSTM, especially for less complex cases.	Intensive computation, especially for large datasets with dimensionality reduction.	Can process large datasets efficiently with dimensionality reduction.
Training Duration	Training time varies; less complex models may train faster.	Can have longer training times due to sophisticated gradient handling.	Trains faster than LSTM due to simpler structure.	Training duration depends on layer quantity; can be extensive.	Weight sharing reduces parameter count, which may help in training efficiency.

CHAPTER 3

STATIC MODEL DEVELOPMENT & EVALUATION

3.1 Introduction

Firstly, one should mention that the research methodology defines the primary path of a project. In the context of machine learning, research methodology is generally integrated into the concept of the Data Science pipeline, which is designed to provide a systematic approach to solving data science problems. The pipeline includes several components, such as data collection, data wrangling, exploratory data analysis, normalization, integration, modeling, validation, and data presentation. Therefore, this project will focus on the essential steps to ensure that the machine learning model is built to be robust.

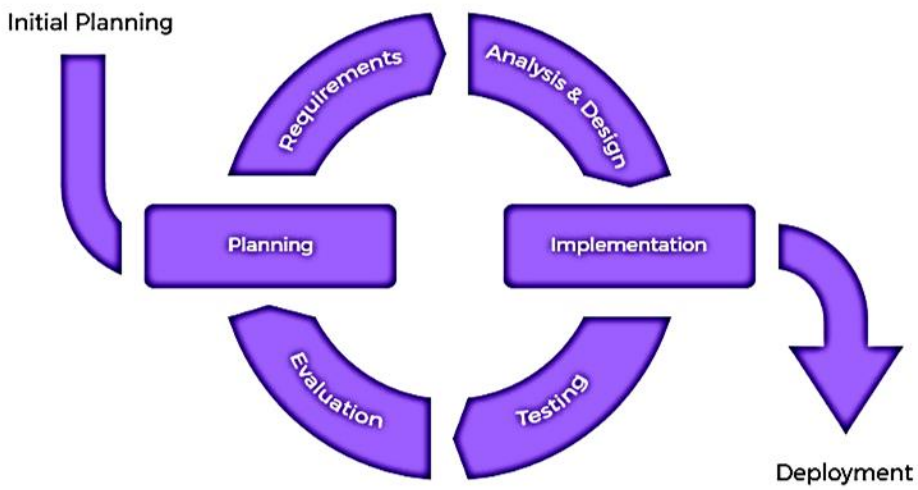


Figure 3.1: Iterative Development in Software Development

The second aspect that will provide significant support for the modeling process is Iterative development from the approach of the software development life cycle framework as shown in Figure 3.1. This approach focuses on the importance of repeated evaluation and risk management, which is suitable for the cyclical nature of evaluating data science projects. By using this approach, will be able to manage risks effectively and refine the model multiple times.

3.2 Flowchart and Work Plan

The purpose of this stage in the methodology is to develop and evaluate static load forecasting models to determine the most suitable approach for predicting power consumption in environments similar to the current research scenario. A static model refers to one that does not adapt or update itself with new data once it has been trained, making it ideal for applications where the underlying data patterns are relatively stable over time. In this phase, various machine learning models, including CatBoost, LSTM, GRU, and CNN-LSTM, are rigorously tested and fine-tuned. The goal is to identify the model that delivers the most accurate and reliable results based on historical data, which will serve as the foundation for further development in the adaptive model phase.

This phase lays the groundwork for forecasting models by optimizing the model structure, hyperparameters, and other factors such as optimizers and activation functions, ensuring that the selected model performs well under static conditions before progressing to adaptive methods.

Figure 3.2 shows the flow chart for the research methodology used in developing various machine learning models. Revised research flow chart encompassing the entire project aimed at enhancing the original model to better address the problem at hand.

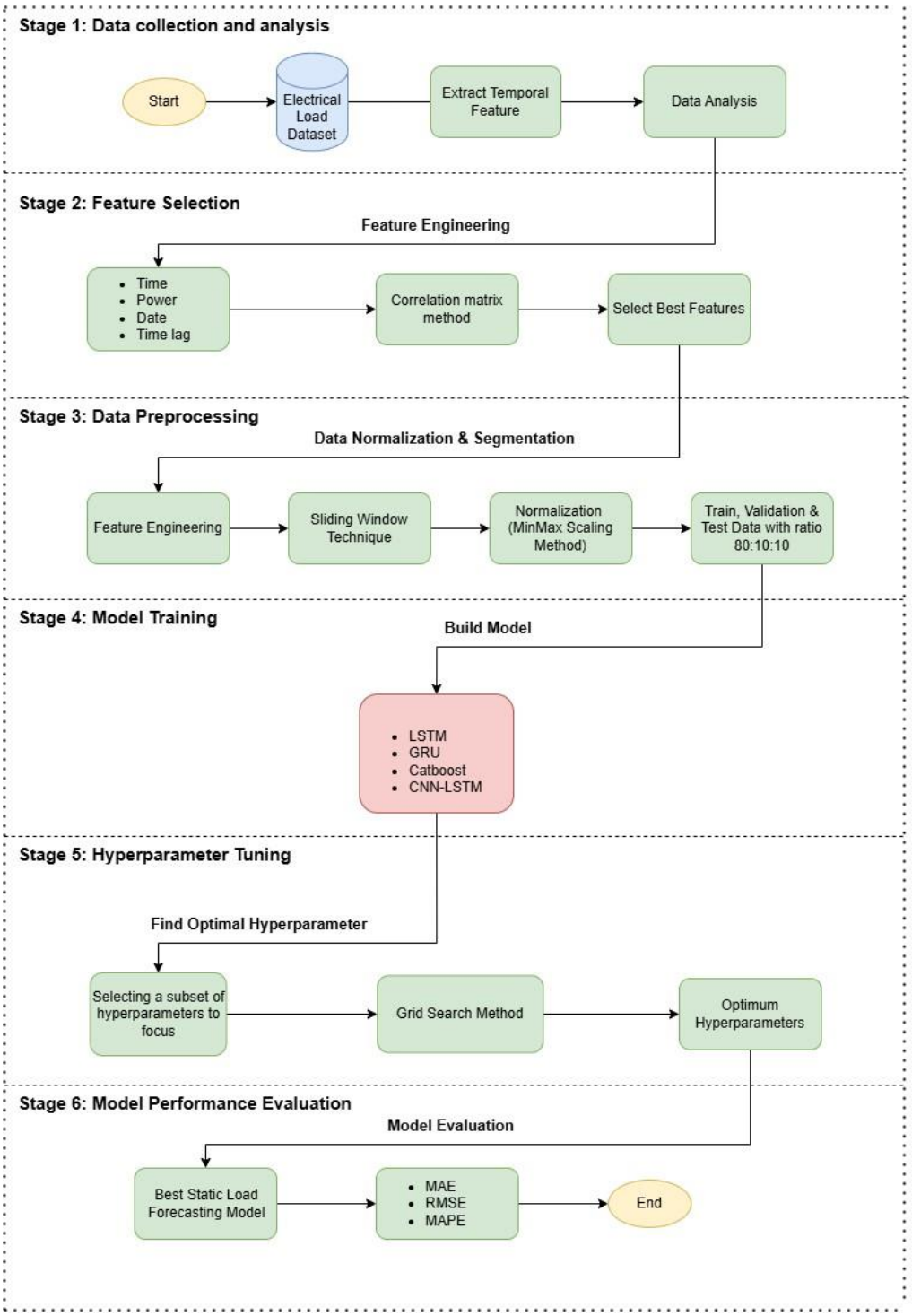


Figure 3.2: Flowchart of Static Model Development

3.3 Data Collection and Analysis

One of the crucial stages of every model development is data collection and analysis. It refers to collection of the data considered relevant for the real world scenario and reviewing it in terms of relevant patterns and trends. In case of machine learning, the quality of data is essential for the ultimate quality of the created model since the dataset ensures that the model will be representative for the real world. Moreover, the detailed analysis of the dataset can result in refining the forecasting algorithms and ensuring greater reliability of the results. In this respect, the elements of the dataset collected, and the analysis conducted will be highlighted in the present section.

3.3.1 Dataset

Dataset is critical for the development of an accurate electrical load forecasting model since the data will be fed into model and train. In this project, the dataset is collected from a two-story research office in every **30-minute** interval, with total 8 variable: time, 3 phases voltage, 3 phase current, and power factor as shown in Table 3.1 which are the sample from the dataset. The time variable delineates temporal intervals at which values needed to compute the power consumption are recorded, while the power variable denotes the corresponding electrical load values observed during these intervals.

Table 3.1: Sample of Dataset Used in this Project.

Date/Time	Voltage Ph-A Avg	Voltage Ph-B Avg	Voltage Ph-C Avg	Current Ph-A Avg	Current Ph-B Avg	Current Ph-C Avg	Power Factor
21/12/2021 1:00	246.55	248.22	246.33	6.98	3.96	6.86	0.97
21/12/2021 1:30	247.03	248.51	246.69	6.86	4.32	5.62	0.98
21/12/2021 2:00	247.63	249.13	247.19	7.17	3.96	6.95	0.97

Table 3.2: Dataset Characteristics.

Dataset Characteristics	Details
Instances	35,517
Time Span	Dec 21, 2021 to Dec 31, 2023
Collect Interval	30 minutes
Missing Values	0
Features	Time, Phase Voltage (A, B, C), Phase Current (A, B, C), Power Factor

By using the 3 phases current, 3 phases voltage, and power factor, the total power consumption and be compute as following:

$$Total\ Power = \begin{bmatrix} (V_{PhA} \times I_{PhA}) + (V_{PhB} \times I_{PhB}) \\ +(V_{Phc} \times I_{PhC}) \end{bmatrix} \times PF \qquad (3.1)$$

where

V_{PhA} = Phase A voltage,

V_{PhB} = Phase B voltage,

V_{Phc} = Phase C voltage,

I_{PhA} = Phase A current,

I_{PhB} = Phase B current,

I_{PhC} = Phase C current,

PF = Power Factor

The dataset does not contain any other features like meteorological data or economic data due to limitation of data collection, its significance lies in providing insights into the temporal load profiles specific to the research office. By focusing solely on time and power variables, the forecasting model can discern inherent patterns and fluctuations in electricity consumption within this environment.

3.3.2 Statistical analysis

Table 3.3 shows the Augmented Dickey-Fuller test results, with a test statistic of -17.2183, which is significantly lower than the critical values at the 1%, 5%, and 10% significance levels. A highly negative test statistic, such as this one, strongly suggests that the time series is stationary, meaning that its statistical properties, like the mean and variance, do not change over time. The associated p-value is extremely small, close to zero, further confirming that the null hypothesis of non-stationarity can be rejected with high confidence.

Table 3.3: Augmented Dickey-Fuller Test

Test Statistic	-17.2183
P-value	6.3×10^{-30}
Lags Used	48.00
Number of Observations Used	3.55×10^{4}
Critical Value (1%)	-3.43
Critical Value (5%)	-2.86
Critical Value (10%)	-2.56

Table 3.4 shows the statistical analysis of the electricity load data reveals an average daily usage of approximately 7,057 watts, with a standard deviation suggesting considerable variability, around 4,040 watts. The minimum recorded load is approximately 2,990 watts, suggesting there are periods of low activity. Conversely, the maximum load observed is about 30,163 watts, this abnormal rise of power may be due to some unusual work that require high demand or data collection error.

The median, or 50th percentile, is approximately 5,276 watts, which provides a more robust sense of a typical day's load compared to the mean due to its resistance to the influence of outliers. Observations lying between the 25th percentile, around 4,442 watts, and the 75th percentile, around 8,038 watts, form the interquartile range, which encompasses the middle 50% of the data and provides a clearer picture of the central distribution of the loads.

Table 3.4: Statistical Analysis Data

count	35566.00
mean	7808.18
std	5619.82
min	2990.87
25%	4509.02
50%	5425.66
75%	8541.45
max	36635.33

3.3.3 Box plots analysis

Preliminary box plots analysis for the dataset of December 2021 to July 2023 is demonstrated in this report. Figure 3.3 shows the electric load pattern of the research office, it is observed that the electrical power consumption started to rise around 8 am and remain at around peak from 10 am to 3 pm, then started to decrease until around minimum on 6 pm. The power consumption of the office was remained minimum from 6 pm unitl to 7 am. This pattern can be explained by the working office hours which is from 8 am to 6pm, it is notable that after office hour, the power cunsumption if not zero, indicating that minimum still required to maintain some of the equipment such as data center in the research office.

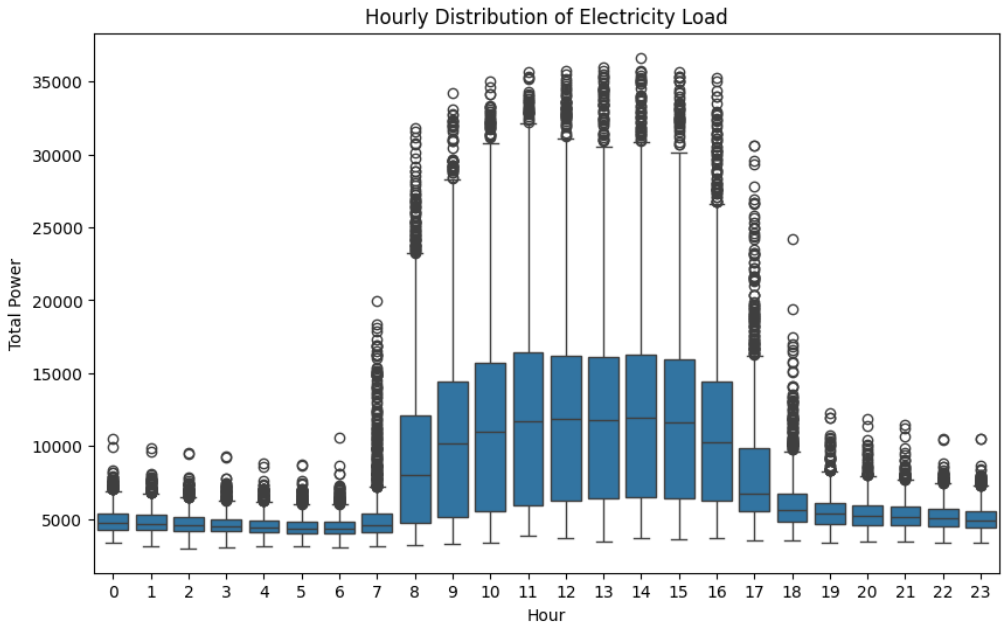


Figure 3.3: Box plot of hourly electric load

The electrical load pattern is more or less the same thoughout the year and months as shown in Figure 3.4 and Figure 3.5, indicating the activities in the research office dones not having much different. Meanwhile, the average load of weekend is found to be much lesser when compared to weekday which can be observed in Figure 3.6, indicating the research office working staff are resting during weekend.

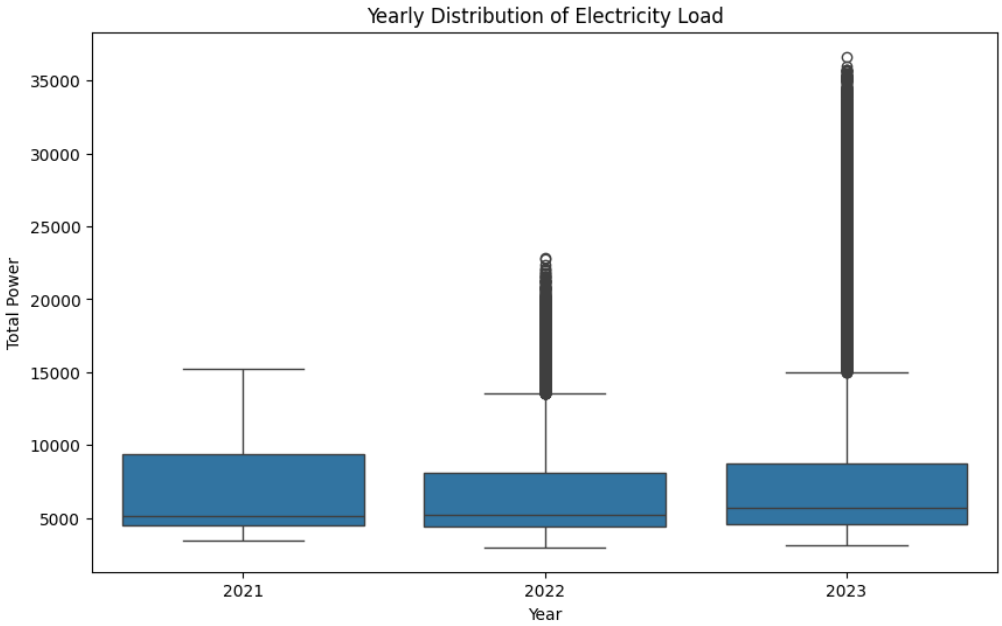


Figure 3.4: Box plot of yearly electric load

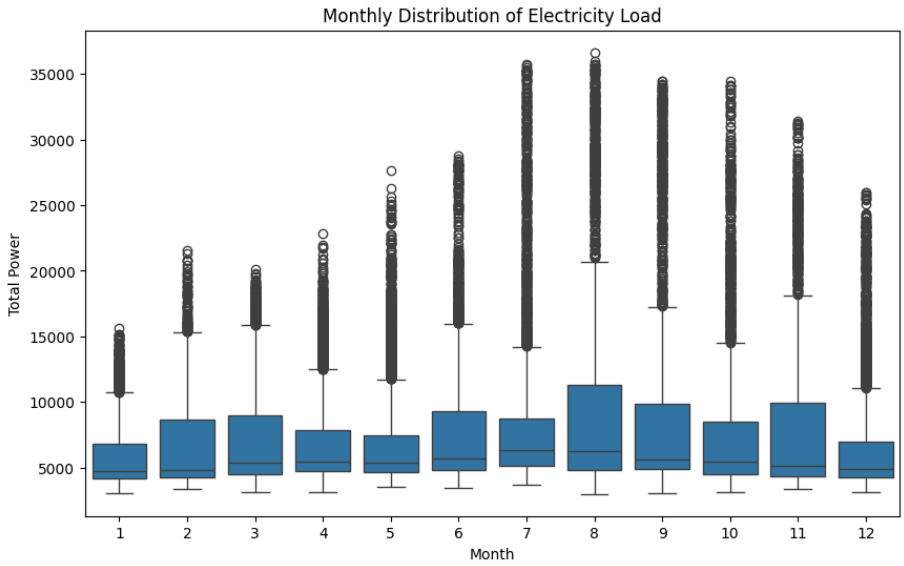


Figure 3.5: Box plot of monthly electric load

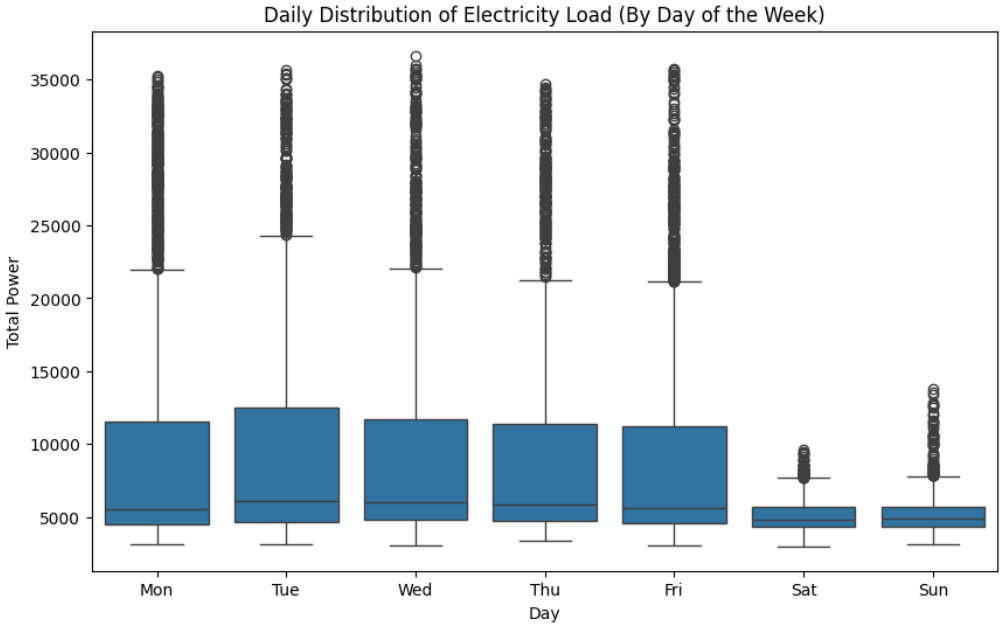


Figure 3.6: Box plot of daily electric load

3.3.4 Autocorrelation Analysis

The Autocorrelation Function (ACF) plot for electrical load displays several spikes that are well above the significance level (outside the shaded area), particularly at the initial lags. This indicates a strong correlation between consecutive observations. The fact that these correlations remain significant over several lags suggests a persistence of influence from past values, which could imply that the power usage has a memory effect where past usage levels influence future usage to some extent.

A noticeable pattern in the ACF plot is the gradual decrease in correlation as the lag increases, which is typical for time series data exhibiting a mix of trend and autocorrelation. However, the presence of significant autocorrelations at higher lags might also hint at a seasonal pattern, as the influence of past values appears to re-emerge at regular intervals.

The ACF shows numerous spikes far above the significance level , especially at the initial lags implying that successive observations are highly dependent. The high and significant negative correlation over several lags may reflect a longstanding impact emanating from the past values. Such behavior is reminiscent of memory, where the current power value is highly influenced by the recent past value and, to some extent, the one before. Another observed

pattern of the ACF is the decline in correlation as the lag increases, as is common in time series with a trend and more autocorrelation. However, the presence of significant autocorrelations at higher lags might also hint at a seasonal pattern, as the influence of past values appears to re-emerge at regular intervals.

The Partial Autocorrelation Function (PACF) plot reveals a sharp cut-off after the first few lags, with the first lag showing a significant spike. This behavior is indicative of an autoregressive (AR) process, where the immediate past value(s) have a strong influence on the current value. The sharp decline in partial autocorrelation after the first lag suggests that the most recent past value is a good predictor of the current value, while the influence of values further in the past becomes negligible once you account for the immediate past. This could inform the selection of an AR model with a low order for modeling the time series.

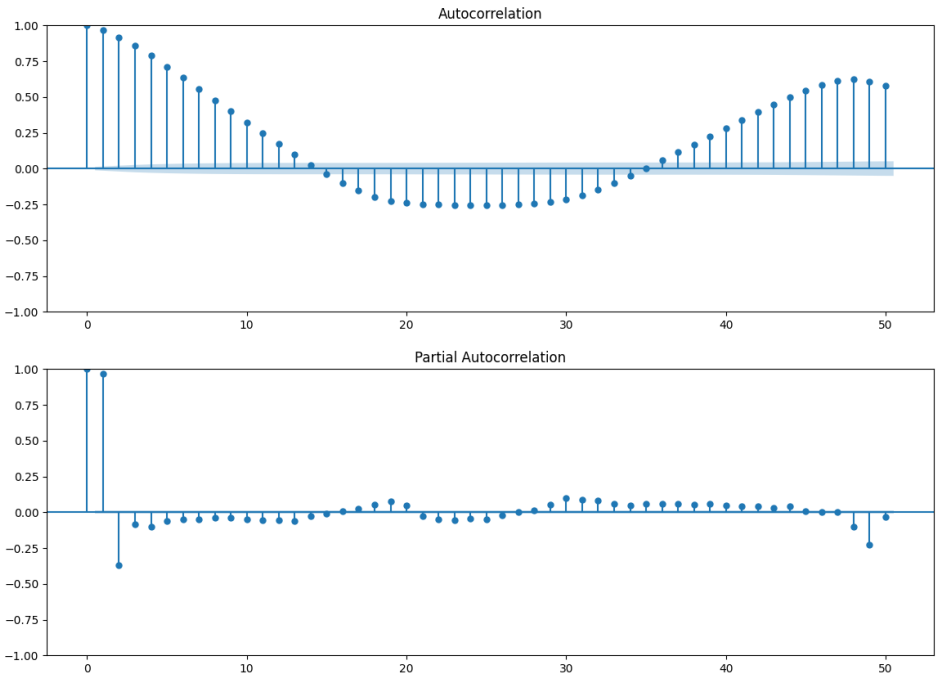


Figure 3.7: Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) Plot

3.4 Machine Learning Algorithm

Machine learning is the concept that tries to minimize human input and create a mainly autonomous process, when the system can operate on its own to solve a problem. With the help of statistical methods, researchers have managed to perfect machine learning algorithms and help them “implement” and support the autonomous learning. In this section, a number of algorithms used in this project will be examined.

3.4.1 Categorical Boosting (CatBoost)

CatBoost (Categorical Boosting) is an advanced gradient boosting algorithm that excels at handling categorical data without the need for extensive preprocessing like one-hot encoding. It's designed to work efficiently with datasets that have a mix of numerical and categorical features, making it particularly useful in a wide range of real-world applications. CatBoost is based on the principle of gradient boosting, where an ensemble of weak learners, typically decision trees, is built sequentially. Each new tree aims to correct the errors made by the previous ones, thereby improving the model's overall accuracy.

CatBoost introduces several key innovations, including an efficient way to deal with categorical features and a technique called Ordered Boosting, which helps prevent overfitting and enhances the model's generalization capabilities. The CatBoost model can be mathematically described as follows:

$$\hat{y} = F_m(x) = \sum_{m=1}^{M} v \cdot h_m(x) \tag{3.2}$$

Where:

$\hat{y} =$ Predicted output,

$F_m(x) =$ Final model after m iterations,

$M =$ Total number of iterations (or trees),

$v =$ Learning rate (controls the contribution of each tree),

$h_m(x) =$ The m_{th} weak learner (tree) trained on the residuals of the previous trees.

3.4.2 Long Short-Term Memory (LSTM)

LSTMs regulate the flow of information using a sequence of gates known as input, output, and forget gates. These gates decide what information is important to keep over time, what to reject, and what to pass through as output. The general equations governing the LSTM unit operations are:

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \quad (3.3)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (3.4)$$

$$o_t = \sigma_o(W_o x_t + U_o h_{t-1} + b_o) \quad (3.5)$$

$$C'_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (3.6)$$

$$C_t = f_t C_{t-1} + i_t C'_t \quad (3.7)$$

$$h_t = o_t \tanh C_t \quad (3.8)$$

Where:

f_t = Forget gate at time t,

i_t = Input gate at time t,

x_t = Input at timestep t,

h_t = Output state of the LSTM at timestep t,

C_t = Cell state at timestep t, representing the memory of the network,

W = Weights corresponding to each gate,

b = Biases corresponding to each gate,

σ = Sigmoid activation function,

σ_o = SoftMax activation function (multiple classes),

Tanh = Hyperbolic tangent function,

U = The time insensitive hidden state matrix.

The equations describe the structure of a simple LSTM model, including the order of operation from equations (3.3) to (3.8) that represents the movement of training data through the model. The LSTM has two major components: the cell state and the hidden state. It begins by computing the three important gates: forget, input, and output. The model then updates the cell state by combining the effects of the current input with the output of the forget gate and the previous cell state. The new hidden state is created by combining the output of the current output gate with the new cell state. Overall, controlling the

cell and hidden states at each timestep t, when combined with gate functions, allows the generation of outputs or updates for the next training phase (Michael et al., 2022).

3.4.3 Gated Recurrent Unit (GRU)

Building on the basic RNN framework, researchers developed the Gated Recurrent Unit (GRU) model to address some of RNN's drawbacks, specifically the vanishing gradient problem, which getting more significant as sequence length rises (Jain et al., 2021). GRU simplifies the LSTM architecture while retaining its ability to understand dependencies in sequence data. The GRU model combines gate functions directly into its architecture, which accelerates the flow of information. The main equations regulating GRU functionality are:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{3.9}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{3.10}$$

$$h'_t = \tanh(W_h x_t + U_h(r_t \cdot h_{t-1}) + b_h) \tag{3.11}$$

$$h_{t(GRU)} = z_t \cdot h_{t-1} + (1 - r_t) \cdot h'_t \tag{3.12}$$

where

z_t = Update gate at time t,

r_t = Reset gate at time t,

h'_t = Candidate hidden state at time t,

$h_{t(GRU)}$ = final hidden state at time t,

σ = sigmoid activation function, used to regulate the gates,

$\tanh$ = hyperbolic tangent function, used for creating the new state vector,

W = Weights for different gates and state updates,

U = Recurrent Weights for different gates and state updates,

b = Biases for different gates and state updates.

b = Biases for different gates and state updates.

These equations (3.9 to 3.12) are structured to show the sequential process of training data of the GRU model. The model efficiently computes the

reset and update gates to determine how much previous information should be passed on. The candidate hidden state is then calculated using these gates, followed by the new hidden state. This simplified method gives a fast learning and updating, making GRU particularly useful for tasks that require long dependencies and also being efficient in computation compared to LSTM.

3.4.4 Hybrid Model

In this project, hybrid machine learning models are used to because of the combined advantages of multiple algorithms, with the goal of improving typical single-model performance. The combination of **CNN-LSTM** models is chosen, as it uses Convolutional Neural Networks (CNNs) for effective spatial feature extraction and Long Short-Term Memory (LSTM) networks to capture long term temporal relationships. This combination can handle data with both spatial and sequential features which improve in extensive feature analysis and dynamic pattern recognition.

3.5 Evaluation Metrics

To assess the effectiveness of any model relative to alternatives, conducting a comparative performance analysis using recognized error or evaluation metrics is essential. In this study, the performance of the load forecasting model that based on different algorithms (CatBoost, LSTM, GRU, CNN-LSTM) are compared using evaluation metrics involving Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). Each of these metrics is valued for its capacity to appropriately assess predictive model precision. MAE represent the average of the absolute errors between predicted and actual load values, RMSE measures average error magnitude, and MAPE expresses mean absolute error as a percentage.

3.5.1 Mean Absolute Error (MAE)

Mean Absolute Error represents the average of the absolute differences between the predicted and actual values without considering the direction. It is calculated as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - y'_i\right| \tag{3.13}$$

where:

$n =$ Total number of observations in the dataset

$y_i =$ The actual observed value for the i^{th} observation

$y'_i =$ The predicted value for the i^{th} observation.

This metric provides an estimation of the error in the same unit as the measured variable, and since it is an average, it offers a singular measure of error across all predictions. Lower MAE values indicate a model with better predictive accuracy, and since it's an absolute measure, it’s not sensitive to the direction of errors, making it particularly valuable for many practical applications.

3.5.2 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE), a closely related metric, takes a slightly different approach by squaring the errors before averaging, thus giving higher weight to larger errors. The RMSE is calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - y'_i\right)^2} \tag{3.14}$$

where:

$n =$ Total number of observations in the dataset

$y_i =$ The actual observed value for the i^{th} observation

$y'_i =$ The predicted value for the i^{th} observation.

This squaring aspect makes RMSE more sensitive to outliers compared to MAE. A lower RMSE value is typically better, indicating that the model's predictions are closer to the actual values. Since RMSE measures the standard deviation of errors, it can give more insight into the variance of the prediction errors. Non-negative values with 0 being ideal, and lower values suggest a tighter fit of the model to the observed data.

3.5.3 Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) translates the error as a percentage of the actual values, allowing for a more intuitive grasp of the model's accuracy. It is calculated with the formula:

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - y'_i}{y_i}\right| \tag{3.15}$$

where

$n =$ Total number of observations in the dataset

$y_i =$ The actual observed value for the i^{th} observation

y'_i = The predicted value for the i^{th} observation.

The metric is always non-negative, with lower values indicating better predictive accuracy, and is presented as a percentage, offering a straightforward indication of model performance. These symbols and the associated calculations together form the basis for evaluating and comparing the performance of predictive models, providing insight into aspects such as the variance explained, average error magnitude, the impact of larger errors, and the relative error size.

3.6 Feature Selection

In time series forecasting model, the temporal features such as year, month and day may affect the performance of the model which are important and needed to be investigate. Temporal features that are extracted from the dataset are quarter of the year, month of the year, day of the month, hour, type of day (weekend or weekday), rolling mean, and rolling standard deviation. Also, some time lags features (Lag 1, 12, 24, 48) are created as a sample to check whether the time lags are the features that is worth to investigate for enhancing the model performance. In the context of time series analysis and forecasting, time lags are defined as a delay or change in observed data over time periods. It is a concept used to find the relationship between a variable's past and future values. The time lags feature are created using sliding window technique, which the details of the technique will be discuss during next section.

In this project, the techniques applied to check the relationship between various parameters and load demand are correlation matrix. Figure 3.8 displays a correlation matrix between load and the potential attributes listed previously.

Each of the values (Correlation coefficients) in Figure 3.8 and Table 3.5, were used to classify the strength of the relationship between load demand and the various features. Features were categorized into different correlation ranges: weak (below 0.4), moderate (0.4 to 0.7), and strong (above 0.7). This analysis was followed by training a Long Short-Term Memory (LSTM) model, where each subset of features, grouped by correlation strength, was used to evaluate model performance in terms of validation Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

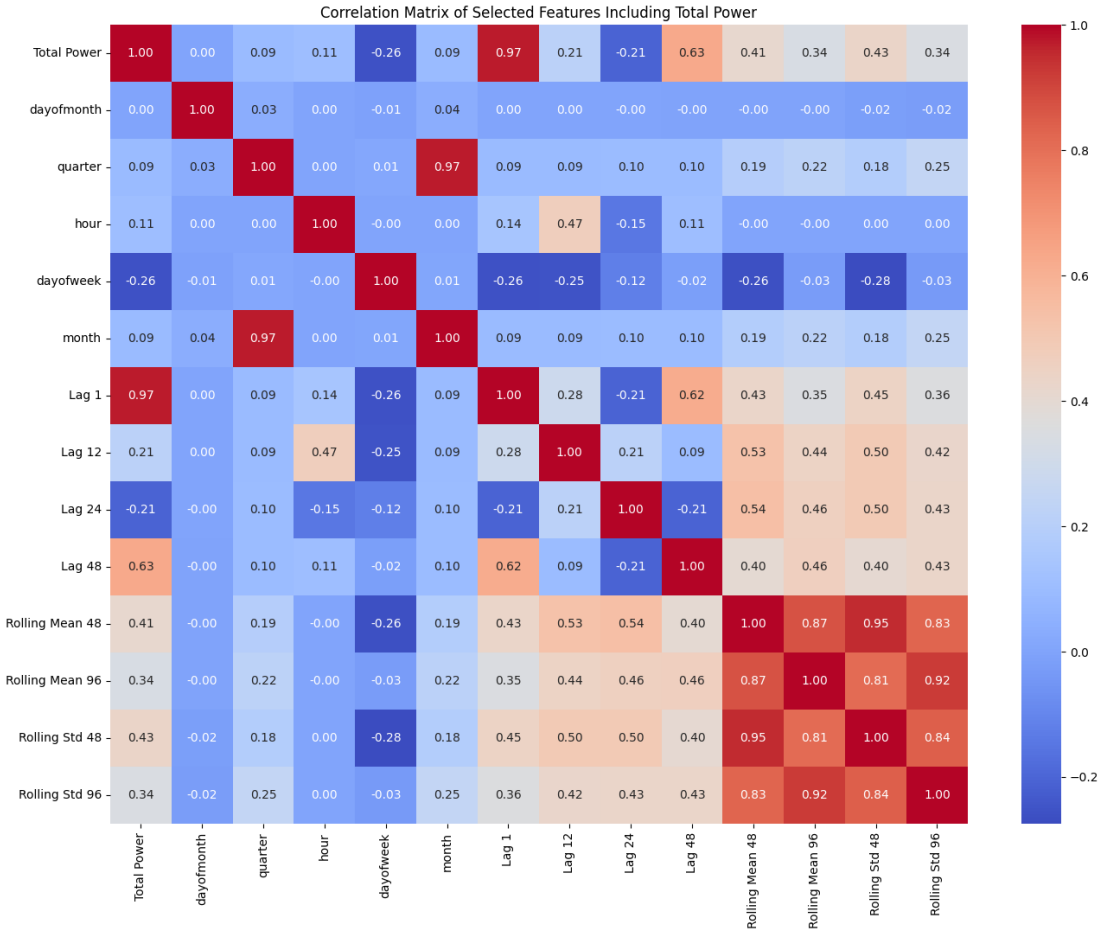


Figure 3.8: Correlation matrix of potential feature

Table 3.5: Feature Categorization by Correlation Strength

Correlation Strength	Feature
Weak	Power Factor Total
	hour
	dayofweek
	quarter
	month
	year
	dayofyear
	dayofmonth
	weekofyear
	Total Power Lag 10 - Lag 41
	Total Power Rolling Mean 96
	Total Power Rolling Std 96
Moderate	Voltage Ph-A Avg
	Voltage Ph-B Avg
	Voltage Ph-C Avg
	Total Power Lag 6 -9
	Total Power Lag 42 - 48
	Total Power Rolling Mean 48
	Total Power Rolling Std 48
Strong	Current Ph-A Avg
	Current Ph-B Avg
	Current Ph-C Avg
	Total Power Lag 1 - 5

3.6.1 LSTM

As shown in Table 3.6 below, the use of features with a higher correlation generally resulted in better model performance, as indicated by lower RMSE, MAE, and MAPE values. The "All Features" scenario, which incorporated the full set of features, yielded the best performance across all metrics. However, the use of only the last 48-time steps produced competitive results, suggesting that shorter time horizons might still be effective in certain contexts.

The findings from this feature selection process demonstrate the importance of including relevant temporal and lag features to improve the predictive power of the model. The next section will further discuss the methodology employed to create lag features and assess the impact on model accuracy.

Table 3.6: Feature Experiment of LSTM model

Algorithm	Feature Correlation	Val RMSE	Val MAE	Val MAPE
LSTM	Weak (Below 0.4)	0.0747	0.0447	0.3283
	Below 0.5	0.0570	0.0320	0.2227
	Below 0.6	0.0544	0.0318	0.2254
	Moderate (0.4 to 0.7)	0.0547	0.0284	0.1983
	Above 0.6	0.0470	0.0208	0.1291
	Strong (Above 0.7)	0.0495	0.0242	0.1707
	All	0.0437	0.0212	0.1472
	Only last 48-time steps	0.0488	0.0240	0.1681

3.6.2 GRU

For the GRU model, Table 3.7 clearly shows the trend of performance improvement as feature correlation increases. Initially, when weakly correlated features are used, the model's RMSE (0.0712) and MAPE (0.3127) are relatively high, indicating that the GRU struggles to make accurate predictions when presented with less relevant information. As the strength of the correlations improves, the model benefits from more informative inputs, reducing the RMSE to 0.0461 and MAPE to 0.1454 when using all features.

The gradual decline in errors with stronger correlations can be attributed to the GRU's ability to capture long-term dependencies more effectively when it has access to more meaningful, high-correlation features. This improvement highlights the importance of using strongly predictive temporal and lag features, which help the model retain essential historical patterns over time.

Interestingly, when only the last 48-time steps are used, the model's performance is still competitive (RMSE = 0.0473), but slightly worse than when all features are available. This indicates that although GRU can handle shorter-term data reasonably well, it benefits from a broader range of time steps and contextual features to fully capture the complexity of the load demand patterns.

Table 3.7: Feature Experiment of GRU model

Algorithm	Feature Correlation	Val RMSE	Val MAE	Val MAPE
GRU	Weak (Below 0.4)	0.0712	0.0426	0.3126
	Below 0.5	0.0555	0.0331	0.2422
	Below 0.6	0.0554	0.0304	0.2110
	Moderate (0.4 to 0.7)	0.0521	0.0271	0.1889
	Above 0.6	0.0498	0.0227	0.1386
	Strong (Above 0.7)	0.0472	0.0231	0.1626
	All	0.0461	0.0228	0.1454
	Only last 48-time steps	0.0473	0.0236	0.1731

3.6.3 CatBoost

In the CatBoost feature experiment, the model's performance improves significantly as feature correlation with the target variable increases. When using highly correlated features (above 0.6 and 0.7), the model shows strong performance, with low RMSE (0.0039) and MAPE (0.0171), demonstrating its ability to effectively capture trends when features have a clear relationship with the target. However, the best overall performance is achieved when all features are used, yielding the lowest RMSE (0.0024) and MAPE (0.0134). This indicates that CatBoost benefits from having access to a diverse set of features, even those that are only weakly correlated, as it still contribute valuable information for prediction.

When only weakly correlated features (below 0.4) are used, the model struggles, with RMSE rising to 0.0772 and MAPE to 0.4581, showing that CatBoost needs sufficient correlation in features to make accurate predictions. Additionally, when restricted to only the last 48-time steps, the model's performance declines further (RMSE = 0.0687, MAPE = 0.4154), highlighting that CatBoost performs best when provided with a wide temporal scope and a comprehensive set of features.

Overall, the results in Table 3.8 underscore CatBoost's reliance on a well-engineered feature set, as it cannot automatically extract patterns from raw data like deep learning models can.

Table 3.8: Feature Experiment of CatBoost model

Algorithm	Feature Correlation	Val RMSE	Val MAE	Val MAPE
CatBoost	Weak (Below 0.4)	0.0771	0.0435	0.4580
	Below 0.5	0.0389	0.0233	0.2611
	Below 0.6	0.0370	0.0224	0.2514
	Moderate (0.4 to 0.7)	0.0619	0.0355	0.4051
	Above 0.6	0.0038	0.0019	0.0170
	Strong (Above 0.7)	0.0041	0.0021	0.0193
	All Features	0.0023	0.0013	0.0134
	Only Last 48-Time Steps	0.0687	0.0368	0.4154

3.6.4 CNN-LSTM

For the CNN-LSTM model, Table 3.9 the performance improves progressively as feature correlation strengthens. When weakly correlated features (below 0.4) are used, the RMSE is relatively high (0.0652), indicating that the model struggles to capture meaningful patterns from such features. As the correlation increases, the error metrics improve, with RMSE dropping to 0.0464 and MAPE to 0.1389 for features with correlations above 0.6.

The best performance is achieved when all features are used, with the lowest RMSE (0.0442) and MAPE (0.1290). This result suggests that the CNN-LSTM model benefits from a wide range of features, including both weakly and strongly correlated ones, as the combination of CNN and LSTM layers allows it to learn spatial patterns (through CNN) and temporal dependencies (through LSTM).

However, when the model is restricted to using only the last 48-time steps, performance significantly declines (RMSE = 0.0801, MAPE = 0.2943), indicating that the model requires more information to capture longer-term trends effectively.

Table 3.9: Feature Experiment of CNN-LSTM model

Algorithm	Feature Correlation	Val RMSE	Val MAE	Val MAPE
CNN-LSTM	Weak (Below 0.4)	0.0651	0.0358	0.2677
	Below 0.5	0.0502	0.0276	0.2060
	Below 0.6	0.0502	0.0287	0.2055
	Moderate (0.4 to 0.7)	0.0552	0.0316	0.2260
	Above 0.6	0.0464	0.0213	0.1389
	Strong (Above 0.7)	0.0513	0.0297	0.2174
	All	0.0442	0.0199	0.1290
	Only last 48-time steps	0.0801	0.0429	0.2943

3.7 Data Preprocessing

Data preparation represents the essential part of preprocessing in the load forecasting workflow, and it serves as a basis for any further analysis and modelling. The preprocessing is carried out through the explicit stages of data cleaning and prior conversion of raw data, feature engineering, data normalization, and dimensionality reduction. Notably, the special attention is given to the step of splitting the data into training and testing sets, as well as ensuring the chronological correctness of time series data, which is crucial for developing forecasting techniques. Overall, the process of data preprocessing is aimed at systematically refining the data to improve the model's ability to recognize and analyses the data more effectively and understand the temporal patterns associated with load forecasting.

3.7.1 Sliding Window

The approach for transforming time series data into a format suitable for multiple-step forecasting is discussed in this section, and Figure 3.9 shows a graphic illustration. X_t represents the dataset at time "t". The conversion method entails restructuring the time series into lagged features, which relocate old data points as input variables to forecast future values. This reshaping is carried out via a sliding window methodology, in which a window of a predetermined size is slid over the time series, yielding sequences of input-output pairings. The

window size was carefully designed to capture the dataset's essential temporal connections and trends.

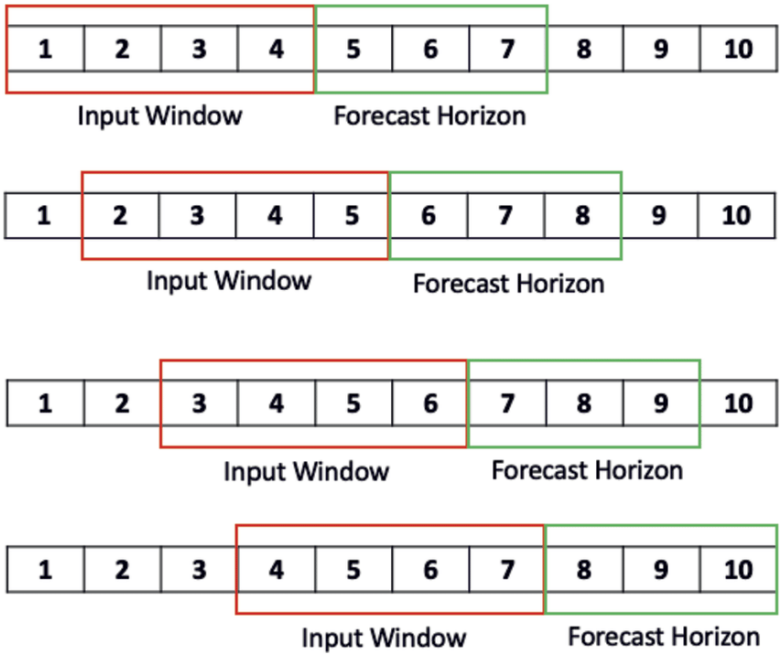


Figure 3.9: Illustration of sliding window technique.

The load predictions is focus on only time variable, which are trained on historical data from time step $t - b$ to t, "t" denotes the current moment, whereas "b" defines the number of preceding time steps, which serve as the model's look-back period.

The sliding window technique is critical in this situation because it ensures that the model receives up-to-date data by moving the window incrementally, one time step at a time. This development enables for the continuous incorporation of recent data, which is critical for capturing the changing trends in the time series for accurate future forecasts.

3.7.2 Min-Max Technique

The Min-Max normalization technique is based on the concept of feature scaling, which involves rescaling data attributes or features so that the values fall inside a specified and predefined interval, most often [0,1]. This normalization method is apply to scales the electrical load data without distorting or losing information. It is done by subtracting the minimum value from each characteristic and then dividing by its range. The mathematical expression as following:

$$x' = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (3.16)$$

where

$X =$ The original value,

$X_{min} =$ The minimum values of the feature,

$X_{max} =$ maximum values of the feature,

$x' =$ Normalized value.

Min-Max method normalize the data such that it has the same scale, which can be benefit for algorithms that are sensitive to data scale. LSTMs and GRUs and many deep learning algorithms work better when features are on a comparable scale. One of the significant advantages of performing this normalization is with respect to the performance of machine learning algorithms especially in terms of convergence speed. Moreover, simple algorithms like k-NN also reap the benefit in the sense that each feature contributes equally to the distance measurements.

3.8 Hyperparameter Tuning

The efficiency of machine learning models is depending on the selection of hyperparameters, select correct parameter can improve the model learning process which can achieve better performance. Due to the complexity of machine learning model structures, determining the best hyperparameter combination is important for achieving optimal performance.

3.8.1 Grid Search Method

In this research, the numbers of neurons in the hidden layer, activation function, batch size, optimizer function, window size are the hyperparameters going to focus and optimize in this research as shown in Table 3.4. Using grid search to explore every potential hyperparameter combination can be time consuming and is not practical for large hyperparameter combination. A more practical technique is to focus on a specific group of hyperparameters and adjust them periodically depending on feedback from each iteration. This incremental

method can get the optimal or near optimal hyperparameters for the load forecasting model in more efficient ways.

3.8.2 LSTM Tunning

3.8.2.1 Phase 1: Initial Investigation

In the first phase, 50 LSTM units, dropout rates of 0.01 and 0.1, and learning rates of 0.1, 0.01, and 0.001 across 1, 2, and 3 layers investigated and shown in Table 3.10. The results showed that a dropout rate of 0.01 consistently outperformed 0.1, especially with more complex configurations, reducing overfitting. The learning rate of 0.001 was optimal, allowing stable convergence, while higher rates caused higher errors. Increasing the number of layers to 2 improved performance, but 3 layers led to overfitting without careful tuning of dropout and learning rates.Increasing the number of LSTM layers to 2 and 3 provided mixed results. Although adding a second layer improved performance in some cases, using 3 layers generally led to overfitting unless combined with lower dropout rates and learning rates. Overall, the key finding in this phase is dropout rate of 0.01 and learning rate of 0.001 yielding a better result in different combinations.

Table 3.10: Hyperparameter Tunning of LSTM model (Phase 1)

Phase 1							
Units	Dropout Rate	Layers	Learning Rate	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
50	0.01	1	0.1	0.0685	0.0301	0.1795	0.1769
			0.01	0.0503	0.0235	0.1565	0.0568
			0.001	0.0506	0.0232	0.1507	0.0535
		2	0.1	0.2777	0.2302	1.9620	2.9690
			0.01	0.0483	0.0234	0.1590	0.0134
			0.001	0.0477	0.0219	0.1533	0.0366
		3	0.1	0.2808	0.2122	1.5679	2.6834
			0.01	0.0485	0.0226	0.1565	0.0404
			0.001	0.0480	0.0221	0.1288	0.0000
	0.1	1	0.1	0.0663	0.0421	0.3222	0.3019
			0.01	0.0530	0.0312	0.1350	0.0347
			0.001	0.0519	0.0229	0.1484	0.0563
		2	0.1	0.0687	0.0410	0.3575	0.3258
			0.01	0.0535	0.0335	0.1387	0.0552
			0.001	0.0495	0.0218	0.1483	0.0409
		3	0.1	0.2851	0.2022	1.3397	2.5306
			0.01	0.0509	0.0326	0.1443	0.0484
			0.001	0.0488	0.0233	0.1312	0.0421

3.8.2.2 Phase 2: Refinement and Further Adjustments

Building on the insights from Phase 1, the second phase involved refining the model by increasing the number of LSTM units to 64 and 128 and further experimenting with the number of layers as shown in Tble 3.10. Building on Phase 1, the number of LSTM units was increased to 64 and 128 (Table 3.11). The best results were obtained with 64 units, 3 layers, 0.01 dropout rate, and a learning rate of 0.001, achieving a validation RMSE of 0.0465. Further increasing the number of units to 128 or layers to 4 did not improve performance and, in some cases, worsened it. Overall, the key finding is that 64 units and 3 layers offered an ideal balance between model complexity and generalization.

Table 3.11: Hyperparameter Tunning of LSTM model (Phase 2)

Phase 2							
Units	Dropout Rate	Layers	Learning Rate	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
64	0.01	3	0.001	0.0465	0.0201	0.1108	0.0000
		4	0.001	0.0472	0.0205	0.1109	0.0049
	0.1	3	0.001	0.0509	0.0216	0.1303	0.0361
		4	0.001	0.0487	0.0241	0.1693	0.0599
128	0.01	3	0.001	0.0495	0.0236	0.1713	0.0619
		4	0.001	0.0506	0.0233	0.1512	0.0542
	0.1	3	0.001	0.0487	0.0219	0.1389	0.0330
		4	0.001	0.0485	0.0216	0.1360	0.0291

3.8.2.3 Optimizer and Activation Function

The LSTM model was then evaluated with four activation functions, ReLU, tanh, sigmoid, and linear, and optimized with various optimizers, including SGD, Adam, Adamax, and Regularizer as shown in Table 3.12. Across all activation functions, the Adam optimizer consistently outperformed the others. For example, when using the ReLU activation function, Adam achieved a validation loss of 0.002111, the lowest for this model. In comparison, SGD produced much higher losses, with a validation loss of 0.065203 for ReLU. Similarly, linear activation combined with Adam resulted in a low validation loss of 0.002271, though still higher than ReLU.

Other activation functions, like tanh and sigmoid, also performed well with Adam, but neither surpassed ReLU or linear in performance. For instance, tanh with Adam achieved a validation loss of 0.002397, while sigmoid reached 0.00249. When using SGD or Regularizer, the model showed significantly worse performance, indicating that these optimizers were less effective for the LSTM model. Overall, Adam with ReLU was the best combination for minimizing validation loss in the LSTM model.

Table 3.12: Optimizer Tunning of LSTM model

Model	Activation Function	Optimizer	Best Validation Loss
LSTM	relu	SGD	0.065203
		Adam	0.002111
		Adamax	0.002229
		Regularizer	0.088231
	tanh	SGD	0.038712
		Adam	0.002397
		Adamax	0.0026
		Regularizer	0.0676
	sigmoid	SGD	0.088701
		Adam	0.00249
		Adamax	0.003156
		Regularizer	0.084181
	linear	SGD	0.036673
		Adam	0.002271
		Adamax	0.00255
		Regularizer	0.010772

3.8.2.4 Summary Flowchart

Figure 3.10 shows the summary process of LSTM hyperparameters tunning process. It is found that the best hyperparameters combination for LSTM under the case study of the double storey research office are **64 neurons units**, **3 LSTM layers**, **dropout 0.01**, **learning rate 0.001**, **RELU activation function** with **Adam optimizer**.

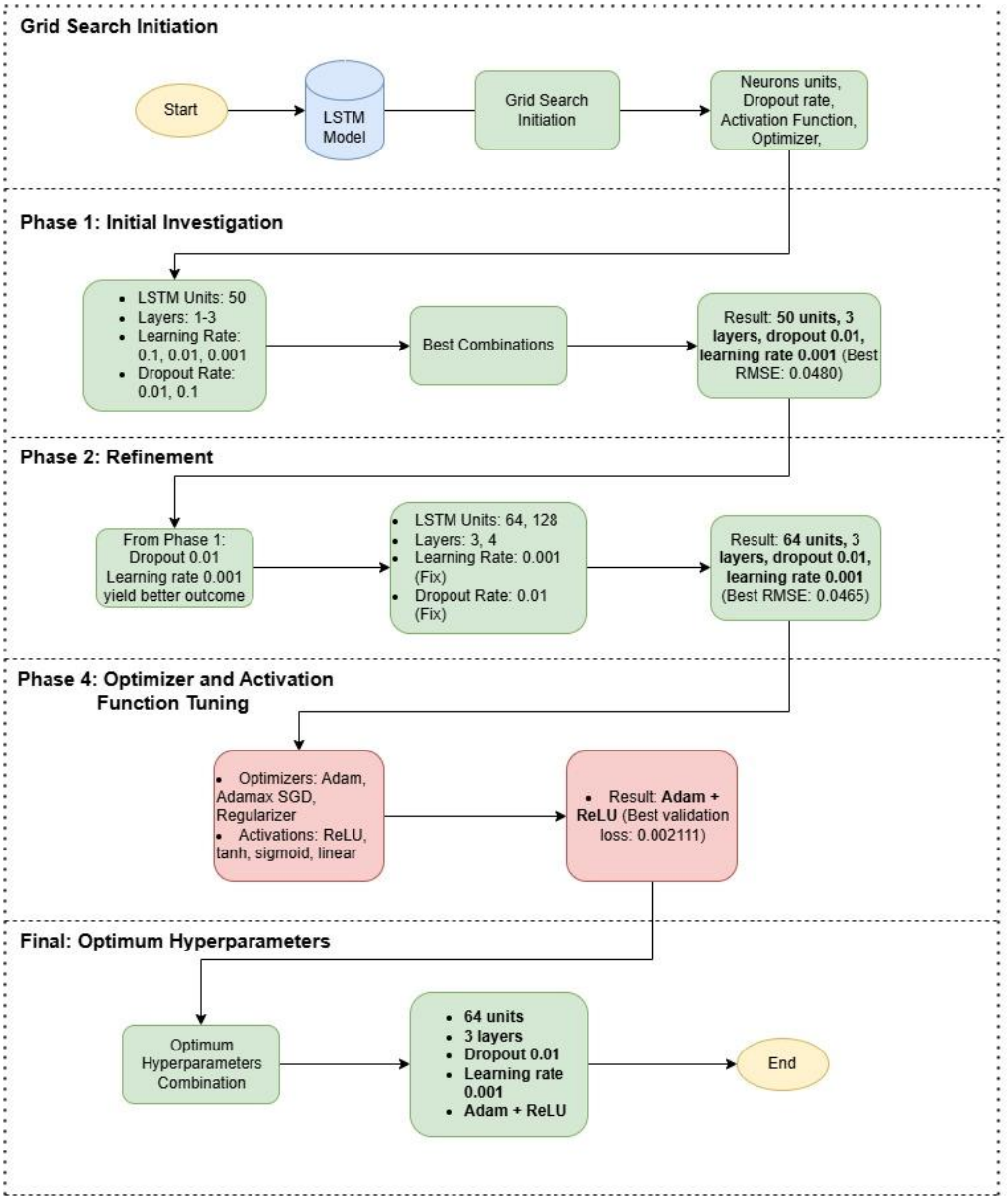


Figure 3.10: Flowchart of LSTM Tunning Process

3.8.3 GRU Tunning

3.8.3.1 Phase 1: Initial Investigation

Phase 1 involved testing the GRU model with 50 units, dropout rates of 0.01 and 0.1, learning rates of 0.1, 0.01, and 0.001, and varying the number of layers between 1, 2, and 3 (Table 3.13). Results indicated that a dropout rate of 0.01 consistently outperformed 0.1, particularly in deeper architectures, as the higher dropout rate caused over-regularization and hindered the model's ability to learn. For example, with 1 layer and a dropout rate of 0.01, the model achieved an RMSE of 0.0503 at a learning rate of 0.001, whereas using a dropout of 0.1 resulted in significantly worse performance (RMSE = 0.2254 for 3 layers).

The learning rate of 0.001 was the most stable, as higher rates like 0.1 led to unstable convergence and overshooting during training. For instance, the model achieved an RMSE of 0.0489 with 2 layers and a learning rate of 0.001, but higher rates produced larger errors. Increasing the number of layers improved performance initially, but adding a third layer often resulted in diminishing returns or overfitting, especially with larger dropout rates. Overall, the key finding in this phase is dropout rate of 0.01 and learning rate of 0.001 yielding a better result in different combinations.

Table 3.13: Hyperparameters Tunning of GRU model (Phase 1)

Phase 1							
Units	Dropout Rate	Layers	Learning Rate	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
50	0.01	1	0.1	0.0594	0.0310	0.2364	0.2217
			0.01	0.0560	0.0297	0.1909	0.1363
			0.001	0.0503	0.0285	0.1450	0.0299
		2	0.1	0.1098	0.0951	1.1493	1.6826
			0.01	0.0697	0.0262	0.2154	0.1236
			0.001	0.0509	0.0219	0.1385	0.0215
		3	0.1	0.2254	0.1535	1.4118	2.9373
			0.01	0.0580	0.0356	0.3175	0.3125
			0.001	0.0499	0.0258	0.1410	0.0000
	0.1	1	0.1	0.0642	0.0355	0.2754	0.3115
			0.01	0.0565	0.0277	0.2237	0.1717
			0.001	0.0511	0.0232	0.1690	0.0670
		2	0.1	0.2101	0.1260	0.4964	1.9352
			0.01	0.0621	0.0299	0.2147	0.2110
			0.001	0.0529	0.0221	0.1569	0.0426
		3	0.1	0.2373	0.1410	0.7684	2.4042
			0.01	0.0601	0.0295	0.1904	0.1788
			0.001	0.0530	0.0278	0.1615	0.0232

3.8.3.2 Phase 2: Investigating Deeper Layers

Building on Phase 1, the second phase involved increasing the number of GRU units to 64 and 128, and testing with 3 and 4 layers as shown in Table 3.14. The best results were obtained with 64 units, 3 layers, dropout rate of 0.01, and a learning rate of 0.001, achieving an RMSE of 0.0480 and MAPE of 0.1310. Increasing the number of units to 128 did not lead to significant improvements, and in some cases worsened performance (e.g., RMSE = 0.0482). Similarly, increasing the number of layers to 4 did not yield meaningful gains and often led to overfitting, as seen with an RMSE of 0.0477 for 4 layers and 128 units.

The dropout rate of 0.01 continued to outperform 0.1, especially in models with more layers, where a higher dropout rate caused increased validation errors. For instance, with 4 layers and a dropout rate of 0.1, the RMSE was 0.0518, compared to lower RMSE values with a dropout of 0.01. The learning rate of 0.001 remained optimal across all configurations, allowing smooth convergence without the instability associated with higher learning rates.

Table 3.14: Hyperparameters Tunning of GRU model (Phase 2)

Phase 2							
64	0.01	3	0.001	0.0480	0.0208	0.1310	0.0027
		4	0.001	0.0489	0.0234	0.1545	0.0454
	0.1	3	0.001	0.0481	0.0219	0.1298	0.0106
		4	0.001	0.0541	0.0254	0.1484	0.0833
128	0.01	3	0.001	0.0482	0.0225	0.1688	0.0461
		4	0.001	0.0477	0.0217	0.1318	0.0082
	0.1	3	0.001	0.0492	0.0230	0.1603	0.0487
		4	0.001	0.0518	0.0244	0.1481	0.0635

3.8.3.3 Optimizer and Activation Function

The GRU model underwent a similar evaluation using the same set of activation functions and optimizers as shown in Table 3.15. Unlike the LSTM model, where Adam was the clear winner, the Adamax optimizer produced the best performance for the GRU model. When paired with the ReLU activation function, Adamax achieved a validation loss of 0.002139, slightly outperforming Adam, which recorded a loss of 0.002233. This suggests that Adamax is marginally more effective in optimizing the GRU model with ReLU.

The tanh activation function also performed reasonably well with Adamax, achieving a validation loss of 0.002634, although it did not match the results of ReLU. On the other hand, the sigmoid activation was the least effective for the GRU model, especially when combined with SGD, which produced the highest validation loss of 0.089876. Overall, Adamax with ReLU emerged as the best combination for minimizing validation loss in the GRU model, offering the lowest validation loss among all tested configurations.

Table 3.15: Optimizer Tunning of GRU model

Model	Activation Function	Optimizer	Best Validation Loss
GRU	relu	SGD	0.023053
		Adam	0.002233
		Adamax	0.002139
		Regularizer	0.009657
	tanh	SGD	0.019836
		Adam	0.002758
		Adamax	0.002634
		Regularizer	0.009821
	sigmoid	SGD	0.089876
		Adam	0.002603
		Adamax	0.002873
		Regularizer	0.082995
	linear	SGD	0.01705
		Adam	0.002659
		Adamax	0.002593
		Regularizer	0.072548

3.8.3.4 Summary Flowchart

Figure 3.11 shows the summary process of GRU model hyperparameters tunning process. It is found that the best hyperparameters combination for GRU under the case study of the double storey research office are **64 neurons units**, **3 GRU layers**, **dropout 0.01**, **learning rate 0.001**, **RELU activation function** with **Adamax optimizer**.

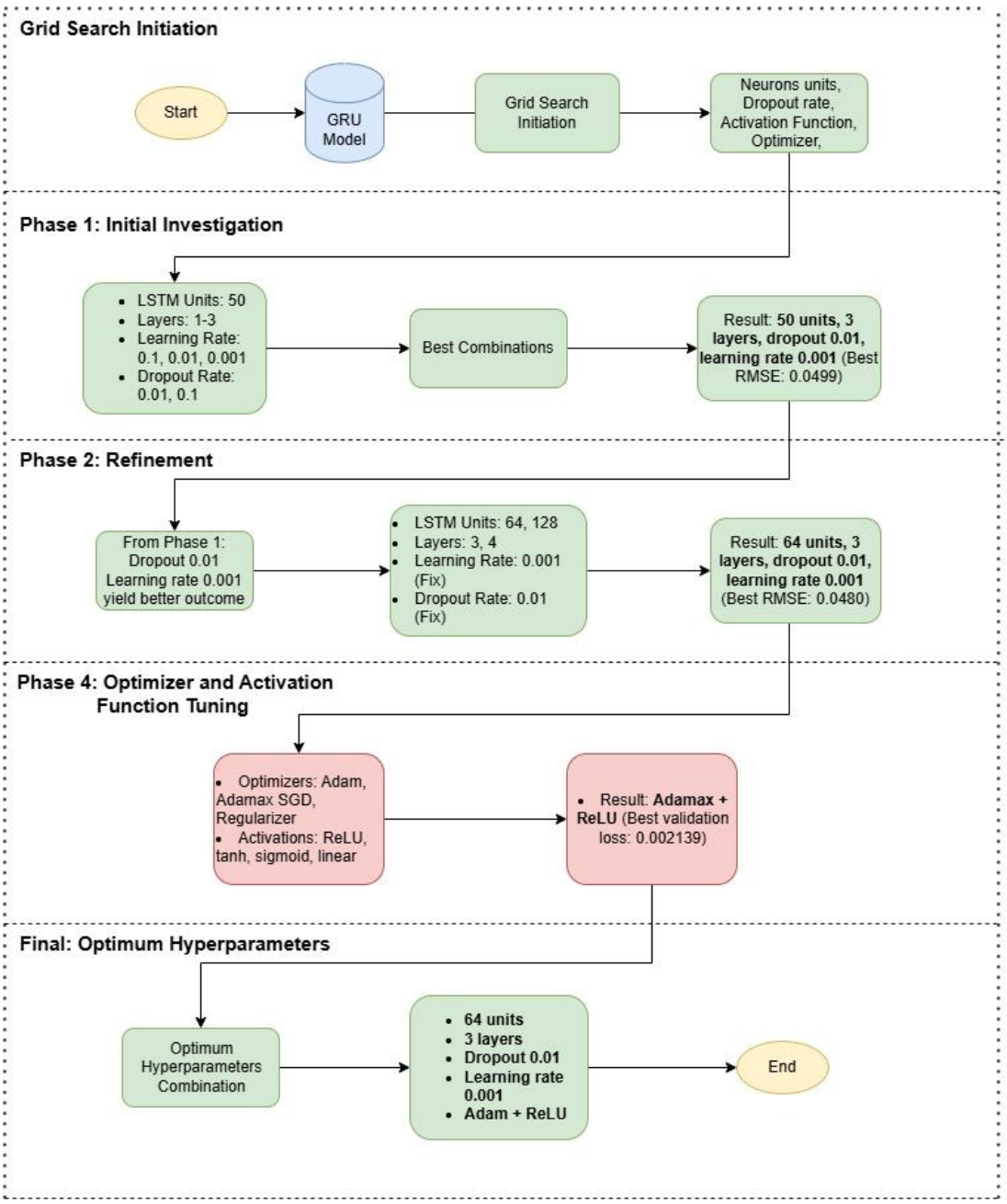


Figure 3.11: Flowchart of GRU Tunning Process

3.8.4 CatBoost Tunning

The hyperparameter tuning for the CatBoost model was conducted across three phases, each progressively refining the search for the optimal configuration. The primary focus was on tuning depth, learning rate, and L2 regularization, followed by additional parameters such as Border Count and Random Seed. The goal throughout was to minimize the validation metrics, including Val RMSE, Val MAE, Val MAPE, and the Total Normalized Score. The overall tunning result is shown in Table 3.14, 3.15 and 3.16.

3.8.4.1 Phase 1: Initial Hyperparameter Search

The first phase focused on finding an effective balance between learning rate (0.001, 0.01, and 0.1) and L2 regularization (1, 3, and 5) with a model depth of 4. Table 3.16 illustrates that a learning rate of 0.001 yielded poor results, with a Val RMSE of 0.0672 and a Total Normalized Score of 1.9253 after 1000 iterations (L2 = 1), indicating that the model was underfitting. Increasing L2 regularization to 3 or 5 did not substantially improve performance.

Switching to a learning rate of 0.01 reduced errors, as the Val RMSE dropped to 0.0062, and the Total Normalized Score improved to 0.1060 (L2 = 1). However, the results still left room for further optimization. The best performance was achieved with a learning rate of 0.1, which lowered the Val RMSE to 0.0024, with a Total Normalized Score of 0.0007 at L2 = 1. Increasing L2 beyond 1 caused slight increases in the error metrics, suggesting that overfitting was beginning to occur. Based on these findings, the combination of depth = 4, learning rate = 0.1, and L2 = 1 was selected as the optimal configuration for this phase.

Table 3.16: Hyperparameters Tunning of CatBoost model (Phase 1)

Phase 1							
Depth	Learning Rate	L2 Leaf Reg	Iterations	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
4	0.001	1	500	0.1043	0.0745	1.0411	2.9967
			1000	0.0672	0.0474	0.6939	1.9253
		3	500	0.1044	0.0746	1.0414	2.9985
			1000	0.0674	0.0474	0.6946	1.9281
		5	500	0.1045	0.0746	1.0416	3.0000
			1000	0.0675	0.0475	0.6945	1.9298
	0.01	1	500	0.0094	0.0060	0.0903	0.2064
			1000	0.0062	0.0039	0.0486	0.1060
		3	500	0.0095	0.0060	0.0911	0.2093
			1000	0.0063	0.0039	0.0485	0.1067
		5	500	0.0095	0.0060	0.0906	0.2088
			1000	0.0063	0.0039	0.0487	0.1074
	0.1	1	500	0.0032	0.0020	0.0205	0.0247
			1000	0.0024	0.0014	0.0134	0.0007
		3	500	0.0033	0.0021	0.0211	0.0272
			1000	0.0025	0.0014	0.0138	0.0028
		5	500	0.0033	0.0021	0.0219	0.0282
			1000	0.0025	0.0014	0.0140	0.0032

3.8.4.2 Phase 2: Investigating Deeper Models

Phase 2 extended the analysis by testing deeper models with depths of 6 and 8 asshown in Tabl 3.17, using the best configurations from Phase 1 (learning rates of 0.001, 0.01, and 0.1; L2 values of 1, 3, and 5). The purpose was to evaluate whether increasing the model depth would yield further improvements.

At depth = 6, the model's performance was very similar to depth = 4, with a Val RMSE of 0.0027 and a Total Normalized Score of 0.0033 (L2 = 1). Although the difference was minor, it indicated that the model's increased complexity was not justified by a significant gain in accuracy. Further increasing the depth to 8 did not result in better performance, with the best Val RMSE reaching 0.0035 and a Total Normalized Score of 0.0129 (L2 = 1). Higher L2 regularization values (3 and 5) led to increased error metrics, signaling that overfitting became more of a concern as depth increased.

The results showed that increasing model depth beyond 4 did not provide meaningful improvements and only introduced the risk of overfitting, with marginal gains in validation metrics. Thus, depth = 4 remained the preferred configuration due to its balance between accuracy and model complexity.

Table 3.17: Hyperparameters Tunning of CatBoost model (Phase 2)

Phase 2							
Depth	Learning Rate	L2 Leaf Reg	Iterations	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
6	0.1	1	1000	0.0027	0.0014	0.0127	0.0033
		3	1000	0.0026	0.0014	0.0136	0.0041
		5	1000	0.0029	0.0015	0.0139	0.0077
8	0.1	1	1000	0.0035	0.0015	0.013	0.0129
		3	1000	0.0037	0.0017	0.0142	0.0187
		5	1000	0.0036	0.0017	0.0146	0.0184

3.8.4.3 Phase 3: Refining the Model with Additional Parameters

In Phase 3, additional parameters like Border Count and Random Seed were explored as shown in Tabl 3.18. The best configuration from Phase 1—depth = 4, learning rate = 0.1, and L2 = 1—was retested, yielding the same optimal results (Val RMSE = 0.0033, Total Normalized Score = 0.0007). Border Count and Random Seed variations had minimal effect on performance, showing that once the core hyperparameters were optimized, these additional parameters did not provide significant gains. The patterns held for depths 6 and 8, where increasing L2 again led to overfitting.

Table 3.18: Hyperparameters Tunning of CatBoost model (Phase 3)

Phase 3								
Depth	Learning Rate	L2 Reg	Border Count	Random Seed	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
4	0.001	1	32	7	0.0674	0.0476	0.6969	2.9934
	0.1	1	128	42	0.0033	0.0018	0.0155	0.0007
	0.05	1	64	7	0.0058	0.0028	0.0241	0.0772
	0.1	5	128	7	0.0035	0.0019	0.0171	0.0032
	0.05	3	128	42	0.0038	0.0021	0.0201	0.0322
6	0.001	1	128	7	0.0658	0.0465	0.6687	2.9064
	0.05	1	64	7	0.0052	0.0024	0.0198	0.0598
	0.1	1	128	42	0.0034	0.0018	0.0144	0.0033
	0.1	5	128	123	0.0035	0.0019	0.0151	0.0051
	0.05	3	64	7	0.0054	0.0025	0.02	0.0627
8	0.001	1	128	42	0.0653	0.0461	0.6561	2.8725
	0.1	1	128	123	0.0035	0.0018	0.0142	0.0129
	0.05	1	64	42	0.0052	0.0026	0.0215	0.0604
	0.1	5	128	42	0.0039	0.0019	0.0174	0.0187
	0.1	3	64	7	0.0036	0.0018	0.0148	0.0096
10	0.001	1	128	42	0.0654	0.0461	0.6474	2.8569
	0.05	1	64	42	0.0055	0.0027	0.021	0.067
	0.1	1	128	7	0.0044	0.0021	0.0164	0.0271
	0.1	5	128	123	0.0049	0.0023	0.0188	0.0387
	0.05	3	64	42	0.0056	0.0028	0.0209	0.0621

3.8.4.4 Optimizer and Activation Function

Optimizer and activation function tuning are not required for CatBoost because it is a gradient-boosting algorithm that operates differently from neural networks. CatBoost optimizes its performance through boosting trees, using gradient descent for decision tree-based learning. Unlike neural networks, which rely on optimizers like Adam or SGD and activation functions like ReLU or sigmoid to update weights and capture non-linear patterns, CatBoost builds a series of decision trees where each tree corrects the errors of the previous one (Hancock & Khoshgoftaar, 2020). Therefore, no activation function or traditional optimizer tuning is necessary for CatBoost models.

3.8.4.5 Summary Flowchart

Figure 3.12 shows the summary process of CatBoost model hyperparameters tunning process. **depth = 4, learning rate = 0.1, and L2 regularization = 1** was selected as the optimal configuration for CatBoost under the case study of the double storey research office.

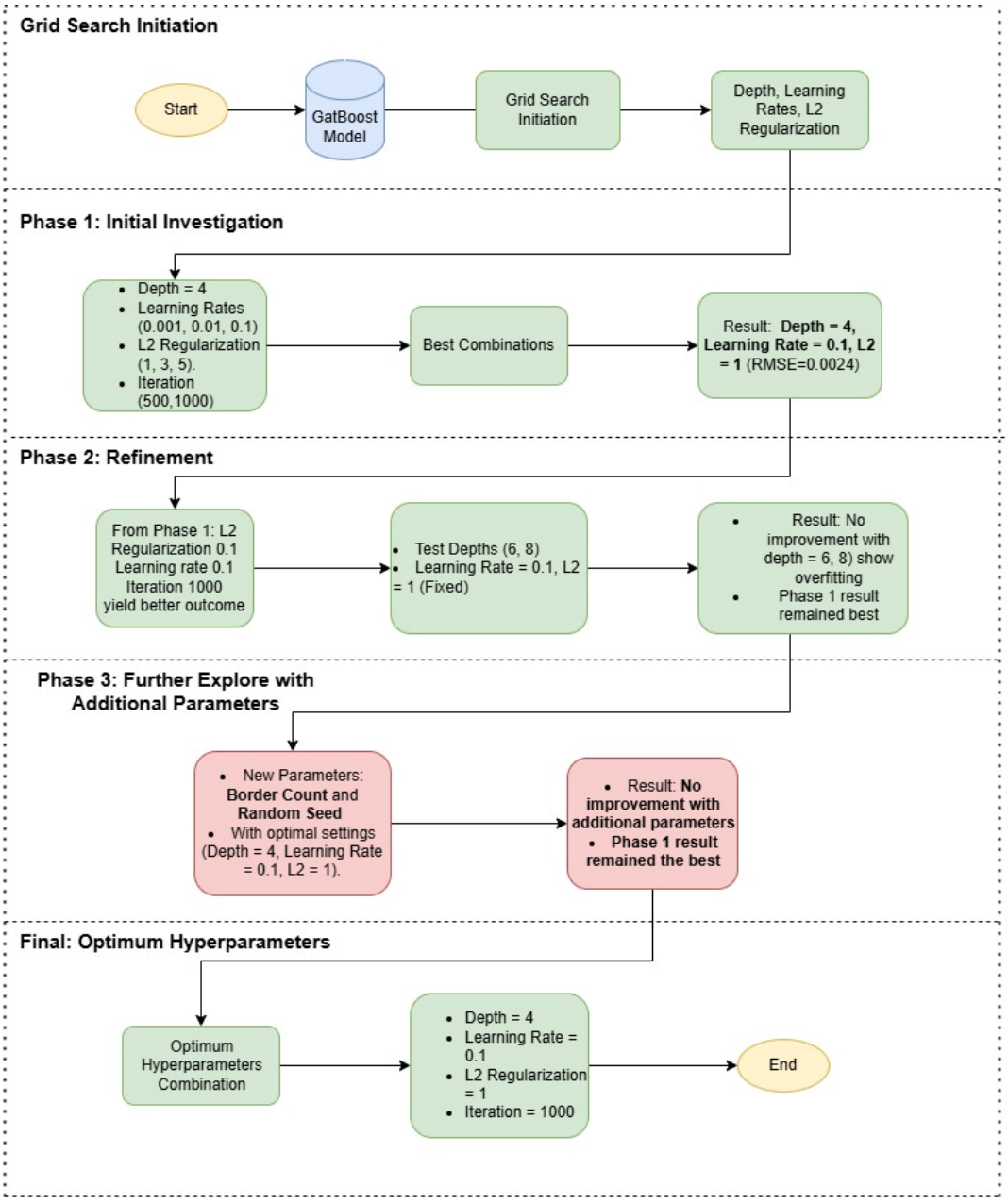


Figure 3.12: Flowchart of CatBoost Tunning Process

3.8.5 CNN-LSTM Tunning

3.8.5.1 Phase 1: Initial Hyperparameter Search

In Table 3.19, several key hyperparameters were tuned for the CNN-LSTM model, including filters, kernel size, LSTM units, dropout rates, learning rates, and the number of convolutional and LSTM layers. Initial configurations, such as 8 filters, 50 LSTM units, and kernel sizes of 1, were fixed, which focusing on the effect of changing dropout rate, learning rate, convolutional layers and LSTM layers.

When increasing the number of convolutional layers from 1 to 2 significantly improved performance, particularly when paired with 1 LSTM layer. For instance, using 8 filters, kernel size 1, 2 convolutional layers, and 1 LSTM layer yielded a validation RMSE of 0.0521, compared to 0.0580 with only 1 convolutional layer. However, adding a second LSTM layer led to inconsistent results, sometimes worsening the RMSE (e.g., 0.0605). Another example which highlighted with green color also shows better performance with 2 convolutional layers and 1 LSTM layer when dropout rate is equal to 0.1. The key finding in this phase is that dropout rate 0.01, learning rate 0.001, 2 convolutional layers and 1 LSTM layers yield a better result.

Table 3.19: Hyperparameters Tunning of CNN-LSTM model (Phase 1)

Phase 1										
Filters	Kernel Size	LSTM Units	Dropout Rate	Learning Rate	Conv Layers	LSTM Layers	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
8	1	50	0.01	0.001	1	1	0.0580	0.0272	0.1719	0.1301
						2	0.0573	0.0284	0.2010	0.1661
					2	1	0.0521	0.0232	0.1288	0.0357
						2	0.0605	0.0286	0.1910	0.1692
				0.01	1	1	0.0600	0.0282	0.1543	0.1248
						2	0.0653	0.0298	0.1835	0.1872
					2	1	0.0582	0.0266	0.1621	0.1164
						2	0.0562	0.0279	0.1844	0.1409
			0.1	0.001	1	1	0.0573	0.0266	0.1681	0.1200
						2	0.0602	0.0288	0.1717	0.1480
					2	1	0.0528	0.0240	0.1453	0.0614
						2	0.0583	0.0296	0.2216	0.1999
				0.01	1	1	0.0642	0.0308	0.1828	0.1878
						2	0.0611	0.0321	0.2129	0.2166
					2	1	0.0593	0.0264	0.1646	0.1228
						2	0.0575	0.0278	0.1646	0.1239

3.8.5.2 Phase 2: Investigating Deeper Models

In the second phase, increasing the number of filters from 8 to 16 produced the best results as show in Table 3.20. The configuration of 16 filters, 64 LSTM units, and 3 convolutional layers delivered a validation RMSE of 0.0446 and validation MAE of 0.0218. Additionally, kernel size 1 outperformed kernel size 3, showing that smaller kernels captured the patterns in the power consumption data more effectively.

The optimal setup, featuring 16 filters, 3 convolutional layers, 1 LSTM layer, 64 LSTM units, kernel size 1, a dropout rate of 0.01, and a learning rate of 0.001, delivered the best overall performance. This configuration balanced complexity and accuracy, making it ideal for the dataset.

Table 3.20: Hyperparameters Tunning of CNN-LSTM model (Phase 2)

Phase 2										
Filters	Kernel Size	LSTM Units	Dropout Rate	Learning Rate	Conv Layers	LSTM Layers	Val RMSE	Val MAE	Val MAPE	Total Normalized Score
8	1	50	0.01	0.001	2	1	0.0510	0.0254	0.1885	2.3965
					3		0.0471	0.0243	0.1589	1.1480
		64	0.01	0.001	2		0.0482	0.0233	0.1572	1.1004
					3		0.0484	0.0245	0.1838	1.7189
	3	50	0.01	0.001	2		0.0466	0.0222	0.1492	0.5052
					3		0.0481	0.0267	0.2144	2.5506
		64	0.01	0.001	2		0.0481	0.0248	0.1514	1.3078
					3		0.0472	0.0225	0.1479	0.6543
16	1	50	0.01	0.001	2		0.0493	0.0248	0.1586	1.5899
					3		0.0491	0.0248	0.1634	1.6422
		64	0.01	0.001	2		0.0473	0.0252	0.1891	1.7812
					3		0.0446	0.0218	0.1400	0.0000
	3	50	0.01	0.001	2		0.0459	0.0234	0.1668	0.8854
					3		0.0456	0.0226	0.1690	0.6998
		64	0.01	0.001	2		0.0482	0.0249	0.1758	1.6679
					3		0.0471	0.0247	0.1852	1.5885
32	1	64	0.01	0.001	3		0.0487	0.0260	0.1836	2.0926
					4		0.0473	0.0232	0.1513	0.8497
			0.01	0.001	3		0.0484	0.0255	0.1864	1.9727
					4		0.0474	0.0253	0.1639	1.4731

3.8.5.3 Optimizer and Activation Function

Table 3.21 The most effective configuration for this model was the Adamax optimizer combined with the ReLU activation function, which achieved a remarkably low validation loss of 0.001266. This made it the top-performing combination not only for CNN-LSTM but across all models evaluated.

The Adam optimizer also performed well, achieving a validation loss of 0.001322 when paired with ReLU, which was slightly higher than Adamax but still highly effective. The linear activation function produced similarly strong results, particularly when combined with Adam or Adamax, both resulting in validation losses below 0.0016. While tanh also delivered decent performance, sigmoid activation was consistently the weakest performer, especially when paired with Regularizer, leading to high validation losses, such as 0.043121. Therefore, the best configuration for the CNN-LSTM model was clearly Adamax with ReLU, delivering the lowest validation loss.

Table 3.21: Optimizer Tunning of CNN-LSTM model

Model	Activation Function	Optimizer	Best Validation Loss
CNN-LSTM	relu	SGD	0.023877
		Adam	0.001322
		Adamax	0.001266
		Regularizer	0.008115
	tanh	SGD	0.020791
		Adam	0.001324
		Adamax	0.001414
		Regularizer	0.010754
	sigmoid	SGD	0.0449
		Adam	0.001693
		Adamax	0.001879
		Regularizer	0.043121
	linear	SGD	0.018816
		Adam	0.001328
		Adamax	0.001598
		Regularizer	0.008122

3.8.5.4 Summary Flowcahrt

Figure 3.13 shows the summary process of CNN-LSTM model hyperparameters tunning process. It is found that 16 filters, 3 convolutional layers, 1 LSTM layer, 64 LSTM units, kernel size 1, a dropout rate of 0.01, and a learning rate of 0.001, delivered the best overall performance.

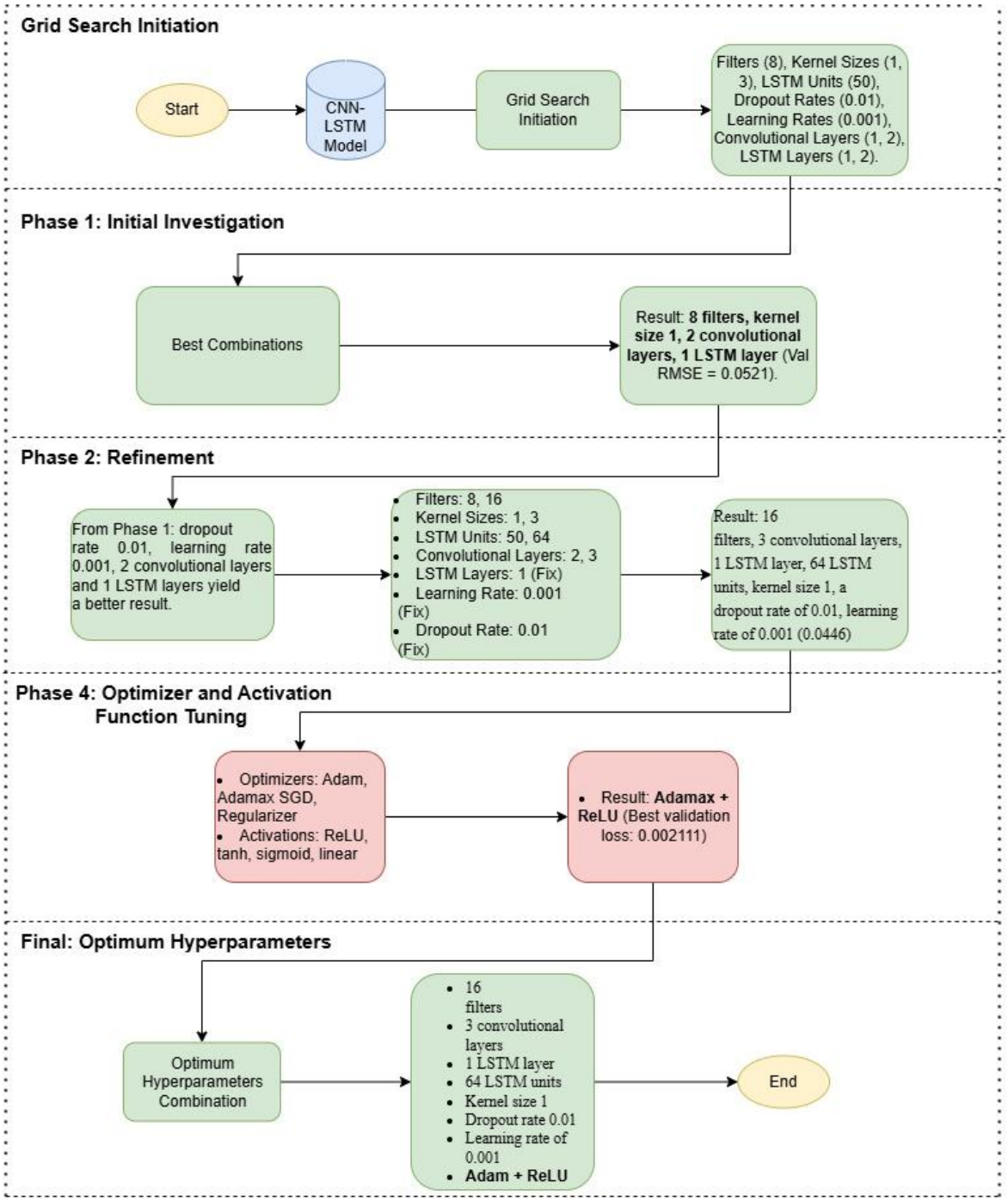


Figure 3.13: Flowchart of CNN-LSTM Tunning Process

3.9 Convergence Analysis

Convergence analysis is a crucial step in evaluating the performance of machine learning models across different algorithms. It assesses how effectively a model minimizes its error or loss during training and how well this performance translates to unseen data. By tracking both the training loss (error on training data) and the validation loss (error on separate validation data), convergence analysis helps determine the model's ability to generalize and identify any signs of overfitting.

The primary objective of convergence analysis is to ensure that both training and validation losses decrease as the training process continues. If the validation loss begins to increase or plateau while the training loss continues to decrease, this is often an indication of overfitting. In such cases, the model becomes too specialized to the training data and fails to generalize effectively.

To quantify the gap between training and validation losses, practical benchmark is applied in convergence analysis. In this project, the used benchmark is that the relative difference between training and validation losses should be less than 10%. However, the exact threshold can vary depending on the specific problem, dataset, and model being used (Owen, 2022). When this gap remains small, it suggests that the model is generalizing well and has converged appropriately. If the gap exceeds 10%, it could indicate overfitting, as the model may be learning patterns too specific to the training data, which do not translate well to new data.

3.9.1 LSTM

Figure 3.14 shows both training and validation losses showed a consistent decrease throughout the training process, with no sharp increases in validation loss, further indicating that the model is not overfitting. Although minor fluctuations in validation loss were observed, these are normal in deep learning models and do not suggest any significant overfitting. The overall downward trend of both losses points to effective learning and strong generalization.

A key benchmark is how closely training and validation loss follow each other. The training loss began at 0.00449 and decreased to 0.00061, while the validation loss started at 0.00232 and dropped to 0.00109. The final difference between the two losses was calculated as 0.00048, or approximately

4.4%, well below the acceptable threshold of 10%. This small gap indicates that the model generalizes well without overfitting. If the validation loss had stagnated or increased while training loss continued to drop, this would have indicated overfitting, but that was not the case here.

The close alignment of training and validation loss, combined with the small magnitudes (0.00061 for training and 0.00109 for validation), suggests that the model makes minimal errors and generalizes well to unseen data. The consistent performance across epochs shows that the model is not overly specialized to the training set, further supporting its robustness for load forecasting tasks.

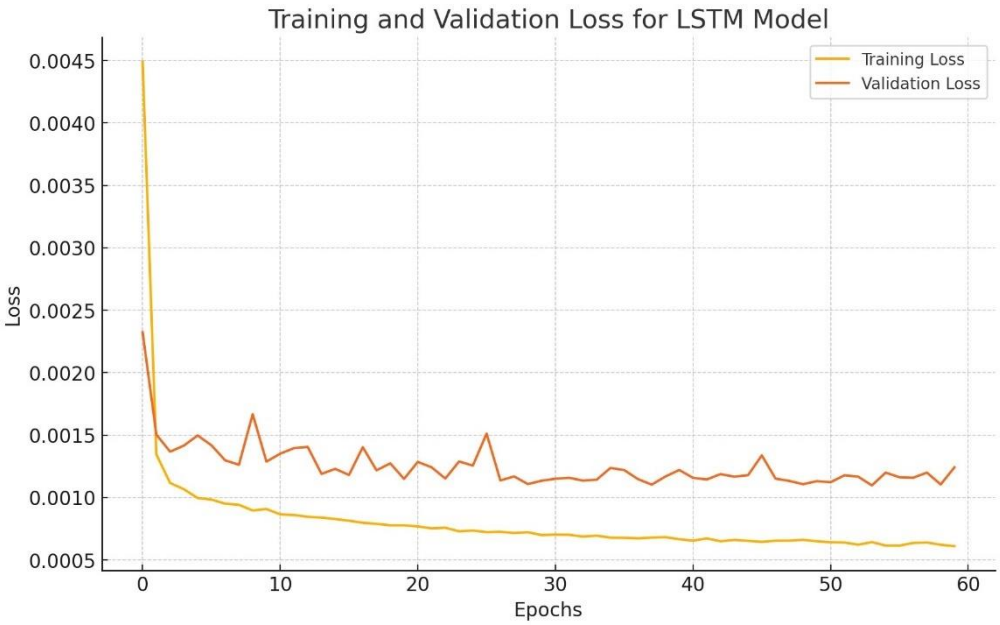


Figure 3.14: Training and validation loss for LSTM

3.9.2 GRU

In Figure 3.15, both training and validation losses for the GRU model exhibited a consistent downward trend throughout the training process, with no significant increases in validation loss, indicating that the model is not overfitting. Although minor fluctuations in validation loss were observed, these variations are normal in deep learning models and do not suggest any substantial overfitting. The general decrease in both training and validation losses underscores effective learning and strong generalization across the dataset.

A key benchmark for convergence analysis is how closely the training and validation losses follow each other. The training loss began at 0.00263 and steadily decreased to 0.00074, while the validation loss started at 0.00164 and dropped to 0.00080 by the end of training. The final difference between the two losses was calculated as 0.00006, which equates to a relative difference of approximately 7.5%, comfortably below the accepted threshold of 10%. This small gap indicates that the model generalizes well without significant overfitting. If the validation loss had stagnated or increased while the training loss continued to drop, this would have indicated overfitting, but that was not the case here.

The close alignment between the training and validation losses, along with the small magnitudes (0.00074 for training and 0.00080 for validation), suggests that the model makes minimal errors and generalizes effectively to unseen data. The consistent performance across epochs demonstrates that the model is not overly specialized to the training set, further supporting its robustness for tasks like load forecasting.

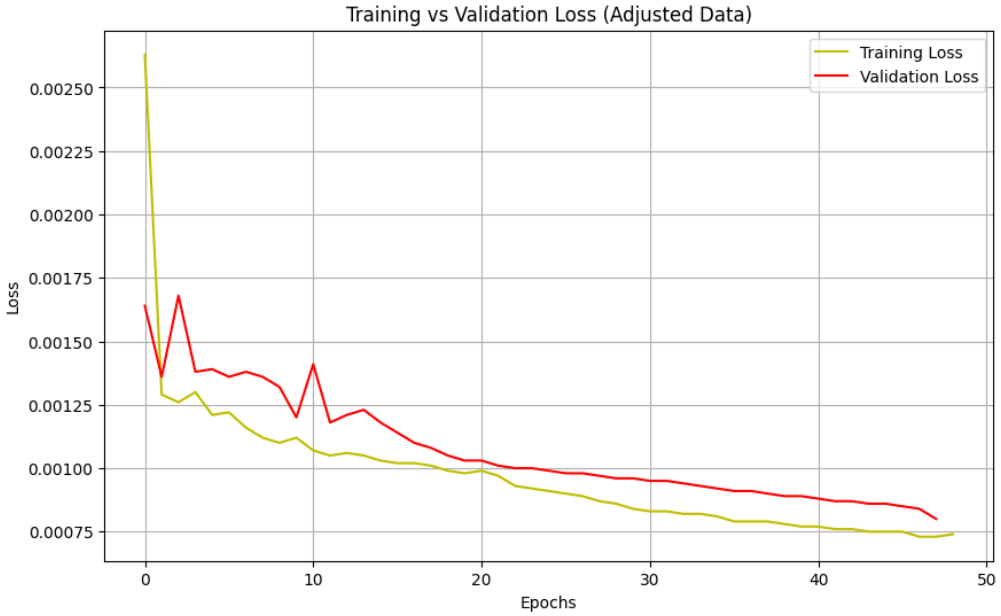


Figure 3.15: Training and validation loss for GRU

3.9.3 CatBoost

he graph in Figure 3.16 illustrates the training and validation loss of the CatBoost model over several epochs for your load forecasting project. The minimal percentage difference between the training and validation losses is approximately 1.5853% at its lowest point, indicating that the model is generalizing very well. Both losses decrease steadily over time, and the near-identical values of 0.0019855 for training loss and 0.0019545 for validation loss reflect that the model is learning the training data effectively while maintaining strong performance on unseen validation data.

This near-perfect alignment between training and validation losses suggests that the CatBoost model is well-optimized for this specific task. Unlike more complex models that may struggle with overfitting or underfitting, the CatBoost algorithm efficiently handles the data, showing excellent convergence and generalization without overfitting. The minimal gap between losses demonstrates that the model captures the underlying patterns in both the training and validation sets, leading to accurate and reliable predictions.

Compared to other models, CatBoost's strong performance here shows its ability to balance complexity and generalization. The decision-tree-based structure of CatBoost, combined with boosting iterations, makes it particularly

well-suited for this type of data. Unlike neural network architectures, which may struggle with small datasets or univariate time-series data, CatBoost adapts effectively without requiring extensive tuning, leading to superior results for load forecasting tasks. The almost negligible gap between the losses further supports its ability to generalize well without sacrificing performance.

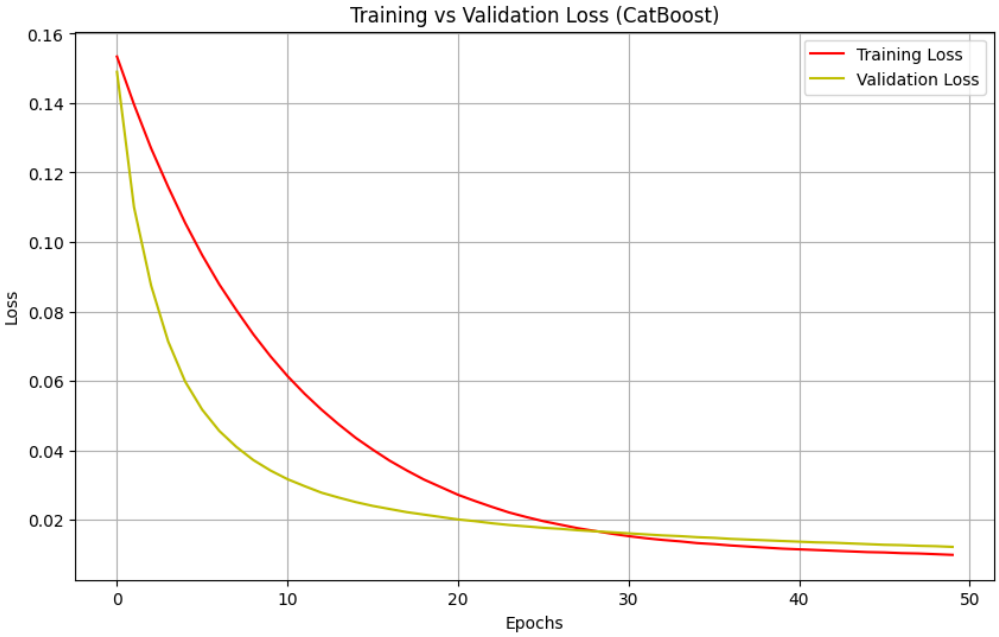


Figure 3.16: Training and validation loss for CatBoost

3.9.4 CNN-LSTM

The graph in Figure 3.17 illustrates the training and validation loss of the CNN-LSTM model over several epochs for your load forecasting project. The minimal percentage difference between the training and validation losses is approximately 18.68% at its lowest point across the epochs, indicating that the model is starting to overfit. While both losses decrease over time, the gap remains consistent, suggesting that the model is learning the training data well but struggling to generalize to unseen validation data.

This performance issue is likely due to the unnecessary complexity introduced by the CNN layers, which are designed for spatial feature extraction but are not suited for your univariate time-series data (power consumption). As a result, the CNN-LSTM model overfits the training data while failing to capture generalizable patterns, leading to slower and less effective convergence.

In contrast, simpler models like LSTM and GRU, which are better suited for handling temporal dependencies in time-series data, would likely exhibit faster convergence and a smaller gap between training and validation loss, leading to better overall performance for load forecasting. The complexity of CNN-LSTM in this context does not provide any additional benefit and instead hinders its ability to generalize.

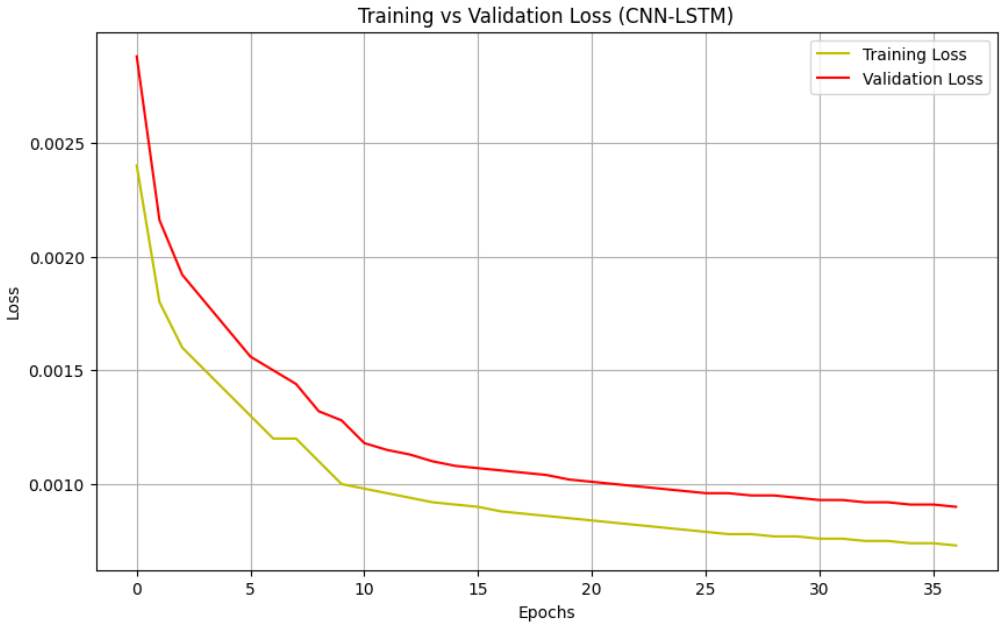


Figure 3.17: Training and validation loss for CNN-LSTM

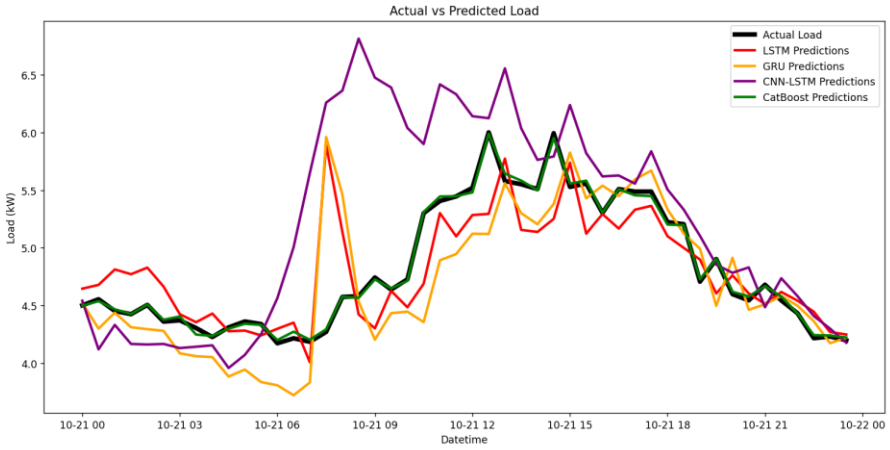
3.10 Performance Measure

The effectiveness of each model will be determined by evaluating its performance in predicting future power consumption. For this purpose, the evaluation will use a testing set, derived using the holdout method, with 10% of the dataset reserved for testing. This ensures that the model is assessed on data it has not encountered during training, providing an accurate measure of its predictive capability. Validation samples will be used for tuning neural network-based models, as these models employ iterative training, while classical machine learning methods such as CatBoost do not require validation during training.

The performance of each model will be analyzed using several metrics, including Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). These metrics provide a clear

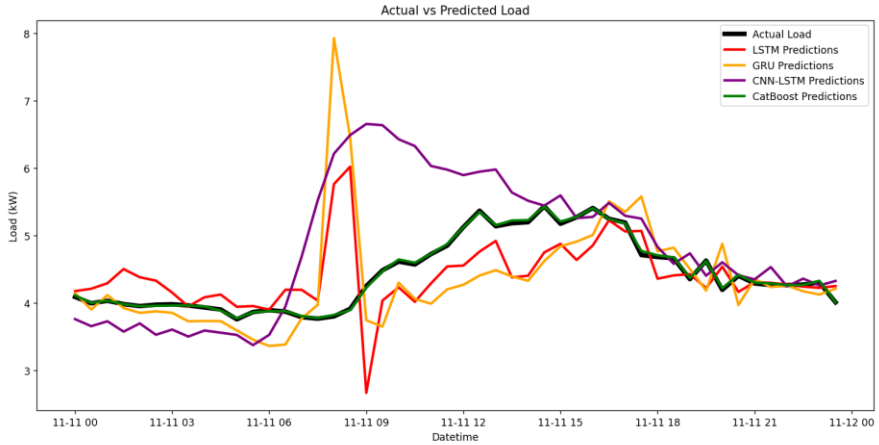
assessment of the model's accuracy in predicting continuous values, with RMSE particularly useful for penalizing larger errors, while MAE and MAPE offer insights into average error magnitudes and percentage-based accuracy.

Figure 3.18 to Figure 3.22 illustrate the prediction outcome of all developed static load forecasting models for random observation periods. It is evident that show whether all the models can trace the load usage pattern of the research office.



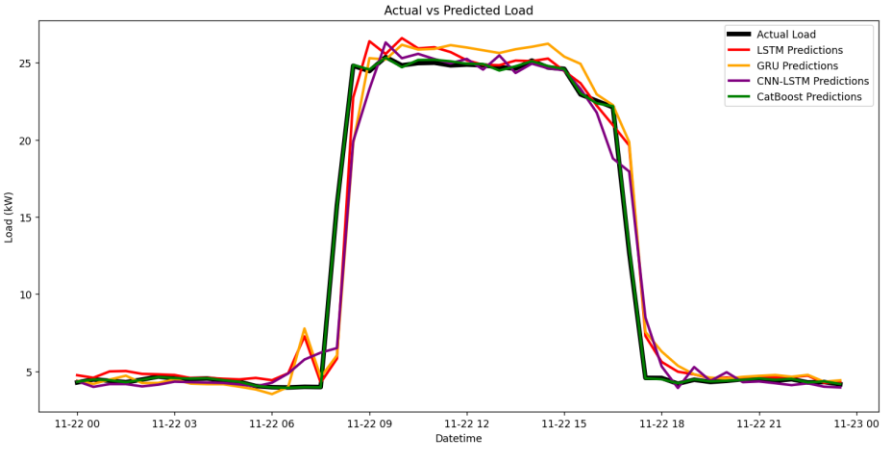
Model	RMSE (kW)	MAE (kW)	MAPE (%)
LSTM	0.3676	0.2561	5.25%
GRU	0.4323	0.3086	6.43%
CNN-LSTM	0.7736	0.5240	11.06%
CatBoost	0.0255	0.0203	0.42%

Figure 3.18: Prediction curves on 21/10/2023



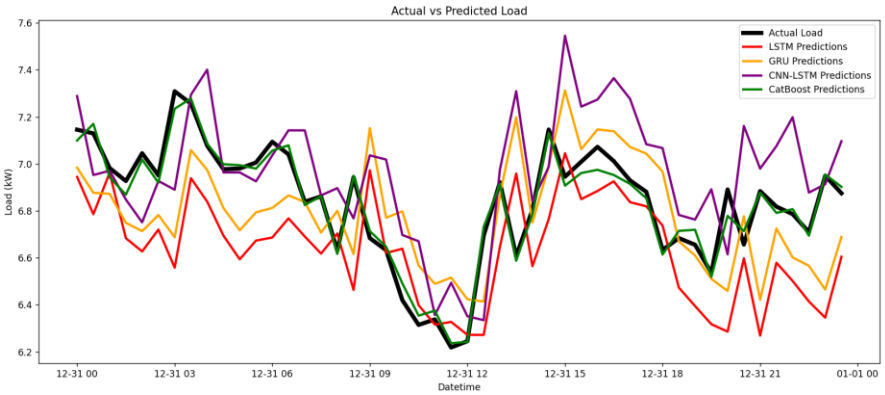
Model	RMSE (kW)	MAE (kW)	MAPE (%)
LSTM	0.5940	0.4001	9.13%
GRU	0.8304	0.4817	11.01%
CNN-LSTM	0.9235	0.6130	14.31%
CatBoost	0.0194	0.0151	0.34%

Figure 3.19: Prediction curves on 11/11/2023



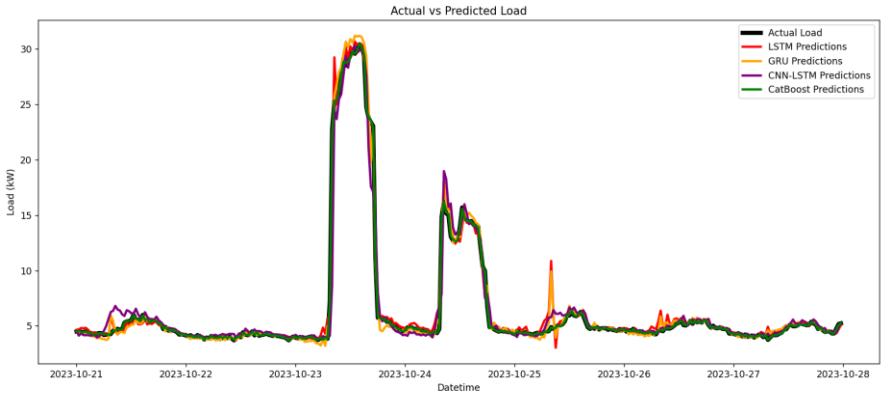
Model	RMSE (kW)	MAE (kW)	MAPE (%)
LSTM	1.9703	0.9268	10.84%
GRU	2.1376	1.1171	12.09%
CNN-LSTM	1.9368	0.9457	11.10%
CatBoost	0.0918	0.0625	0.56%

Figure 3.20: Prediction curves on 22/11/2023



Model	RMSE (kW)	MAE (kW)	MAPE (%)
LSTM	0.3053	0.2558	3.70%
GRU	0.2596	0.2152	3.15%
CNN-LSTM	0.2737	0.2192	3.23%
CatBoost	0.0394	0.0316	0.46%

Figure 3.21: Prediction curves on 31/12/2023



Model	RMSE (kW)	MAE (kW)	MAPE (%)
LSTM	0.7374	0.3853	6.29%
GRU	0.8121	0.4079	6.20%
CNN-LSTM	1.1921	0.4896	6.99%
CatBoost	0.0507	0.0302	0.45%

Figure 3.22: Prediction curves on 21/10/2023 to 27/10/2023 (one week)

The overall performance of the static load forecasting models, as shown in Figure 3.23 and Table 3.22, was evaluated on a 10% test set covering data from October 20, 2023, to December 31, 2023. Key metrics used for assessment were Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE), which provide insights into each model's ability to predict future power consumption with minimal input variables.

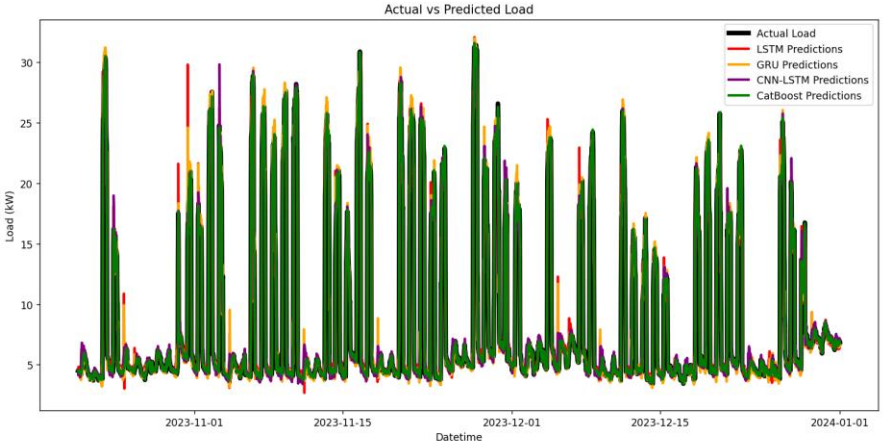
The LSTM model performed well, achieving an RMSE of 1.2579, MAE of 0.5799, and MAPE of 7.2875. These results indicate that LSTM effectively captured the power consumption patterns, with relatively low prediction errors. LSTM's ability to model long-term dependencies in time-series data allowed it to perform well despite using only datetime and power data.

The CNN-LSTM model had a higher RMSE of 1.4381, MAE of 0.6630, and MAPE of 8.2021, showing that it struggled more compared to LSTM. The addition of convolutional layers did not improve performance, likely because the limited input variables made it difficult to leverage CNN's strengths.

The GRU model, a simpler alternative to LSTM, performed similarly with an RMSE of 1.2771, MAE of 0.5821, and MAPE of 7.6171. While GRU handled the sequential data efficiently, LSTM had a slight advantage in capturing more complex temporal patterns.

The CNN model recorded the poorest performance, with an RMSE of 1.4614, MAE of 0.7164, and MAPE of 8.6425. As CNNs are designed to capture spatial patterns, it struggled to model the sequential nature of power consumption using only datetime and power, leading to higher error metrics.

Surprisingly, the CatBoost model delivered the best performance, with the lowest RMSE of 0.9621, MAE of 0.4217, and MAPE of 5.6212. CatBoost’s gradient-boosting approach efficiently handled the limited input data, consistently making accurate predictions with smaller errors than other models. Its MAPE of 5.6212% further demonstrated its reliable percentage-based predictions, making it the most effective model.



Model	RMSE (kW)	MAE (kW)	MAPE (%)
LSTM	1.2579	0.5799	7.29%
GRU	1.3158	0.6185	7.49%
CNN-LSTM	1.4381	0.6630	8.20%
CatBoost	0.0819	0.0474	0.51%

Figure 3.23: Prediction curves on 20/10/2023 to 31/12/2023 (full test data)

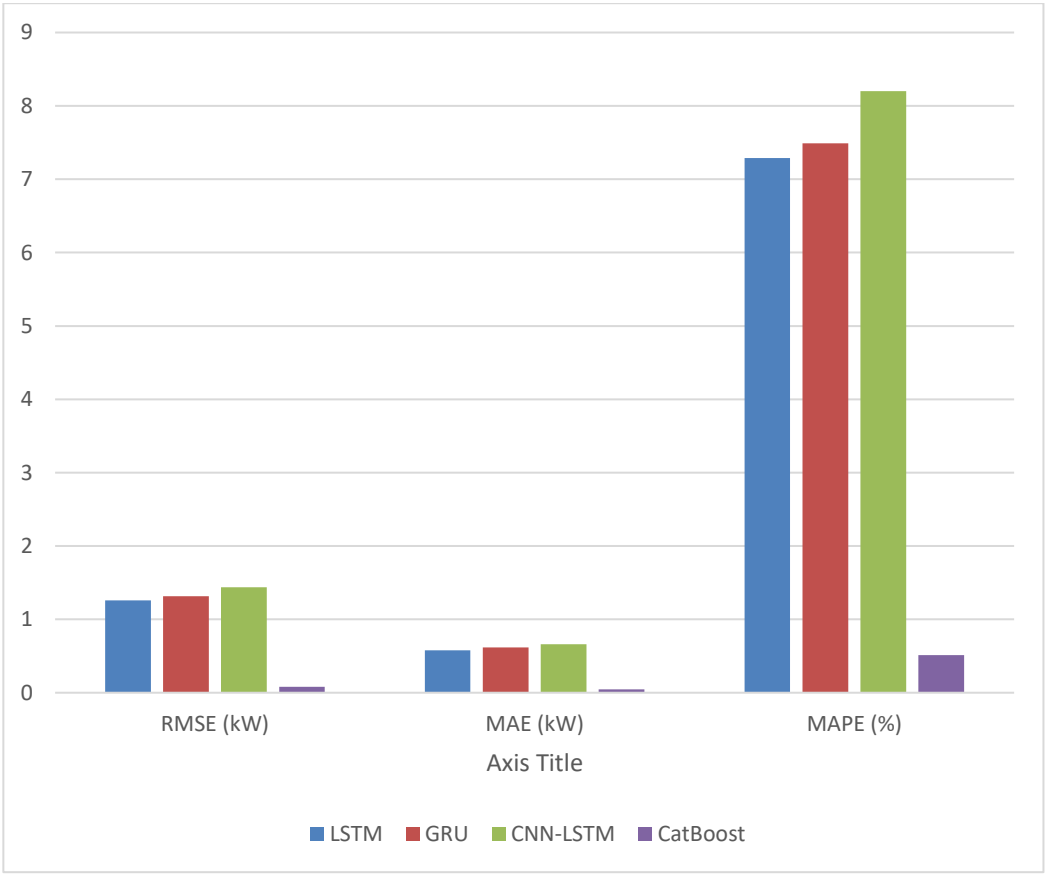


Figure 3.24: Prediction curves on 31/12/2023

Table 3.22: Overall Comparison between LSTM, GRU, CNN-LSTM, CatBoost on Test Data

Model	RMSE (kW)	MAE (kW)	MAPE (%)
LSTM	1.2579	0.5799	7.2875
GRU	1.3158	0.6185	7.4891
CNN-LSTM	1.4381	0.6630	8.2021
CatBoost	0.0819	0.0474	0.5127

In summary, CatBoost outperformed all models in forecasting power consumption with limited input variables. Among the neural networks, LSTM performed best, followed by GRU, while CNN-LSTM and CNN underperformed due to added complexity not suited for the limited dataset. CatBoost is the recommended model for this task, with LSTM and GRU as strong alternatives for time-series data.

3.11 Limitations of Static models

The static load forecasting models, as shown in Figure 3.25, and Figure 3.26 have several limitations, particularly when applied to time periods with dynamic or evolving power consumption patterns. These models are trained on historical data from a fixed time span (in this case, 2021 to 2023) and are not updated as new data becomes available. As the load profile evolves, especially with significant changes in usage patterns, the performance of static models tends to degrade. This is evident in the rising RMSE, MAE, and MAPE values over time, as depicted in the figure, where the errors progressively increase from February to June 2024. The pattern of degrading can be explain by Figure 3.27 which is the load power consumption of whole dataset. It shows that the load power consumption of the research office was keep changing.

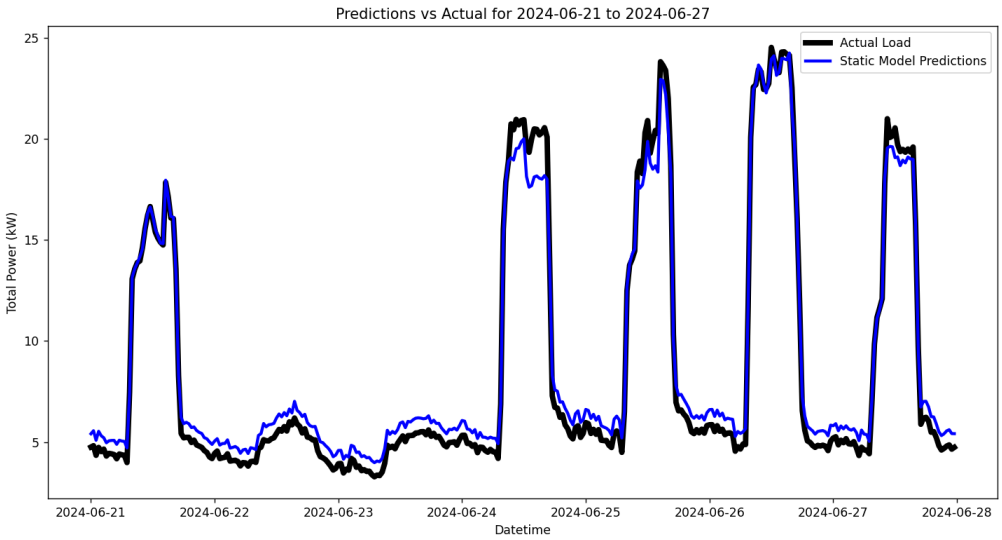


Figure 3.25: Prediction curves on 21/06/2024 – 27/06/2024

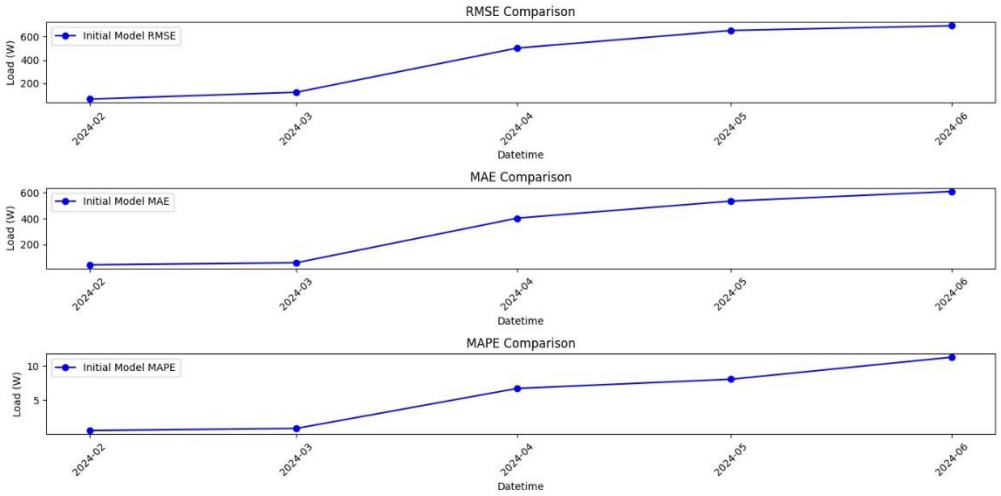


Figure 3.26: Plotting of Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2024 Onwards)

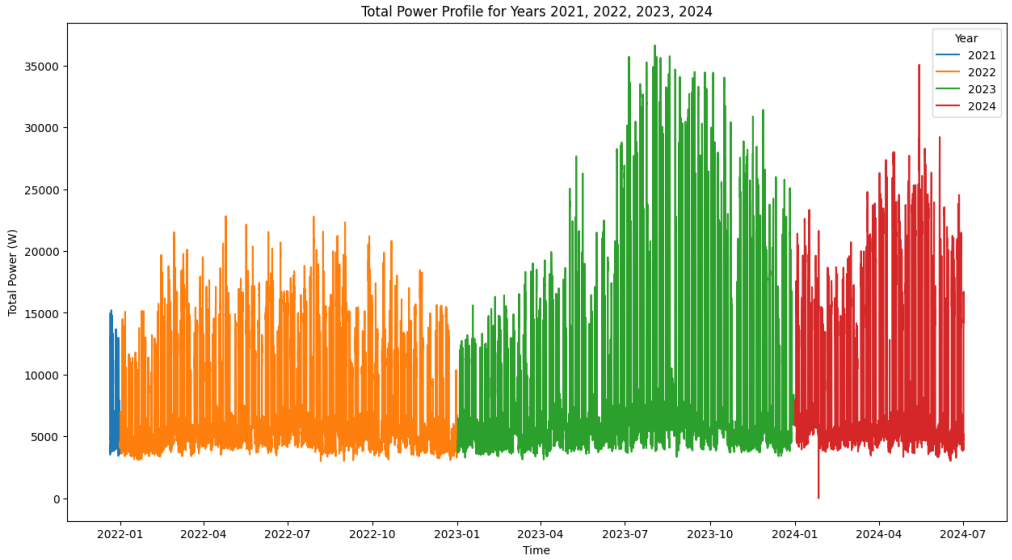


Figure 3.27: Load Power Consumption of Whole Dataset (Dec 2021 to June 2024)

One of the key limitations is the inability of static models to adapt to changing trends or seasonal variations. Without the ability to retrain or adjust based on new data, these models cannot capture shifts in power consumption patterns effectively. Additionally, static models may struggle with unseen conditions or data distributions that differ from the training period, leading to overfitting to historical trends but underperforming in future periods.

In summary, while static models can provide reasonable accuracy in the short term, the effectiveness diminishes over time as it is unable to dynamically adapt to the evolving load profile. This limitation underscores the need for adaptive models that can update and refine themselves as new data is introduced, ensuring more accurate and reliable forecasts in changing environments.

CHAPTER 4

ADAPTIVE CATBOOST DEVELOPMENT & EVALUATION

4.1 Flowchart & Work Plan

The purpose of this stage in the methodology is to develop an adaptive load forecasting model using the CatBoost algorithm, which was identified as the best-performing model during the static development phase. Unlike the static model, which is fixed after training, the adaptive model is designed to adjust to new data, ensuring it can handle changing consumption patterns over time. This adaptability is essential for accounting for shifts in future power consumption trends, which may differ from historical patterns. By building an adaptive CatBoost model, the goal is to maintain high forecasting accuracy even in dynamic environments where consumption behavior evolves over time. The flowchart and workplan is shown in Figure 4.1.

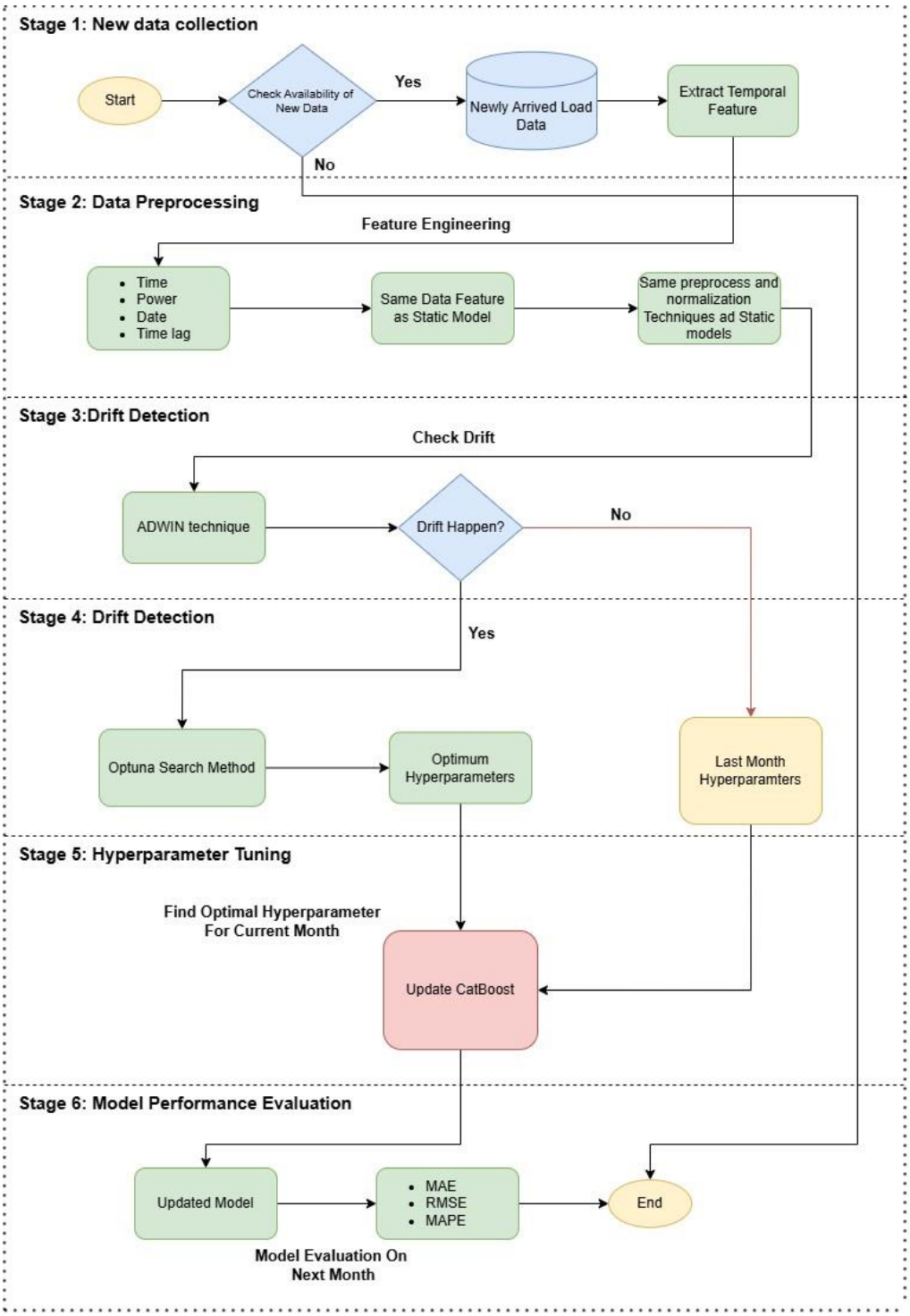


Figure 4.1: Flowchart of Adaptive CatBoost

4.2 Dataset

For the development of the adaptive CatBoost model, the dataset will consist of power consumption data collected from January 2024 to June 2024 from the

same two-story research office used in the static model development. This dataset shares the same structure and variables as the previous dataset, ensuring consistency in the modeling process while reflecting updated power consumption trends for 2024. The adaptive model will be updated monthly, allowing it to account for potential changes in consumption patterns that may differ from those observed between December 2021 and December 2023. The total power is computed in same manner as in static model development stage.

Table 4.1: Dataset Characteristics (January 2024 – June 2024)

Dataset Characteristics	**Details**
Instances	35,517
Time Span	Jan 1, 2024 to June 30, 2024
Collect Interval	30 minutes
Missing Values	0
Features	Time, Phase Voltage (A, B, C), Phase Current (A, B, C), Power Factor

4.3 Preprocessing

In the preprocessing phase of developing the adaptive CatBoost model, the preprocessing step kept consistent with that used in the static model development to ensure uniformity and reliability. This consistency is crucial for maintaining the comparability of results across different stages of model evolution. The preprocessing begins with the integration of newly collected monthly data with the existing historical dataset. This approach allows us to build upon the previous data, ensuring that the model continuously learns from the most recent trends. Missing values in the dataset are addressed using imputation methods that were applied in the static model. For numerical features, imputation is carried out by filling in missing values with the mean or median of the respective features, while categorical features are handled according to the specific characteristics.

Normalization and scaling are essential steps to prepare the data for effective model training. In this regard, MinMaxScaler was apply to normalized feature values to a range between 0 and 1, ensuring that all features contribute

equally to the model training process. Feature engineering is maintained in the same manner as the static model, with the same set of features being used to ensure that the model's performance can be consistently evaluated over time. Data splitting is performed by dividing the dataset into training and testing sets following the same proportions as used in the static model, allowing us to validate the model on a representative sample and ensure its robustness.

4.4 Drift Detection

Drift detection is a vital aspect of maintaining the accuracy and reliability of predictive models, especially in environments where the underlying data distribution can change over time—a phenomenon known as concept drift. In the context of the adaptive CatBoost model for power consumption forecasting, the detection technique being employed is the ADWIN (Adaptive Windowing) algorithm to detect such drifts effectively.

ADWIN is a robust and efficient method designed specifically for streaming data applications. Its core functionality revolves around maintaining a dynamically sized sliding window over the data stream, which, in this case, consists of the sequence of absolute errors between the model's predictions and the actual observed values. Unlike fixed-size windowing techniques, ADWIN adjusts the window size adaptively based on the statistical properties of the incoming data. This adaptability allows it to be sensitive to both gradual and abrupt changes in the data distribution.

The mechanism by which ADWIN detects drift involves several key steps. Firstly, as new data points arrive, the absolute error was computed between each predicted value and its corresponding true value. These error values are sequentially fed into the ADWIN algorithm. ADWIN continuously updates its internal statistics with each new error, recalculating the mean and variance within the current window. This continuous monitoring is crucial for timely detection of any significant changes in the error distribution.

Secondly, ADWIN employs statistical hypothesis testing to compare the distribution of errors in different segments of the window. Specifically, it splits the window into two sub-windows at every possible point and evaluates whether the difference in the mean errors between these sub-windows is

statistically significant. It utilizes Hoeffding's Inequality to calculate confidence intervals, which helps determine whether the observed difference could be due to random fluctuations or represents a genuine shift in the data distribution.

A significant advantage of ADWIN is its use of dynamic thresholds derived from the data itself, rather than relying on manually set fixed thresholds. This means that the algorithm calculates thresholds based on the observed variance and desired confidence levels, making it highly adaptable to various types of drift and noise levels in the data. If the statistical test indicates that the difference between sub-windows exceeds the calculated threshold, ADWIN concludes that a concept drift has occurred.

4.5 Optuna Optimization

Optuna is a powerful hyperparameter optimization framework designed to enhance model performance through systematic and efficient search techniques. In this project, Optuna is employed to fine-tune the hyperparameters of the CatBoost model, aiming to optimize its forecasting accuracy for power consumption. The core objective of using Optuna is to identify the optimal set of hyperparameters that improves the model's predictive accuracy while maintaining computational efficiency.

Optuna employs a Bayesian optimization approach, which builds a probabilistic model of the objective function and iteratively refines it based on previous trial results. This method is advantageous over traditional grid search or random search approaches as it explores the hyperparameter space more intelligently. The optimization begins with an initial set of hyperparameters, which are evaluated based on model performance metrics. Optuna then updates its probabilistic model to suggest new configurations that are likely to enhance performance.

A notable feature of Optuna is its trial management capability. It handles multiple trials by sampling different hyperparameter configurations and assessing the performance. Each trial involves training the CatBoost model with a specific set of hyperparameters and measuring its performance on validation data. Based on these results, Optuna guides subsequent trials towards more promising configurations. Additionally, Optuna includes a pruning mechanism

that allows early termination of underperforming trials, conserving computational resources by focusing on more promising trials.

Optuna also provides visualization tools that help understand the optimization process and the impact of various hyperparameters on model performance. By utilizing Optuna's advanced search algorithms and pruning features, the goals is to enhance the CatBoost model's performance, leading to more accurate and reliable power consumption forecasts.

4.5.1 Updates Starting in 2024 (Scenario 1)

In Scenario 1, where the model was initially trained on data from 2021 to 2023 and updates started in January 2024, Optuna consistently selected a depth of 6 and a learning rate around 0.049 for most months as shown in Table 4.2. Although the L2 regularization and iterations varied slightly with each update, this variability shows that tuning occurred during each update to reflect changes in the dataset. The adjustments made by Optuna for every update ensured the model stayed optimized and adapted well to the new data, maintaining high accuracy.

Table 4.2: Adaptive Catboost Optimum Parameter Tunned by Optuna Optimization for Each Updating (Update from Jan 2024 Onwards)

Data used to Update	Depth	Learning_rate	L2 Regularization (L2_leaf_reg)	Lterations
2024-01	5	0.0494	3.6631	1785
2024-02	5	0.0499	1.3292	1958
2024-03	6	0.0359	0.2528	1999
2024-04	6	0.0491	0.1068	2000
2024-05	6	0.0495	0.3351	2000

4.5.2 Updates Starting in 2023 (Scenario 2)

Similarly, in Scenario 2, where the model was initially trained on data from 2021 to 2022 and updates began in January 2023, Optuna again tuned the model's hyperparameters for each update as shown in Table 4.3. The depth of 6 remained constant, but adjustments in learning rate, L2 regularization, and iterations occurred monthly, showing that the model was continually optimized as it incorporated new data. This fine-tuning process during each update further confirms that the model was regularly recalibrated to handle shifting consumption patterns.

Table 4.3: Adaptive Catboost Optimum Parameter Tunned by Optuna Optimization for Each Updating (Update from Jan 2023 Onwards)

Date	depth	learning_rate	l2_leaf_reg	iterations
2023/01	6	0.0489	2.0293	1755
2023/02	6	0.0498	0.2216	1870
2023/03	6	0.049	3.3553	1918
2023/04	6	0.05	0.3612	1886
2023/05	6	0.0498	0.1128	1870
2023/06	6	0.0456	0.9556	1997
2023/07	6	0.0495	0.1918	1680
2023/08	6	0.05	0.1047	1928
2023/09	6	0.0498	0.7588	1954
2023/10	6	0.0455	0.1952	1724
2023/11	6	0.045	0.1711	1807
2023/12	6	0.0495	4.2478	1930
2024/01	6	0.0469	1.2794	1885
2024/02	6	0.0496	0.3521	1814
2024/03	5	0.0498	0.1522	1960
2024/04	6	0.0488	3.1979	1899
2024/05	6	0.05	0.4013	1930

4.6 Updating Size

The decision to update the adaptive CatBoost model on a monthly basis was made to balance responsiveness, stability, and computational efficiency. Power consumption patterns often follow monthly cycles due to factors like weather changes, holidays, and billing periods. By updating the model monthly, it can capture these recurring trends without overreacting to short-term fluctuations

that could occur weekly. Weekly updates might cause the model to respond to noise, reducing its stability and accuracy.

A key reason for avoiding weekly updates is that it don’t provide enough new data for the model to learn. Power consumption doesn’t change significantly week to week, so updating the model weekly would not give it enough information to learn meaningful patterns. In contrast, monthly updates offer sufficient data, allowing the model to adjust more effectively and improve its predictions.

Monthly updates also align well with seasonal variations in power consumption, such as increased energy demand during summer and winter. This helps the model adapt to these shifts in a timely manner, whereas yearly updates might miss mid-year changes, leading to outdated forecasts. Weekly updates could overreact to minor variations, missing the larger seasonal trends.

Finally, monthly updates align with typical business and operational cycles, where power consumption is often tracked and billed monthly. This makes the updates more relevant for decision-makers and ensures that forecasts stay in sync with practical needs.

4.7 Updating Model Evaluation

The evaluation process for the adaptive model will differ significantly from the approach used for the static model, given the model's continuous updates over time. To appropriately assess the adaptive model's performance, a rolling window evaluation will be employed. This method is particularly suitable for time series forecasting models, where predictions are made iteratively with the inclusion of newly available data.

Initially, the static model will be used to predict power consumption from January 2024 onwards. This static model is trained on historical data from 2021 to 2023 and will serve as the baseline for comparison. Since the static model does not incorporate any data from 2024 or beyond, its predictions will be made based solely on the historical patterns learned during the training period. This establishes a reference point against which the performance of the adaptive model will be measured.

For the adaptive model, beginning in January 2024, the model will be updated monthly with newly available data. After each monthly update, the model will generate predictions for the subsequent month. For example, the model updated at the end of January 2024 will be used to predict power consumption for February 2024, and the model updated at the end of February 2024 will be used to forecast March 2024, and so on. This approach ensures that the model adapts to evolving trends in power consumption and is responsive to any changes in underlying data patterns.

The performance of both the static and adaptive models will be assessed using standard forecasting metrics, including Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). These metrics will be calculated for each month, enabling a direct comparison of how the static model, with its fixed knowledge of past data, performs relative to the adaptive model, which integrates new information on a rolling basis. The focus of the evaluation will be on how well the adaptive model reduces forecasting errors over time, and whether its ability to learn from recent data leads to improved accuracy compared to the static model.

4.8 Performance Measure Result

4.8.1 Updates Starting in 2024 (Scenario 1)

Figure 4.2 to Figure 4.5 illustrate the prediction curve of initial static model and the Adaptive CatBoost model. From these Figure, the performance difference of both static and updated model is being visualized.

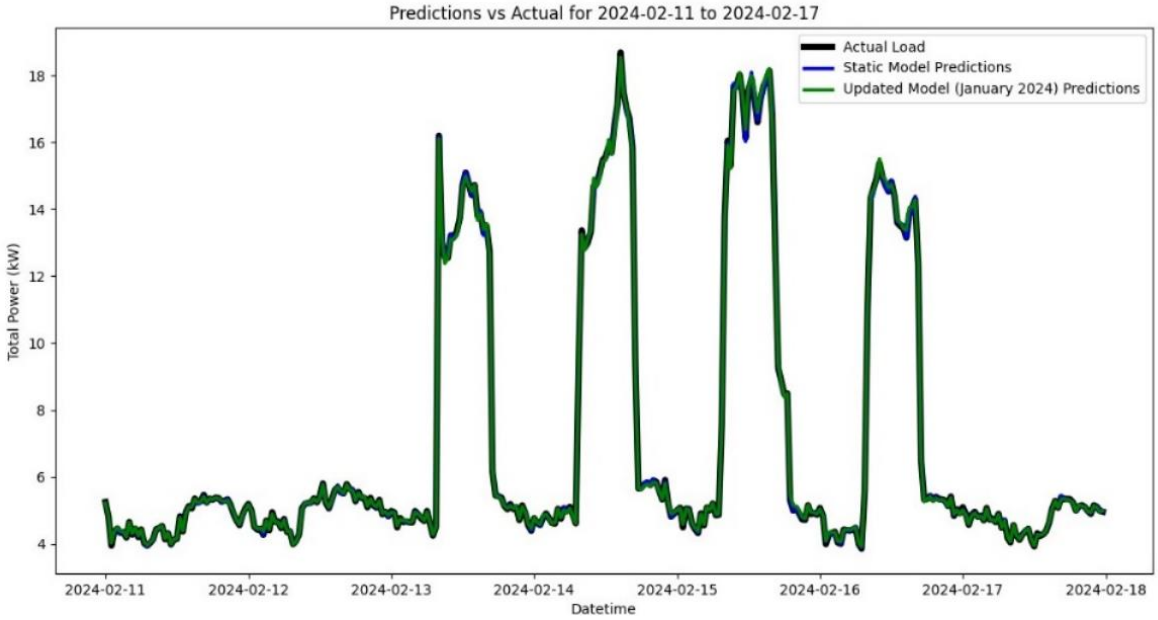


Figure 4.2: Prediction curves on 11/02/2024 to 17/02/2024 (one week)

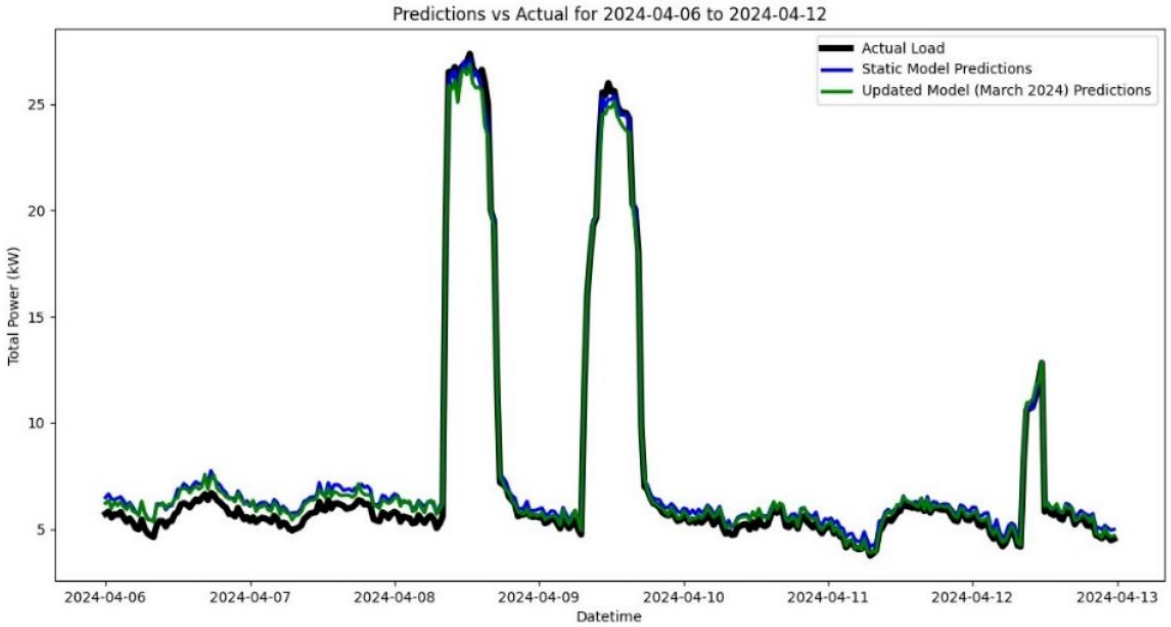


Figure 4.3: Prediction curves on 06/05/2024 to 12/04/2024 (one week)

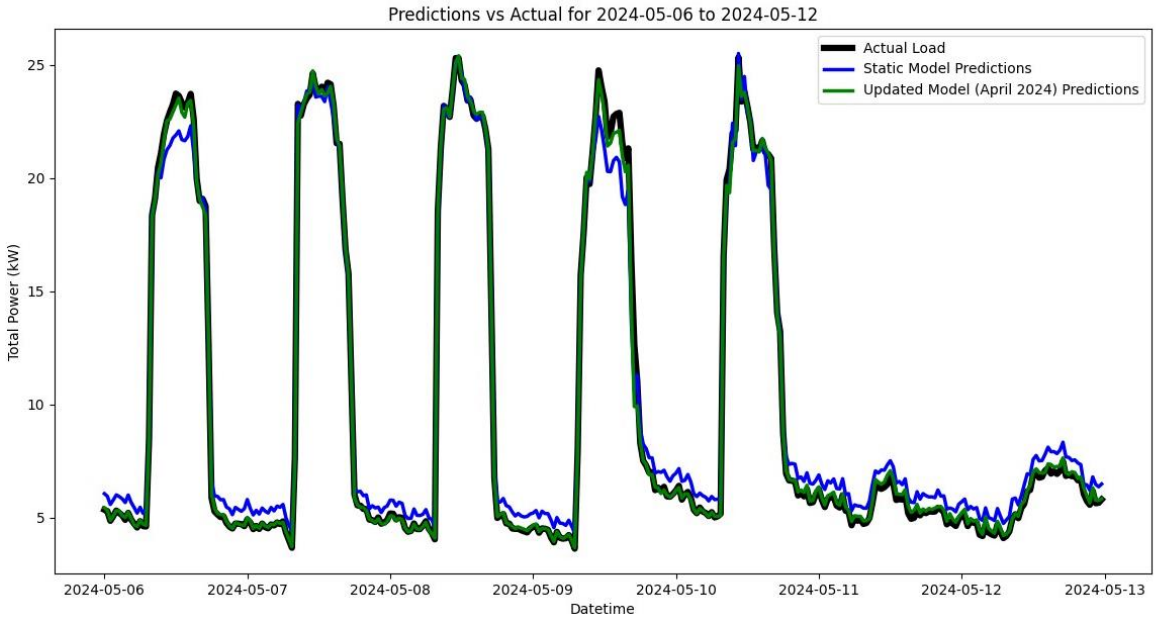


Figure 4.4: Prediction curves on 06/05/2024 to 12/05/2024 (one week)

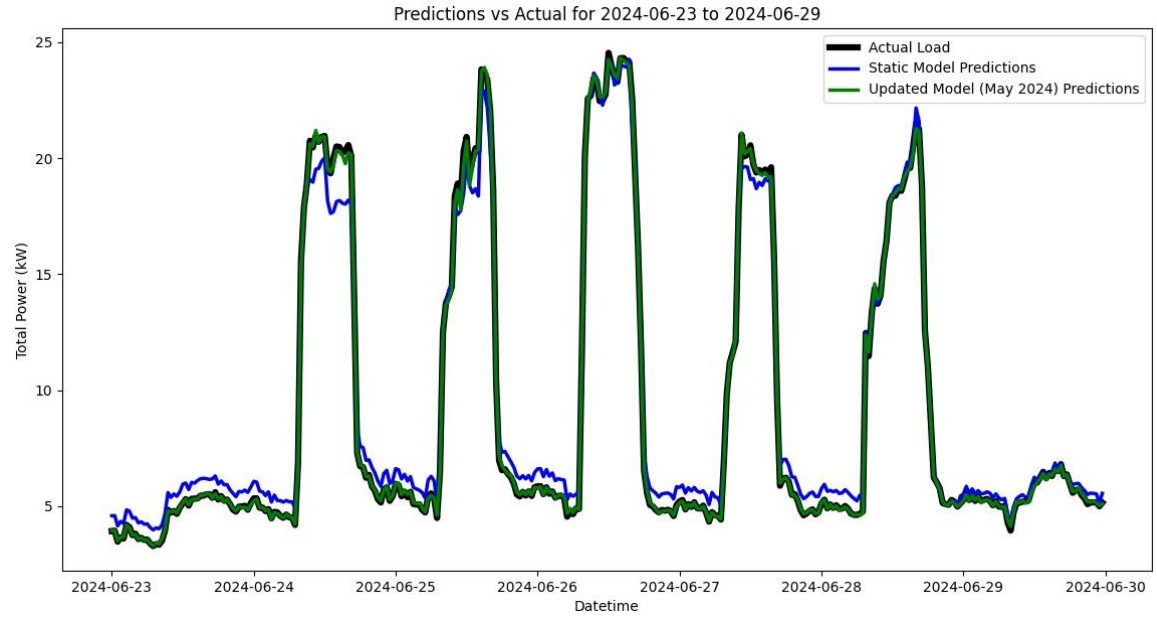


Figure 4.5: Prediction curves on 23/06/2024 to 29/06/2024 (one week)

After each updating, the models was used to predict power consumption for the subsequent month's power consumption, At the same time, the **initial static model**, which was not updated, continued to predict the same months, serving as a benchmark to compare the performance of the static and updated models. Key differences in performance were observed at specific data points, reflecting the advantages of the adaptive model. The overall performance of Adaptive CatBoost is shown in Figure 4.6 and Table 4.4.

In February 2024, after being updated with January's data, the adaptive model achieved an RMSE of 48.64, significantly lower than the static model's 70.13. This suggests that the adaptive model quickly captured recent consumption patterns, improving its prediction accuracy early on.

The most notable improvement occurred in April 2024, where the adaptive model's RMSE dropped to 282.86, compared to a much higher 405.40 for the static model. The static model struggled here, likely due to evolving power consumption trends, whereas the adaptive model successfully adjusted to these changes through monthly updates.

Similarly, in April 2024, the MAPE for the adaptive model was 1.43%, compared to 3.98% for the static model. This significant reduction shows that the adaptive model provided more accurate percentage-based forecasts, effectively accounting for shifts in power usage patterns.

By May 2024, although the error margins slightly increased, the adaptive model still outperformed the static model across all metrics, including MAE, which was 132.34 compared to 250.71 for the static model. This indicates that even with fluctuations, the adaptive model maintained better accuracy.

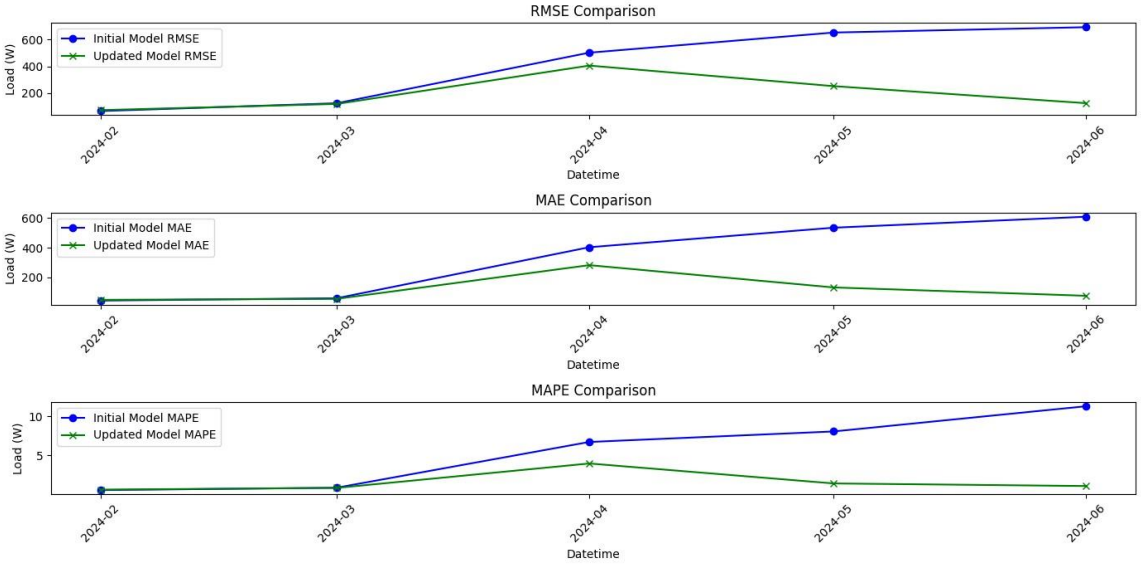


Figure 4.6: Plotting of Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2024 Onwards)

Table 4.4: Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2024 Onwards)

Evaluation Month	RMSE(kW)		MAE(kW)		MAPE(%)	
	Static Model	Updated Model	Static Model	Updated Model	Static Model	Updated Model
2024/02	0.0632356	0.0701327	0.0437327	0.0486426	0.5829	0.6407
2024/03	0.121942	0.1167044	0.0594443	0.055297	0.8854	0.8608
2024/04	0.5025032	0.4054039	0.4037901	0.2828642	6.7277	3.9805
2024/05	0.6539747	0.2507122	0.5357126	0.1323437	8.0718	1.4314
2024/06	0.6942722	0.1224752	0.6098815	0.0761539	11.2927	1.1038

4.8.2 Updates Starting in 2023 (Scenario 2)

Figure 4.7 to Figure 4.10 illustrate the prediction curve of initial static model and the Adaptive CatBoost model. From these Figure, the performance difference of both static and updated model is being visualized.

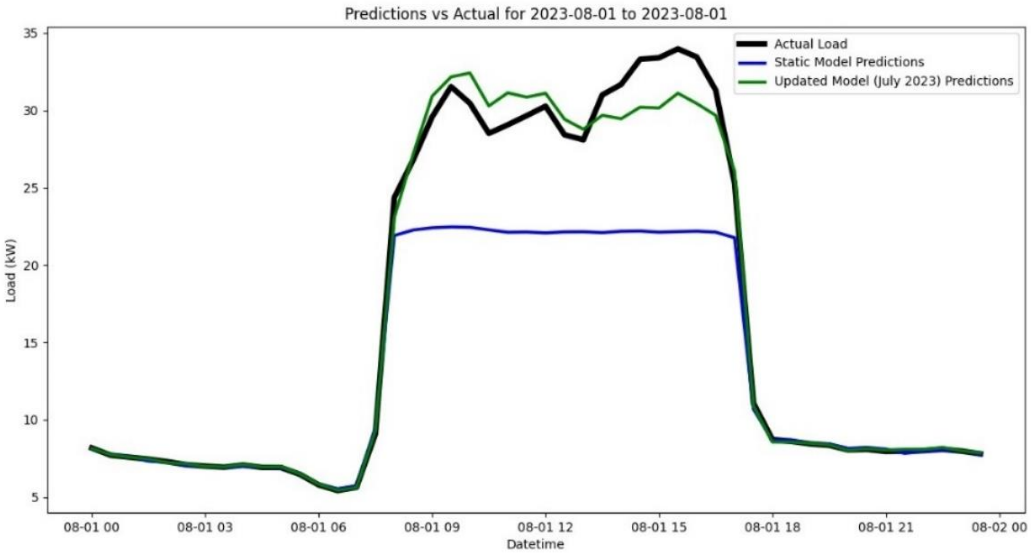


Figure 4.7: Prediction curves on 01/08/2023

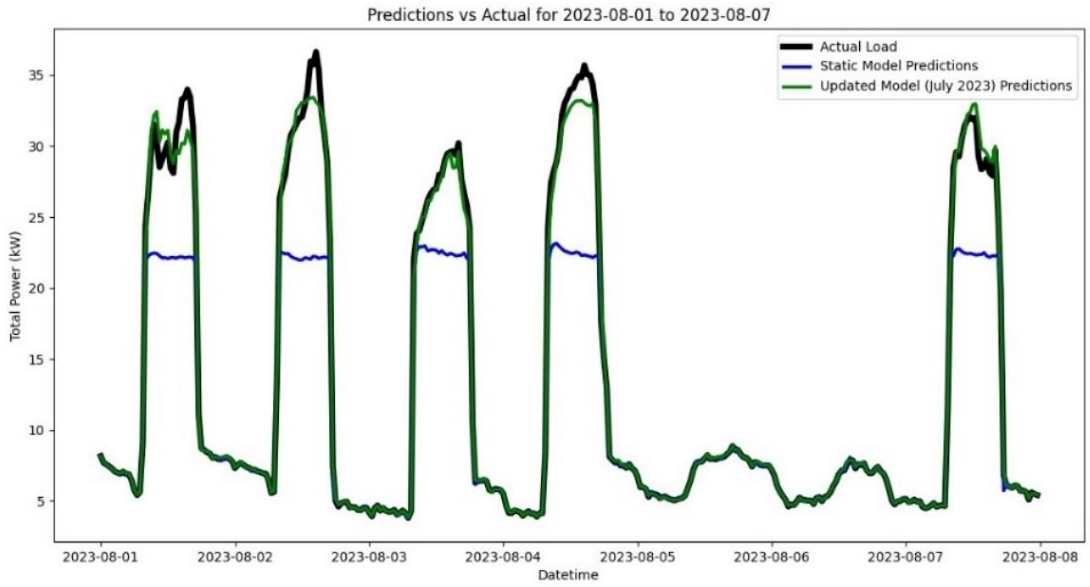


Figure 4.8: Prediction curves on 01/08/2023 to 07/08/2023 (one week)

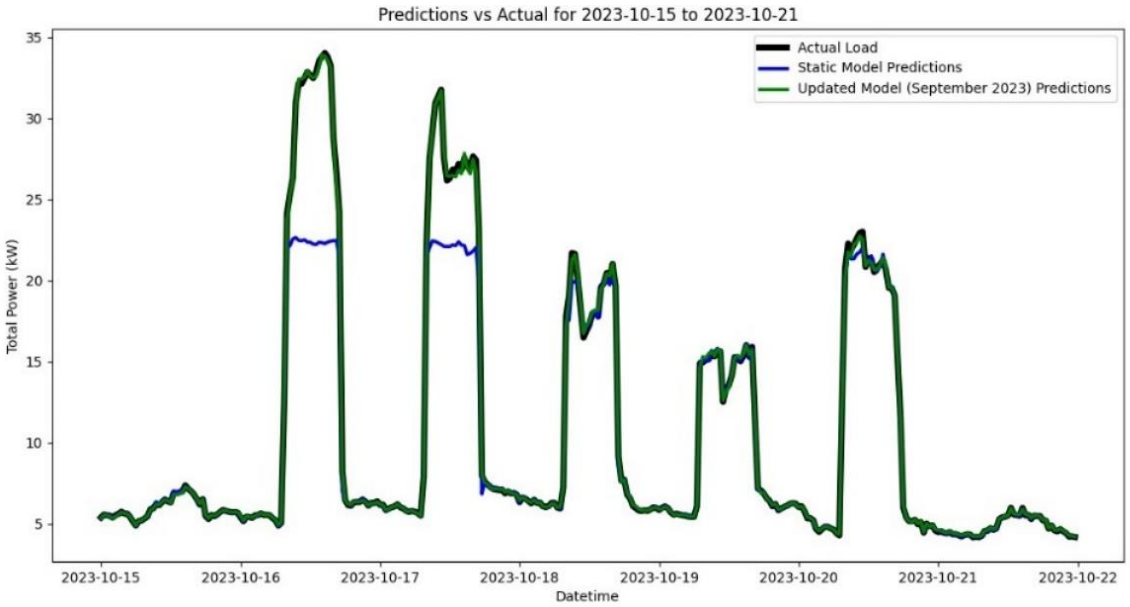


Figure 4.9: Prediction curves on 15/10/2023 to 22/10/2023 (one week)

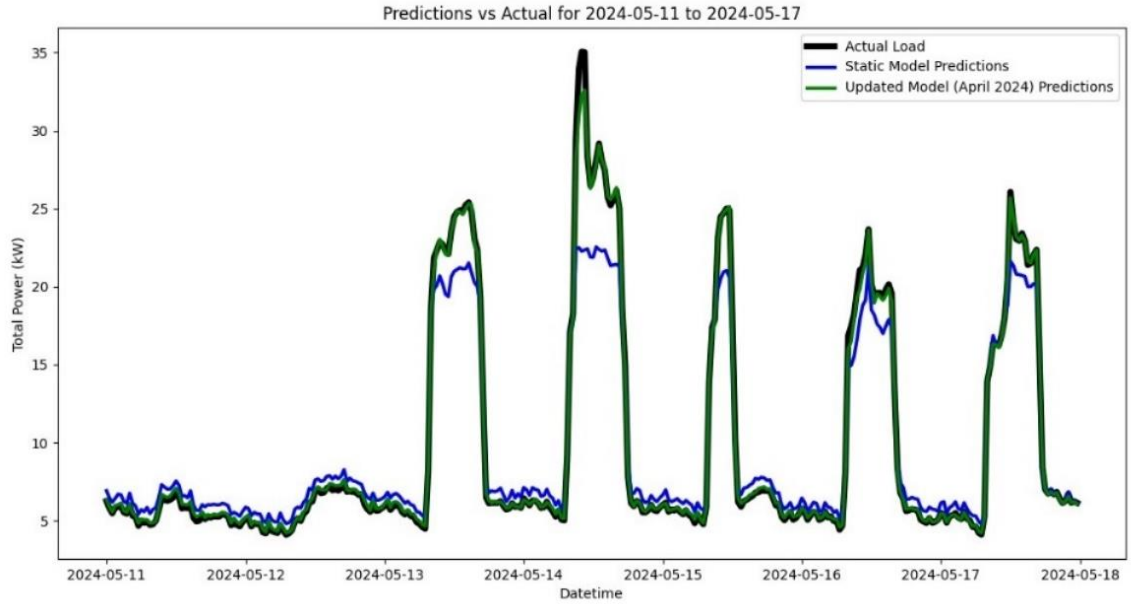


Figure 4.10: Prediction curves on 11/05/2024 to 17/05/2024 (one week)

In Scenario 2, the Adaptive CatBoost model was trained on data from 2021 to 2022, with monthly updates starting in January 2023. Each update was used to predict power consumption for the following month, with the static model serving as a benchmark. Performance was evaluated using RMSE, MAE, and MAPE, showing clear improvements with the adaptive model. The result was shown in Figure 4.11 and Table 4.5.

The RMSE results highlight significant gains for the adaptive model. In July 2023, the static model recorded a high RMSE of 2.7612, while the adaptive model reduced this error to 1.2127. Similarly, in August 2023, the static model had an RMSE of 3.7131, which the adaptive model brought down to 0.5597. These reductions demonstrate the adaptive model’s ability to adjust to changing consumption patterns effectively.

In later months, especially from October to December 2023, the adaptive model continued to outperform the static model. For instance, in November 2023, the static model's RMSE was 1.4768, while the adaptive model achieved a much lower 0.1754. These improvements show how the adaptive model better captured evolving trends in power consumption.

The MAE results follow a similar pattern. In June 2023, the adaptive model reduced the MAE to 0.1482, compared to 0.2388 for the static model. By November 2023, the MAE for the adaptive model was 0.0936, compared to 0.5372 for the static model, proving the adaptive model's precision in forecasting.

The MAPE values further emphasize the adaptive model's advantage. In August 2023, the static model's MAPE was 1.6637%, while the adaptive model reduced it to 0.2409%. Even in June 2024, where the static model’s MAPE surged to 11.9732%, the adaptive model maintained a much lower error of 1.0841%. These results highlight the adaptive model's superior percentage-based accuracy over time.

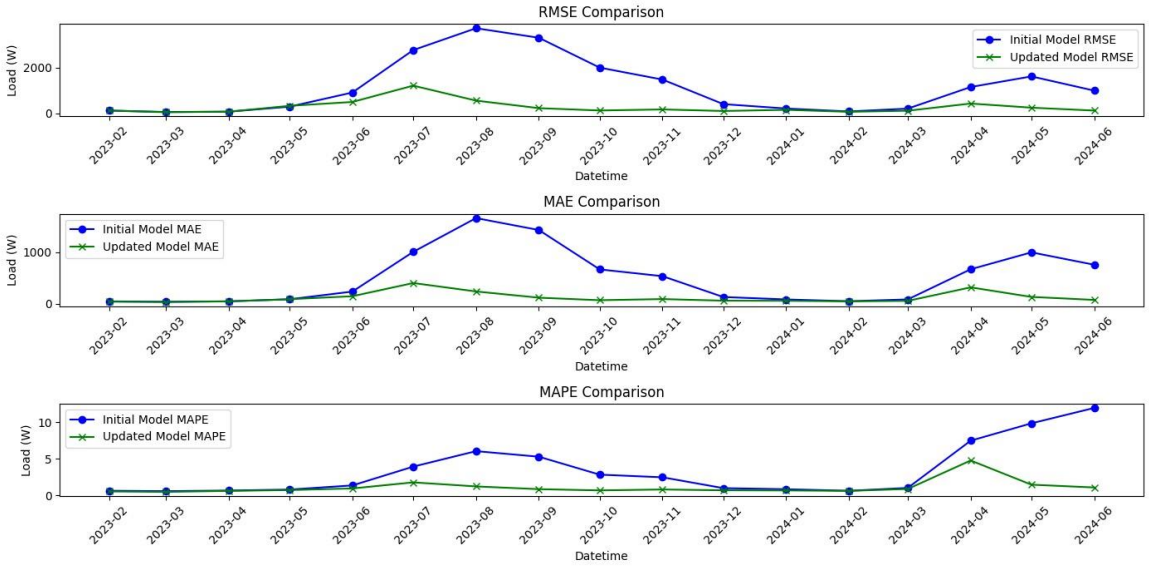


Figure 4.11: Plotting of Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2023 Onwards)

Table 4.5: Overall Performance of Static Model and Updated Models Across Each Months (Start Update from 2023 Onwards)

Evaluation Month	RMSE(kW)		MAE(kW)		MAPE(%)	
	Static Model	Updated Model	Static Model	Updated Model	Static Model	Updated Model
2023/02	0.1268	0.1267	0.0486	0.0431	0.6327	0.5422
2023/03	0.0638	0.0556	0.0417	0.0358	0.5771	0.4799
2023/04	0.0777	0.0789	0.0498	0.0482	0.6610	0.6168
2023/05	0.2961	0.3323	0.0893	0.0911	0.7964	0.7286
2023/06	0.9126	0.5014	0.2388	0.1482	1.3644	0.9505
2023/07	2.7612	1.2127	1.0085	0.4044	3.9361	1.7803
2023/08	3.7131	0.5597	1.6637	0.2409	6.0567	1.2274
2023/09	3.2976	0.2318	1.4319	0.1200	5.2929	0.8522
2023/10	1.9976	0.1294	0.6687	0.0716	2.8380	0.6822
2023/11	1.4768	0.1754	0.5372	0.0936	2.4775	0.8068
2023/12	0.4069	0.1072	0.1331	0.0622	0.9928	0.6981
2024/01	0.2161	0.1588	0.0863	0.0601	0.8469	0.6750
2024/02	0.0868	0.0727	0.0515	0.0490	0.6262	0.6193
2024/03	0.2119	0.1200	0.0871	0.0585	1.0256	0.8781
2024/04	1.1519	0.4323	0.6709	0.3206	7.4883	4.7778
2024/05	1.6152	0.2525	0.9993	0.1349	9.8654	1.4634
2024/06	0.9946	0.1259	0.7584	0.0774	11.9732	1.0841

4.9 Summary

In both scenarios, the Adaptive CatBoost model proved to be more effective than the static model in forecasting power consumption over time. The primary difference between the two scenarios was the initial training period: in Scenario 1, the model was trained on data from 2021 to 2023, while in Scenario 2, it was trained on data from 2021 to 2022, with updates beginning earlier in 2023.

Across both scenarios, the monthly updates of the adaptive model allowed it to continuously improve and adjust to changes in consumption patterns. This adaptive capability resulted in consistently lower errors compared to the static model, which struggled to maintain accuracy as time progressed and consumption trends shifted. The adaptive model's ability to incorporate new data each month made it more responsive to changes, leading to more reliable predictions.

Overall, the results from both scenarios demonstrate the effectiveness of the adaptive approach, with the Adaptive CatBoost model outperforming the static model in predicting future power consumption, particularly as consumption patterns evolved over time. This underscores the importance of periodic model updates in dynamic forecasting environments.

CHAPTER 5

CONCLUSION

5.1 Conclusion

This project successfully demonstrated the potential of various machine learning and deep learning models in forecasting power consumption using limited data, specifically datetime and power variables. The models tested included CatBoost, LSTM, GRU, CNN, and CNN-LSTM, each providing insights into different aspects of load forecasting. Among them, the adaptive CatBoost model proved to be the most effective in capturing evolving consumption patterns by updating monthly, significantly outperforming the static models, which struggled to generalize to future data. Both LSTM and GRU models exhibited strong performance in handling temporal dependencies, while CNN-LSTM showcased its capability in combining spatial and temporal features. However, CNN alone underperformed, indicating that pure spatial feature extraction may not be sufficient for this type of forecasting task. Overall, the project demonstrated that machine learning models, even when limited to minimal input features, can provide robust and accurate predictions, particularly when dynamically updated to reflect changing trends in power consumption.

5.2 Recommendations for Future Work

To build upon the findings of this project and further enhance the forecasting models, several key areas of improvement are recommended. First, expanding the dataset to include additional features such as weather data, occupancy levels, or economic indicators could significantly improve model performance. These additional variables would provide the models with more context and enable them to capture more complex consumption patterns, thereby improving both short-term and long-term predictions.

Second, future work should explore advanced deep learning architectures, particularly transformer-based models, which have shown promising results in time-series forecasting tasks. The self-attention mechanism in transformers could potentially improve the handling of long-term

dependencies and provide more accurate predictions for datasets with limited features, like the one used in this project. Integrating these architectures could offer a significant leap in forecasting accuracy.

Finally, implementing an ensemble approach, combining the strengths of multiple models, could provide more robust predictions. For instance, combining the adaptive capabilities of CatBoost with the temporal learning strengths of LSTM or GRU could yield a more reliable system that excels at both trend detection and short-term fluctuations. This multi-model ensemble could offer better generalization and accuracy than relying on a single model, especially in dynamic environments where consumption patterns change frequently.

REFERENCES

Ahmad, T., Madoński, R., Zhang, D., Huang, C., & Mujeeb, A. ， 2022. Data-driven probabilistic machine learning in sustainable smart energy/smart energy systems: Key developments, challenges, and future research opportunities in the context of smart grid paradigm. Elsevier BV. https://doi.org/10.1016/j.rser.2022.112128

Al-Shatri, H., & Awad, M.， 2024. Time series forecasting of electricity consumption using hybrid model of recurrent neural networks and genetic algorithms. Elsevier BV, 100004-100004. https://doi.org/10.1016/j.meaene.2024.100004

Bian, J., Wang, J., & Yece, Q., 2024. A novel study on Power Consumption of an HVAC system using CatBoost and AdaBoost algorithms combined with the Metaheuristic Algorithms. Energy, 302, 131841. https://doi.org/10.1016/j.energy.2024.131841

Bouktif, S., Fiaz, A., Ouni, A., & Serhani, M A.，2019. Single and Multi-Sequence Deep Learning Models for Short and Medium Term Electric Load Forecasting. Multidisciplinary Digital Publishing Institute, 12(1), 149-149. https://doi.org/10.3390/en12010149

Deng, X., Ye, A., Zhong, J., Xu, D., Yang, W., Song, Z., Zhang, Z., Guo, J., Wang, T., Tian, Y., Pan, H., Zhang, Z., Wang, H., Wu, C., Shao, J., & Chen, X. (2022). Bagging–XGBoost algorithm based extreme weather identification and short-term load forecasting model. Energy Reports, 8, 8661–8674. https://doi.org/10.1016/j.egyr.2022.06.072

Duan, X.， 2020. Application of Deep Learning in Power load Analysis. https://doi.org/10.46300/9106.2020.14.92

Elamin, N., & Fukushige, M.， 2018. Modeling and forecasting hourly electricity demand by SARIMAX with interactions. Elsevier BV, 165, 257-268. https://doi.org/10.1016/j.energy.2018.09.157

Eren, Y., & Küçükdemiral, İ B., 2024. A comprehensive review on deep learning approaches for short-term load forecasting. Elsevier BV, 189, 114031-114031. https://doi.org/10.1016/j.rser.2023.114031

Fan, G., Han, Y., Wang, J., Jia, H., Peng, L., Huang, H., & Hong, W., 2023. A new intelligent hybrid forecasting method for power load considering uncertainty. Elsevier BV, 280, 111034-111034. https://doi.org/10.1016/j.knosys.2023.111034

Fekri, M N., Patel, H R., Grolinger, K., & Sharma, V., 2021. Deep learning for load forecasting with smart meter data: Online Adaptive Recurrent Neural Network. Elsevier BV. https://doi.org/10.1016/j.apenergy.2020.116177\

Hadjout, D., Torres, J F., Troncoso, A., Sebaa, A., & Martínez‐Álvarez, F., 2022. Electricity consumption forecasting based on ensemble deep learning with application to the Algerian market. Elsevier BV, 243, 123060-123060. https://doi.org/10.1016/j.energy.2021.123060

Hancock, J.T. and Khoshgoftaar, T.M. (2020a) CatBoost for Big Data: An interdisciplinary review - journal of big data, SpringerLink. Available at: https://link.springer.com/article/10.1186/s40537-020-00369-8 (Accessed: 17 September 2024).

Huang, Y., & Huang, S. (2020, September 1). A Short-Term load forecasting model based on improved random Forest algorithm. IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/9356670

Kim, T Y., & Cho, S B., 2019. Predicting residential energy consumption using CNN-LSTM neural networks. Elsevier BV, 182, 72-81. https://doi.org/10.1016/j.energy.2019.05.230

Kolter, J Z., & Maloof, M A., 2004. Dynamic weighted majority: a new ensemble method for tracking concept drift. https://doi.org/10.1109/icdm.2003.1250911

Li, Y., Zhang, F., Liu, Y., Liao, H., Zhang, H., & Chung, C., 2023. Residential Load Forecasting: An Online-Offline Deep Kernel Learning Method. Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/tpwrs.2023.3299637

Lin, J., Ma, J., Zhu, J., & Chen, Y., 2022. Short-term load forecasting based on LSTM networks considering attention mechanism. Elsevier BV, 137, 107818-107818. https://doi.org/10.1016/j.ijepes.2021.107818

Massaoudi, M., Refaat, S S., Chihi, I., Trabelsi, M., Abu‐Rub, H., & Oueslati, F S., 2020. Short-Term Electric Load Forecasting Based on Data-Driven Deep Learning Techniques. https://doi.org/10.1109/iecon43393.2020.9255098

Michael, N E., Mishra, M., Hasan, S., & Al‐Durra, A., 2022. Short-Term Solar Power Predicting Model Based on Multi-Step CNN Stacked LSTM Technique. Multidisciplinary Digital Publishing Institute. https://doi.org/10.3390/en15062150

Owen, L. (2022, July). Hyperparameter Tuning with Python. Google Books. https://books.google.com.my/books?hl=en&lr=&id=CqF-EAAAQBAJ&oi=fnd&pg=PP1&dq=What%27s+the+optimal+machine+learning+data+split+ratio+and+how+to+achieve+it%3F+Learn+how+to+avoid+overfitting+&ots=RPIBpjE2lF&sig=XipcHVvm1op8m90ufAnx2g2MXfo&redir_esc=y#v=onepage&q&f=false

Qin, S. J., MacGregor, J. F., Ge, Z., Daszykowski, M., Maronna, R. A., Serneels, S., Templ, M., Downs, J. J., Luo, L., Nomikos, P., Kano, M., Cai, B., Yu, H., Yu, J., & Bao, S., 2019. Robust monitoring of industrial processes using process data with outliers and missing values. Chemometrics and Intelligent Laboratory Systems. https://www.sciencedirect.com/science/article/abs/pii/S0169743919300735

Sajjad, M., Khan, Z A., Ullah, A., Hussain, T., Ullah, W., Lee, M Y., & Baik, S W., 2020. A Novel CNN-GRU-Based Hybrid Approach for Short-Term Residential Load Forecasting. Institute of Electrical and Electronics Engineers, 8, 143759-143768. https://doi.org/10.1109/access.2020.3009537

Taleb, I., Guérard, G., Fauberteau, F., & Nguyen, N., 2022. A Flexible Deep Learning Method for Energy Forecasting. Multidisciplinary Digital Publishing Institute, 15(11), 3926-3926. https://doi.org/10.3390/en15113926

Tong, X., Wang, J., Zhang, C., Wu, T., Wang, H., & Wang, Y., 2022. LS-LSTM-AE: Power load forecasting via Long-Short series features and LSTM-Autoencoder. Elsevier BV, 8, 596-603. https://doi.org/10.1016/j.egyr.2021.11.172

Xu, J., & Baldick, R., 2019. Day-Ahead Price Forecasting in ERCOT Market Using Neural Network Approaches. https://doi.org/10.1145/3307772.3331024

Zhang, N., Li, Z., Zou, X., & Quiring, S. M. (2019, October 17). Comparison of three short-term load forecast models in Southern California. Energy. https://www.sciencedirect.com/science/article/abs/pii/S0360544219320535

Zhou, Y., Liu, Y., Wang, D., & Liu, X. (2021). Comparison of machine-learning models for predicting short-term building heating load using operational parameters. Energy and Buildings, 253, 111505. https://doi.org/10.1016/j.enbuild.2021.111505

Zulfiqar, M H., Kamran, M., Rasheed, M B., Alquthami, T., & Milyani, A H., 2022. A Short-Term Load Forecasting Model Based on Self-Adaptive Momentum Factor and Wavelet Neural Network in Smart Grid. Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/access.2022.3192433

APPENDICES

Appendix A: Codes

Appendix A : LSTM Model Code

```
import os
import numpy as np
import pandas as pd
import json
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense,
Dropout, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import re
from sklearn.metrics import mean_squared_error,
mean_absolute_error, mean_absolute_percentage_error
import csv

# Load and preprocess the dataset
file_path = 'fyp_dataset(2021-2023).csv'  # Replace
with your dataset path
df = pd.read_csv(file_path)
df['Date/Time'] = pd.to_datetime(df['Date/Time'],
dayfirst=True)  # Corrected date parsing
df.set_index('Date/Time', inplace=True)
df.sort_index(inplace=True)

# Function to remove non-numeric characters and
convert to float
def clean_column(column):
    return column.apply(lambda x: float(re.sub(r
'[^-?\d.]+', '', str(x))))

columns_to_clean = ['Voltage Ph-A Avg', 'Voltage Ph-B
 Avg', 'Voltage Ph-C Avg',
                    'Current Ph-A Avg', 'Current Ph-B
 Avg', 'Current Ph-C Avg',
                    'Power Factor Total']
for col in columns_to_clean:
    df[col] = clean_column(df[col])
df['Power Factor Total'] = df['Power Factor Total'].
abs()
df['Total Power'] = (df['Voltage Ph-A Avg'] * df['
```

```
32 Current Ph-A Avg'] * df['Power Factor Total'] +
33                       df['Voltage Ph-B Avg'] * df['
   Current Ph-B Avg'] * df['Power Factor Total'] +
34                       df['Voltage Ph-C Avg'] * df['
   Current Ph-C Avg'] * df['Power Factor Total'])
35 total_power = df[['Total Power']]
36 scaler = MinMaxScaler(feature_range=(0, 1))
37 total_power_scaled = scaler.fit_transform(total_power
   )
38
39 # Create sequences for the LSTM model
40 def create_dataset(dataset, time_step=1):
41     dataX, dataY = [], []
42     for i in range(len(dataset) - time_step):
43         a = dataset[i:(i + time_step), 0]
44         dataX.append(a)
45         dataY.append(dataset[i + time_step, 0])
46     return np.array(dataX), np.array(dataY)
47
48 time_step = 48  # One day
49 X, y = create_dataset(total_power_scaled, time_step)
50 X = X.reshape((X.shape[0], X.shape[1], 1))
51
52 # Split the dataset into training, validation, and
   test sets (80:10:10)
53 train_size = int(len(X) * 0.8)
54 val_size = int(len(X) * 0.1)
55 test_size = len(X) - train_size - val_size
56
57 X_train, X_val, X_test = X[:train_size], X[train_size
   :train_size + val_size], X[train_size + val_size:]
58 y_train, y_val, y_test = y[:train_size], y[train_size
   :train_size + val_size], y[train_size + val_size:]
59
60 # Model parameters
61 units = 64
62 dropout_rate = 0.01
63 layers = 3
64 learning_rate = 0.001
65
66 # Build the LSTM model using the specified structure
```

```
model = Sequential()
model.add(Input(shape=(time_step, 1)))
model.add(LSTM(units=units, return_sequences=(layers > 1)))
model.add(Dropout(dropout_rate))
for i in range(1, layers):
    model.add(LSTM(units=units, return_sequences=(i < layers - 1)))
    model.add(Dropout(dropout_rate))
model.add(Dense(units=1))
model.compile(optimizer=Adam(learning_rate=learning_rate), loss='mean_squared_error')

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, mode='min', restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train, y_train, epochs=60, batch_size=20, validation_data=(X_val, y_val), verbose=1, callbacks=[early_stopping])

# Save the final model
model_save_path = 'D:/Uni/fyp dataset/final_LSTM_load_forecasting_model_best_filtered.h5'
model.save(model_save_path)

# Save the history
history_save_path = 'D:/Uni/fyp dataset/LSTM_training_history_best_filtered.json'
with open(history_save_path, 'w') as f:
    json.dump(history.history, f)

# Plot training and validation loss
plt.figure(figsize=(14, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
```

```
 98 plt.ylabel('Loss')
 99 plt.legend()
100 plt.show()
101
102 # Evaluate the model on the test set
103 test_predictions = model.predict(X_test)
104 test_predictions = scaler.inverse_transform(
    test_predictions)
105 y_test_inv = scaler.inverse_transform(y_test.reshape
    (-1, 1))
106
107 # Calculate test metrics
108 test_rmse = np.sqrt(mean_squared_error(y_test_inv,
    test_predictions))
109 test_mae = mean_absolute_error(y_test_inv,
    test_predictions)
110 test_mape = mean_absolute_percentage_error(
    y_test_inv, test_predictions)
111
112 print(f'Test RMSE: {test_rmse}')
113 print(f'Test MAE: {test_mae}')
114 print(f'Test MAPE: {test_mape}')
115
```

Appendix B : GRU Model Code

```
import os
import numpy as np
import pandas as pd
import json
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense,
Dropout, Input
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import re
from sklearn.metrics import mean_squared_error,
mean_absolute_error, mean_absolute_percentage_error
import csv
from tensorflow.keras.callbacks import EarlyStopping

# Load and preprocess the dataset
file_path = 'D:/Uni/fyp dataset/fyp_dataset(2021-2023
).csv'  # Replace with your dataset path
df = pd.read_csv(file_path)
df['Date/Time'] = pd.to_datetime(df['Date/Time'],
dayfirst=True)  # Corrected date parsing
df.set_index('Date/Time', inplace=True)
df.sort_index(inplace=True)

# Function to remove non-numeric characters and
convert to float
def clean_column(column):
    return column.apply(lambda x: float(re.sub(r
'[^-?\d.]+', '', str(x))))

columns_to_clean = ['Voltage Ph-A Avg', 'Voltage Ph-B
 Avg', 'Voltage Ph-C Avg',
                    'Current Ph-A Avg', 'Current Ph-B
 Avg', 'Current Ph-C Avg',
                    'Power Factor Total']
for col in columns_to_clean:
    df[col] = clean_column(df[col])
df['Power Factor Total'] = df['Power Factor Total'].
abs()
df['Total Power'] = (df['Voltage Ph-A Avg'] * df['
```

```
32 Current Ph-A Avg'] * df['Power Factor Total'] +
33                       df['Voltage Ph-B Avg'] * df['
   Current Ph-B Avg'] * df['Power Factor Total'] +
34                       df['Voltage Ph-C Avg'] * df['
   Current Ph-C Avg'] * df['Power Factor Total'])
35 total_power = df[['Total Power']]
36 scaler = MinMaxScaler(feature_range=(0, 1))
37 total_power_scaled = scaler.fit_transform(total_power
   )
38
39 # Create sequences for the GRU model
40 def create_dataset(dataset, time_step=1):
41     dataX, dataY = [], []
42     for i in range(len(dataset) - time_step):
43         a = dataset[i:(i + time_step), 0]
44         dataX.append(a)
45         dataY.append(dataset[i + time_step, 0])
46     return np.array(dataX), np.array(dataY)
47
48 time_step = 48  # One day
49 X_train, y_train = create_dataset(total_power_scaled
   , time_step)
50 X_train = X_train.reshape((X_train.shape[0], X_train.
   shape[1], 1))
51
52 # Hyperparameter grid search focusing on the most
   promising options
53 units_options = [50]
54 dropout_options = [0.01]  # Focused on the best-
   performing dropout rate
55 layers_options = [3, 4]  # Extended to investigate
   deeper models
56 learning_rate_options = [0.001]  # Focused on the
   best-performing learning rate
57
58 best_rmse = float('inf')
59 best_params = {}
60
61 # Initialize CSV file to log search results,
   appending data instead of overwriting
62 log_file = 'D:/Uni/fyp dataset/
```

```
gru_hyperparameter_search_log_filtered.csv'
if not os.path.exists(log_file):
    with open(log_file, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['Units', 'Dropout Rate', 'Layers', 'Learning Rate', 'RMSE', 'MAE', 'MAPE'])

for units in units_options:
    for dropout in dropout_options:
        for layers in layers_options:
            for lr in learning_rate_options:
                # Build the GRU model
                model = Sequential()
                model.add(Input(shape=(time_step, 1)))  # Use Input layer to define input shape
                model.add(GRU(units=units, return_sequences=(layers > 1)))
                model.add(Dropout(dropout))
                for i in range(1, layers):
                    model.add(GRU(units=units, return_sequences=(i < layers - 1)))
                    model.add(Dropout(dropout))
                model.add(Dense(units=1))
                model.compile(optimizer=Adam(learning_rate=lr), loss='mean_squared_error')

                # Train the model
                history = model.fit(X_train, y_train, epochs=60, batch_size=20, verbose=0)

                # Evaluate the model
                train_predictions = model.predict(X_train)
                train_predictions = scaler.inverse_transform(train_predictions)
                y_train_inv = scaler.inverse_transform(y_train.reshape(-1, 1))
                train_rmse = np.sqrt(mean_squared_error(y_train_inv, train_predictions))
                train_mae = mean_absolute_error(
```

```
 91 y_train_inv, train_predictions)
 92                 train_mape =
    mean_absolute_percentage_error(y_train_inv,
    train_predictions)
 93
 94                 # Log the hyperparameters and
    metrics, appending data
 95                 with open(log_file, mode='a',
    newline='') as file:
 96                     writer = csv.writer(file)
 97                     writer.writerow([units, dropout
    , layers, lr, train_rmse, train_mae, train_mape])
 98
 99                 print(f"Units: {units}, Dropout: {
    dropout}, Layers: {layers}, Learning Rate: {lr},
    RMSE: {train_rmse}, MAE: {train_mae}, MAPE: {
    train_mape}")
100
101                 if train_rmse < best_rmse:
102                     best_rmse = train_rmse
103                     best_params = {
104                         'units': units,
105                         'dropout': dropout,
106                         'layers': layers,
107                         'learning_rate': lr
108                     }
109
110 print(f"Best Parameters: {best_params}")
111 print(f"Best RMSE: {best_rmse}")
112
113 # Build and train the final model with the best
    parameters, including early stopping
114 model = Sequential()
115 model.add(Input(shape=(time_step, 1)))  # Use Input
    layer for the best model
116 model.add(GRU(units=best_params['units'],
    return_sequences=(best_params['layers'] > 1)))
117 model.add(Dropout(best_params['dropout']))
118 for i in range(1, best_params['layers']):
119     model.add(GRU(units=best_params['units'],
    return_sequences=(i < best_params['layers'] - 1)))
```

```
120     model.add(Dropout(best_params['dropout']))
121 model.add(Dense(units=1))
122 model.compile(optimizer=Adam(learning_rate=
    best_params['learning_rate']), loss='
    mean_squared_error')
123
124 # Implement early stopping
125 early_stopping = EarlyStopping(monitor='val_loss',
    patience=10, restore_best_weights=True)
126
127 # Split the data into training and validation sets (
    80/20 split)
128 train_size = int(len(X_train) * 0.8)
129 X_train_split, X_val_split = X_train[:train_size],
    X_train[train_size:]
130 y_train_split, y_val_split = y_train[:train_size],
    y_train[train_size:]
131
132 # Train the model with early stopping and validation
     data
133 history = model.fit(X_train_split, y_train_split,
    validation_data=(X_val_split, y_val_split), epochs=
    60, batch_size=20, verbose=1, callbacks=[
    early_stopping])
134
135 # Save the final model
136 model_save_path = 'D:/Uni/fyp dataset/
    final_GRU_load_forecasting_model_best_filtered.h5'
137 model.save(model_save_path)
138
139 # Save the history
140 history_save_path = 'D:/Uni/fyp dataset/
    GRU_training_history_best_filtered.json'
141 with open(history_save_path, 'w') as f:
142     json.dump(history.history, f)
143
144 # Plot training and validation loss
145 plt.figure(figsize=(14, 5))
146 plt.plot(history.history['loss'], label='Training
    Loss')
147 plt.plot(history.history['val_loss'], label='
```

```
147 Validation Loss')
148 plt.title('Training and Validation Loss')
149 plt.xlabel('Epoch')
150 plt.ylabel('Loss')
151 plt.legend()
152 plt.grid(True)
153 plt.show()
154
155 # Make predictions on the training dataset
156 train_predictions = model.predict(X_train)
157 train_predictions = scaler.inverse_transform(
    train_predictions)
158 y_train_inv = scaler.inverse_transform(y_train.
    reshape(-1, 1))
159 train_rmse = np.sqrt(np.mean((train_predictions -
    y_train_inv) ** 2))
160 print(f'Training RMSE with best parameters: {
    train_rmse}')
161
```

Appendix C : CatBoost Model Code

```
import os
import numpy as np
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error,
mean_absolute_error, mean_absolute_percentage_error
from catboost import CatBoostRegressor, Pool
import warnings
import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")

# Load and preprocess the dataset
file_path = 'D:/Uni/fyp dataset/fyp_dataset(2021-2023
).csv'
df = pd.read_csv(file_path)
df['Date/Time'] = pd.to_datetime(df['Date/Time'],
dayfirst=True)
df.set_index('Date/Time', inplace=True)
df.sort_index(inplace=True)

# Remove unit and clean columns
def clean_column(column):
    return column.apply(lambda x: float(re.sub(r
'[^-?\d.]+', '', str(x))) if re.sub(r'[^-?\d.]+', ''
, str(x)) else np.nan)

columns_to_clean = ['Voltage Ph-A Avg', 'Voltage Ph-B
 Avg', 'Voltage Ph-C Avg',
                    'Current Ph-A Avg', 'Current Ph-B
 Avg', 'Current Ph-C Avg',
                    'Power Factor Total']
for col in columns_to_clean:
    df[col] = clean_column(df[col])

# Compute total power
df['Power Factor Total'] = df['Power Factor Total'].
abs()
df['Total Power'] = (df['Voltage Ph-A Avg'] * df['
```

```
33 Current Ph-A Avg'] * df['Power Factor Total'] +
34                      df['Voltage Ph-B Avg'] * df['
   Current Ph-B Avg'] * df['Power Factor Total'] +
35                      df['Voltage Ph-C Avg'] * df['
   Current Ph-C Avg'] * df['Power Factor Total'])
36
37 # Feature engineering
38 def create_features(df):
39     df['hour'] = df.index.hour
40     df['dayofweek'] = df.index.dayofweek
41     df['quarter'] = df.index.quarter
42     df['month'] = df.index.month
43     df['year'] = df.index.year
44     df['dayofyear'] = df.index.dayofyear
45     df['dayofmonth'] = df.index.day
46     df['weekofyear'] = df.index.isocalendar().week
47     return df
48
49 df = create_features(df)
50
51 # Create lag features and rolling statistics
52 for lag in range(1, 49):  # Lag features for up to 24
    hours
53     df[f'Total Power Lag {lag}'] = df['Total Power'].
   shift(lag)
54
55 window_sizes = [48, 96]  # Rolling windows for 24
   hours and 48 hours
56 for window in window_sizes:
57     df[f'Total Power Rolling Mean {window}'] = df['
   Total Power'].rolling(window=window).mean()
58     df[f'Total Power Rolling Std {window}'] = df['
   Total Power'].rolling(window=window).std()
59
60 df.dropna(inplace=True)
61
62 # Data normalization for feature columns and target
   variable
63 feature_cols = ['Voltage Ph-A Avg', 'Voltage Ph-B Avg
   ', 'Voltage Ph-C Avg',
64                 'Current Ph-A Avg', 'Current Ph-B Avg
```

```
', 'Current Ph-C Avg',
                 'Power Factor Total', 'hour', '
dayofweek', 'quarter',
                 'month', 'year', 'dayofyear', '
dayofmonth', 'weekofyear'] + \
                [f'Total Power Lag {lag}' for lag in
range(1, 49)] + \
                [f'Total Power Rolling Mean {window}'
 for window in window_sizes] + \
                [f'Total Power Rolling Std {window}'
for window in window_sizes]

X = df[feature_cols]
y = df['Total Power']

# Scaler for X and y
scaler_X = MinMaxScaler(feature_range=(0, 1))
X_scaled = scaler_X.fit_transform(X)

scaler_y = MinMaxScaler(feature_range=(0, 1))
y_scaled = scaler_y.fit_transform(y.values.reshape(-
1, 1))

# First split: 90% training + validation, 10% test (
final evaluation, unseen data)
X_train_val, X_test, y_train_val, y_test =
train_test_split(X_scaled, y_scaled, test_size=0.1,
random_state=1)

# Second split: From 90%, split into 80% training
and 10% validation
X_train, X_val, y_train, y_val = train_test_split(
X_train_val, y_train_val, test_size=0.1111,
random_state=1)
# 10% / 90% = 0.1111 to ensure correct ratio

# Convert to CatBoost Pool
train_pool = Pool(X_train, y_train)
val_pool = Pool(X_val, y_val)

# Use the best parameters directly (final model
```

```
 92 training)
 93 best_params = {
 94     'depth': 4,
 95     'learning_rate': 0.1,
 96     'l2_leaf_reg': 1,
 97     'iterations': 1000,
 98     'eval_metric': 'RMSE',
 99     'verbose': 0,
100     'use_best_model': True
101 }
102
103 # Train the final model
104 final_model = CatBoostRegressor(**best_params)
105 final_model.fit(train_pool, eval_set=val_pool,
    verbose=1)
106
107 # Model evaluation on the test set using best
    parameters
108 y_test_pred = final_model.predict(X_test)
109 y_test_pred_unscaled = scaler_y.inverse_transform(
    y_test_pred.reshape(-1, 1))  # Unscale predictions
110 y_test_unscaled = scaler_y.inverse_transform(y_test
    )  # Unscale true test values
111
112 test_rmse = np.sqrt(mean_squared_error(
    y_test_unscaled, y_test_pred_unscaled))
113 test_mae = mean_absolute_error(y_test_unscaled,
    y_test_pred_unscaled)
114 test_mape = mean_absolute_percentage_error(
    y_test_unscaled, y_test_pred_unscaled)
115
116 print(f'Test RMSE (Unscaled): {test_rmse}')
117 print(f'Test MAE (Unscaled): {test_mae}')
118 print(f'Test MAPE (Unscaled): {test_mape}')
119
120 # Save the final model
121 final_model.save_model('D:/Uni/fyp dataset/
    final_catboost_model.cbm')
122
123 # Plot the results
124 plt.figure(figsize=(14, 5))
```

```
plt.plot(y_test_unscaled, color='blue', label='Actual Load', linewidth=2)
plt.plot(y_test_pred_unscaled, color='red', label='Predicted Load (CatBoost)', linewidth=2)
plt.title('Load Prediction for Test Set')
plt.xlabel('Time')
plt.ylabel('Load')
plt.legend()
plt.show()

# Plot training and validation loss
eval_results = final_model.get_evals_result()
plt.figure(figsize=(14, 5))
plt.plot(eval_results['learn']['RMSE'], label='Training Loss')
plt.plot(eval_results['validation']['RMSE'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

Appendix D : CNN-LSTM Model Code

```
import os
import numpy as np
import pandas as pd
import json
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense,
Dropout, Conv1D, MaxPooling1D
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error,
mean_absolute_error, mean_absolute_percentage_error
import tensorflow as tf
import tensorflow_probability as tfp
from tensorflow.keras import backend as K
from tensorflow.keras.metrics import
RootMeanSquaredError
import csv
import re

# Load and preprocess the dataset
file_path = 'D:/Uni/fyp dataset/fyp_dataset(2021-2023
).csv'
df = pd.read_csv(file_path)
df['Date/Time'] = pd.to_datetime(df['Date/Time'],
dayfirst=True)
df.set_index('Date/Time', inplace=True)
df.sort_index(inplace=True)

# Function to remove non-numeric characters and
convert to float
def clean_column(column):
    return column.apply(lambda x: float(re.sub(r
'[^-?\d.]+', '', str(x))))

columns_to_clean = ['Voltage Ph-A Avg', 'Voltage Ph-B
 Avg', 'Voltage Ph-C Avg',
                    'Current Ph-A Avg', 'Current Ph-B
 Avg', 'Current Ph-C Avg',
```

```
                    'Power Factor Total']
for col in columns_to_clean:
    df[col] = clean_column(df[col])
df['Power Factor Total'] = df['Power Factor Total'].
abs()
df['Total Power'] = (df['Voltage Ph-A Avg'] * df['
Current Ph-A Avg'] * df['Power Factor Total'] +
                     df['Voltage Ph-B Avg'] * df['
Current Ph-B Avg'] * df['Power Factor Total'] +
                     df['Voltage Ph-C Avg'] * df['
Current Ph-C Avg'] * df['Power Factor Total'])
total_power = df[['Total Power']]

# Split the data into training, validation, and test
sets
n = len(total_power)
n_train = int(0.8 * n)
n_val = int(0.1 * n)
n_test = n - n_train - n_val

train_data = total_power[:n_train].values
val_data = total_power[n_train:n_train + n_val].
values
test_data = total_power[-n_test:].values

# Normalize the data using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
train_data = scaler.fit_transform(train_data)
val_data = scaler.transform(val_data)
test_data = scaler.transform(test_data)

# Create sequences for the CNN-LSTM model
window_size = 48  # One day

def create_window_dataset(data, window_size):
    X, y = [], []
    for i in range(window_size, len(data)):
        X.append(data[i - window_size:i])
        y.append(data[i])
    return np.array(X), np.array(y)

```

```
X_train, y_train = create_window_dataset(train_data, window_size)
X_val, y_val = create_window_dataset(val_data, window_size)
X_test, y_test = create_window_dataset(test_data, window_size)

# Custom correlation coefficient metric
def cc(y_true, y_pred):
    cov = tfp.stats.covariance(y_true, y_pred, sample_axis=0)
    std_true = K.std(y_true)
    std_pred = K.std(y_pred)
    return cov / (std_true * std_pred)

# Build the CNN-LSTM model with 'relu' activation and 'Adamax' optimizer
model = Sequential()

# Add Conv1D layers
model.add(Conv1D(filters=16, kernel_size=1, activation='relu',
                 input_shape=(window_size, 1)))
model.add(MaxPooling1D(pool_size=2))

# Add LSTM layers
model.add(LSTM(64, return_sequences=False))
model.add(Dropout(0.01))

# Add Dense layers
model.add(Dense(32, activation='relu'))

model.add(Dense(1))  # Output layer

# Compile the model with Adamax optimizer
optimizer = Adamax(learning_rate=0.001)
model.compile(optimizer=optimizer,
              loss=tf.keras.losses.Huber(),
              metrics=[RootMeanSquaredError(), cc])

# Early stopping
```

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='min', restore_best_weights=True)

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val),
                    verbose=1, callbacks=[early_stopping])

# Evaluate the model
test_predictions = model.predict(X_test)
test_predictions = scaler.inverse_transform(test_predictions)
y_test_inv = scaler.inverse_transform(y_test)

# Calculate metrics
test_rmse = np.sqrt(mean_squared_error(y_test_inv, test_predictions))
test_mae = mean_absolute_error(y_test_inv, test_predictions)
test_mape = mean_absolute_percentage_error(y_test_inv, test_predictions)

print(f'Test RMSE: {test_rmse}')
print(f'Test MAE: {test_mae}')
print(f'Test MAPE: {test_mape}')

# Save the final model
model_save_path = 'D:/Uni/fyp dataset/final_CNN_LSTM_load_forecasting_model_relu_Adamax.h5'
model.save(model_save_path)

# Save the training history
history_save_path = 'D:/Uni/fyp dataset/CNN_LSTM_training_history_relu_Adamax.json'
with open(history_save_path, 'w') as f:
    json.dump(history.history, f)

# Plot training and validation loss
```

```
plt.plot(y_test_unscaled, color='blue', label='Actual Load', linewidth=2)
plt.plot(y_test_pred_unscaled, color='red', label='Predicted Load (CatBoost)', linewidth=2)
plt.title('Load Prediction for Test Set')
plt.xlabel('Time')
plt.ylabel('Load')
plt.legend()
plt.show()

# Plot training and validation loss
eval_results = final_model.get_evals_result()
plt.figure(figsize=(14, 5))
plt.plot(eval_results['learn']['RMSE'], label='Training Loss')
plt.plot(eval_results['validation']['RMSE'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.show()

```