TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A THESIS REPORT ON

SHORT-TERM ELECTRICAL LOAD FORECASTING FOR BANESHWOR FEEDER USING MACHINE AND DEEP LEARNING MODELS

SUBMITTED BY:

SUJIT KOIRALA

(PUL075MSPSE016)

SUPERVISED BY:

PROF. DR. ROGERS S PRESSMAN

SUBMITTED TO:

DEPARTMENT OF ELECTRICAL ENGINEERING

December, 2025

Declaration

I hereby declare that this study/research entitled [**Put the title of the project/thesis here....**] is based on our original research work. Related works on the topic by other researchers have been duly acknowledged. I owe all the liabilities relating to the accuracy and authenticity of the data and any other information included hereunder.

Name of the Student (Roll no)

Date:

Recommendation

This is to certify that this project report entitled [**Put the title of the project/thesis here**] prepared and submitted by [**Put the name & roll no of the student here**], in partial fulfillment of the requirements of the Master degree of Engineering in......, awarded by Tribhuvan University, has been completed under my/our supervision. I/we recommend the same for acceptance by Tribhuvan University.

Name of the Supervisor: ABC
Designation: ABC
Organization: ABC
Date: ABC

Name of the Co-supervisor: ABC
Designation: ABC
Organization: ABC
Date: ABC

Page of Approval

TRIBHUVAN UNIVERSIY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

This project/thesis entitled [**Put the title of the project/thesis here**] prepared and submitted by [**Put the Name and Roll no of the student here**] has been examined by us and is accepted for the award of the Master's degree in [Put name of the program] by Tribhuvan University.

....	....
Supervisor	External examiner
Name	**Name**
Designation	Designation
Organization Name and Address.	Organization Name and Address.

....
Co-Supervisor (if Any)
Name
Designation
Organization Name and Address.

Date of approval:

Copyright

Acknowledgments

I would like to express my sincere gratitude to my supervisor and faculty members of the Department of Electrical Engineering for their valuable guidance, continuous support, and encouragement throughout the course of this project. Their technical insights and constructive feedback were instrumental in shaping this work. I am also thankful to the Nepal Electricity Authority and relevant data-providing institutions for making the load and meteorological data available for this study. Their cooperation greatly contributed to the successful completion of the analysis. Special thanks go to my friends and colleagues for their support, discussions, and motivation during the project period. Finally, I would like to express my heartfelt appreciation to my family for their constant encouragement and support throughout my academic journey.

Sujit Koirala (PUL075MSPSE016)

Abstract

Accurate short-term electrical load forecasting plays a crucial role in the efficient planning and operation of modern power systems. With increasing load variability influenced by weather conditions, temporal patterns, and socio-economic activities, traditional statistical methods often struggle to capture complex and nonlinear demand behavior. This project focuses on short-term electrical load forecasting for the Lekhnath Feeder using machine learning–based approaches.

Historical hourly load data, along with meteorological variables such as air temperature, global solar radiation, and relative humidity, were used to develop predictive models. Comprehensive data preprocessing was performed, including missing value imputation, outlier treatment, temporal feature extraction, and cyclical encoding of time-based variables. Several machine learning models were implemented and evaluated, including Linear Regression, Ridge Regression, Support Vector Regression, Random Forest, Gradient Boosting, and XGBoost. Hyperparameter tuning was applied to improve model performance.

The models were assessed using standard evaluation metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), and R-squared (R^2). The results show that ensemble-based models, particularly tuned XGBoost and Random Forest models, significantly outperform linear and baseline methods. The findings highlight the effectiveness of machine learning techniques for feeder-level short-term load forecasting and provide valuable insights for operational planning and decision-making in power distribution systems.

Keywords: *short-term load forecasting, machine learning, XGBoost, Random Forest, power distribution systems*

Contents

Declaration **ii**

Recommendation **iii**

Page of Approval **iv**

Copyright **v**

Acknowledgements **vi**

Abstract **vii**

Contents **x**

List of Figures **xi**

List of Tables **xii**

List of Abbreviations **xiii**

List of Units and conversions **xiv**

1 Introduction **1**

1.1 Background 1

1.2 Problem Statement 2

1.3 Objectives 2

1.4 Scope 3

1.5 Limitation 3

2 Literature Review **5**

2.1 Related work 5

2.1.1 Machine Learning Approaches 5

2.1.2 Deep Learning Architectures 6

2.1.3 Hybrid and Advanced Architectures 7

2.1.4 Transformer-Based Models 7

2.1.5 Comparative Studies and Ensemble Methods . . . 8

3 Methodology 9

3.1 Overall Workflow . . . 9

3.2 Data Acquisition . . . 10

3.2.1 Data Sources . . . 10

3.2.2 Load Data Acquisition and Structuring Process . . . 11

3.2.3 Weather Data Acquisition and Structuring . . . 12

3.2.4 Final Merging of Load and Weather Data . . . 12

3.3 Data Preprocessing . . . 13

3.3.1 Timestamp Parsing and Standardization . . . 13

3.3.2 Handling Missing Values . . . 13

3.3.3 Outlier Detection and Treatment . . . 14

3.3.4 Weather Data Cleaning and Alignment . . . 14

3.3.5 Temporal Feature Engineering . . . 14

3.3.6 Correlation Analysis and Feature Selection . . . 14

3.3.7 Train–Test Split . . . 15

3.3.8 Final Processed Dataset Structure . . . 15

3.4 Exploratory Data Analysis (EDA) . . . 15

3.4.1 Overview of the Dataset . . . 15

3.4.2 Time-Series Visualization . . . 16

3.4.3 Hourly Load Pattern Analysis . . . 16

3.4.4 Monthly Load Trends . . . 16

3.4.5 Weather Variable Behaviour . . . 16

3.4.6 Load vs. Weather Relationships . . . 17

3.4.7 Correlation Analysis . . . 17

3.5 Model Development . . . 17

3.5.1 Machine Learning Models . . . 17

3.5.2 Deep Learning Models . . . 22

3.6 Model Training and Validation . . . 24

3.6.1 Training of Machine Learning Models . . . 25

3.6.2 Training of Deep Learning Models . . . 26

3.6.3 Validation Approach . . . 28

3.7 Performance Evaluation . . . 28

3.7.1 Mean Absolute Error (MAE) . . . 28

3.7.2 Mean Squared Error (MSE) . . . 29

3.7.3 Root Mean Squared Error (RMSE) 29
3.7.4 Mean Absolute Percentage Error (MAPE) 30
3.7.5 Coefficient of Determination (R^2 Score) 30

4 Results & Discussion 31
4.1 Results 31
4.1.1 Machine Learning Model Performance 31
4.1.2 Deep Learning Model Performance 32

5 Conclusions & Recommendations 34

6 Limitations and Future enhancement 35
References 35
APPENDICES 37

List of Figures

List of Tables

4.1 Machine Learning Models Evaluation Matrix 31
4.2 Deep Learning Models Evaluation Matrix 32

List of Abbreviations

NEA	Nepal Electricity Authority
STLF	Short-Term Load Forecasting
ML	Machine Learning
DL	Deep Learning
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
MLP	Multi-Layer Perceptron
SVR	Support Vector Regression
RF	Random Forest
GBR	Gradient Boosting Regressor
XGBoost	Extreme Gradient Boosting
MAE	Mean Absolute Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAPE	Mean Absolute Percentage Error
SMAPE	Symmetric Mean Absolute Percentage Error
$\mathbf{R^2}$	Coefficient of Determination
MW	Megawatt
BS	Bikram Sambat (Nepali Calendar)
AD	Anno Domini (Gregorian Calendar)
EDA	Exploratory Data Analysis
IQR	Interquartile Range
TFT	Temporal Fusion Transformer
API	Application Programming Interface

List of units and conversions

m^3	Meter cube (Cubic meter)
Sq.ft	Square feet
add	more

1. Introduction

This project focuses on short-term electrical load forecasting at the feeder level using data-driven machine learning and deep learning techniques. Historical load data combined with weather and temporal features were used to model and predict hourly power demand. Multiple forecasting models were developed and evaluated to identify the most effective approach for accurate and reliable load prediction.

1.1 Background

Electricity demand is never constant. It rises and falls with daily routines, temperature changes, business hours, and countless other factors. For a power system operator, being able to predict this demand even just a few hours ahead can make a huge difference. Accurate short-term forecasting helps optimize generation schedules, reduce operational costs, manage peak hours more confidently, and maintain a reliable supply.

Short-Term Load Forecasting (STLF) typically focuses on horizons ranging from one hour to a day ahead. These forecasts are critical for economic dispatch, unit commitment, load flow analysis, and real-time operation. Traditionally, utilities relied on statistical approaches such as linear regression, ARIMA, exponential smoothing, and Holt-Winters. These techniques can work well when patterns are simple, but they struggle with real-world load curves that are nonlinear, noisy, and influenced by many interacting variables.

Machine Learning models like Random Forest, Support Vector Regression, and XGBoost have shown strong results in several energy-related forecasting tasks. Their ability to capture nonlinear relationships makes them a natural fit for electricity load prediction. Likewise, Deep Learning approaches, especially recurrent neural networks such as LSTM and GRU, can learn temporal dependencies more effectively than traditional models.

The Baneshwor Feeder of the Nepal Electricity Authority serves a mixed group of consumers in the Baneshwor region. Its load pattern reflects residential lifestyles, commercial activity, seasonal tourism impacts, and local weather changes. Daily and weekly cycles are clearly visible, but there are also irregularities that simple models fail to capture. As power consumption continues to grow and diversify, the ability to forecast the feeder's short-term load accurately has become even more important. This creates a strong motivation to investigate how modern ML and DL models can improve forecasting performance for this specific feeder.

1.2 Problem Statement

The current forecasting practices for the Baneshwor Feeder rely heavily on manual estimation or basic statistical techniques. These methods do not fully capture the nonlinear and dynamic nature of the load profile, especially when multiple influencing factors, like temperature, humidity, rainfall, weekends, and special events come into play. As a result, prediction errors tend to increase during peak hours, sudden weather changes, and seasonal transitions.

Inaccurate short-term forecasts have several consequences. They can affect how generation is scheduled, leading to either unnecessary reserve margins or inadequate supply. They may increase operational costs and technical losses at the distribution level. In the worst cases, poor foresight during high-demand periods can create voltage drops, reliability concerns, or inefficient load-shedding decisions.

Despite the availability of historical load and weather data, there has not been a systematic study applying and comparing advanced machine learning and deep learning approaches specifically for the Baneshwor Feeder. The lack of a data-driven forecasting system means operators do not yet benefit from models that are capable of learning complex relationships within the data.

This thesis aims to address these gaps by building a complete forecasting framework using multiple ML and DL models, evaluating their performance, and identifying the most suitable approach for accurate short-term load prediction of the Baneshwor Feeder.

1.3 Objectives

To develop and evaluate machine learning and deep learning models for short-term electrical load forecasting of the Baneshwor Feeder to improve prediction accuracy and operational efficiency.

1. To collect and preprocess historical load data and relevant influencing factors such as weather variables and calendar effects for the Baneshwor Feeder.
2. To analyze consumption patterns and influencing factors (e.g., time, weather, festivals).
3. To implement various machine learning models, including Support Vector Regression (SVR), Random Forest (RF), and XGBoost, for load forecasting.
4. To design and train deep learning models such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks to capture temporal dependencies in load data.

5. To evaluate and compare the performance of ML and DL models using standard error metrics (e.g., RMSE, MAPE, MAE, MSE, R-squared).

6. To compare and select the most effective forecasting model.

7. To recommend the most suitable forecasting model for operational use in the Baneshwor Feeder.

1.4 Scope

This study is geographically limited to the Baneshwor Feeder under the Nepal Electricity Authority. The temporal scope focuses on short-term load forecasting with a prediction horizon of up to 24 hours ahead, utilizing historical hourly load data as the primary foundation for model development. The dataset encompasses historical load data from the Baneshwor Feeder, complemented by weather data including temperature, humidity, and rainfall, along with calendar data distinguishing weekdays, weekends, and holidays.

From a technical perspective, the research implements several machine learning models including Support Vector Regression (SVR), Random Forest, and XGBoost, alongside deep learning architectures such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks. The performance of these models is rigorously evaluated using standard metrics including Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), Mean Squared Error (MSE), and the coefficient of determination (R-squared). It should be noted that the accuracy of forecasts is inherently dependent on the quality and completeness of the historical data available. Additionally, this study does not extend to medium-term or long-term forecasting horizons, and the scope explicitly excludes renewable generation forecasting from its analysis.

1.5 Limitation

Despite the promising results obtained in this study, certain limitations were encountered, primarily related to data availability, model assumptions, and scope of analysis.

1. **Dependence on data quality:** Forecast accuracy is limited by the completeness and reliability of the historical load and weather data. Missing values, sensor errors, or inconsistent reporting can influence model performance.

2. **Model sensitivity to sudden changes:** Unexpected events such as outages, festivals, abrupt weather shifts, or abnormal consumption patterns are difficult for data-driven models to predict accurately.

3. **Deep learning computation constraints:** Training LSTM and GRU models requires more computational resources and time compared to ML models. Their performance may vary depending on the hardware used.

4. **Limited feature diversity:** Although weather and calendar data are included, other influential factors like economic activities, special events, or industrial load profiles are not part of the dataset.

5. **Generalization across feeders:** The models developed in this study are tailored specifically to the Baneshwor Feeder and may not generalize directly to other feeders without retraining or adaptation.

2. Literature Review

The literature review presents a comprehensive overview of existing research related to short-term electrical load forecasting using machine learning and deep learning techniques. It examines previously published studies to understand commonly used methodologies, datasets, and performance evaluation approaches in the domain of power system load forecasting. Reviewing prior work helps to identify current research trends, strengths, and limitations of existing models, while also highlighting gaps that motivate the need for this study. By situating the present research within the context of established knowledge, this chapter provides a foundation for model selection and methodological design adopted in this work.

2.1 Related work

There are many previous works done for electrical load forecasting from short-term electrical load forecasting, to medium-term and long-term. Most of the studies have done short-term load forecasting.

2.1.1 Machine Learning Approaches

Singla et al. (2019) employed Artificial Neural Networks for 24-hour short-term load forecasting, utilizing dew point temperature, dry bulb temperature, and humidity as input features. Their work demonstrated the effectiveness of ANN in capturing the relationship between weather variables and electrical load demand. Similarly, Desai et al. (2021) utilized the Prophet model from Meta to perform short-term load forecasting, incorporating time, temperature, humidity, and weather forecast data as features. The Prophet model's ability to handle seasonal patterns and missing data made it suitable for load forecasting applications.

Matrenin et al. (2022) conducted a study on medium-term load forecasting using ensemble machine learning models. They compared XGBoost and AdaBoost against traditional methods including SVR, decision trees, and Random Forest. Their results highlighted the superior performance of gradient boosting techniques for capturing complex load patterns. Aguilar Madrid & Antonio (2021) tested five machine learning models and found XGBoost to be the most accurate for predictions, using historical load data, weather information, and holiday indicators as input features. Their comprehensive evaluation demonstrated XGBoost's ability to handle diverse feature sets effectively.

Guo et al. (2021) analyzed three popular ML methods for load forecasting: Support Vector Machine, Random Forest, and LSTM. They proposed a fusion forecasting approach that combined outputs from all three models, demonstrating that ensemble methods could improve prediction accuracy beyond individual model performance. Saglam et al. (2024) performed a comparison between optimization methods (Particle Swarm Optimization, Dandelion Optimizer, Growth Optimizer) and machine learning models (SVR, ANN) for instantaneous peak electrical load forecasting. They found that ANN combined with Growth Optimizer outperformed other models and identified a strong positive correlation between GDP and peak load demand.

Jain & Gupta (2024) conducted a comprehensive evaluation of various machine learning algorithms for power load prediction, including Support Vector Machines, LSTM, ensemble classifiers, and Recurrent Neural Networks. Their study emphasized the importance of data preprocessing methods, feature selection strategies, and performance assessment metrics in achieving accurate forecasts. The research demonstrated that ensemble methods and deep learning approaches consistently outperformed traditional statistical models.

2.1.2 Deep Learning Architectures

Chapagain et al. (2021) explored time series regression along with machine learning and deep learning models for electricity demand forecasting in Kathmandu Valley. They found LSTM demonstrating outstanding performance in terms of MAPE and RMSE, using deterministic variables such as day type and temperature. Their work validated the effectiveness of recurrent architectures for capturing temporal dependencies in load data.

Acharya et al. (n.d.) performed short-term electrical load forecasting for the Gothatar feeder using six input features. They found that Recurrent Neural Networks outperformed baseline methods including Single Exponential Smoothing, Double Exponential Smoothing, and Holt-Winter's method. This study confirmed that RNNs could better model the non-linear and time-dependent characteristics of feeder-level load patterns.

Cordeiro-Costas et al. (2023) conducted a comprehensive comparison of load forecasting methods, including Random Forest, SVR, XGBoost, Multi-Layer Perceptron, and LSTM. They also explored Conv-1D models and found that LSTM achieved the lowest error rates across multiple evaluation metrics. Their research highlighted the trade-off between model complexity and forecasting accuracy in practical applications.

Dong et al. (2024) provided a comprehensive survey on deep learning-based short-term electricity load forecasting covering the past decade. They examined the entire forecasting process, including data preprocessing, feature extraction, deep learning modeling and optimization, and results evaluation. The survey identified CNN-LSTM hybrid architectures as widely adopted solutions due to exceptional performance in capturing both spatial and temporal features. Their analysis revealed that most recent studies focused on short-term horizons ranging from one hour to several days ahead.

2.1.3 Hybrid and Advanced Architectures

Wen et al. (2024) proposed a hybrid deep learning model combining Gated Recurrent Units and Temporal Convolutional Networks with an attention mechanism for short-term load forecasting. The GRU captured long-term dependencies in time series data, while TCN efficiently learned patterns and features. The attention mechanism automatically focused on input components most relevant to the prediction task, significantly enhancing model performance. Their approach demonstrated superior accuracy compared to standalone architectures.

Alhussein et al. (2020) developed a hybrid CNN-LSTM framework for short-term individual household load forecasting. The model used CNN layers for feature extraction from input data and LSTM layers for sequence learning. Evaluated on the Smart Grid Smart City dataset, the hybrid model achieved an average MAPE of 40.38%, outperforming standalone LSTM models that obtained 44.06% MAPE. This work demonstrated the effectiveness of combining convolutional and recurrent architectures for handling high volatility in household-level load data.

Hasanat et al. (2024) proposed a parallel multichannel network approach using 1D CNN and Bidirectional LSTM for load forecasting in smart grids. Unlike traditional stacked CNN-LSTM architectures that use convolutions as preprocessing steps, their model independently processed spatial and temporal characteristics through parallel channels. The research addressed the issue of temporal feature neglect in existing models and incorporated cyclic features through trigonometric transformations, achieving superior accuracy on diverse building types.

2.1.4 Transformer-Based Models

Chan & Yeo (2024) proposed a sparse transformer-based approach for electricity load forecasting that addressed the computational complexity limitations of standard transformer architectures. Their model applied sparse attention mechanisms to capture temporal dependencies more efficiently, achieving comparable accuracy to RNN-based state-of-the-art methods while being up to 5 times faster during inference. The model was enhanced to support multivariate inputs including weather data, demonstrating versatility in forecasting loads from individual households to city levels.

Zhang et al. (2022) developed a Time Augmented Transformer model for short-term electrical load forecasting, incorporating temporal features and self-attention mechanisms to capture complex dynamic nonlinear sequence dependencies. Their experimental results showed that multivariate inputs including weather and calendar features produced significantly better predictions than univariate approaches. The attention mechanism's capacity to capture complex dynamical patterns in multivariate data contributed to improved forecasting accuracy.

Lu & Chen (2024) proposed a multivariate data slicing transformer neural network for load forecasting in power systems with high-penetration renewables. The transformer model excelled in capturing spatiotemporal relationships by modeling global correlations through self-attention mechanisms. Their approach demonstrated superior performance in handling the intermittency and volatility characteristics brought by renewable energy integration, outperforming traditional statistical models and conventional machine learning methods.

2.1.5 Comparative Studies and Ensemble Methods

Banik & Biswas (2024) developed an enhanced stacked ensemble model combining Random Forest and XGBoost for renewable power and load forecasting. The Random Forest model first forecasted the target variable, followed by XGBoost improving predictions through combination of RF outputs. A meta-model using logistic regression then learned the optimal combination, achieving 99% accuracy on R^2 evaluation metrics for both short-term and long-term predictions in Agartala City dataset.

Kwon et al. (2020) conducted extensive research on learning models combined with data clustering and dimensionality reduction for short-term electricity load forecasting. They adapted k-means clustering for data grouping and utilized kernel PCA, UMAP, and t-SNE for dimensionality reduction. Applied to neural network-based models on large-scale electricity usage data from 4,710 households, their approach demonstrated improved forecasting performance through effective data preprocessing and feature engineering.

Nabavi et al. (2024) combined Discrete Wavelet Transform with LSTM to improve electricity load forecasting accuracy. The DWT decomposed load series into multiple frequency components, allowing LSTM to learn from denoised and structured representations. Their research demonstrated that preprocessing techniques significantly enhanced deep learning model performance, particularly for datasets with high noise levels and irregular patterns.

3. Methodology

This chapter describes the overall research methodology adopted to achieve the objectives of the thesis. It outlines the research design, data acquisition process, preprocessing techniques, and modeling approaches used for short-term electrical load forecasting. The methodology is structured to ensure a systematic analysis, with each step directly linked to the research objectives. A flowchart is used to summarize the overall framework, followed by a detailed explanation of the selected machine learning and deep learning models and the evaluation techniques employed in this study.

3.1 Overall Workflow

The forecasting framework used in this thesis follows a clear and systematic workflow. Each stage builds on the previous one to ensure that the final model is both reliable and reproducible. The complete process can be broken into six major steps:

1. **Data Acquisition:** Hourly historical load data from the Baneshwor Feeder is collected alongside weather variables (temperature, humidity, rainfall) and calendar information (weekday/weekend, holidays). These serve as the core inputs to the forecasting models.

2. **Data Preprocessing:** The raw dataset often contains missing readings, outliers, and format inconsistencies. This stage involves cleaning the data, handling missing values, treating outliers, converting timestamps, and engineering new time-based and cyclical features. This step ensures that the dataset is suitable for model development.

3. **Exploratory Data Analysis (EDA):** The cleaned dataset is examined to understand load trends, seasonal patterns, hourly variations, and correlations with weather variables. EDA helps identify which features influence load demand the most and guides feature selection.

4. **Model Development:** Both machine learning models (SVR, Random Forest, XGBoost) and deep learning models (LSTM, GRU) are designed. Input features are standardized and structured depending on the model type—tabular matrices for ML models and sequential inputs for DL models.

5. **Model Training and Validation:** Models are trained using training data and validated using walk-forward validation or hold-out testing. Hyperparameters are op-

timized (GridSearchCV for ML models and iterative tuning for DL models). This ensures that the models generalize well and avoid overfitting.

6. **Performance Evaluation and Comparison:** All models are evaluated using RMSE, MAE, MAPE, MSE, and R^2. Their forecasting accuracy, stability, and computational efficiency are compared. The best-performing model is then recommended for operational forecasting in the Baneshwor Feeder.

This workflow provides a complete pipeline from raw data to final recommendation and supports both machine learning and deep learning approaches. The overall diagram of the methodology is shown in Figure 3.1.

3.2 Data Acquisition

The first step of our methodology is data acquisition. Hourly historical load data from Baneshwor Feeder will be collected. Weather data such as temperature, humidity, and rainfall data will be collected from the Department of Hydrology and Meteorology, Nepal. And finally, the information on weekdays, weekends are obtained from official government calendars. The study relies on two primary datasets:

1. Hourly electrical load data for the Baneshwor Feeder

2. Hourly weather data (temperature, humidity, and solar radiation)

Since both datasets were obtained as raw Excel files with irregular formats, inconsistent timestamps, missing entries, and multiple sheets per day/month, a multi-stage acquisition and structuring pipeline was developed. The sources of both datasets are listed below.

3.2.1 Data Sources

a) **Electrical Load Data:** The electrical load data used in this study was obtained from the Baneshwor Substation, which operates under the Nepal Electricity Authority (NEA). Hourly feeder load readings were collected from archived operational log sheets maintained by the substation for the years 2079 to 2082 BS. These records provided raw POWER (MW) measurements for the Baneshwor Feeder, along with associated timestamp information. Since the data originated from manually recorded and distributed Excel files, several preprocessing steps—such as header correction, timestamp standardization, and quality checks—were required before the dataset could be used for modeling. This substation-provided dataset forms the core of the forecasting analysis, representing real operational feeder behavior across multiple years.

b) **Weather Data:** Weather data was sourced from Nepal's Department of Hydrology and Meteorology (DHM), the official governmental agency responsible for climate and atmospheric measurements. The dataset included hourly records of air temperature, relative humidity, and global solar radiation for the corresponding study period. These variables were essential for capturing the environmental conditions influencing electricity consumption patterns. The DHM dataset required timestamp alignment, interpolation for missing values, and smoothing of extreme readings to ensure compatibility with the load dataset. Once cleaned and synchronized, the weather data served as an important set of exogenous features for both machine learning and deep learning models.

Because the raw files came in varying formats—different months, unpredictable sheet names, Bikram Sambat (BS) dates, mixed day formats, half-hour readings, inconsistent header rows, and multiple sheets per month—a custom data acquisition pipeline was required.

3.2.2 Load Data Acquisition and Structuring Process

The original NEA-provided Excel files were not uniform. Each month contained multiple workbooks, each workbook contained several sheets, and each sheet sometimes mixed headers, irrelevant rows, or inconsistent timestamp formats. To convert all raw data into a single unified structure, the following pipeline was implemented:

a) **Month-wise Sheet Extraction:** The script automatically scanned each monthly folder, such as those inside dataset/2082MW, and identified all sheets whose names contained "11KV" or variations of it. From there, it matched the sheet names to the correct year–month to ensure proper alignment during extraction. Only rows with timestamps ending in HH:00 were selected, and half-hour readings like 7:30 were intentionally ignored. For every sheet, the script compiled a list of valid hourly entries and then stored the extracted results month-wise inside the corresponding extracted-Dataset/YYYYMM folders. This step produced a clean per-month dataset, but timestamps were still in Nepali calendar (BS) and formats varied.

b) **Converting BS Dates, Normalizing Hours, and Structuring Daily Sheets:** The process began by extracting the BS date encoded inside each sheet name, such as 2079.02.15, and converting it to the AD calendar using Nepali date conversion libraries. Each sheet was read without assuming a fixed header position, allowing the script to dynamically detect both the Time column and the appropriate POWER (MW) value column. Every day was normalized to 24 hourly records by indexing hours from 1 to 24, and any missing hours were filled using linear interpolation. Clean timestamps were then constructed in the formats YYYY-MM-DD HH:00 and YYYY-MM-DD 24:00 for the final hour. Finally, one clean sheet was generated per day in the AD calendar, ensuring that each day contained exactly twenty-four readings aligned to a consistent timestamp system.

c) **Combining All Structured Monthly Files:** The script scanned the four yearly folders from 2079 to 2082, and for each folder it opened every Excel file, extracted the valid Time and POWER (MW) columns, removed any remaining header fragments, and concatenated all sheets in that folder into a single file. After processing all four years, the resulting files were merged into one master dataset. At this stage, the study had a fully consolidated file containing all hourly POWER (MW) readings for the entire study period.

3.2.3 Weather Data Acquisition and Structuring

Weather data was also provided in raw format with mixed timestamps. Two scripts were developed to clean and align it with the load data.

a) **Extracting and Cleaning Raw Weather File:** The script located the correct columns for time, temperature, humidity, and solar radiation, then removed any unusable rows. All timestamps were parsed into a consistent datetime format, after which the weather data was filtered to match the exact date range of the load dataset. The timestamps were then formatted as YYYY-MM-DD HH:MM. This stage produced a clean hourly weather dataset.

b) **Structuring Weather Timestamp Alignment:** Timestamps were shifted so that values such as "HH:45" were aligned to the next hour at "HH+1:00," and all "24:00" rollover cases were handled correctly. Missing or zero weather values were replaced using nearest-neighbor averages, while NaN solar radiation entries were set to zero. These steps produced the final clean weather file and ensured that all weather variables followed the exact hourly structure required for forecasting.

3.2.4 Final Merging of Load and Weather Data

The process involved parsing all load timestamps, including proper handling of "24:00," and parsing the weather timestamps in a consistent format. Weather rows were then aligned precisely to the corresponding load timestamps, and any gaps in the weather variables were filled using linear interpolation. Finally, both datasets were merged into a single unified file. This produced the final dataset containing:

1. Time
2. Power (MW)
3. Air Temperature
4. Global Solar Radiation
5. Relative Humidity

This dataset is the core input for all ML/DL models used in the thesis.

3.3 Data Preprocessing

Once the load and weather datasets were fully acquired and merged into a single hourly dataset, several preprocessing steps were applied to prepare the data for machine learning and deep learning models. These steps ensure data consistency, remove inconsistencies introduced during manual logging, and improve the overall reliability of the models.

3.3.1 Timestamp Parsing and Standardization

The merged dataset initially contained timestamps in multiple formats, such as "YYYY-MM-DD HH:MM," "YYYY-MM-DD HH:MM:SS," and special cases like "24:00." To prevent errors during model training, the Time column was fully standardized. All timestamps were parsed using a custom parser capable of handling "24:00" by shifting it to 00:00 of the following day, and the final format was normalized to "YYYY-MM-DD HH:00." This ensured that every row represented a clean and unambiguous hourly reading.

3.3.2 Handling Missing Values

During the data acquisition process, several power (MW) entries were found to be missing due to incomplete feeder log sheets, damaged or empty records, and files containing no valid measurements. Since the dataset represents time-series load data, preserving temporal continuity was a key consideration during preprocessing.

To address this issue, a forward-fill followed by backward-fill imputation strategy was employed. In the forward-fill step, missing values were replaced using the most recent valid observation, ensuring continuity with past load behavior. Subsequently, any remaining missing values, typically occurring at the beginning of the dataset, were filled using the next available valid observation through backward filling. This combined approach effectively maintains the temporal structure of the load data while avoiding the introduction of artificial fluctuations or unrealistic load variations.

3.3.3 Outlier Detection and Treatment

Some load values contained extreme spikes caused by Excel merging errors, mistyped POWER (MW) entries, or abnormally large or zero readings recorded during festivals or outages. Outlier detection was performed using the Interquartile Range (IQR) method, where Q1, Q3, and the IQR were computed, and any values falling below $Q1 - 1.5 \times IQR$ or above $Q3 + 1.5 \times IQR$ were flagged as outliers. Although approximately 0.24 percent of the data was identified as anomalous, these points were not removed, as discarding them would break the continuity of the time-series structure.

3.3.4 Weather Data Cleaning and Alignment

After merging the weather data with the load dataset, any NaN weather values were filled using linear interpolation. Extreme or zero readings for humidity and temperature were smoothed using nearest-neighbor values, while missing solar radiation entries were replaced with zero under the assumption of nighttime conditions. All weather variables were then aligned precisely to the hourly timestamps. This produced a fully synchronized dataset in which every hour contained both load and weather information.

3.3.5 Temporal Feature Engineering

To help ML/DL models learn daily/weekly seasonal patterns, several time-based features were added. The timestamp was used to extract several temporal features, including Hour (0–23), Day (1–31), Month (1–12), DayOfWeek (0–6), WeekOfYear, and an IsWeekend flag. These features were designed to capture the natural daily and weekly patterns present in electricity consumption. Since time progresses cyclically—where hour 23 transitions directly back to hour 0—cyclical encodings were also applied. Sine and cosine transformations were created for the hour values, such as $Hour\_sin = \sin(2\pi \times Hour/24)$ and $Hour\_cos = \cos(2\pi \times Hour/24)$, and similar cyclic encodings were generated for Month and DayOfWeek. These transformations help machine learning and deep learning models learn smooth periodic relationships instead of treating time variables as abrupt linear steps.

3.3.6 Correlation Analysis and Feature Selection

Before modeling, a correlation heatmap was created to examine the relationships among variables. Hour showed the strongest correlation with load, while global solar radiation exhibited a moderate positive correlation. Temperature demonstrated a weak but meaningful correlation, and humidity showed a weak negative correlation. All features were retained based on their correlation patterns and domain relevance. The final feature set consisted of fourteen engineered features that combined both temporal components and weather variables.

A correlation matrix was computed using all numerical features in the dataset. The analysis showed that Hour had the strongest correlation with POWER (MW) at +0.457, while global solar radiation contributed a moderate positive correlation of +0.307. Temperature and humidity exhibited weaker correlations, ranging from about +0.18 to –0.18, and calendar-related features such as Month, IsWeekend, and DayOfWeek displayed weak but meaningful trends. Overall, the correlation analysis supported the decision to retain both weather variables and time-based features in the modeling process.

3.3.7 Train–Test Split

Since machine learning models depend on random data splits, while deep learning models typically require sequential splits, the dataset was divided into 80 percent for training and 20 percent for testing. Shuffling was enabled for the ML models to prevent seasonal clustering from introducing bias during training.

3.3.8 Final Processed Dataset Structure

After all preprocessing steps were completed, each row of the dataset contained the timestamp, POWER (MW) load value, air temperature, global solar radiation, relative humidity, and fourteen engineered temporal features. The dataset had no missing values, no irregular timestamps, and no half-hour entries, resulting in a fully cleaned, capped, and time-aligned series. This final consolidated dataset was then used for developing the machine learning and deep learning models.

3.4 Exploratory Data Analysis (EDA)

Exploratory Data Analysis was carried out to understand the statistical behaviour of the load and weather variables, identify underlying temporal patterns, and observe the relationships between different features. This step provides important insights that guide model selection, feature engineering, and the choice of evaluation methods.

3.4.1 Overview of the Dataset

The final dataset consisted of hourly timestamps spanning the entire study period, feeder load values in megawatts, and the key weather variables of air temperature, global solar radiation, and relative humidity. It also included a comprehensive set of engineered temporal features such as hour, day, month, weekday, and the corresponding cyclical encodings. A quick inspection confirmed that there were no missing values after imputation, the dataset maintained a uniform hourly structure with no half-hour gaps, the POWER (MW) readings were clean after capping and smoothing, and all weather and load timestamps were fully synchronized.

3.4.2 Time-Series Visualization

To understand how the load varies over time, the hourly POWER (MW) values were resampled into daily averages and visualized across the entire study period. The resulting trend showed clear daily, weekly, and seasonal fluctuations in the feeder's behavior. Winter months displayed slightly lower solar radiation levels along with moderately higher load during certain intervals. Occasional dips in the curve aligned with known outages or special events. Solar radiation exhibited a strong daytime pattern, reinforcing its moderate correlation with load. Overall, this broad visualization confirmed that the Baneshwor Feeder operates as a typical mixed-load distribution feeder with pronounced daily cycles and noticeable seasonal influences.

3.4.3 Hourly Load Pattern Analysis

The hourly load values were averaged across the full dataset to understand the feeder's daily consumption pattern. The analysis showed that the minimum load typically occurs around 3:00 AM, which reflects low residential and commercial activity during that time. Load levels begin to rise through the morning and reach a peak at around 19:00, with the average peak load reaching approximately 3.16 POWER (MW). This aligns with evening lighting needs and heightened residential usage. A boxplot comparing load against hour of day further illustrated that evening hours exhibit higher variance, while midnight to early-morning hours display more stable and lower demand. These observations confirm that the hour of the day is one of the strongest predictors of load in this feeder.

3.4.4 Monthly Load Trends

Monthly averages revealed that warmer months experience higher temperatures, although the corresponding load behavior varies across the year. Seasonal patterns are present but not as dominant as the daily cycles observed in the feeder. Consumption typically increases during festival seasons when household activity rises. These seasonal shifts are effectively captured through the Month feature and its corresponding Month_sin and Month_cos cyclical encodings.

3.4.5 Weather Variable Behaviour

All weather variables were examined individually to understand their behavior and potential influence on load. Air temperature ranged from roughly 1°C to 33°C and followed a clear daily cycle, with warmer afternoons and cooler nights. Global solar radiation showed a distinct daytime-only pattern, peaking sharply around midday and dropping to zero during nighttime hours. Relative humidity tended to be higher during nighttime and rainy months and displayed a slight inverse relationship with temperature.

3.4.6 Load vs. Weather Relationships

Scatter plots were used to examine how load responds to different weather variables. The temperature–load relationship showed a weak positive correlation, with load increasing moderately as temperatures rise, which is typical for mixed-load areas where fans and cooling appliances see greater use. Solar radiation displayed a moderate positive correlation with load, as higher midday radiation often coincides with active residential and commercial activity. Humidity exhibited a weak negative correlation, since high humidity is generally associated with cloudy or rainy conditions during which daytime load may decrease slightly.

3.4.7 Correlation Analysis

A correlation matrix was computed using all numerical features in the dataset. The analysis showed that Hour had the strongest correlation with POWER (MW) at +0.457, while global solar radiation contributed a moderate positive correlation of +0.307. Temperature and humidity exhibited weaker correlations, ranging from about +0.18 to –0.18, and calendar-related features such as Month, IsWeekend, and DayOfWeek displayed weak but meaningful trends. Overall, the correlation analysis supported the decision to retain both weather variables and time-based features in the modeling process.

3.5 Model Development

Machine learning models were first developed to establish strong baseline performance and evaluate how well traditional ML algorithms can capture the nonlinear, multi-feature dependencies present in feeder load data. All models were trained on the cleaned and feature-engineered dataset described in the earlier sections. Each model received the same input feature set to ensure fair comparison.

The machine learning pipeline involved standard scaling of the numerical features, an 80–20 train–test split, and hyperparameter tuning performed using GridSearchCV. Model performance was evaluated using RMSE, MAE, MAPE, MSE, and R^2. The following subsections describe each machine learning model used in the study.

3.5.1 Machine Learning Models

Linear Regression

Linear Regression is the simplest baseline model that assumes the relationship between input features and the target load value is linear. Although load patterns are nonlinear, this model helps establish a reference point for evaluating more complex methods.

Mathematical Formulation:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \tag{3.1}$$

where:

- β_0 is the intercept (bias term)
- β_i are the coefficients (weights) learned via ordinary least squares minimization
- x_i are the input features

Ridge Regression

Ridge Regression adds L2 regularization to the linear model to reduce overfitting and stabilize coefficient estimates. It is more robust than standard Linear Regression when dealing with many correlated features, which is the case in this study.

Mathematical Formulation:

$$\min_{\beta} \left(\|y - X\beta\|^2 + \alpha \|\beta\|^2 \right) \tag{3.2}$$

where:

- α controls the strength of L2 regularization
- $\|y - X\beta\|^2$ is the residual sum of squares
- $\|\beta\|^2$ is the L2 norm of the coefficient vector

Key Hyperparameters (Tuned):

- `alpha`: Regularization strength
- `fit_intercept`: Whether to calculate the intercept
- `solver`: Algorithm used for optimization

Support Vector Regression (SVR)

SVR models nonlinear relationships by mapping features into a high-dimensional space using kernel functions. It works well for complex regression problems with moderate dataset sizes.

Mathematical Formulation:

SVR finds a function $f(x)$ by solving the following optimization problem:

$$\min_{w,b,\xi,\xi^*} \left(\frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*) \right) \tag{3.3}$$

subject to the constraints:

$$y_i - (w \cdot x_i + b) \leq \varepsilon + \xi_i$$
$$(w \cdot x_i + b) - y_i \leq \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \geq 0$$

where:

- w is the weight vector
- b is the bias term
- C is the regularization parameter controlling the trade-off between flatness and tolerance of deviations
- ε defines the epsilon-insensitive tube
- ξ_i and ξ_i^* are slack variables for points outside the tube

Kernel Used:

- RBF (Radial Basis Function): Best suited for capturing nonlinear patterns in load data

Hyperparameters (Tuned):

- `C`: Regularization parameter
- `epsilon`: Width of the epsilon-insensitive tube
- `gamma`: Kernel coefficient for RBF
- `kernel`: Set to 'rbf'

Random Forest Regressor

Random Forest is an ensemble method consisting of multiple decision trees. Each tree is trained on a random subset of features and samples (bootstrap aggregating). It is robust, stable, and handles nonlinearity effectively.

Mathematical Formulation:

The Random Forest prediction is the average of all individual tree predictions:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} f_t(x) \tag{3.4}$$

where:

- T is the total number of trees in the forest
- $f_t(x)$ is the prediction of the t-th decision tree

Hyperparameters (Tuned):

- `n_estimators`: Number of trees in the forest
- `max_depth`: Maximum depth of each tree
- `min_samples_split`: Minimum samples required to split an internal node
- `min_samples_leaf`: Minimum samples required at a leaf node
- `max_features`: Number of features considered for the best split

Gradient Boosting Regressor

Gradient Boosting builds trees sequentially, with each new tree correcting the errors (residuals) of the previous ensemble. It is particularly effective for structured tabular data like load forecasting.

Mathematical Formulation:

At each boosting iteration m, the model is updated as:

$$F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x) \tag{3.5}$$

where:

- $F_{m-1}(x)$ is the ensemble prediction from the previous iteration
- $h_m(x)$ is the new tree fitted to the negative gradient (pseudo-residuals)
- ν is the learning rate (shrinkage factor) that controls the contribution of each tree

Hyperparameters (Tuned):

- `n_estimators`: Number of boosting stages
- `learning_rate`: Shrinkage factor
- `max_depth`: Maximum depth of individual trees

XGBoost Regressor

XGBoost (Extreme Gradient Boosting) is an optimized and regularized implementation of gradient boosting designed for efficiency, scalability, and high accuracy. It was one of the best-performing ML models in this study.

Mathematical Formulation:

XGBoost minimizes the following regularized objective function:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{3.6}$$

where:

- $l(y_i, \hat{y}_i)$ is the loss function measuring the difference between actual and predicted values
- $\Omega(f_k)$ is the regularization term for the k-th tree, defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{3.7}$$

where:

- T is the number of leaves in the tree
- w_j is the weight (score) of the j-th leaf
- γ controls the minimum loss reduction required to make a split
- λ is the L2 regularization term on leaf weights

Hyperparameters (Tuned):

- `n_estimators`: Number of boosting rounds
- `learning_rate`: Step size shrinkage
- `max_depth`: Maximum tree depth
- `subsample`: Fraction of samples used per tree
- `colsample_bytree`: Fraction of features used per tree
- `gamma`: Minimum loss reduction for a split

3.5.2 Deep Learning Models

To model the nonlinear, temporal, and sequential characteristics of feeder load data, three deep learning architectures were implemented. These models were trained on sliding windows of historical hourly sequences, allowing the networks to learn both short-term and long-term dependencies in the dataset.

The models developed are:

1. Multi-Layer Perceptron (MLP)
2. Long Short-Term Memory (LSTM)
3. Gated Recurrent Unit (GRU)

All models were trained with the Adam optimizer, early stopping, and sequence-based inputs where applicable.

Multi-Layer Perceptron (MLP)

The MLP is a feed-forward network consisting of multiple fully connected layers. Although it does not inherently model temporal order, it can extract nonlinear patterns from feature-engineered inputs. In this study, the MLP receives sequences flattened into a fixed-size input vector.

Mathematical Formulation:

Hidden layer activation:

$$h = \sigma(W_1 x + b_1) \tag{3.8}$$

Output layer:

$$\hat{y} = W_2 h + b_2 \tag{3.9}$$

where:

- σ is the activation function (ReLU in this implementation)
- W_1, W_2 are weight matrices for the hidden and output layers respectively
- b_1, b_2 are bias terms
- x is the input feature vector
- h is the hidden layer output

Long Short-Term Memory (LSTM)

LSTM networks are designed to maintain contextual memory over long sequences through a gating mechanism that controls information flow. This makes them naturally suited for load forecasting, where consumption patterns depend on previous hours. The LSTM architecture addresses the vanishing gradient problem that affects standard RNNs.

Mathematical Formulation:

At each time step t, the LSTM computes the following:

Forget gate (determines what information to discard from cell state):

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.10}$$

Input gate (determines what new information to store):

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.11}$$

Candidate cell state (creates new candidate values):

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{3.12}$$

Cell state update (combines old and new information):

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{3.13}$$

Output gate (determines what to output):

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.14}$$

Hidden state (final output at time step t):

$$h_t = o_t \odot \tanh(c_t) \tag{3.15}$$

where:

- σ is the sigmoid activation function
- tanh is the hyperbolic tangent activation function
- $\odot$ denotes element-wise (Hadamard) product
- $[h_{t-1}, x_t]$ represents concatenation of the previous hidden state and current input
- W_f, W_i, W_c, W_o are weight matrices for each gate
- b_f, b_i, b_c, b_o are bias vectors for each gate
- c_t is the cell state that carries long-term memory
- h_t is the hidden state output

Gated Recurrent Unit (GRU)

GRU is a streamlined version of LSTM with fewer parameters, combining the forget and input gates into a single update gate and merging the cell state with the hidden state. It often trains faster while achieving comparable performance to LSTM.

Mathematical Formulation:

At each time step t, the GRU computes the following:

Update gate (controls how much past information to keep):

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{3.16}$$

Reset gate (determines how much past information to forget):

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{3.17}$$

Candidate hidden state (computes new candidate activation):

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \tag{3.18}$$

Final hidden state (interpolates between previous and candidate state):

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{3.19}$$

where:

- σ is the sigmoid activation function
- tanh is the hyperbolic tangent activation function
- $\odot$ denotes element-wise (Hadamard) product
- $[h_{t-1}, x_t]$ represents concatenation of the previous hidden state and current input
- W_z, W_r, W_h are weight matrices for the update gate, reset gate, and candidate state
- b_z, b_r, b_h are bias vectors
- z_t controls the balance between old and new information
- r_t controls how much of the previous state influences the candidate

3.6 Model Training and Validation

This stage focuses on how both machine learning (ML) and deep learning (DL) models were trained, tuned, validated, and prepared for final performance comparison. Since ML and DL models require different handling, the training process is presented in two separate parts.

3.6.1 Training of Machine Learning Models

The machine learning models trained in this study include:

1. Support Vector Regression (SVR)
2. Random Forest Regressor (RF)
3. Gradient Boosting Regressor (GBR)
4. Extreme Gradient Boosting (XGBoost)
5. Linear Regression and Ridge Regression (baseline models)

All models used the same final preprocessed dataset described earlier, containing weather features, temporal encodings, and cleaned load values.

Input Preparation

For ML models, the dataset was used in a tabular format:

1. **Features (X):** Time features (hour, month, weekday), cyclical encodings, weather variables, and lag features if applied.
2. **Target (y):** Load at the next hour.

Since ML models do not operate on sequences, no sliding window was required.

Data Splitting

The dataset was split into 80 percent for training and 20 percent for testing. Shuffling was applied during the split to prevent temporal clustering and to ensure that the machine learning models were exposed to a diverse mix of seasonal and temporal patterns during training.

Feature Scaling

Some models required normalized inputs, so StandardScaler was applied to Linear Regression, Ridge, and SVR. Tree-based models such as Random Forest, Gradient Boosting, and XGBoost did not require any scaling.

Training Procedure

Each model was trained using its corresponding optimization approach:

- Least squares optimization for Linear Regression and Ridge
- Kernel-based optimization for SVR
- Ensemble tree learning for Random Forest and Gradient Boosting
- Gradient-boosted tree optimization for XGBoost

The training process involved fitting the models on the training set, generating predictions on the test set, and evaluating performance using RMSE, MAE, MAPE, MSE, and R^2.

Hyperparameter Tuning

All machine learning models were fine-tuned using GridSearchCV, which tested different combinations of hyperparameters:

- **SVR:** C, epsilon, and gamma
- **Random Forest and Gradient Boosting:** n_estimators, max_depth, and min_samples_split
- **XGBoost:** learning_rate, max_depth, subsample, and colsample_bytree
- **Ridge:** alpha

The best configurations were selected based on the lowest validation error.

3.6.2 Training of Deep Learning Models

Three deep learning models were implemented:

1. Multi-Layer Perceptron (MLP)
2. Long Short-Term Memory (LSTM)
3. Gated Recurrent Unit (GRU)

Since deep learning models learn from sequences rather than static features, the training process follows a different pipeline.

Sequence Construction

A sliding window method was used in which the model received the past N hours as input and predicted the load for the next hour. A typical window size of 24 hours was used, although this value can be adjusted in the implementation.

Train–Validation–Test Split

Deep learning models require sequential integrity, so the dataset was split chronologically:

- 80 percent for training
- 10 percent for validation
- 10 percent for testing

No shuffling was applied, ensuring that the model learned from the natural temporal progression of the data.

Model Training Configuration

All deep learning models were trained with the following configuration:

- **Optimizer:** Adam
- **Loss Function:** Mean Squared Error (MSE)
- **Batch Size:** Typically 32 or 64
- **Epochs:** Training continued for multiple epochs until early stopping criteria were met
- **Weight Initialization:** Xavier/Glorot initialization (TensorFlow defaults)

Regularization and Stability

To avoid overfitting, several regularization techniques were applied:

- Dropout layers were included in the MLP and LSTM-based models
- Batch normalization was applied where appropriate
- Early stopping was used to monitor validation loss, and training automatically stopped when the validation loss stopped improving for several consecutive epochs

Model-Specific Training Notes

- **MLP:** Used flattened inputs from the sliding window and relied on dense layers to learn nonlinear mappings. Generally converged quickly during training.
- **LSTM:** Captured long-term temporal dependencies but required more computation. Performed best when extended historical context was important.
- **GRU:** Provided a faster and lighter alternative to LSTM, often achieving comparable accuracy while offering a strong balance for medium-complexity time-series tasks.

3.6.3 Validation Approach

A consistent evaluation strategy was applied across all models:

Machine Learning Models:

- Validated using GridSearchCV with five-fold cross-validation
- Best hyperparameters were chosen based on the minimum validation loss

Deep Learning Models:

- Validated using a 10 percent validation split
- Early stopping was used to prevent overfitting
- Best-performing model weights were preserved through checkpointing

3.7 Performance Evaluation

To compare the machine learning and deep learning models fairly, a consistent set of standard error metrics was used. These metrics measure the difference between the predicted load values and the actual observed load. In this study, five commonly used regression evaluation metrics were selected:

1. Mean Absolute Error (MAE)
2. Mean Squared Error (MSE)
3. Root Mean Squared Error (RMSE)
4. Mean Absolute Percentage Error (MAPE)
5. Coefficient of Determination (R^2 Score)

These metrics help assess accuracy, robustness, and the overall predictive performance of the forecasting models.

3.7.1 Mean Absolute Error (MAE)

MAE represents the average magnitude of errors between forecasted and actual values, without considering direction. It is simple and easy to interpret.

Formula:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{3.20}$$

where:

- n is the total number of samples
- y_i is the actual load value
- $\hat{y}_i$ is the predicted load value

Interpretation: Lower MAE indicates that the model's predictions are consistently closer to the actual load. This metric is robust against the influence of individual large errors.

3.7.2 Mean Squared Error (MSE)

MSE squares the prediction errors before averaging, which penalizes larger deviations more strongly.

Formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.21}$$

where:

- n is the total number of samples
- y_i is the actual load value
- $\hat{y}_i$ is the predicted load value

Interpretation: This metric is very sensitive to large errors, making it particularly useful in situations where significant forecasting deviations are unacceptable.

3.7.3 Root Mean Squared Error (RMSE)

RMSE is the square root of MSE. It is one of the most widely used metrics in load forecasting, and its units are the same as the original POWER (MW) values.

Formula:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.22}$$

where:

- n is the total number of samples
- y_i is the actual load value
- $\hat{y}_i$ is the predicted load value

Interpretation: Lower RMSE reflects a more accurate model overall. Because it strongly penalizes large errors, it serves as a stricter evaluation metric than MAE.

3.7.4 Mean Absolute Percentage Error (MAPE)

MAPE expresses the prediction error as a percentage of the actual value. This makes it easy to compare forecasting performance across different feeders or scales.

Formula:

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{3.23}$$

where:

- n is the total number of samples
- y_i is the actual load value
- $\hat{y}_i$ is the predicted load value

Interpretation: This metric shows the average percentage deviation between predictions and actual values, with lower values indicating better performance. It becomes undefined when the actual load equals zero, though this was not an issue here since the feeder load never drops to zero.

3.7.5 Coefficient of Determination (R² Score)

R^2 indicates how much of the variance in the actual load values is explained by the model.

Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.24}$$

where:

- y_i is the actual load value
- $\hat{y}_i$ is the predicted load value
- $\bar{y}$ is the mean of actual values

Interpretation: An R^2 value of 1 represents perfect prediction, while a value of 0 indicates that the model performs no better than simply using the average load. Higher R^2 values therefore indicate better model performance. Negative R^2 values are possible when the model performs worse than the mean baseline.

4. Results & Discussion

This chapter presents the performance of all machine learning and deep learning models trained for short-term load forecasting of the Baneshwor Feeder. All models were evaluated using the same performance metrics (MAE, RMSE, MSE, MAPE, R^2), ensuring a fair comparison.

4.1 Results

4.1.1 Machine Learning Model Performance

All machine learning models were trained on the final feature-engineered dataset and evaluated on the test set. Below is the summary table for ML models.

Table 4.1: Machine Learning Models Evaluation Matrix

Rank	**Model**	**MAE**	**RMSE**	**MAPE**	**R^2**
1	XGBoost (Tuned)	0.257	0.384	12.693	0.831
2	Random Forest (Tuned)	0.294	0.435	14.290	0.783
3	Random Forest	0.305	0.444	14.825	0.774
4	XGBoost	0.313	0.449	15.242	0.769
5	Gradient Boosting	0.330	0.469	16.062	0.749
6	SVR	0.318	0.483	15.193	0.732
7	Ridge Regression	0.502	0.649	25.021	0.518
8	Linear Regression	0.502	0.649	25.021	0.518

Discussion of Results

The machine learning models were evaluated using MAE, RMSE, MAPE, R-squared (R^2), and prediction accuracy to assess their forecasting effectiveness. Linear Regression and Ridge Regression served as baseline models and produced relatively higher error values and lower R^2 scores, indicating their limited ability to capture nonlinear relationships between load demand and influencing variables. Support Vector Regression improved upon linear models by reducing error metrics; however, its performance remained inferior to ensemble-based approaches, particularly in terms of RMSE and prediction accuracy.

Tree-based ensemble models demonstrated superior performance across all evaluation metrics. Random Forest and Gradient Boosting significantly reduced MAE and RMSE while achieving higher R^2 values, reflecting improved generalization and better handling of nonlinear patterns. Among these, the tuned XGBoost model outperformed all other machine learning models, achieving the lowest RMSE (0.384 MW), the highest R^2 (0.831), and the best prediction accuracy (55.77%). The improvement over other ensemble models can be attributed to XGBoost's gradient-based optimization, regularization mechanisms, and effective handling of feature interactions, which collectively enhanced model robustness and accuracy.

Overall, the comparative evaluation confirms that ensemble-based machine learning models, particularly XGBoost, provide the most reliable and accurate predictions for feeder-level short-term load forecasting.

4.1.2 Deep Learning Model Performance

Deep learning models were trained using sliding-window sequences to capture temporal dependencies in the feeder load data. Three architectures were implemented: LSTM, GRU, and MLP. After conducting extensive training using the cleaned dataset, the models were evaluated using the same metrics as the ML models.

The results obtained for the deep learning models are presented below.

Table 4.2: Deep Learning Models Evaluation Matrix

Rank	**Model**	**MAE**	**RMSE**	**MAPE**	**R^2**	**Accuracy**
1	MLP	0.618	0.499	20.062	0.011	77.890
2	LSTM	0.467	0.498	14.155	0.014	83.308
3	GRU	0.482	0.499	15.186	0.012	82.754

Discussion of Results

Deep learning models, including MLP, LSTM, and GRU, were evaluated using the same performance metrics for a fair comparison. Among these models, the MLP produced marginally better results than sequence-based architectures; however, its error values remained high and R^2 scores were close to zero, indicating weak explanatory power. Sequence-based models such as LSTM and GRU did not show meaningful improvement over MLP, despite their capability to model temporal dependencies.

Both LSTM and GRU exhibited high MAE and RMSE values, with RMSE remaining close to 0.499 MW, and very low prediction accuracy. These results suggest that the deep learning models were unable to effectively learn stable temporal patterns from the available dataset. Compared to machine learning models, deep learning approaches showed limited generalization and higher sensitivity to data volume and structure. The lack of performance gains over simpler architectures indicates that increased model complexity did not translate into improved forecasting accuracy in this case.

In comparison to machine learning models, deep learning models were clearly outperformed across all evaluation metrics. This highlights that, for the given data scale, feature representation, and forecasting horizon, traditional machine learning models with explicit feature engineering are more suitable than deep learning architectures for feeder-level short-term load forecasting.

5. Conclusions & Recommendations

The conclusion is an integration of various issues covered in the body of the thesis. The conclusion includes noting any implications resulting from the discussion and making policy recommendations and the need for further research. Hence, the conclusion should be a logical ending to what has been previously discussed. It must pull together all parts of the argument and refer the reader back to the focus you have outlined in your introduction and to the central topic. Never present any new information in this section. Thus, the conclusion and recommendation of the study must be limited within the scope of the research.

6. Limitations and Future enhancement

This chapter should contain the major limitations of the project and the further enhancement of the project/research shortly with a different but related approach. **Referencing checking here**[1]

References

[1] Santosh Giri and Basanta Joshi. Transfer learning based image visualization using cnn. *International Journal of Artificial Intelligence & Applications*, 10(4):47–55, 2019.

Appendices

This page contains a data sheet, coding, procedure, photograph, questionnaire, and other essential documents. This page should be started from an odd page and APPENDIX numbering should be A, B, C, etc.