Short Term Load Forecasting

Using Machine learning and Artificial Intelligence

Presented by:
Blessed Mukundi Mangena

Prepared for:
Professor Komla Folly
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Electrical and Computer Engineering

October 9, 2025

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:.....
B. M. M̆angena

Date:.....

Terms of Reference

The terms of reference page is an agreement between yourself and your supervisor outlining what is expected of you in your final year project. Please make sure that this is discussed and written at the beginning of your thesis project.

Acknowledgments

Make relevant acknowledgments to people who have helped you complete or conduct this work, including sponsors or research funders.

Abstract

- Open the **Project Report Template.tex** file and carefully follow the comments (starting with %).
- Process the file with **pdflatex**, using other processors may need you to change some features such as graphics types.
- Note the files included in the **Project Report Template.tex** (with the .tex extension excluded). You can open these files separately and modify their contents or create new ones.
- Contact the latex namual for more features in your document such as equations, subfigures, footnotes, subscripts & superscripts, special characters etc.
- I recommend using the **kile** latex IDE, as it is simple to use.

Contents

Chapter 1

Introduction

1.1 Background to the study

A very brief background to your area of research. Start off with a general introduction to the area and then narrow it down to your focus area. Used to set the scene .

1.2 Objectives of this study

1.2.1 Problems to be investigated

Description of the main questions to be investigated in this study.

1.2.2 Purpose of the study

Give the significance of investigating these problems. It must be obvious why you are doing this study and why it is relevant.

1.3 Scope and Limitations

Scope indicates to the reader what has and has not been included in the study. Limitations tell the reader what factors influenced the study such as sample size, time etc. It is not a section for excuses as to why your project may or may not have worked.

1.4 Plan of development

Here you tell the reader how your report has been organised and what is included in each chapter.

I recommend that you write this section last. You can then tailor it to your report.

Chapter 2

Literature Review

2.0.1 Introduction

An odd power surge phenomenon was recognized during the 2008 Olympics in the UK. At certain times in the evening there would be massive power surges at what seemed like random times in the evening. Though they seemed random to the utility companies they realized something peculier was happening at these random times. Making a cup of tea is deeply ingrained in the english culture, after looking at the data they realized that the power surge was happening during times when a TV commercial would come on. Whenever a commercial came on TV multiple families would switch on their kettles to make a cup of tea. Though a single kettle may seem harmless, hundreds of thousands of kettles switched on at the same time would cause a large effect on the load of the power grid. This phenomenon was called the ***'Great British Kettle Surge'*** [1].

Whenever the kettles were turned on the load on the grid would increase. Electric load refers to the power demanded by consumers at any given time, typically measures in MegaWatts(MW). It represents the combined consumption of household, businesses and industries connected in a grid and it should meet the requirements of all end users [2]. Provision of enough electricity to meet the load demand of the grid is crucial for safe, reliable and economic operation of power systems [2].

The effect of this great kettle surge can be very harmful to the grid if necessary steps are not taken to increase the grids capacity in times when we expect the load to increase. This is where the concept of load forecasting comes into play.

Electric load forecasting is the process of predicting how much electricity will be needed at a given time and how that demand will affect the utility grid [3]. In the South African context, this issue is particularly relevant, as the country has faced persistent challenges in meeting the energy needs of its population. These difficulties can be attributed, in part, to the government's slow response to expert recommendations in load forecasting and energy planning, despite growing demand. A growing population results in a growth in the demand of electricity. If the country does not build facilities to supply enough electricity for the load requirements then load-shedding becomes the only solution to protect the grid. In a perfect country the advice from load forecasters in the early 2000's would have been taken into consideration and built more facilities to meet this demand. The above example shows the hand in hand relationship between load forecasts and their economic impacts. Large forecasting errors may lead to either risky ory conservative scheduling, which can in turn result in undesirable economic penalties[4]. Reducing prediction error on a 10GW power generation facility by 1% can save the station up to $1.6 million a year [5]. This means that there is a push towards finding the best load forecasting techniques that can be accurate and have minimal errors all year round.

Load forecasting is separated into 3 main categories which are short , medium and long term load forecasting. the major differentiator between the three is the duration in which the forecasting is predicted for.

Long Term Load Forecasting(LTLF) considers periods that are more than a year. LTLF mainly considers factors such as demographic changes, economic growth and energy policy impacts [3]. This forecasting helps utilities think of what can be done to improve their systems to meet the increasing demand of the grid in the future.

Medium Term Load Forecasting (MTLF) forecasts look at periods between a few months and a year. MTLF is important for demand side management , storage maintenance and scheduling of power [6].

Short term load forecasting(STLF) which looks at shorter time periods from hourly, daily all the way up to a week of load prediction. STLF is essential for the daily operation performance of a power grid, estimating how many power generators can be used in a particular day [7]. Finding accurate prediction models can not only save money but it will sustain the reliability and stability of a grid while increasing its operational efficiency [8]. STLF can also aid in resource management on the generation side as well as demand side management of electricity [9], this will help the demand side to plan usage and the supply side use their resources for generation efficiently.

The demand for more accurate prediction models has been continuously growing as the utilities focus on environmentally friendly and efficient power generation. An efficient and accurate prediction model eliminated the problems such as *british kettle surge* as utilities can plan ahead by ensuring more generators are operational when the demand for power rises. STLF can also ensure that the grid has a reliable continuous power flow during power shortages or outages[10].

In this study we look at and evaluate different models that have been used for STLF. Recent studies have shown a promise in using machine learning(ML) and artificial intelligence(AI) models offer more accurate forecasts than traditional statistical models [11]. Though these models come with a higher complexity and higher price for computation, however their increased accuracy offsets the negatives.

In this literature review we will look at the statistical, intelligent and hybrid models that have been used for the purpose of prediction. We will also look at the different methods that have been used for data pre-processing. This is because most models strive to get the highest accuracy value whilst, treating all data equally in the training process and not looking at the anomalies and different seasonal variations that represent themselves in data [12].

2.1 Statistical Models

Statistical methods have been the backbone of STLF before the prevalence of machine learning technologies. Different statistical models boast the ability to catch correlation in time series data. The methods have successfully been able to predict load with minimal error in the short and long term dimension. Though statistical methods are very old and have more cons that newer machine learning and hybrid models, they are still very relevant. Rusina et al, [13] highlights that despite their disadvantages and low performances for time series data, statistical models still have a, "fast implementation in practice, a wide range of use and are well studied". In this section we will review some statistical models that have been used and are still in use and look at how they hold up with current advancements.

Regression Models

Regression algorithms are mathematical tools that establish statistical correlations between variables, providing an insight on how each of the data parameters are related to the output data [14]. It also allows using a lot of predictors and it will predict outcome when there are a lot of independent predictors. A regression model being fundamentally based on the principal of finding correlation between input parameters and the output [15], it works well only until our dataset has flactuating data. Dhaval et al, [16] used Multiple Linear Regression (MLR) in their study which was successfully trained and optimized to give a 95% accuracy. Datasets for the STLF usually contains a lot of different data points and having an MLR allows us to consider each of the features to be added to the contribution.

Gochhait et al ,[14] did a study to evaluate the most effective model out of 24 statistical models for their functionality. Amongst them were regression trees , linear trees exponential and quadratic trees just to name a few. These different regression models were tested under the same hardware conditions and the data pre-processing techniques and only 6 out of the 24 were shortlisted as the best performing , however the Rational Quadratic Gaussian Process Regression and the Exponential Gaussian Process Regression were the best rated regression models to use. These two models showed low error percentages. They benchmarked these models using a Support Vector Machine(SVM) model and the regression models outperformed the all the other models that were tested. Though the regression model produces what may look like good results they have limitations. Most regression models are ,"parametric linear models and can capture only linear dependencies between the current sample and historical data",[11]. This means that regression models struggle when the data has nonlinear dependencies. The solution to the ineffectiveness of linear regression models could be using SVM. This is because SVMs help in classification problems where LR models fail to provide clear boundaries. An SVM is capable of finding the best separating boundary by identifying an optimal hyperplane that distinguishes between data points or fits a regression function [17]. This will then classify the input data buy the separation brought by the hyperplane. In the case of data that is not perfectly separable, SVMs can find soft margins to classify data points. We can understand the nonlinear aspects of the forecasts using SVMs, however they become ineffective with very large datasets [15].

Exponential Smoothing

Exponential smoothing(ES) is a simple, low cost and adaptable statistical method that is used for time series forecasting. The concept behind ES is that the weights of the observed time series are exponentially decreased as observations come further in the past [18]. The most recent time points get the highest weights while the older time get progressively smaller weights [18]. There are different version of exponential smoothing. The first one being Simple Exponential Smoothing(SES), it puts focus on estimating the smoothed value of the series at a given time. The forecast that the SES will give for the next time step is determined by a smoothing constant [19]. Ramos et al [18] explains in his paper how ES methods are differentiated by their components and they simplified the components into three. The level components which would focus on the smoothed average of the series, which would be the basis of all models. The second component is trend, which is the rate of change in the data and finally the seasonality component which finds the recurring patterns in the data over a period. A combination of each of these components will end up giving us different ES models that specify on a particular component. The SES only uses the level component and this is a problem as it assumes that the data fluctuates around the same mean with no trend and seasonality, which in reality is not true because load data is volatile and irregular [20].

Rendon et al. [21], using the component model, also applied the Holt-Winters model, which is well-suited for seasonal time series data and can be extended with an auto-regressive error correction term. Unlike Simple Exponential Smoothing, which only captures the level component, the Holt-Winters model includes parameters for both the level and seasonal components. This allows it to better handle series where seasonality plays a major role. Furthermore, they explored the Double-Seasonal Holt-Winters-Taylor model, which is specifically designed to capture two different seasonal patterns. This feature makes it particularly effective for short-term load forecasting (STLF), where electricity demand often exhibits both daily and weekly seasonal cycles.

ES models have several advantages. They are very simple to set up and have a quick learning capability , they have a straightforward structure [11].These models are also capable of capturing seasonal and trend variations. They are well suited for time series data that exhibits a strong trend of seasonal patterns which are very common in electric load data [18].

Despite the strengths ES models face limitations when looking at the complex characteristics of electric load data.These models are essentially based on linear analysis and struggle to capture the inherent non-linearity and high volatility present in electricity load time

series, which are influenced by numerous factors like weather, holidays, and user habits [11].Their performance can deteriorate with highly irregular and random time series data [22] , essentially saying there are sensitive to noise and irregularities.Finally ES models generally tend to produce less accurate results that the "black box" methods used by utility companies[23].

Auto Regressive Integrated Moving Average

The Autoregressive Integrated Moving Average (ARIMA) model is a widely recognised statistical approach employed for time series forecasting, particularly in STLF within power systems. It is considered a linear model that combines autoregressive(AR), difference(l) and moving average (MA) components [24]. It is valued as a simple and efficient model and is primarily excellent at capturing linear relations ad periodic patterns in time series data [18].

The ARIMA framework is often referred to as the Box-Jenkins model, it is systematic and practical and involves 3 iterative steps as discussed below.

Step 1 : Model Identification The initial stage involves analyzing time series data to determine the appropriate orders (p,d , q) for the AR , l and MA components.The AR component signifies that the current value of the time series (Yt) is expressed linearly in terms of its 'p' previous values and a random noise component [25].The MA component denotes that the current value of Yt is a linear combination of white noise error terms (et) from previous time steps. The integrated (I) component (represented by 'd') signifies that the time series has been differenced 'd' times to achieve stationarity [19].

Step 2 : Parameter Estimation Once the model structure (p, d, q) is identified, the parameters for the AR and MA components are estimated. This is often achieved by maximising the log-likelihood function [18].

Step 3 : Model Diagnosis The final stage involves assessing the fit of the model by checking if the residuals are uncorrelated (i.e., behave like white noise). If the residuals are not uncorrelated, the model needs to be refined, requiring a return to the identification or estimation steps [18].

ARIMA has been extended in multiple studies to enhance its capabilities . The Seasonal Auto-Regressive Integrated Moving Average (SARIMA) , is especially designed to handle seasonality in the data such as looking at daily patterns recognized in the data

[26].Eventually a SARIMA model can effectively eliminate the influence of periodicity in the prediction and it has shown great results in the past tests [27].

In a study conducted by Wang et al [27] where they were looking at the performance of three residual modifications for improving SARIMA, they implemented an optimized fourier residual modification and this residual improved the outputs accuracy more than the normal SARIMA. The SARIMA/ARIMA are statistical models and carry the same disadvantages, even though it has higher accuracy that ES models it is still inferior to artificial neural network(ANN) and SVM models [28].This is mainly because of its inability to capture non-linearity in the data [29] and they are also limited in capturing long term dependencies [26]. In the study by Jiang et al [28] they found that ARIMA is less accurate, but notably more computationally efficient (11.25 seconds for ARIMA versus 683.62 seconds for ANN and 1412.7 seconds for GA-SVM). Hybrid ARIMA models would outperform standard ARIMA models however they would still not perform on the same level to machine learning models.

2.2 Intelligent Models

The emergence of intelligent methods in STLF has significantly advanced the field , moving beyond traditional statistical models to embrace computational techniques that can better handle the complex and non-linear nature of electricity demand [30].

At the dawn of Artificial intelligence(AI) were foundational models and tools that were used and implemented for STLF to ensure accuracy.The early AI applications marked a crucial shift from purely statistical approaches that were limited by computing capabilities and often struggled with the non-linear features of time series data [31]. These models aimed to address these limitations by leveraging their ability to learn patterns from complex data but they did not do so well due to non-linearity of the data.

We will start by looking at very fundamental models that have been used for prediction and how they have impacted the industry , we will expand further into looking at the more advanced models that are currently being implemented and tested in modern day technology.

2.2.1 Early Artificial Intelligence(AI) and Machine Learning Models

Support Vector Machines (SVMs) SVMs and their variants such as the Support Vector Regression(SVR) and Least Squares SVMs (LSSVM) were among the first prominent techniques used in STLF [31].The basic idea of an SVM for regression , is to ,"map the data x into a high-dimensional feature space via a nonlinear mapping and to perform a linear regression in this feature space"[32]. This basically simplifies to an SVM being used in a classification task to choose a boundary that will maximize the margin that classifies an element in the dataset.

SVMs are renowned for their kernel trick that effectively handles non-linear input spaces and provides proficient prediction models for regression problems like load forecasting[17].They overcome overfitting issues through kernel methods and regularization techniques [17] , however a drawback is identified in their ineffectiveness or intolerable long training times when dealing with large datasets [33].Despite this SVMs have shown superiority compared to traditional statistical methods when it comes to load forecasting [14].

Fuzzy Logic methods also emerged as early AI tools for STLF. Fuzzy logic is an extension of boolean logic however instead of having true or false(0 or 2) , fuzzy logic allows variability of the truth ranging between 0 and 1. For example a pot can be 0.7 for "hot" and 0.3 for "warm".Fuzzy logic approches used fuzzy rules to integrate historical load data with time and day characteristics to determine probable load curves[9].

Artificial Neural Networks(ANNs) represented a significant leap forward due to their ability to model complex non-linear relationships between loads and related factors [33]. Mimicking the human brain's information processing, ANNs can approximate non-linear functions with high fidelity and accuracy, making them well suited for load forecasting [30]. ANNs gained their popularity due to their flexibility and robustness in handling diverse input-output projections and architectures. The earliest forms of ANNs for short term load forecasting showed up in the early 90's in the form of Multi Layer Perceptrons(MLP) [30].Park et al [30] provided a 3 layer perceptron that used historical load data and temperature data to predict peak load, total daily load, and hourly load.Mandal et al also utilized a back-propagation neural network and this was noted to be the firstly widely used ANN method for STLF [8].Early ANN models have evolved to become more complex and capable for tackling task as load forecasting with ease.

Challenges and Drawbacks of Early AI Models

Despite the initial promise , standard ANNs presented key problems that limited their efficiency for time series tasks like STLF. These Methods had *inefficiencies of Back-propagation.* The training of ANNs, especially those relying on backpropagation (BP) algorithms, suffered from issues such as slow convergence rates and high computational costs [33]. ANNs were also prone to getting stuck in local optima and exhibiting overfitting, which could lead to poor generalization on new, unseen data [5]. The final results of ANN models couldd also be dependent on the initial random weights and thresholds, contributing to forecast instability [30].

The early models also had a *Lack of Temporal Memory*, which is a model's capacity to retain and utilise information from previous time steps when processing current and future data, which is crucial for capturing long-term dependencies and patterns that evolve over time[34]. A fundamental limitation of standard ANNs for time series forecasting tasks was always their inability to learn from the sequential nature of the load data.Typical these models would treat each of the time steps individually when the time points are largely correlated.This would result in failure to memorize past information or account for long-term dependencies inherent in time varying electricity consumption patterns [5]. This meant that they struggled to capture the influence of previous timesteps on current or future load values , hindering accurate predictions for dynamic systems.

2.2.2 Advanced Deep Learning Architecture for Enhanced STLF

Deep learning techniques, characterized by a significantly higher number of layers, offered a powerful solution to these limitations, enabling models to deal with complicated non-linear patterns and learn complete probability distributions with temporal memory from vast datasets [11].The vast data that is generated daily by smart grids and Internet of Things (IOT) devices makes deep learning models ideal due to their scalability.Deep learning methods often offer better performance than more conventional machine learning methods [35].

Recurrent Neural Networks and Long Short Term Memory

Recurrent Neural Networks (RNNs) were introduced as a direct solution to the temporal memory problem in neural networks.Unlike the feed-forward networks, RNNs establish

weight connections between layers over time, allowing them to reflect sequential information and possess a "memory ability" [31].This made these suitable for time series data, where the current outputs depend on the previous inputs and hidden states.This is because RNNs introduce a loop so that the network can remember information from previous steps.

However, original RNNs still faced the vanishing gradient problem, where the ability of later time nodes to perceive previous ones decreased as the network deepened, limiting their performance with long time sequences [31].

To address this, the Long Short-Term Memory (LSTM) network architecture was proposed, enhancing RNNs with recurrent gates (input, output, and forget gates) to solve the vanishing gradient problem and effectively deal with long-term dependencies [5].LSTM networks contain memory cells that store information over random time intervals, and gates that trace the flow of input and output data from the cell, allowing them to determine what information to discard, store, or output selectively [29].Studies have consistently demonstrated LSTM's effectiveness, showing lower errors and higher accuracy in STLF compared to traditional ANN approaches [19][20]. They are particularly capable of handling more complex time series load data with long-term dependencies [9].LSTM has shown superior performance for longer forecast horizons compared to ARMA models [11] and this is because of their temporal capabilities.

Bidirectional LSTM (Bi-LSTM)

Bidirectional LSTM (Bi-LSTM) networks significantly improve upon the standard LSTM by processing data in both forward and backward directions [5]. This architecture consists of two independent LSTM layers that run in opposite directions, both connected to the same output layer [36]. This allows the model to leverage both past context (from the forward pass) and future context (from the backward pass) for more accurate and robust predictions. The bidirectional movement and interconnected structure help eliminate problems such as missing data and overfitting in the training phase [36]. Research has shown Bi-LSTM to achieve superior performance, often with significantly lower Mean Absolute Percentage Error (MAPE) values, compared to unidirectional LSTMs and other methods [35].

Deep Belief Network (DBN)

Deep Belief Networks (DBNs) emerged as a crucial solution to address some of the challenges associated with backpropagation, particularly the difficulty of finding optimal initial parameters and the problem of local optima [37]. DBNs are generative probabilistic models composed of stacked Restricted Boltzmann Machines (RBMs) and a classifier. DBNs combine deep learning with feature learning , giving them powerful fitting capabilities to quickly analyze large amounts of data [37].The multiple layers in DBNs help make them better in handling multifactor relationships in comparison to single layer networks.Their greedy, layer-wise unsupervised pre-training mechanism helps in identifying better initial parameters, which are then fine-tuned through supervised learning [37]. This pre-training process allows DBNs to learn patterns progressively and overcome the disadvantage of being easily trapped in local optima due to random weight initialization. By converting continuous input features into binomial distribution features (e.g., using Gauss-Bernoulli RBMs), DBNs can improve their effectiveness in processing real-valued data [37]. This approach leads to faster convergence and improved prediction performance in various fields, including load forecasting. This makes them ideal for uncovering electricity usage patterns and are considered ideal due to their scalability [20].

Other Advanced Models

Gated Recurrent Unit (GRU) neural networks were proposed as a simpler alternative to LSTM , combining the input and forget gates into a single "update gate" [31]. GRUs can achieve similar accuracy to LSTMs with less computational cost and faster convergence [38]. They are effective at extracting temporal features and preserve important features while using fewer parameters [38].

Convolutional Neural Networks (CNNs) have been frequently used in load prediction due to their excellent ability to capture the trend of load data. CNNs are particularly effective for feature extraction and dimension reduction of original data [38]. CNN's have a lack of temporal memory that put them at a disadvantage in forecasting tasks separately that require memory.They are often integrated with other models like LSTMs to leverage both spatial feature extraction and temporal modeling capabilities [9]. A method by Shafiul Hasan Rafi proposes an integrated CNN and LSTM network for STLF in the Bangladesh power system, reporting higher precision and accuracy than existing approaches like LSTM, Radial Basis Function Network (RBFN), and Extreme

Gradient Boosting (XGBoost) [9].

2.3 Hybrid Models

Hybrid models have emerged as a prominent approach in Short-Term Load Forecasting (STLF) to address the inherent complexities of electricity demand prediction, such as its non-linear and non-stationary characteristics [2].These models combine the strengths of various algorithms and techniques to enhance forecasting accuracy and reliability, overcoming the limitations often encountered by single, standalone models.

The fundamental motivation is that no single forecasting technique is superior in all cases [39]. By combining methods, hybrid models can effectively handle the non-linear and multivariate characteristics of load data, extract complex patterns, and improve prediction accuracy and stability [39]. Conceptually, they operate by breaking down the forecasting problem, applying specialized techniques to different aspects (e.g., feature extraction, temporal learning, error correction), and then integrating the results. This often involves preprocessing data, applying deep learning for feature learning and sequence modeling, and optimizing parameters [37].

2.3.1 Decomposition-Based Hybrid Models

A common approach involves decomposing the original, complex load data into simpler, more stable components using techniques like Variational Mode Decomposition (VMD) [29],Empirical Mode Decomposition (EMD), or complete ensemble empirical mode decomposition with adaptive noise (CEEMDAN) [38].These methods will solve the problem of analyzing time series data directly and solve this by splitting the data into intrinsic components that are easier for machine learning models to learn from. Each component is then predicted using an appropriate model like an LSTM for high-frequency components, MLR for low-frequency components [40] and the results are aggregated for the final forecast. This pre-processing step significantly reduces volatility and improves prediction accuracy [5].

2.3.2 Deep Learning Combination Hybrid Models

CNN-LSTM/GRU

Hybrid models like CNN-LSTM and CNN-GRU are frequently used. The CNN module excels at extracting local features and patterns from time series data, while the LSTM or GRU module is proficient in capturing long-term temporal dependencies[34].The combination of such models help get the benefits of both models in a prediction task.

The CNN module is typically used to extract local features and patterns from load data [8]. This is achieved through pooling and convolution layers which process two-dimensional data and flatten it into a one-dimensional feature vector.This feature vector is then fed as input to the LSTM or GRU layers, which are good at capturing long-term dependencies and sequential information in time series data [34]. LSTM mechanisms allow for the retaining of long-term information and address the vanishing gradient problem [34]. The GRU would combine with the CNN in the same way as an LSTM as they have a similar structure just simpler and fewer parameters, making it faster[31].

This combination improved the forecast accuracy, reducing prediction errors and outperforming single models. In the tests done by Rafi et al [9] , they found that CNN-LSTM models have a higher precision and accuracy in short-term load forecasting compared to standalone LSTM, Radial Basis Function Network (RBFN), and Extreme Gradient Boosting (XGBoost) approaches. A CNN-LSTM model provided 173.76 MW less Mean Absolute Error (MAE), 330.2 MW less Root Mean Squared Error (RMSE), and 3.07% less Mean Absolute Percentage Error (MAPE) on average than a standalone LSTM network.In another study done by Danish et al [41] they found that the GRU-CNN model outperfomed a GRU , CNN and a Back Propagation Neural Network(BPNN) and this was due to its capability to learn from both time sequence data and spatiotemporal data.

DBN-RNN/Bi-RNN

Deep Belief Networks (DBNs) are integrated with Recurrent Neural Networks (RNNs) or Bidirectional RNNs (Bi-RNNs). This hybrid model typically leverages teg DBNs capability for unsupervised pre-training and optimal parameter seeking with the RNN's strength in processing sequential and time-dependent data [42].

A DBN being a generative and probabilistic model composed of multiple layers of RBM.

These RBM combine the traditional neural networks with energy and probabilistic models [2].The RBM is crucial for initialising parameters and this pretraining helps overcome issues like local minimisation and slow convergence seen in some other neural network [43]. On the other side the RNN is specifically designed to process sequence data due to their internal loop structure that allows information to persist across time steps [31] and this makes them highly effective for STLF where mining large quantities of temporal data is essential. A Bi-RNN is just a structural improvement that will process the input simultaneously in two opposite directions (forward and backward) [42] . This bidirectional structure gives the RNN complete past and future context information for each point in the input layer , enhancing the validity of the forecasting model [42].

In the test done by Tang et al [42] they used the DBN to perform the unsupervised greedy pre-training layer by layer to obtain initial weight parameters, making it easier to approach optimal values.The pretraining phase helped in handling parameters initialization problem in a large dataset. After this pre-training the parameters are supervisedly adjusted through the Bi-RNN, which then propagates learned time information in both directions to derive the final forecasted value. In addition to model design, the study also incorporated data preprocessing techniques. Specifically, the bisecting k-means algorithm was applied for clustering, while Ensemble Empirical Mode Decomposition (EEMD) was used to decompose the load data into intrinsic mode functions (IMFs). This decomposition helped reduce noise and enhance feature selection, with methods such as the Pearson correlation coefficient further refining the input features. This data preprocessing played a part in the final perfomance of this hybrid model. The results of this study show that DBN with Bi-RNN reduces the MAPE to 1.85% from 2% which is a move towards the right direction.The DBN-Bi-RNN models have been shown to outperform unidirectional LSTM, SVR, and BPNN [42].

The DBN-RNN/Bi-RNN hybrid models provide a robust and accurate solution for STLF by combining DBN's parameter optimisation and feature learning capabilities with RNN/Bi-RNN's strength in handling temporal dependencies and contextual information. They often outperform traditional and even some advanced deep learning models in terms of forecasting accuracy and computational efficiency [2].

2.4 Data Pre-Processing Techniques

While machine learning and statistical models are powerful, their effectiveness depends heavily on the quality and structure of the input data. Proper data preprocessing is

therefore essential to ensure that models operate at their full potential. In the context of electricity load forecasting, preprocessing addresses challenges arising from the complex, non-linear, non-stationary, and often noisy nature of load data factors influenced by weather conditions, consumer behavior, and other external variables [24]. A variety of preprocessing methods and techniques have been developed to tackle these challenges, and in this section we highlight some of them, examining their role and impact on model performance.

Data Cleaning and Outlier Detection

Large datasets often contain incorrect, missing, or anomalous values, commonly referred to as data outliers. In machine learning models such as LSTMs, which depend heavily on the accuracy of historical data, the presence of outliers can significantly degrade model performance. To address this, techniques like the Hampel Identifier (HI) and the Interquartile Range (IQR) method are commonly applied to detect and correct or smooth anomalous data points [38].

The HI method was used in a study by Wang et al [38]. This method applies a sliding window mechanism and replaces the extreme values with more representative estimates. For each data point , the median of the surrounding window is calculated along with the median absolute deviation(MAD), which will serve as a variability measure. They then scale the MAD to approximate the standard deviation.If a sample differs from the local median by more than three scaled deviations, it is identified as an outlier and replaced with the window median. Incorporating HI into data pre-processing ensures that outliers are corrected, thereby preventing disruptions in model training and improving the overall fitting performance.

To handle missing values in the dataset which may be very common due to defective sensors or erronous recordings, these can be addressed by filling in these missing values using zeros, means, median, linear interpolations or by just outright deleting the data points [24].

Cleaning and detecting and removing outliers improve the quality and accuracy of the data preventing them from disrupting the model training, negatively impacting forecast accuracy [38]. Pre-processing can reduce the data complexity, as evidenced by a lowered sample entropy (SampEn) value in [38] indicating a higher degree of self-similarity and better data quality. Pre-processing with outlier correction improves the non-linear fitting performance of data and enhances prediction accuracy of a model.

Data Normalization and Transformation

The dataset that is collected has multiple data points that are assigned different values and weights. Each of the featurea should be treated withing the same scaling of importance. "The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values",[44]. This ensures that no single data point carries an advantage because of its numerical size.

Normalization will scale numerical input features to a uniform range, typically between 0 and 1, or to have zero mean and unit variance [2]. Normalization can also be implemented using the min-max normalization method that will scale the data to a specified range. In [2] this method is used to preprocess historical load, temperature and other features and it is very important in neural networks because of their sensitivity to variation in the input data. Feature scaling can also be done and this generally refers to scaling independent features to prevent biases from varying dimensions and ensure features are on a uniform scale [34].

Feature Engineering and Selection

Feature Engineering is the process of transforming raw data into meaningful features that better represent the underlying patterns [20]. Different prediction models perform best with different types of data. For example a CNN would perform best for image recognition meaning if we can change our raw data into an image. We can use a CNN for detection and prediction. Feature engineering can also be done by using some signal processing techniques to turn your data into signals that would then be used to train the models.

Variation Mode Decomposition (VMD) is one of the methods used for feature engineering. VMD is an adaptive signal decomposition method that iteratively searches the variation mode to decompose an original time series signal into a discrete number of modes, each with a limited bandwidth and a corresponding center frequency [40]. The primary objective of VMD is to decompose the original signal into a set of modes where each mode has the smallest possible frequency range, while ensuring that all these modes, when put back together, exactly reproduce the original signal [5]. VMD works by iteratively finding the optimal centre frequency and bandwidth for each mode and it does this by shifting the frequency of each mode to a "baseband" using a specific mathematical adjustment. After this it measures the spread of frequencies (bandwidth) for that shifted mode using a

smooth, bell-shaped curve and this process is repeated many times [5] until a predefined stop condition is met, ensuring adaptive decomposition of the signal [5].

when we use VMD on data we have a very strong anti noise capability and it reduces the complexity that is present in natural time series data [40]. VMD overcomes the modal mixing problem that is inherent in Empirical Mode Decomposition(EMD) and we can artificially change the number of mode decomposition which is an advantage over EMD [40].

2.4.1 Summary

Studies around the area of STLF have shown a very clear path towards machine learning and AI algorithms being the future of prediction. The promise of these methods have been significantly highlighted for delivering more accurate load forecasts compared to traditional statistical methods. While the newer models may involve higher complexity and computational costs, their enhanced accuracy is seen as offsetting the negatives.There has also been a clear overarching between the statistical, hybrid and intelligent models along with data preprocessing techniques with the sole goal of higher accuracy and handling data anomalies and seasonal variations in the data.

The review has recognized LSTMs for their satisfactory performance in short term forecasting and capabilities to capture non-linear and temporal characteristics present in data. It has been shown that a combination of an LSTM with a CNN can also bring about a combination of benefits possessed by both models. This model can also be combined with purely statistical models to create hybrid models. A variation of LSTM the Bi-LSTM was also identified as a potential for the future of LSTM in STLF in microgrids.Each of the variations of the STLF came with its own benefits in advancing load foresting. DBNs also showed a potential of being developed further to provide quality results that would be implemented into an accurate model of the future. DBNs also perform a lot better when they are presented in the Bi-Directional version of themselves.

This literature review has shown the capabilities of these advanced models effectively capturing non-linearities, long-range dependencies, and handle large-volumes of time-series data, often benefiting from hybrid approaches and optimization techniques to deliver superior predictive performance.

Chapter 3

Methodology

The goal for this research is to investigate the impact of ML methods in STLF. The literature review shows improved accuracy when using ML models. It also highlights the effectiveness of statistical models and SVM's in temporal prediction tasks. It further shows the increased accuracy in ML and AI models, showing how hybrid models can capture the benefits of different models in one. In this section we will focus on the data processing steps, the theoretical background of the models and the methodology followed to produce them.

3.1 Data Collection and Description

The dataset that is used in the experiments handled in this research was collected by the Panama City government in central America as an initiative to research and improve methods for short term load forecasting. The dataset is available on Kaggle and Mendeley Data and provides historical records of electrical load data and relevant weather variables of Panama city[45].

The dataset consists of hourly observations spanning the period from January 2015 to December 2019, yielding approximately 48048 data points. The target variable is the load demand measured in megawatts(MW), while the other features include multiple weather and environmental parameters recorded at different stations across Santiago, Tocumen and David.

The dataset has the following features as mentioned in [45]:

- datetime - The date and time sampled every hour from
- T2M – Temperature at 2 meters above ground (°C)
- QV2M – Specific humidity at 2 meters (%) sampled in the three cities
- TQL – Total cloud water content (liters/m^2) sampled n the three cities
- W2M – Wind speed at 2 meters (m/s) sampled in the three cities
- Holiday_ID - a unique value representing a particular holiday in the country
- holiday - a binary value representing whether the day is a holiday or not
- school - a binary representing schools being open or closed

The environmental features in the dataset are sampled hourly in the three cities and are represented as toc, san and dav as the extensions of the feature name (e.g. T2M_dav would be the temperature in David). These meteorological inputs have been selected due to their established influence on electricity demand, as load consumption often correlates with environmental factors such as ambient temperature, humidity, and weather patterns. The relationship between temperature and load has been established with Hor et al [46] seeing a correlation in the increase in temperature with the increase in the load demand.

The dataset has been cleaned previously by the initial collectors to ensure minimum erroneous values. The dataset has 4 separate folders namely:

- continuous-dataset.csv - containing data sampled every hour for the duration of the collection
- test_dataframes.xslx - a test dataset prepared with historical load from 4 weeks previously [45]
- train_dataframes.xslx - a train dataset prepared with historical load from 4 weeks previously with hourly granularity [45]
- weekly_pre_dispatch_forecast.csv - contains the load forecast from the weekly pre-dispatch report

To train the models in this research we chose to use the continuous_dataset.csv. This is because of the comprehensive feature set in the dataset. This dataset also has continuous unsplit time series data allowing for testing of custom train test splits that align with the model designs. Finally the flexibility for feature engineering on this dataset is enhanced. It can allow you to generate custom lagged features and rolling statistics giving the researcher transparency and reproducibility.

3.2 Data Preprocessing

Data preprocessing is an important step in training of ML and statistical models as it ensures that the data being used is free of noise and outliers. Preprocessing also includes feature engineering which help identify and retain the most influential features, hence simplifying models, reducing redundancy and improving performance [43].

3.2.1 Choice of programming language

The two options for the programming language to use for this problem were Python and MATLAB. Both these choices offered pros and cons, shown in table 3.1.

Aspect	**Python**	**MATLAB**
Data Processing	Rich data processing framework(Pandas, NumPy).	Built-in data handling but less flexible than Python's frameworks.
Machine Learning & AI	Strong libraries: TensorFlow, PyTorch, Scikit-learn.	ML toolboxes available, but limited adoption compared to Python.
Community Support	Huge global community; extensive tutorials, forums, and open-source contributions.	Strong academic/engineering base, but smaller community outside academia.
Flexibility of production environment	Works well for online development online with google collab offering CPU and GPU services.	Better integration with control simulations, and hardware prototyping.
Ease of Use	Easy-to-read syntax, intuitive for beginners, widely taught.	Powerful for numerical computing syntax less intuitive for general-purpose tasks.
Performance	Fast with optimized libraries (NumPy, TensorFlow GPU acceleration).	Highly optimized for matrix operations sometimes faster out-of-the-box for linear algebra.

Table 3.1: Comparison of Python and MATLAB for data analysis and machine learning.

Matlab is powerful in numerical computing and is widely used in academia and engi-

neering settings, python was chosen for this research due to several key reasons. Python provides robust data processing frameworks, which make data cleaning, transformation and preprocessing efficient and straightforward. Python supports industry standard ML frameworks such as TensorFlow and Scikit-learn that were extensively used in this research for setting up and training the models. These models offer a ease of use and advanced modeling and experimentation. Python has a massive global community offering more resources and troubleshooting support ~~more~~ than MATLAB. After careful consideration of the pros and cons, python was used for modeling.

3.2.2 Feature Engineering

The data handling framework that is used in the project is python Pandas. Pandas offers a wide range of processes to feature engineer your dataset. It also sets the dataset into a format that is user friendly for training and testing both statistical and machine learning models.

Forward Filling The first step taken to ensure that the dataset is complete and there are no missing values, was the forward fill method. Forward fill is a method where you fill a missing value with the last known value. The last known value is carried forward to replace the missing value. An illustration of how this method works is shown in figure 2. This method is simple and effective for filling in values such as temperature which have low variance.

Day of Week Encoding A `day_of_week` column was added to the dataset to represent the weekday on which each observation occurred. This feature allows the models to capture temporal seasonality related to human activity patterns that typically vary across weekdays and weekends. For instance, energy consumption or cooling demand may be systematically higher on weekdays compared to weekends.

Cyclical Encoding : To avoid discontinuities and ensure the network learns smooth transition of time eg from 11PM to 12 AM , cyclical encoding is applied to hour, day of week and month. With cyclical encoding the times 11PM and 12AM are represented as being close to each. This is performed by using sine and cosine transformation to the dataset columns associated.

Weekend and Weekday Tag Building on the day-of-week information, a binary tag distinguishing between weekdays and weekends was introduced. This feature reduces the dimensionality of temporal effects and allows the models to easily account for systematic differences in behaviour between weekdays and weekends.

Month Identifier To incorporate longer-term seasonal variations, a `month` identifier was added. This feature makes it possible to detect monthly or seasonal cycles in the data, such as weather-related patterns or operational schedules that repeat on a monthly basis.

Holiday Combination Feature The dataset included two holiday-related variables: `holiday_ID`, which was a categorical value ranging from 1 to 22 representing different types of holidays, and `holiday`, which was a binary value indicating whether a given day was a holiday or not. To enhance the usability of these features, a combined feature was created by multiplying `holiday_ID` with `holiday`. This ensured that non-holiday days were represented as zero, while holidays retained their unique identifiers. This transformation simplified model training by consolidating redundant information into a single, more informative feature.

3.2.3 Outlier Detection and Removal using the Hampel Identifier Method

Outliers can significantly bias statistical and machine learning models if not properly addressed. To mitigate this, the Hampel identifier(HI) method was employed for outlier detection. The HI method was chosen because it is a robust statistical technique based on the median and the median absolute deviation (MAD)[38]. The method is also robust to heavy tailed data and skewed distributions of raw data. Below are the equations used for the implementation as taken from [38].

Consider the input sequence $A = [a_1, a_2, \ldots, a_k]$. For each sample a_i, a symmetric sliding window of length $w = 2n + 1$ centered at i is defined. Within this window:

$$m_i = \text{median}(a_{i-n}, \ldots, a_i, \ldots, a_{i+n}) \tag{1}$$

The median absolute deviation (MAD) is calculated as:

$$\text{MAD}_i = \text{median}\left(|a_j - m_i| : j \in [i-n,\, i+n]\right) \tag{2}$$

To estimate the standard deviation, the MAD is scaled using a constant $\alpha = 0.6745$ (consistent with normal distribution assumptions):

$$\sigma_i = \frac{\text{MAD}_i}{\alpha} \tag{3}$$

An observation is identified as an outlier if it deviates from the local median beyond three times the estimated standard deviation:

$$|a_i - m_i| > 3\sigma_i \tag{4}$$

If an outlier is detected, the original value a_i is replaced with the local median m_i. This correction preserves the temporal structure of the data while reducing the influence of anomalies. Incorporating HI ensures that erroneous spikes caused by equipment errors, missing sensor calibration, or human errors do not affect model training and forecasting. This HI formulation was shown to improve quality of raw data and reduce the occurrence of spikes in the data [38].

Data Normalization

To ensure that the features contributed proportionally during model training, all numerical variables in the dataset were normalized, with the exception of the `datetime` column, which was retained in its original format for temporal referencing and indexing the dataset. Normalization prevents features with larger absolute values from dominating those with smaller ranges, and it improves the convergence behavior of machine learning algorithms.

Several normalization and standardization approaches exist, such as z-score standardization, robust scaling, and Min–Max scaling. In this work, the Min–Max scaling method was used due to its effectiveness in re-scaling features to a fixed range while preserving the original distribution's shape. The transformation is defined as:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{5}$$

where x is the original feature value, $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the feature, and x' is the normalized value re-scaled to the interval $[0, 1]$.

The Min–Max scaling method was chosen because it is well-suited for neural networks and distance-based models, particularly those relying on gradient descent, as these algorithms perform optimally when features are restricted to a bounded range [47]. Unlike z-score normalization, which re-centers data around the mean, Min–Max scaling preserves the original relative spacing between values, making it appropriate when proportional differences carry meaning. Furthermore, by rescaling all features to the interval $[0, 1]$, it provides a consistent and interpretable representation that simplifies visualization and ensures that all features contribute fairly during training.

By applying Min–Max scaling, the dataset was transformed into a consistent format across all features, thereby improving model training stability and ensuring fair contribution of each feature to the learning process.

The steps taken for the data preprocessing are shown in the fig 1 in appendix 7.

3.3 Evaluation Metrics

The models were compared using standard error metrics and information criteria that are widely used in forecasting literature. These metrics provide complementary views on model performance in terms of accuracy, error magnitude, and model complexity. The chosen metrics are briefly described below.

Mean Absolute Percentage Error (MAPE)

MAPE expresses forecast accuracy as a percentage, making it easy to interpret across different scales. Lower values indicate better performance.

$$\text{MAPE} = \frac{100}{n} \sum_{t=1}^{n} \left| \frac{Y_t - \hat{Y}_t}{Y_t} \right| \tag{6}$$

Mean Squared Error (MSE)

MSE penalises larger errors more heavily by squaring them, making it sensitive to outliers.

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(Y_t - \hat{Y}_t)^2 \tag{7}$$

Root Mean Squared Error (RMSE)

RMSE is the square root of MSE and brings the error measure back to the same scale as the original data.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(Y_t - \hat{Y}_t)^2} \tag{8}$$

Coefficient of Determination (R^2)

R^2 measures how well the forecasts explain the variance in the actual data. Higher values (closer to 1) indicate better fit.

$$R^2 = 1 - \frac{\sum_{t=1}^{n}(Y_t - \hat{Y}_t)^2}{\sum_{t=1}^{n}(Y_t - \bar{Y})^2} \tag{9}$$

Akaike Information Criterion (AIC)

AIC balances model fit and complexity. It penalizes over-parameterized models, with lower values indicating a better trade-off.

$$\text{AIC} = 2k - 2\ln(\hat{L}) \tag{10}$$

where k is the number of model parameters and $\hat{L}$ is the maximized likelihood.

3.4 Statistical and Machine Learning Models

3.4.1 Exponential Smoothing

Exponential smoothing has been widely used in STLF due to its low computational needs and fast processing times. The core idea of this method is to produce a forecast from an exponentially weighted average of past observations, giving the largest weight to the most recent data and exponentially decreasing weights to older data [48]. This method's functionality has been discussed extensively in section 2.1. In this section we will focus on the theoretical fundamentals and the choice of model we used in our research.

There are multiple forms of ES, each designed to capture different components of a time series data.

Simple Exponential Smoothing(SES) SES is designed for time series datasets that lack seasonality and trend. The forecast is based solely on the weighted average of past observations. If $\hat{Y}_t$ is the value at time t and α is the smoothing parameter between 0 and 1. The equation for the smoothed value S_t would be 11 as adapted from [48].

$$\hat{S}_{t+1} = \alpha Y_t + (1 - \alpha)\hat{Y}_t \tag{11}$$

Though the simplicity of SES is good for very low computational tasks, it is not favorable because of its low accuracy and failure to identify presence of the trend and seasonality in the data.

Double Exponential Smoothing (DES) method introduces capturing of trend in the time series data. This is in addition to the already existing level component in SES 3.4.1. The equation of DES as adapted from [49] would be:

$$S_t = \alpha Y_t + (1 - \alpha)(S_{t-1} + b_{t-1}) \tag{12}$$

$$b_t = \gamma(S_t - S_{t-1}) + (1 - \gamma)b_{t-1} \tag{13}$$

where b_t would be the estimated trend or slope of the dataset and γ would be the smoothing parameter for the trend between 0 and 1. The data used in this research shows a

clear flat trend over the long-term and seasonality every 24 hours with a fast rise and peak demand during midday shown in image 3.1. This 24 hour seasonality brings us to the third ES model which is the Triple Exponential Smoothing (TES).

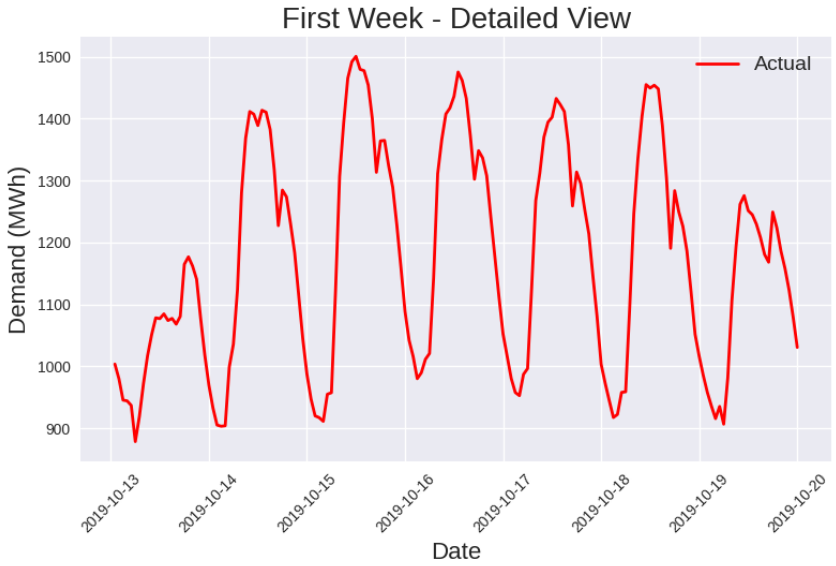

Figure 3.1: Real electricity demand from Sunday 13th to Saturday 20th of October 2019

Triple Exponential Smoothing introduces the equation that takes care of the seasonality component [49]. TES inherits equation 12 and 13 and introduces equation 14 for seasonality adapted from [49].

$$I_t = \beta \frac{Y_t}{S_t} + (1 - \beta) I_{t-m} \tag{14}$$

I_t is the estimated seasonal component and β is the seasonality constant that is between 0 and 1. With the combination of equation 11 , 13 and 14 the overall smoothed value equation would be 15.

$$S_t = \alpha \frac{Y_t}{I_{t-L}} + (1 - \alpha)(S_{t-1} + b_{t-1}) \tag{15}$$

Damping is a mechanism used to reduce or dampen the trend in forecasts over long horizons [50]. This method makes the forecast trend more conservative, preventing it from overshooting the actual data, which is a common problem in ES methods projecting a linear trend indefinitely in the future [50]. The equation below is for the smoothed value inclusive of the damping factor ϕ which is between 0 and 1, adapted from [50]. With equation 16 being the additive in level and equation 17 being the multiplicative in

level.

$$S_t = \alpha \frac{Y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} + \phi\, b_{t-1}) \tag{16}$$

$$S_t = \alpha \frac{Y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} \cdot b_{t-1}^{\phi}) \tag{17}$$

Choice of best Exponential Smoothing model

ES was used as a benchmark for testing the ML models. However because of the different variations of ES an algorithm was developed to help choose the best performing ES model for the research. The models performance were compared on their MAPE, MAE, Mean Squared Error(MSE), RMSE and the AIC.

This algorithm would create a model using *statsmodel holtwinters* [51] and test performance of forecasts o. Different model's performance would be compared to each other. Image 3 in appendix A shows a the algorithm followed to choose the best ES model. Table 3.2 shows the different models that were tested to find the best model.

Model	Trend	Seasonality	Seasonality Period	Damped	MAPE (%)	MSE (MWh)	AIC
Simple	None	None	–	No	13.76%	180.98	343990.96
Double	Add	None	–	No	15.14%	172.11	327001.68
Triple_Add	Add	Add	24	No	10.56%	125.56	277981.02
Triple_Mul	Mul	Mul	24	No	34.48%	408.23	272265.85
Triple_Add_Damped	Add	Add	24	Yes	9.99%	120.81	277708.98
Triple_Mul_Damped	Mul	Mul	24	Yes	10.01%	119.49	272266.18

Table 3.2: Exponential Smoothing Models chosen for benchmarking and their performance.

Comparison of performance metrics through the algorithm in figure 3 was the Triple Multiplicative Damped algorithm with a seasonality period of 24 data points. Since the dataset is sampled hourly the seasonality was set to 24 hours. The MAPE and MSE are very close to each other with a difference of about 0.01 each. However the AIC value for *Triple_Mul_Damped* is lower than the one for *Triple_Add_Damped*, this showing a better model fit for *Triple_Mul_Damped*. The final choice for the ES model was *Triple_Mul_Damped* and it served as a benchmark for testing performance of other models used in the experiment.

Implementation of the Triple Multiplicative Damped ES model

3.4.2 Deep Belief Network

A DBN is a probabilistic generative model composed of multiple layers of stochastic, latent variables [52]. It is constructed by stacking multiple layers of RBMs on top of each other [53]. This model benefits from its capabilities of limiting the occurrence of the local minima by pre-training RBMs using unsupervised training to adjust weights and parameters to ensure an enhanced performance in actual prediction use cases.

Restricted Boltzmann Machine (RBM)

A Restricted Boltzmann Machine (RBM) is a two-layer probabilistic generative model designed to learn the underlying probability distribution of input data. RBMs are widely used as building blocks for constructing DBNs [2]. The architecture consists of a *visible layer*, which receives the observed data, and a *hidden layer*, which captures latent features and higher-order correlations in the data. Learning in RBMs is unsupervised, the model attempts to represent the structure of the input data by adjusting its parameters weights and biases so as to maximize the likelihood of the training samples [52].

An RBM is defined by a weight matrix that connects each visible unit to each hidden unit in a bipartite manner, together with bias vectors for both layers. No intra-layer connections are permitted, which simplifies inference and training. The structure is shown in figure 3.2, adapted from Zhang et al. [52].

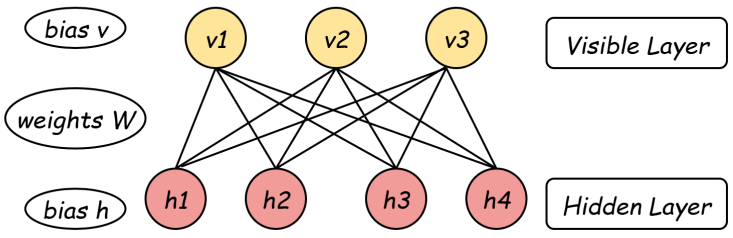

Figure 3.2: Illustration of an RBM with visible and hidden layers, including weights and biases

Energy-Based Formulation: An RBM is an energy-based model, meaning it assigns a scalar energy value to each configuration of visible and hidden units. Configurations with lower energy are more likely under the model. The energy function is given by [37, 53]:

$$E(v, h; \theta) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i W_{ij} h_j \tag{18}$$

where

- v_i: the state of visible unit i,
- h_j: the state of hidden unit j,
- a_i: the bias associated with visible unit i,
- b_j: the bias associated with hidden unit j,
- W_{ij}: the weight between visible unit i and hidden unit j,
- $\theta = \{W, a, b\}$: the model parameters.

Probability Distribution: Using the energy function, the joint probability of a visible hidden configuration is defined by the Boltzmann distribution [54]:

$$P(v, h) = \frac{e^{-E(v,h)}}{Z} \tag{19}$$

where Z is the *partition function*, given by summing over all possible visible and hidden states:

$$Z = \sum_{v,h} e^{-E(v,h)} \tag{20}$$

The probability of a visible vector v is obtained by marginalizing out the hidden layer:

$$P(v) = \frac{1}{Z} \sum_h e^{-E(v,h)} \tag{21}$$

Training RBMs: RBMs are typically trained using the *Contrastive Divergence* (CD) algorithm which is a version for forward and back propagation. Training consists of two phases:

1. **Positive phase:** a visible layer vector from the data activates the hidden layer, and correlation outputs $\langle v_i h_j \rangle_{\text{data}}$ are computed.
2. **Negative phase:** the hidden units output are used to reconstruct the visible layer, producing a reconstructed vector. From this, correlations $\langle v_i h_j \rangle_{\text{recon}}$ are obtained.

The difference between the two phases which is error provides the learning signal for updating weights:

$$\Delta W_{ij} = \eta\Big(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{recon}}\Big) \tag{22}$$

where η is the learning rate. In this way, the RBM iteratively learns to reduce the gap between the distribution of the training data and the distribution it models. The feed-forward (data-to-hidden) and feed-backward (hidden-to-reconstruction) passes, together with weight updates, constitute the CD training loop [55].

DBN (Stacked RBMs) When 1 or more RBMs are stacked together they create a DBN. This DBN has the advantage of pre-trained RBMs that have adjusted their weightings to minimize error in the training phase. Stacking of RBMs is as shown in the figure 3.3. This illustration highlights how the hidden layer of the first RBM is the

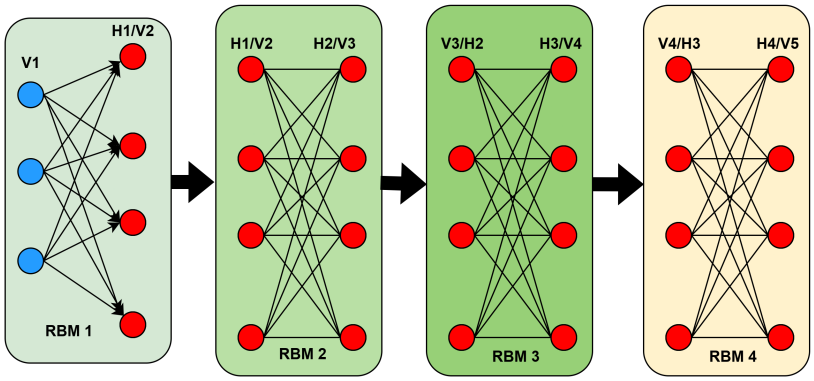

Figure 3.3: An illustration of stacked RBMs making a DBN

visible layer of RBM 2. During pretraining each RBM is trained separately. RBM1 will receive the original data and perform CD on RBM1, the recreated output on the hidden layer 1(HD1) becomes the input of RBM2 through the visible layer 2(V2). RBM2 will go through CD and continues the loop until the full RBM has been trained and the weights have been adjusted to fit the data.

Implementation of DBN architecture for STLF

The proposed DBN designed for the STLF task, leverages its hierarchical feature extraction capabilities to model temporal and non linear dependencies present in the data.

The DBN was implemented using *tensorflow.keras.callbacks* for fitting, evaluating and prediction functionality of the model. The DBN also uses *tensorflow.keras.layers* for creating the different RBM layers in the model. Scikit-learn was also used for data scaling and evaluating performance using features such as the *MinMaxScaler* and performance metrics.

Input Feature Engineering The original dataset went through its initial feature engineering as explained in section 3.2.3, however further feature engineering is performed to enhance performance.

1. **Lagged Load Values :** This is essential for capturing time series auto correlation, to do so the network uses lagged values of nat_demand at τ = [1,2,3,7,24,168] hours . These lags will enable capturing past demand for 1-3 hours, 24 hours and 168 hours which is a week. The lags are important for capturing seasonality and aligns with the seasonality benchmark in the ES implementation.
2. **Rolling Statistics :** Mean and standard deviation of nat_demand over 7 and 24-hour windows are included to capture recent trend and volatility.

All missing values resulting from lag and rolling window operations were handled through forward and backward filling, followed by complete case removal to ensure data integrity. The final feature set comprised approximately 20-25 input variables depending on the available data columns.

DBN Structure

The DBN structure comprises a stacking of 3 RBMs, forming a deep architecture representation. Table 3.3 shows the representation of each layer.

The architecture had 3 RBMs setup with 256, 128 and 64 units respectively. Each of these layers were trained using CD, before being stacked and initialized for the deep network.

Layer Type	**Units (n_h)**	**Activation (RBM Pre-training)**	**Activation (Fine-Tuning)**
RBM 1 (H1)	256	Sigmoid (σ)	ReLU
RBM 2 (H2)	128	Sigmoid (σ)	ReLU
RBM 3 (H3)	64	Sigmoid (σ)	ReLU
Fine-Tuning Layer	32	N/A	ReLU
Output Layer	1	-	Linear (None)

Table 3.3: DBN Layer configuration

The final layer added a dense layer of 32 ReLU units for finetuning and a linear regression output layer for predicting continuous load values. The ReLu activation function is chosen due to its ability to mitigate to the disappearing gradient problem [33].
The fine tuning layer acts as the combiner of RBM-initialized layers, putting them together before the output layer. This design implementation was chosen to allow the network to learn task specific transformations on top of the general features learned during unsupervised learning.
The final output layer is a single dense linear unit without an activation function, producing a continuous regression output representing the forecasted electricity demand. This type of output is appropriate for regression tasks as it allows the network to predict values across the full range of the target value without constraints. Figure 3.4 shows the implementation of the network.

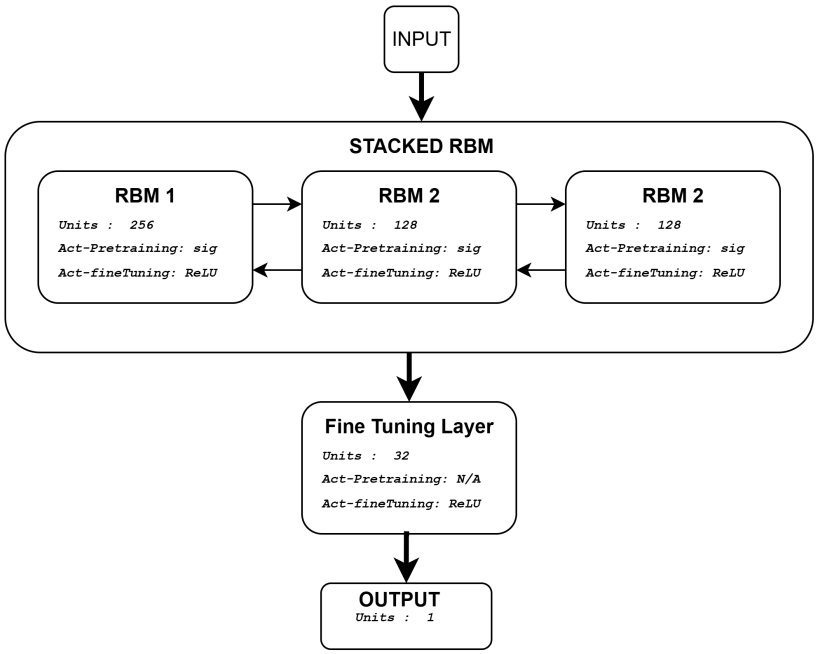

Figure 3.4: Image showing the implementation of the DBN used in the research

Training Procedure

DBN training is split into two sections, unsupervised learning for the RBMs and supervised learning for the stacked RBMs. These procedures will effectively initialize a deep forward network to a good starting point.

Pre-training (Unsupervised) Each of the RBMs are trained sequentially using the input data in an unsupervised manner to learn a robust and hierarchical representation of the input features. The CD algorithm is used to train each RBM, approximating the gradient using the error equation 22. A fixed momentum term of 0.9 is set to accelerate convergence and dampen oscillation.
The learning rate was implemented during the pretraining, starting at 0.01 for the first layer and decaying by a factor or 0.8 for each subsequent RBM layer. This decaying factor is to counter the increasing abstraction in higher layer which will require the updates of the layers to be more conservative. Each RBM is trained for 25 epochs, this is because there was a negligible change in the final prediction with increased epochs.

Fine-Tuning (Supervised) Following the supervised pre-training, the network is unrolled into a standard feed-forward neural network and fine tuned using supervised learning. The learned RBM weights and biases from the pre-training are transferred to initialize the corresponding dense layers in the supervised network. This initialization provides the network with meaningful feature representation. After the transfer the network will learn through supervised learning using standard back-propagation method which will train the network end-to-end.
The Adam optimizer is chosen with a base learning rate of 0.001. Adam is preferred over standard SGD for deep learning due to its adaptive learning rates for individual parameters, leading to faster and more stable convergence. The network is then compiled using MSE as its loss function, which is appropriate for regression tasks and is minimized during training.

Regularization Strategies Multiple regularization strategies were employed to prevent overfitting and improve generalization.

- **L2 Weight Decay** : Applied to the kernel weights of all dense layers with a factor of 0.001 to prevent overfitting by penalizing large weights.

- **Dropout** : A dropout rate of 0.2 is applied between the hidden layers, and a lower rate (0.1) is applied to the input layer and the final fine-tuning layer to prevent over-reliance on specific neurons.
- **Early Stopping** : Training terminates if the validation loss) does not improve for 15 consecutive epochs. The best weights corresponding to the lowest validation loss are restored.
- **Reduce LR On Plateau** : The learning rate is dynamically halved (factor=0.5) if the validation loss plateaus for 8 epochs, ensuring the model can escape local minima.

The DBN was evaluated using three complementary metrics to capture the different aspects of forecasting accuracy. MAPE, MAE and RMSE were used to evaluate and compare the DBN with the other models. These evaluations were performed on inverse transformed predictions to ensure the validation occurred in the original load scale rather than the normalized space used for training. Performance was also assessed separately on both training and test sets to monitor potential overfitting and evaluate generalization capability.

3.4.3 Long Short Term Memory

LSTM is a specialized type of RNN that is particularly effective for time series data prediction [9]1. The LSTM networks are designed to overcome long-term dependency issues such as vanishing and exploding gradients that can occur when processing long data sequences [20]. The LSTM was chosen for its effectiveness is overcoming long term dependency.

The LSTM is versatile and also potentially produces more accurate solutions than statistical methods. Since our data is time series data and any prediction thereof should consider past data points to make an accurate prediction make this model a top choice for evaluation.

LSTM Theoretical Background

AN RNN Background is essential to understand how an LSTM model functions. An RNN is a model that utilizes past sets of information to make a prediction of the

present. They achieve this feat by using a feedback loop. The RNN tracks context or previous data on a hidden state at each time step [56]. This hidden state then works as the memory of the system. Imagine A is a single unit of the RNN receiving input X_t , A will have a hidden state H_t that will store information about the previous input X_{t-1} and it will output Y_t. Image 3.5 adapted from [57] shows how the single unit looks like.

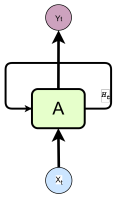

Figure 3.5: A single unit of a recurrent neural network

If the single unit in figure 3.5 is unrolled we will end up with a simple RNN as shown in 3.6 as adapted from [57].

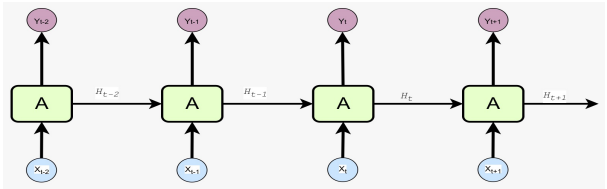

Figure 3.6: An unrolled rnn

An LSTM uses a similar mechanism to remember past data. It introduces gates which act as a control mechanism for the what data to keep and discard as we process information. The LSTM has four main components the cell state, forget gate, input gate and output gate. These control the full mechanism of this neural network.

Cell State

The cell state is a block of information that propagates all the units of the LSTM. This unit contains the data from previous states that will be used as context in future data cells.

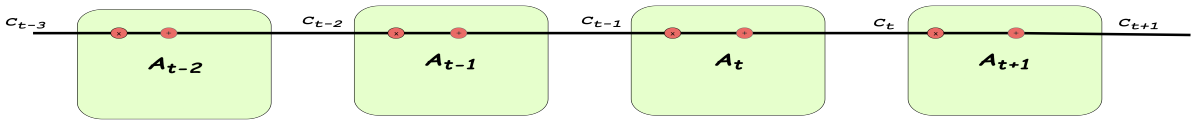

Figure 3.7: Cell state propagating through different units of the LSTM carrying data from previous states to future units

The image 3.7 above shows the cell state C_t propagating through the basic unit of an LSTM A_t. In each of these units through an additive and multiplicative process data is added and removed from the cell state.

Forget gate The cell state contains previous information however there needs to be a way to determine what information is important. The forget gate decides what information to discard from the previous cell state C_{t-1} [34]. During back-propagation the forget gate plays a crucial role. If it learns that certain information is important and should be kept for long, the forget gates activation function will be close to 1 and 0 if the information is not very important [34]. Equation 23 below adapted from [57] shows how the forget gate works.

$$f_t = \sigma\Big(W_f \cdot [h_{t-1}, x_t] + b_f\Big) \tag{23}$$

- f_t: Forget gate activation vector at time step t (values between 0 and 1).
- σ: Sigmoid activation function.
- W_f: Weight matrix for the forget gate.
- $[h_{t-1}, x_t]$: Concatenation of the previous hidden state h_{t-1} and the current input x_t.
- b_f: Bias vector for the forget gate.

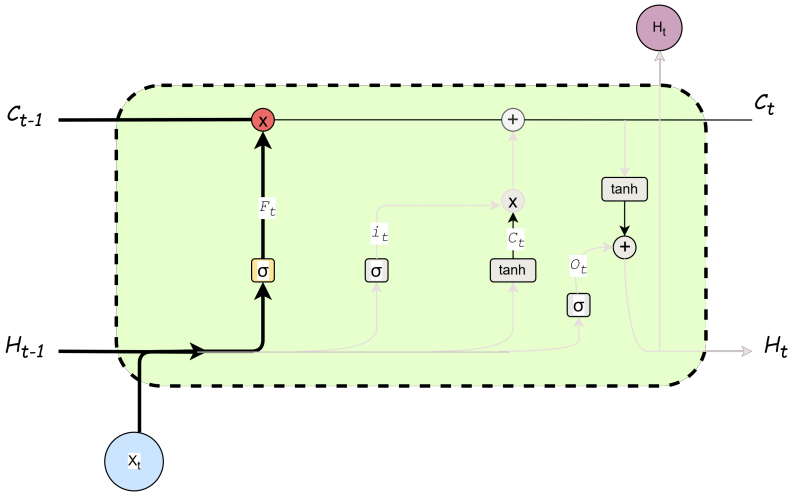

Figure 3.8: Forget gate of an LSTM

Figure 3.8 adapted from [57] shows how the hidden state h_{t-1} and input x_t concatenate together. The concatenated hidden state and input are multiplied with a weight W_f

associated with the input.This sigmoid function also has a bias input vector b_f that is added to the weighted hidden state and input . The two are passed through a sigmoid function using the ReLU activation function giving a number between 0 and 1 as the output. Finally the function f_t is multiplied with c_{t-1} to determine if we add or subtract from the cell state.

$$C_{t-1} * f_t = 0..... \text{ if} f_t = 0 \text{ (forget everything)} \tag{24.a}$$

$$C_{t-1} * f_t = 1..... \text{ if} f_t = 1 \text{ (forget nothing)} \tag{24.b}$$

Equation 24.a and 24.b adapted from [56] shows how the forget gate will be applied to the cell state through the multiplicative process.

Input Gate The input gate determines what new information from the current input and previous hidden state should be stored in the current cell state [9]. The input gate quantifies the importance of new data carried in the input [56]. The input layer is split into two parts, the output of the input gate i_t and a candidate cell state $\tilde{C}_t$ used to update the global cell state. $\tilde{C}_t$ allows retaining of important information and effectively updating the global cell state [34].

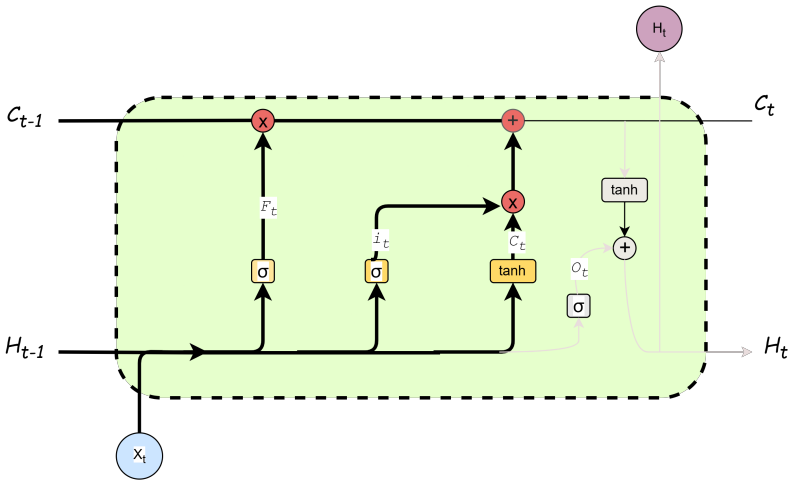

Figure 3.9: An LSTM unit with the forget and input gate

Figure 3.9 shows how the forget and input gates. Below are the equations that determine

the functionality of the input gate.

$$i_t = \sigma\Big(W_i \cdot [h_{t-1}, x_t] + b_i\Big) \tag{25}$$

$$\tilde{C}_t = \tanh\Big(W_c \cdot [h_{t-1}, x_t] + b_c\Big) \tag{26}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{27}$$

- i_t: Input gate output at time step t.
- σ: Sigmoid activation function.
- W_i, W_c: Weight matrices for the input gate and candidate cell state.
- b_i, b_c: Bias vectors for the input gate and candidate cell state.
- $\tilde{C}_t$: Candidate cell state (potential new memory).
- tanh: Hyperbolic tangent activation function.
- C_t: Updated cell state at time step t.
- f_t: Forget gate activation vector at time step t.
- C_{t-1}: Previous cell state.
- $\odot$: Element-wise product.

As shown in figure 3.9 the candidate cell state and the input output gate are combined through an element wise multiplication. The global cell state is also updated by using an additive operation of the forget gate output and the output of the multiplication in the input gate.

Output Gate The output gate determines what information from the cell state should be passed onto the hidden state [34]. The hidden state acts as the output of the unit of processing and will also be used in the following units of processing.

Figure 3.10 shows the complete unit of processing with the output gate. The hidden state takes a sigmoid processed concatenated h_{t-1} and x_t with a tanh processed C_t the output

will be through a multiplicative output.

$$o_t = \sigma\Big(W_o \cdot [h_{t-1}, x_t] + b_o\Big) \tag{28}$$

$$h_t = o_t \odot \tanh(C_t) \tag{29}$$

- o_t: Output gate activation vector at time step t.
- W_o: Weight matrix for the output gate.
- b_o: Bias vector for the output gate.
- h_t: Hidden state at time step t (output of the LSTM block).
- C_t: Updated cell state at time step t.
- tanh: Hyperbolic tangent activation function.

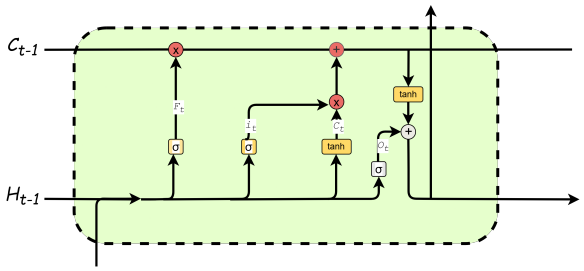

Figure 3.10: Complete LSTM unit with forget, input and output gates

Equation 28 shows the combination for the output and 29 shows the output of the LSTM unit.

A combination of the three gates and their interaction of the cell state form a single unit for an LSTM network. Figure 3.11 shows multiple units of an LSTM and how they relate to each other in a neural network. The LSTM model uses the power of remembering

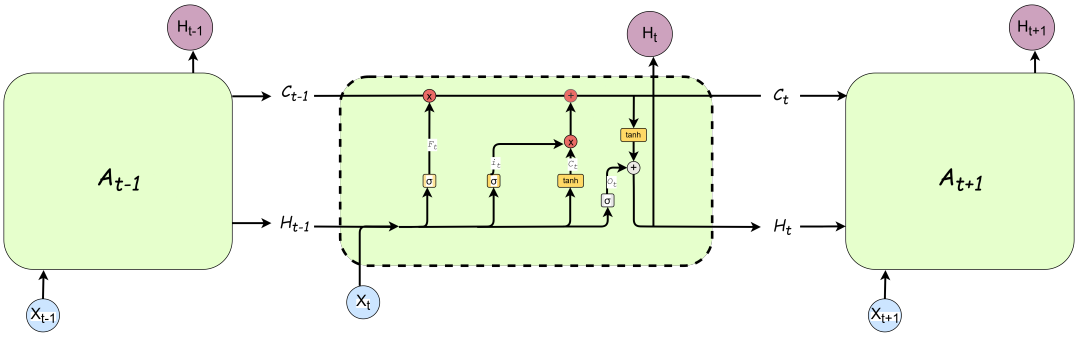

Figure 3.11: full lstm

past data and filtering out important and unnecessary data to the given task to make predictions. In STLF a model capable of finding the important data points in past data to predict for a future data point is crucial. The LSTM model perfects remembrance through its cell state and gate system making it effective for STLF.

LSTM Architecture for STLF

The architecture for the LSTM model used in this research used tensorflow and scikitlearn frameworks to build and train the models. The model followed the same data preprocessing procedures mentioned in section 3.2.3. In addition to the data pre-processing steps a lag feature was added for the STLF. LSTM requires past data to make an accurate prediction and by adding lag features to the data helps the model train with context of the past data. The lag features included the past demand values at 1-hour, 24-hours and 168-hours intervals. Introduction of lag features ensured that the model had access tp both short term and long term temporal dependencies. The dataset was further split into 80% training and 20% testing subsets, to monitor the models generalization.

Sequence Formation

The input of the LSTM model was reshaped into sequences of historical observation to future forecasts.

- **Input sequence length** : 168 hours (7 days)
- **Forecast Horizon** : 24 hours (1 day)

Each training instance therefore contained 168 past observations of all features as input and the next 24-hours of demand as the target output. All the rows containing missing values introduced by lag and rolling operations were removed.

Model Architecture

The proposed LSTM model was created using a stacked architecture to capture short and long term dependencies in the time series data. The architecture comprised:

- **LSTM Layer 1**: 128 units followed by batch normalization and 20% dropout.

- **LSTM Layer 2**: 64 units with Batch Normalization and 20% Dropout.
- **LSTM Layer 3**: 32 units with Batch Normalization and 10% Dropout.
- **Dense Layer**: 50 neurons with ReLU activation and 10% Dropout.
- **Output Layer**: Linear activation with 24 neurons, corresponding to the 24-hour forecast horizon.

The model was compiled using the Adam optimizer with a learning rate of 0.001 and trained to minimize the MSE loss function. The MAE was tracked as a secondary performance metric.

Model Training

The model was trained on the prepared training set with followed parameters.

- **Epochs**: 50-100
- **Batch Size**: 32
- **Validation split** : 0.2

In order to better convergence and prevent overfitting of the model we used call back methods.

- **EarlyStopping** with a 15 epochs stopping patience and best model restoration.
- **ReduceLROnPlateau** to dynamically reduce the learning rate upon validation loss stagnation.
- **ModelCheckpoint** to save the model automatically with the lowest validation loss.

Training progress was monitored through loss and MAE curves for training and validations sets. The model was evaluated using the evaluation metrics mentioned in section 3.4.1.

Flowchart 4 in appendix A shows a clear methodology followed to make the LSTM model. This methodology ensured a systematic approach to implementing an LSTM-based forecasting framework, emphasizing robust feature engineering, careful data preprocessing, and a well-regularized deep learning model architecture optimized for temporal sequence learning.

3.4.4 Hybrid Model : CNN-LSTM

A CNN-LSTM hybrid model is a deep learning architecture designed to combine the strengths of Convolutional Neural Networks (CNN) and LSTM networks, typically to enhance forecasting accuracy and handle complex, non-linear time series data, such as electrical load data sequences [34]. The model takes advantage of the CNNs capability of feature extraction and the LSTMs ability to capability to extract sequence patterns and long-short term dependencies [58]. The LSTM theoretical background was covered in section 3.4.3, therefore we will only look at the theoretical background of a CNN.

CNN Theoretical Background

A CNN is considered the most representative neural network, this is because of its inspiration by biological nervous systems such as the brain [59]. A CNN is composed of neurons that self optimize through learning [59].Their architecture is formed by stacking three primary types of layers:

Convolutional Layer : This is the fundamental component that perform feature extraction using learnable kernels or filters [60]. The kernels are usually small spatial dimensions but extend themselves across their entire depth of the input volume [61]. The kernels themselves are optimizable feature extractors, containing sets of learnable parameters [60].

The layer convolves each filter across the dimensions of the input volume. This is a specialized type of linear operation. As the kernel glides over the input, the scalar product between the kernel's weights and the local region of the input volume is calculated and summed to obtain a single output value.

Activation Layer with Rectified linear Unit layer is usually applied to every element after the convolution function [59].The purpose of the ReLU is to introduce non-linearity into the network, which is essential because of the highly non linear format of the data CNN's process [62]. The ReLU equation 30 as adapted from [62].

$$f_t = \max(0, x) \tag{30}$$

Pooling Layers aim to gradually reduce the dimensionality of the representation, subsequently lowering the number of parameters and the model's computational complexity [63]. In simple terms it reduces the size of the feature maps while keeping the most important information. The most common type of pooling is *Max Pooling*, which takes the maximum value from each window of the output which in turn helps in reduction of computation and overfitting.

Fully Connected Layer is the layer that performs the classification task in a CNN [64]. This will be based on the features that are extracted in the pooling and convolutional layers. This layer performs as a standard neural network, producing class scores from activation [59].

Figure 3.12 is the architecture of a CNN as adapted from [59].

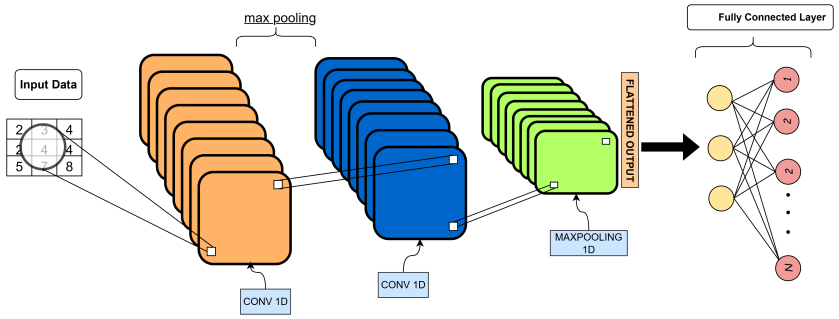

Figure 3.12: An illustration of the CNN model

CNN-LSTM Architecture and Methodology

Data Preparation and Framing

This model uses the continuous_dataset.csv converted into pandas, this is to maintaining uniformity across the different models tested. The data is processed following the steps mentioned in section 3.2.3. Adding onto the already extensive data processing a *lookback window* of 168 hours was used to predict the next 24-hours which is the forecast horizon. This means that each input consisted of 168 hourly observations while the corresponding label contained the next 24 hourly demand values.

Constructing Multi Step Time series

In order to model multi hour ahead forecast, a sliding window approach was used to construct the input output pairs. For every time step t an input matrix

$$X_t = [x_{t-167}, x_{t-166}, ..., x_t]$$

was created to predict the target vector

$$y_t = [y_{t+1}, y_{t+2}, ..., y_{t+24}]$$

corresponding to the next 24 hours.This process generated overlapping input–output pairs that captured both short-term fluctuations and weekly seasonal patterns. The resulting sequences were reshaped into the three-dimensional format *(samples,timesteps,features)* required for deep learning models in TensorFlow.

CNN-LSTM Model Architecture

The proposed model combines a CNN and LSTM as implemented by [9]. The CNN layers act as feature extractors while the LSTM capture the longer term temporal relationships. The dense layers are used to integrate the learned features and generate multi-step forecasts for the next 24 hours.

- **CNN Block**
  - 64 filter Layer 1-D Convolutional layer
  - 32 filter Layer 1-D Convolutional layer
  - Kernel size = 3 , Padding = 3 (for temporal dimension preservation)
  - ReLU Activation
  - MaxPooling 1D layer
  - Dropout rate = 0.2 (enhances generalization)
- **LSTM Block**
  - 50 unit Layer 1 for hierarchical feature extraction
  - 50 unit Layer 2 for sequential output and passes its representation to dense layers.
  - Dropout rate = 0.2 also used between layers for enhanced generalization.

- **Dense Layers (Fully Connected Block)**
  - 64 neuron Dense layers with RelU activation
  - Dropout layer for reduction of overfitting
  - 32 neuron Dense layer
  - 24 neuron Output layer , with each neuron corresponding to a forecasted hour.

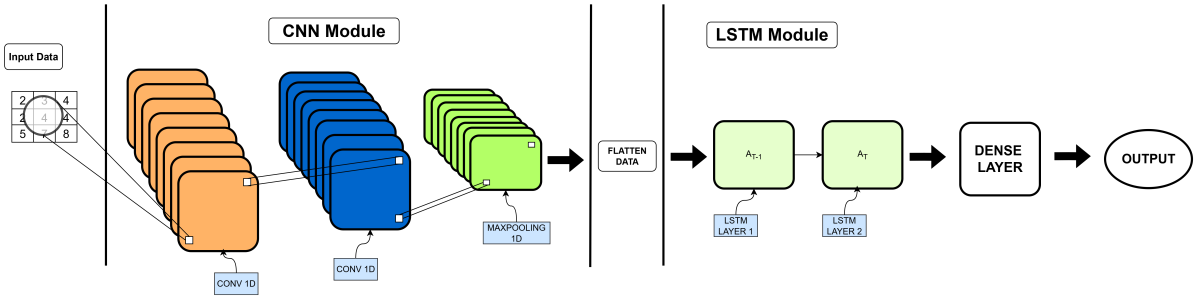

Figure 3.13: Illustration of the CNN-LSTM hybrid model with 2 CNN layers and 2 LSTM layer and a dense layer that produces the output

The model was compiled with the Adam Optimizer with a learning rate of 0.001. MAE, MAPE, MSE were used as performance metrics to evaluate the model. Figure 3.13 illustrates the cnn-lstm hybrid model as adapted from [9].

Model Training and Validation

The CNN-LSTM model was trained on an 80/20 split train/test split. Training was run for 100 epochs with a batch size of 32. We use two callback functions to improve performance.

- **EarlyStopping:** Monitored validation loss and halted training when no improvement was observed for 15 consecutive epochs, restoring the best weights.
- **ReduceLROnPlateau:** Reduced the learning rate by a factor of 0.5 if validation loss plateaued for five epochs, preventing overfitting and ensuring stable convergence.

The training and validation losses were plotted to monitor model convergence and ensure that the model generalized well to unseen data. The callback functions helped in ensuring that our computational resources- are not wasted during the training of the models.

Chapter 4

Results

These are the results I found from my investigation.

Present your results in a suitable format using tables and graphs where necessary. Remember to refer to them in text and caption them properly.

4.1 Simulation Results

4.2 Experimental Results

Chapter 5

Discussion

Here is what the results mean and how they tie to existing literature...

Discuss the relevance of your results and how they fit into the theoretical work you described in your literature review.

Chapter 6

Conclusions

These are the conclusions from the investivation and how the investigation changes things in this field or contributes to current knowledge...

Draw suitable and intelligent conclusions from your results and subsequent discussion.

Chapter 7

Recommendations

Make sensible recommendations for further work.

Bibliography

[1] R. BURDETT-GARDINER, "The great british kettle surge." https://www.renewableenergyhub.co.uk/blog/the-great-british-kettle-surge, 2023. Accessed: 2025-08-05.

[2] Y. Dong, Z. Dong, T. Zhao, Z. Li, and Z. Ding, "Short term load forecasting with markovian switching distributed deep belief networks," *International Journal of Electrical Power & Energy Systems*, vol. 130, p. 106942, 2021.

[3] A. McGrath, "What is load forecasting?." https://www.ibm.com/think/topics/load-forecasting, 2024. Accessed: 2025-08-05.

[4] S. Tzafestas and E. Tzafestas, "Computational intelligence techniques for short-term electric load forecasting," *Journal of Intelligent and Robotic Systems*, vol. 31, no. 1, pp. 7–68, 2001.

[5] N. Wang and Z. Li, "Short term power load forecasting based on bes-vmd and cnn-bi-lstm method with error correction," *Frontiers in Energy Research*, vol. 10, p. 1076529, 2023.

[6] L. Han, Y. Peng, Y. Li, B. Yong, Q. Zhou, and L. Shu, "Enhanced deep networks for short-term and medium-term load forecasting," *IEEE Access*, vol. 7, pp. 4045–4055, 2018.

[7] J. Shohan, M. Faruque, and S. Foo, "Forecasting of electric load using a hybrid lstm–neural prophet model. energies 2022, 15, 2158," 2022.

[8] L. Wu, C. Kong, X. Hao, and W. Chen, "A short-term load forecasting method based on gru-cnn hybrid neural network model," *Mathematical problems in engineering*, vol. 2020, no. 1, p. 1428104, 2020.

[9] S. H. Rafi, S. R. Deeba, E. Hossain, *et al.*, "A short-term load forecasting method using integrated cnn and lstm network," *IEEE access*, vol. 9, pp. 32436–32448, 2021.

[10] C. Tarmanini, N. Sarma, C. Gezegin, and O. Ozgonenel, "Short term load forecasting based on arima and ann approaches," *Energy Reports*, vol. 9, pp. 550–557, 2023.

[11] O. T. Tshipata, D. T. Kazumba, P. S. Nzakuna, V. Paciello, and A. K. Lusala, "Multi-horizon short-term electrical load forecasting: a comparative analysis of statistical models and deep neural networks," in *2024 IEEE International Symposium on Measurements & Networking (M&N)*, pp. 1–6, IEEE, 2024.

[12] C. Wang, Y. Zhou, Q. Wen, and Y. Wang, "Improving load forecasting performance via sample reweighting," *IEEE Transactions on Smart Grid*, vol. 14, no. 4, pp. 3317–3320, 2023.

[13] A. Rusina, T. Osgonbaatar, and P. Matrenin, "Short-term load forecasting using statistical methods for the central power system of mongolia," *2022 IEEE International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, pp. 2030–2035, 2022.

[14] S. Gochhait and D. Sharma, "Regression model-based short-term load forecasting for load dispatch centre," *Journal of Applied Engineering and Technological Science (JAETS)*, vol. 4, no. 2, pp. 693–710, 2023.

[15] B. S. Vardhan, M. Khedkar, I. Srivastava, P. Thakre, and N. D. Bokde, "A comparative analysis of hyperparameter tuned stochastic short term load forecasting for power system operator," *Energies*, vol. 16, no. 3, p. 1243, 2023.

[16] B. Dhaval and A. Deshpande, "Short-term load forecasting with using multiple linear regression," *International Journal of Electrical and Computer Engineering*, vol. 10, pp. 3911–3917, 2020.

[17] M. Hussien, W. Yehia, and A. B. El-Sisi, "A comparative study of machine learning algorithms for short-term electrical load forecasting," *IJCI. International Journal of Computers and Information*, vol. 8, no. 2, pp. 32–37, 2021.

[18] P. Ramos, N. Santos, and R. Rebelo, "Performance of state space and arima models for consumer retail sales forecasting," *Robotics and computer-integrated manufacturing*, vol. 34, pp. 151–163, 2015.

[19] R. Ahmed, V. Sreeram, Y. Mishra, and M. Arif, "A review and evaluation of the state-of-the-art in pv solar power forecasting: Techniques and optimization," *Renewable and sustainable energy reviews*, vol. 124, p. 109792, 2020.

[20] P. Boopathy, M. Liyanage, N. Deepa, M. Velavali, S. Reddy, P. K. R. Maddikunta, N. Khare, T. R. Gadekallu, W.-J. Hwang, and Q.-V. Pham, "Deep learning for

intelligent demand response and smart grids: A comprehensive survey," *Computer science review*, vol. 51, p. 100617, 2024.

[21] J. F. Rendon-Sanchez and L. M. de Menezes, "Structural combination of seasonal exponential smoothing forecasts applied to load forecasting," *European Journal of Operational Research*, vol. 275, no. 3, pp. 916–924, 2019.

[22] R. Wang, J. Wang, and Y. Xu, "A novel combined model based on hybrid optimization algorithm for electrical load forecasting," *Applied Soft Computing*, vol. 82, p. 105548, 2019.

[23] H. Takeda, Y. Tamura, and S. Sato, "Using the ensemble kalman filter for electricity load forecasting and analysis," *Energy*, vol. 104, pp. 184–198, 2016.

[24] V. Revathi, K. Prashant, A. Singla, B. Boddu, A. A. Hameed, S. Kalyani, and K. Pandey, "Short-term load forecasting for virtual power plants using time series analysis and open energy data," in *2025 International Conference on Cognitive Computing in Engineering, Communications, Sciences and Biomedical Health Informatics (IC3ECSBHI)*, pp. 741–746, IEEE, 2025.

[25] S. Dai Haleema, "Short-term load forecasting using statistical methods: A case study on load data," *Int. J. Eng. Res. Technol*, vol. 9, pp. 516–520, 2020.

[26] M. Abbas, Y. Che, S. Maqsood, M. Z. Yousaf, M. Abdullah, W. Khan, S. Khalid, M. Bajaj, and M. Shabaz, "Self-adaptive evolutionary neural networks for high-precision short-term electric load forecasting," *Scientific Reports*, vol. 15, no. 1, p. 21674, 2025.

[27] Y. Wang, J. Wang, G. Zhao, and Y. Dong, "Application of residual modification approach in seasonal arima for electricity demand forecasting: A case study of china," *Energy Policy*, vol. 48, pp. 284–294, 2012.

[28] H. Jiang, Y. Zhang, E. Muljadi, J. J. Zhang, and D. W. Gao, "A short-term and high-resolution distribution system load forecasting approach using support vector regression with hybrid parameters optimization," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3341–3350, 2016.

[29] F. He, J. Zhou, Z.-k. Feng, G. Liu, and Y. Yang, "A hybrid short-term load forecasting model based on variational mode decomposition and long short-term memory networks considering relevant factors with bayesian optimization algorithm," *Applied energy*, vol. 237, pp. 103–116, 2019.

[30] A. Arvanitidis, D. Bargiotas, A. Daskalopulu, V. Laitsos, and L. Tsoukalas, "Enhanced short-term load forecasting using artificial neural networks. energies 2021, 14, 7788," 2021.

[31] Y. Wang, M. Liu, Z. Bao, and S. Zhang, "Short-term load forecasting with multi-source data using gated recurrent unit neural networks," *Energies*, vol. 11, no. 5, p. 1138, 2018.

[32] M. Mohandes, "Support vector machines for short-term electrical load forecasting," *International Journal of Energy Research*, vol. 26, no. 4, pp. 335–345, 2002.

[33] X. Dong, L. Qian, and L. Huang, "Short-term load forecasting in smart grid: A combined cnn and k-means clustering approach," in *2017 IEEE international conference on big data and smart computing (BigComp)*, pp. 119–125, IEEE, 2017.

[34] J. Zhu, J. Yang, X. Cui, M. Peng, and X. Liang, "A novel adaptive adjustment kolmogorov-arnold network for heat load prediction in district heating systems," *Applied Thermal Engineering*, p. 126552, 2025.

[35] B. Ibrahim, L. Rabelo, E. Gutierrez-Franco, and N. Clavijo-Buritica, "Machine learning for short-term load forecasting in smart grids," *Energies*, vol. 15, no. 21, p. 8079, 2022.

[36] A. Moradzadeh, H. Moayyed, S. Zakeri, B. Mohammadi-Ivatloo, and A. P. Aguiar, "Deep learning-assisted short-term load forecasting for sustainable management of energy in microgrid," *Inventions*, vol. 6, no. 1, p. 15, 2021.

[37] X. Kong, C. Li, F. Zheng, and C. Wang, "Improved deep belief network for short-term load forecasting considering demand-side management," *IEEE transactions on power systems*, vol. 35, no. 2, pp. 1531–1538, 2019.

[38] J. Wang, H. Liu, G. Zheng, Y. Li, and S. Yin, "Short-term load forecasting based on outlier correction, decomposition, and ensemble reinforcement learning," *Energies*, vol. 16, no. 11, p. 4401, 2023.

[39] S. Li, J. Wang, H. Zhang, and Y. Liang, "Short-term load forecasting system based on sliding fuzzy granulation and equilibrium optimizer," *Applied Intelligence*, vol. 53, no. 19, pp. 21606–21640, 2023.

[40] W. Huang, Q. Song, and Y. Huang, "Two-stage short-term power load forecasting based on ssa–vmd and feature selection," *Applied Sciences*, vol. 13, no. 11, p. 6845, 2023.

[41] M. U. Danish and K. Grolinger, "Kolmogorov–arnold recurrent network for short term load forecasting across diverse consumers," *Energy Reports*, vol. 13, pp. 713–727, 2025.

[42] X. Tang, Y. Dai, Q. Liu, X. Dang, and J. Xu, "Application of bidirectional recurrent neural network combined with deep belief network in short-term load forecasting," *IEEE Access*, vol. 7, pp. 160660–160670, 2019.

[43] Y. Gao, Y. Hang, and M. Yang, "A cooling load prediction method using improved ceemdan and markov chains correction," *Journal of Building Engineering*, vol. 42, p. 103041, 2021.

[44] S. Kappal *et al.*, "Data normalization using median median absolute deviation mmad based z-score for robust predictions vs. min–max normalization," *London Journal of Research in Science: Natural and Formal*, vol. 19, no. 4, pp. 39–44, 2019.

[45] E. Aguilar Madrid, "Short-term electricity load forecasting (panama case study)." Mendeley Data, V1, 2021.

[46] C.-L. Hor, S. J. Watson, and S. Majithia, "Analyzing the impact of weather variables on monthly electricity demand," *IEEE transactions on power systems*, vol. 20, no. 4, pp. 2078–2085, 2005.

[47] A. Vidhya, "What is feature scaling and why is it important?," 2020. Accessed: 2025-09-20.

[48] E. Ostertagová and O. Ostertag, "The simple exponential smoothing model," in *The 4th International Conference on modelling of mechanical and mechatronic systems, Technical University of Košice, Slovak Republic, Proceedings of Conference*, pp. 380–384, 2011.

[49] N. e-Handbook of Statistical Methods, "Double exponential smoothing." `https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc433.htm`, 2025. Accessed: 2025-09-23.

[50] J. W. Taylor, "Exponential smoothing with a damped multiplicative trend," *International journal of Forecasting*, vol. 19, no. 4, pp. 715–725, 2003.

[51] Statsmodels Developers, "statsmodels.tsa.holtwinters.ExponentialSmoothing documentation." Web documentation, 2025. Accessed: 24 September 2025.

[52] B. Zhang, X. Xu, H. Xing, and Y. Li, "A deep learning based framework for power demand forecasting with deep belief networks," in *2017 18th international conference on parallel and distributed computing, applications and technologies (PDCAT)*, pp. 191–195, IEEE, 2017.

[53] X. Zhang, R. Wang, T. Zhang, and Y. Zha, "Short-term load forecasting based on a improved deep belief network," in *2016 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*, pp. 339–342, IEEE, 2016.

[54] GeeksforGeeks, "Deep belief network (dbn) in deep learning." `https://www.geeksforgeeks.org/deep-learning/deep-belief-network-dbn-in-deep-learning/`, [2023]. Accessed: [29 Sept. 2025].

[55] GeeksforGeeks, "Restricted boltzmann machine." `https://www.geeksforgeeks.org/machine-learning/restricted-boltzmann-machine/`, [2023]. Accessed: [29 Sept. 2025].

[56] C. Stryker, "What is a recurrent neural network (rnn)?." `https://www.ibm.com/think/topics/recurrent-neural-networks`, —. Accessed: 2025-09-29.

[57] C. Olah, "Understanding lstm networks." `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, Aug.27 2015. Accessed: 2025-09-29.

[58] M. Alhussein, K. Aurangzeb, and S. I. Haider, "Hybrid cnn-lstm model for short-term individual household load forecasting," *Ieee Access*, vol. 8, pp. 180544–180557, 2020.

[59] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[60] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[61] A. Ajit, K. Acharya, and A. Samanta, "A review of convolutional neural networks," in *2020 international conference on emerging trends in information technology and engineering (ic-ETITE)*, pp. 1–5, IEEE, 2020.

[62] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[63] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021.

[64] IBM, "What are convolutional neural networks?." IBM Think, 2025. Accessed: 2025-10-08.

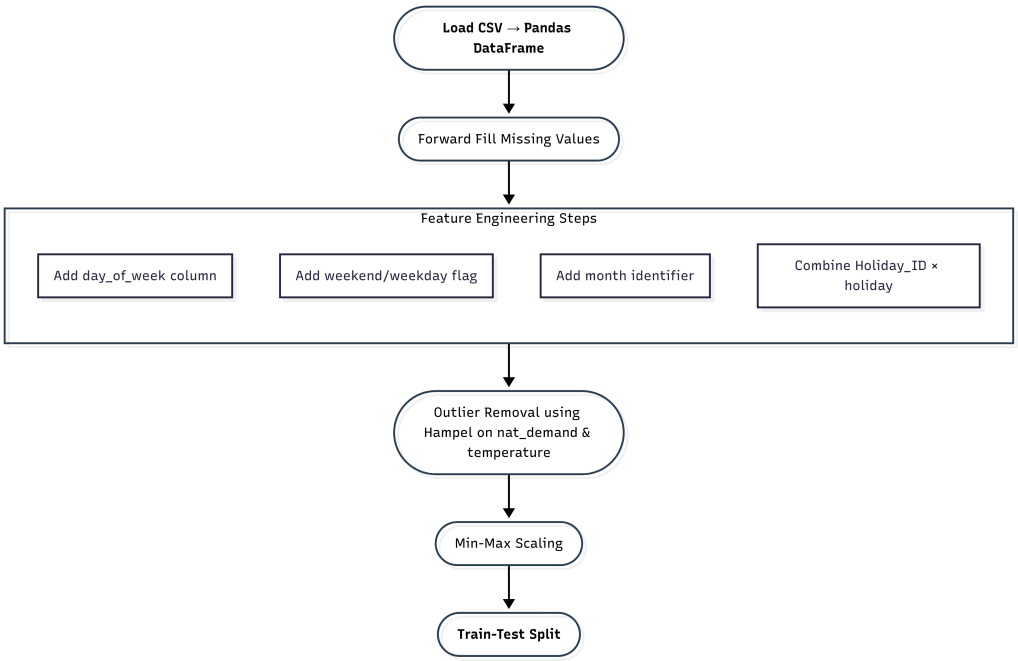

Figure 1: The data preprocessing steps taken to process the data for model training

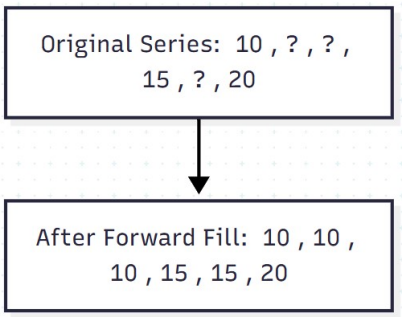

Figure 2: Forward Fill algorithm demonstration

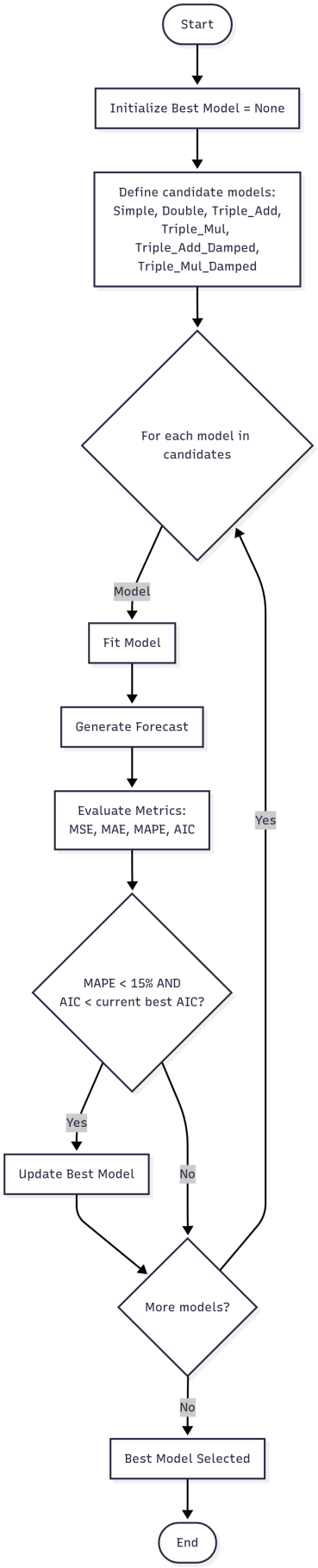

Figure 3: Exponential smoothing model selection design choices.

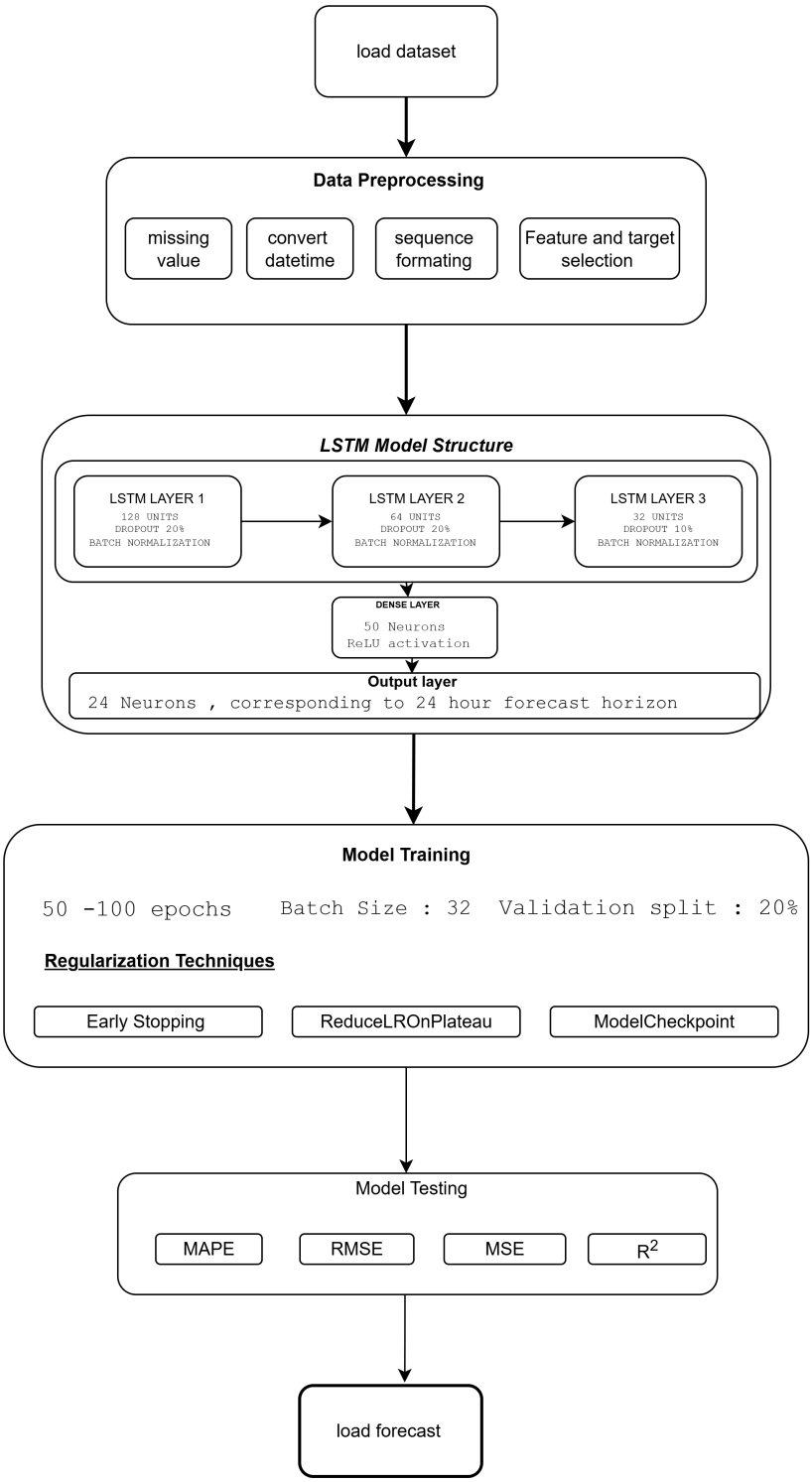

Figure 4: full flowchart of the lstm model

Appendix A

Addenda

A.1 Ethics Forms