Electrical and Computer Engineering
University of Thessaly (UTH)

ECE461 - Machine Learning (ML)

Fall Semester — Educational year 2024-2025

Final Project

Time Series Forecasting for household devices

Themistoklis Proko - AEM: 03301

Georgios Kapakos - AEM: 03165

Eleni Athanailidi - AEM: 03453

Instructor: **Ilias Houstis**

Submission Date: *January 20, 2025*

Contents

Abstract

This research project employs various methods to predict the electrical power consumption (Wh) of a house. To forecast the load demand for the next month, a wide range of models has been utilized. The current work is a continuation of the study titled "Data-driven prediction models of energy use of appliances in a low-energy house," with pre-processing primarily based on the principles presented in that paper. Firstly, a feature elimination method was implemented to remove less important features, aiming to improve forecasting performance. Secondly, the Interquartile Range (IQR) algorithm was applied for outlier removal; it was observed that outliers had a significant influence on the predicted output. Lastly, feature engineering techniques, such as rolling means and lag features, were employed. Based on the performance of the algorithms, these techniques resulted in substantial improvements. The evaluation metrics for each model vary, with the deep learning models achieving the best results (e.g. , LSTM: $R^2 = 0.997$ in testing set).The machine learning models also performed exceptionally well, outperforming those presented in the referenced paper (e.g., XGBoost : $R^2 = 0.77$). Baseline methods, however, did not perform as effectively.

1 Introduction

Load forecasting is a critical process concerning the management and operation of energy systems. It involves predicting the future energy demand over various time horizons—ranging from short-term (hours or days) to long-term (months or years). Accurate load forecasting helps utilities, businesses, and homeowners make informed decisions about energy distribution, infrastructure investment, and operational planning. It also plays a pivotal role in ensuring the reliability, efficiency, and sustainability of energy systems.

In our case, we try to predict the energy consumption of household appliances in a low-energy house. These appliances include electronic devices such as refrigerators, washing machines, dryers, and ovens. According to the paper, approximately 20-30% of the total electricity demand in a household is attributed to these appliances, highlighting the critical need for accurate predictions. Having a high-performing prediction model at our arsenal offers significant advantages in optimizing various aspects of energy management, such as demand-side response, load balancing, and the efficient utilization of renewable energy sources like photovoltaics.

Why do we approach and model those kind of problems using time series? It is because energy consumption data is sequential and influenced by temporal factors.Factors such as time of day, occupancy patterns, and weather conditions all exhibit trends and cycles that affect energy use. By leveraging time series analysis, the study can model these patterns to provide accurate short-term and long-term energy consumption forecasts.This approach is valuable for tasks like predictive control and optimization of energy resources in homes and on the grid.The time series approcach is also valubale for Demand-Side Management cause it enables utilities and homeowners to anticipate peaks in demand, reduce strain on energy infrastructure, and avoid costly energy usage.Lastly the time series analysis aslo helps in integrating renewable energy systems by matching energy production with consumption patterns.

By analyzing historical data and identifying key predictors (e.g., indoor temperature, humidity, and external weather conditions), time series forecasting provides a robust framework to understand and predict energy demand dynamics.

As mentioned earlier, the most accurate predictions in this work were achieved by the deep learning models. These models include LSTM, GRU, TCN, and Transformers, and their per-

formance is significantly better than that of other models in this study. All of them approach the energy demand curve with such high accuracy that the actual and predicted values are almost identical.The Transformer model achieved $R^2 = 0.9466$ on the testing set,the TCN model $R^2 = 0.9503$,while the GRU and LSTM both scored the best results $R^2 = 0.9976$ for each.

2 Preprocessing

Two factors have an enormous impact on achieving accurate results: the selection of a suitable prediction model for the dataset and the pre-processing of the dataset. Of these two, pre-processing is the most critical factor because it is relatively easy to test a limited number of prediction models and identify the best one. In contrast, there are countless pre-processing techniques, and it can often be extremely challenging, to determine the most effective way to pre-process the data for optimal performance.

The pre-processing in this work was initially based on the techniques used in the referenced paper. However, a range of new techniques was introduced in this project to achieve even better evaluation scores.In order to have a better aspect of the problem, the dataset and its features are presented below.

2.1 Dataset Description

The dataset used in this study is the `Energydata_complete.csv`, which contains time-series data representing energy consumption and environmental conditions in a household. The dataset consists of the following key features:

- **Timestamp:** A column providing the time of each recorded observation, sampled at 10-minute intervals.
- **Appliances:** Energy consumption of household appliances (in Wh), representing the dependent variable for prediction tasks.
- **Lights:** Energy consumption of lighting systems (in Wh).
- **Environmental Conditions:** Features such as temperature (in °C), humidity (in %), atmospheric pressure (in mmHg), wind speed (in m/s), visibility (in km), and dew temperature (in °C).Although there is no weather station outside the house, weather data for the nearest airport weather station (Chièvres Airport, Belgium) is merged by date and time in this study to evaluate its impact on the prediction of the energy consumption of appliances.
- **Temperatures in the House:** dataset includes temperature measurements (in °C) for nine different locations within the house, where electrical power is consumed. These measurements are recorded as $T1, T2, T3, T4, T5, T6, T7, T8, T9$
- **Humidity in the House:** Similarly, the dataset records the humidity levels (in %) for the same nine locations in the house. These measurements are represented as $R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$.
- **Random variables**:For testing reasons 2 random variables are introduced (rv1,rv2).

This comprehensive dataset provides a rich foundation for analyzing and predicting energy consumption patterns in residential settings.

A clear view of the Appliances consumption is presented in Figure 1:

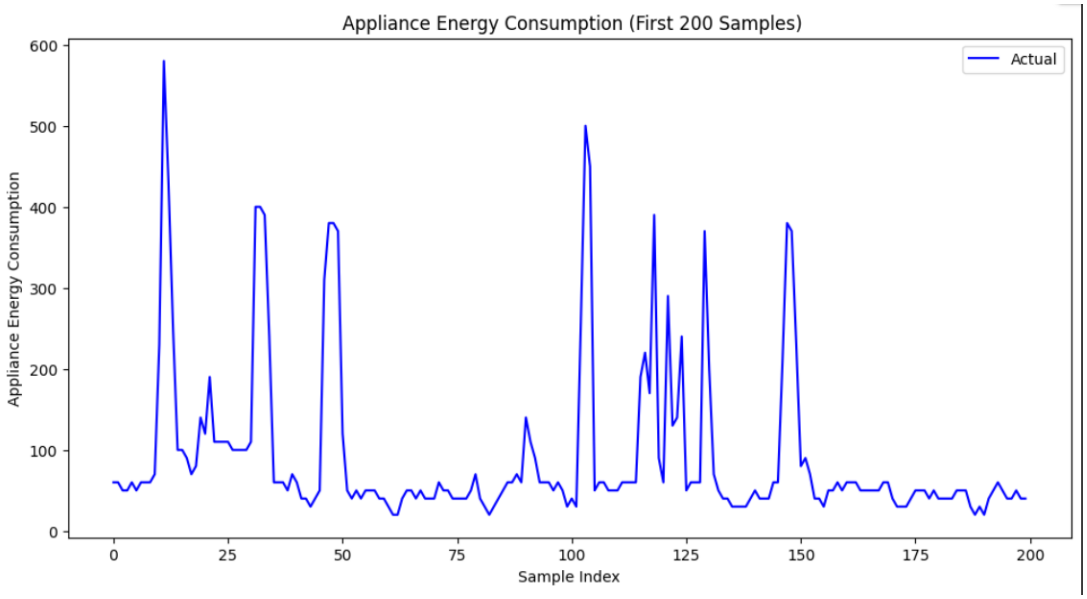

Figure 1: Appliances Energy Consumption for the first 200 samples.

The **Appliances** are plotted on the y-axis, while the x-axis represents the samples. Each sample corresponds to an observation taken every 10 minutes. The total duration of the dataset is approximately 4.5 months, which results in a total of 19,440 samples.

From the **date/time** variable, additional features were generated to enhance the dataset:

- **Number of Seconds from Midnight (NSM):** Represents the number of seconds elapsed since midnight for each observation.
- **Week Status:** A binary feature indicating whether the observation was made on a weekday or a weekend. The column has binary values where 0 represents weekends and 1 represents weekdays.
- **Day of the Week:** One-hot encoding was applied to represent the day of the week, resulting in the creation of seven new binary columns. Each column corresponds to a specific day of the week, with a value of 1 indicating that the observation was made on that day and 0 otherwise.

Based on the referenced paper, we conclude that certain features need to be eliminated from the dataset in order to minimize the loss. The variables to be dropped include 'lights', 'rv1', 'rv2', and 'visibility'.

2.2 ACF Plot

An ACF (Autocorrelation Function) plot shows the correlation of a time series with its own past values at different lags. It helps identify patterns like seasonality or autocorrelation in the data, aiding in model selection for time series analysis. Based on the acf plot we decide to create

the different lags and moving average columns, that will help us improve the predictive power of our model.

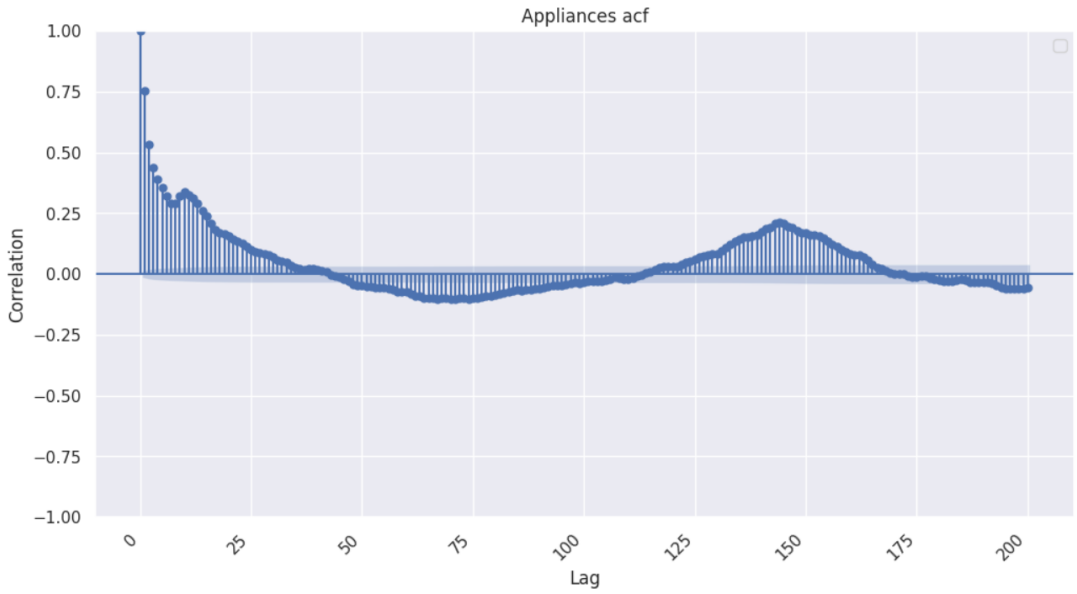

Figure 2: ACF Plot for Appliances

From this plot, we decide to add lags every:

- Day (144): Captures daily patterns or seasonality.
- 2 days (288): Highlights trends or cycles spanning 48 hours.
- 3 days (432): Identifies longer periodic variations over 72 hours.
- 12 hours (72): Useful for detecting half-day cycles.
- 6 hours (36): Captures finer-grained intraday patterns.
- 3 hours (18): Highlights shorter-term fluctuations within a day.
- 2 hours (12): Provides a closer look at very short-term patterns.
- 1 hour (6): Focuses on highly localized trends in the data.

The lag values were chosen based on the observed seasonality and periodic patterns in the data, ensuring that important cycles (daily, multi-day, and hourly variations) are captured effectively.

We also decide to add rolling averages every:

- Approximately every 2 hours (10): Smooths very short-term noise.
- Every 5 hours (30): Balances noise reduction with trend visibility.
- Approximately every 8 hours (50): Highlights intraday trends.
- Every day (144): Captures daily averages for broader insights.
- Every 2 days (288): Highlights longer-term trends while smoothing fluctuations.

The rolling average values were selected to smooth the data at varying granularities, balancing noise reduction with trend visibility across different time scales.

2.3 Feature Engineering Techniques

After eliminating irrelevant features from the dataset, we proceed with two critical feature engineering techniques that greatly enhance the performance of our prediction model: **lag features** and **rolling window** techniques. Both of these techniques capture important temporal dependencies in the data and provide additional context for future predictions.

2.3.1 Lag Features

Lag features are created by shifting the target variable (in this case, `Appliances`) by a specified number of time steps. This allows the model to learn from past observations when making future predictions. By incorporating lagged values, the model can identify patterns and trends that are critical for time-series forecasting.

For example, a lag feature such as `Appliances_lag_144` represents the energy consumption of appliances from 144 time steps (equivalent to 24 hours) ago. These lag features enable the model to understand how past consumption levels influence future energy usage. Including lagged values improves the model's ability to predict future values with greater accuracy, as it leverages the temporal dependencies within the data.

2.3.2 Rolling Window

The rolling window technique involves creating moving averages of the target variable over a specified window of time. These rolling means help smooth out fluctuations in the data, making it easier for the model to capture long-term trends and remove short-term noise.

For instance, features such as `Appliances_rolling_mean_10` represent the average energy consumption of appliances over the last 10 time steps. By using rolling windows, the model learns from smoothed, long-term patterns instead of reacting to every minor fluctuation in the data. This technique is particularly valuable for capturing trends over longer periods and improving the stability and robustness of the predictions.

2.3.3 Importance of Lag Features and Rolling Windows

Both lag features and rolling windows play a significant role in enhancing the prediction performance of the model. Lag features allow the model to recognize patterns in historical data and understand how previous energy consumption values influence future values. Meanwhile, rolling windows help smooth the data, reduce noise, and highlight meaningful trends.

Together, these techniques provide the model with a more robust build, resulting in more accurate and predictions for energy consumption.

2.4 Missing Values Handling

The next step involves handling the missing values in the dataframe. It was observed that the majority of models performed poorly when the data contained `NaN` values. To address this issue, two potential approaches were considered. The first approach involved detecting and dropping rows with missing values, while the second approach involved replacing the missing values with the mean of the corresponding column. The second method was chosen, as dropping rows generally results in a loss of information, which is particularly critical in time series tasks where maintaining the temporal structure of the data is essential.

2.5 Outliers Handling

Handling outliers is a crucial step to ensure that models achieve accurate predictions. In the current project, three different techniques for managing outliers were examined: Z-score, Winsorization, and the Interquartile Range (IQR) method. For each method, any value exceeding the threshold was replaced with the upper bound.

2.5.1 Z-score Method

The Z-score method identifies outliers by measuring how far a data point deviates from the mean in terms of standard deviations. A Z-score is calculated as follows:

$$Z = \frac{x - \mu}{\sigma}$$

where x is the data point, μ is the mean of the dataset, and σ is the standard deviation. A common threshold for detecting outliers is $|Z| > 3$, meaning any data point more than three standard deviations away from the mean is considered an outlier. This method assumes that the data follows a normal distribution, making it less effective for skewed data.

2.5.2 Winsorization

Winsorization is a technique that reduces the effect of extreme values by capping them at a specified percentile. For example, the top 5% of data values might be replaced with the value at the 95th percentile, and the bottom 5% might be replaced with the value at the 5th percentile. This approach preserves the size of the dataset while limiting the influence of outliers, making it useful when dealing with extreme values without discarding data points.

2.5.3 Interquartile Range (IQR) Method

The IQR method identifies outliers by examining the range between the first quartile (Q1) and the third quartile (Q3). The IQR is defined as:

$$IQR = Q3 - Q1$$

Outliers are defined as data points that fall outside the range:

$$[Q1 - 1.5 \times IQR,\ Q3 + 1.5 \times IQR]$$

This method is robust to skewed data and extreme values, making it widely applicable. It removes points that are unusually far from the central distribution, improving the accuracy of models by focusing on representative data.

2.5.4 Conclusion

Each of these techniques has its strengths and weaknesses, and the choice of method depends on the nature of the dataset and the goals of the analysis. In this project, these techniques were evaluated to ensure that the data used for modeling was free from extreme outliers that could negatively impact predictions. After extensive testing with different thresholds, it was concluded that the highest prediction accuracy was achieved using the **IQR** method with the threshold value set to 1. Consequently, this technique was chosen for implementation across all problems in this study.

.

2.6 Scaling

The final step before proceeding with the model implementation is scaling the data. Two different scaling techniques were examined: standard scaling and Min-Max scaling. Both techniques were applied to evaluate their impact on the performance metrics of the model (XGBoost). It was observed that the Min-Max scaler produced better results for this task, and therefore, it was selected as the scaling method for subsequent steps.

3 Forecasting Models

Time series forecasting is not a new concept; it has long been a fundamental task in the fields of engineering and energy management. However, in the last decade, the advancement of computational power, coupled with significant progress in data science, has shifted the scientific community's focus toward more efficient approaches for time series forecasting, such as machine learning and deep learning models. These methods offer greater flexibility and improved performance compared to traditional techniques. Forecasting models can generally be classified into four main categories: Baseline Models, Traditional Models, Machine Learning Models, and Deep Learning Models. In this section, each method is presented in detail to help the reader gain a deeper understanding of their concepts and applications.

3.1 Baseline Models

Baseline model in machine learning is a simple, often naïve model used as a reference point to compare the performance of more complex models. Baseline models are useful for benchmarking, early-stage testing, and identifying potential issues like overfitting. They are not very accurate, but they provide a point of comparison between them and more complex models, if the more complex models underperform in comparison to the naive models it means that further development in feature engineering, model choice, or data pre-processing.

3.1.1 Naive Model

A naive model is a type of forecasting model that makes predictions based solely on the most recent value or observation, without considering any complex patterns, trends, or other features in the data.

$$\hat{y}_{t+1} = y_t$$

where:

- y_t is the observed value at time t,
- $\hat{y}_{t+1}$ is the predicted value for the next time step.

3.1.2 Moving Average Model

The Moving Average (MA) model is a time series forecasting technique that predicts future values based on the average of past observations over a fixed window of time. The window(lag), can vary in size, and the model computes the average of the data points within this window to make predictions. It can be useful in identifying trends in data that exhibit cyclical or repetitive patterns, helping to mitigate noise by averaging out irregularities.

$$\hat{y}_t = \frac{1}{k} \sum_{i=t-k+1}^{t} y_i$$

where:

- $\hat{y}_t$ is the predicted value for time t,
- y_i is the observed value at time i,
- k is the window size, which determines how many past observations are used for the average.

3.1.3 Seasonal Naive Model

The Seasonal Naive model is an extension of the naive model that takes into account the seasonality in time series data. Instead of simply predicting the next value based on the most recent observation, the seasonal naive model predicts future values based on the corresponding value from the same season or period in the previous cycle. The seasonal naive model is simple and easy to implement, making it a good baseline for seasonally periodic data.

$$\hat{y}_{t+s} = y_t$$

where:

- $\hat{y}_{t+s}$ is the predicted value for time $t + s$,
- y_t is the observed value at time t,
- s is the seasonal period, which represents the number of time steps after which the seasonality repeats (e.g., 12 for monthly data with yearly seasonality).

3.2 Traditional Forecasting Models

Traditional forecasting models are statistical approaches used to analyze and predict future values based on past data. These models are grounded in mathematical principles and assumptions about the underlying data, such as stationarity, seasonality, and linearity. They are widely used for time series analysis and forecasting and often serve as benchmarks for comparing more advanced machine learning models.However, it was observed that their performance during testing was poor; therefore, they were not further considered in this work.

3.2.1 ARIMA (AutoRegressive Integrated Moving Average)

ARIMA is a popular and versatile statistical model used for time series forecasting. It combines three key components: autoregression (AR), differencing to ensure stationarity (I), and a moving average (MA). The ARIMA model is defined by three parameters:

- p: The number of lag observations included in the model (autoregressive order).
- d: The number of times the data needs to be differenced to achieve stationarity (integration order).
- q: The size of the moving average window (moving average order).

The general form of the ARIMA model is:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where:

- y_t: The observed value at time t,
- ϕ_i: The coefficients of the autoregressive terms,
- θ_i: The coefficients of the moving average terms,
- ϵ_t: The error term (white noise) at time t,
- c: A constant term.

ARIMA is effective for forecasting univariate time series data with linear patterns. However, it requires the data to be stationary, and additional preprocessing (e.g., differencing or transformation) may be needed to meet this condition.

3.2.2 SARIMA (Seasonal ARIMA)

SARIMA extends the ARIMA model by incorporating seasonal patterns in time series data. It is particularly useful for datasets with periodic fluctuations. SARIMA introduces additional seasonal parameters (P, D, Q, s), where:

- P: The number of seasonal autoregressive terms,
- D: The number of seasonal differences needed to make the data stationary,
- Q: The number of seasonal moving average terms,
- s: The seasonal period (e.g., 12 for monthly data with yearly seasonality).

The SARIMA model is expressed as:

$$SARIMA(p, d, q)(P, D, Q)_s$$

The general form incorporates seasonal terms in addition to the ARIMA structure:

$$y_t = c + \Phi_1 Y_{t-s} + \Phi_2 Y_{t-2s} + \ldots + \Phi_P Y_{t-Ps} + \Theta_1 \epsilon_{t-s} + \Theta_2 \epsilon_{t-2s} + \ldots + \Theta_Q \epsilon_{t-Qs} + \epsilon_t$$

where:

- Φ_i: The seasonal autoregressive coefficients,
- Θ_i: The seasonal moving average coefficients,
- Y_{t-s}: The seasonal lagged values,
- ϵ_t: The error term (white noise),
- c: A constant term.

SARIMA accounts for both short-term and seasonal dependencies, making it a robust choice for datasets with seasonal trends. However, it involves more complex parameter tuning compared to ARIMA, as both non-seasonal and seasonal parameters must be optimized.

Disclaimer: In this project we did not end up using any of these traditional models, as we tried to incorporate them into the code and they did not perform well at all. The result findings from these models was unsignificant and therefore we did not report the times and their accuracies.

3.3 Machine Learning Models

Machine learning models are an alternative approach to load forecasting. They are generally preferred over statistical and baseline methods due to their greater flexibility and superior performance. Additionally, they can process larger and more complex datasets that may not exhibit a linear relationship between input and output and are often characterized by fluctuations and variability. In this work, we implemented three machine learning models: XGBoost, Random Forest, and Decision Trees.

3.3.1 Decision Trees

Decision Trees are one of the simplest and most intuitive machine learning algorithms. They operate by splitting the data into subsets based on specific conditions defined by feature values. The model constructs a tree-like structure, where each internal node represents a decision rule on a feature, each branch represents the outcome of the rule, and each leaf node represents the predicted output. Decision Trees are particularly effective for datasets with non-linear relationships between features and the target variable. However, they are prone to overfitting, especially when the tree becomes too deep. To address this, hyperparameters such as maximum depth and minimum samples per leaf can be tuned.

The splitting of nodes is typically guided by a metric that measures the ”purity” of the split. Common metrics include:

- **Gini Impurity**:

$$G = 1 - \sum_{i=1}^{C} p_i^2$$

  where p_i is the proportion of samples belonging to class i, and C is the total number of classes. The Gini impurity is minimized during splits to create purer child nodes.

- **Entropy** (used in information gain):

$$H = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

  The information gain (IG) for a split is calculated as:

$$IG = H_{\text{parent}} - \sum_{k=1}^{K} \frac{|N_k|}{|N|} H_k$$

  where H_{parent} is the entropy of the parent node, H_k is the entropy of the k-th child node, $|N_k|$ is the number of samples in the k-th child node, and $|N|$ is the total number of samples in the parent node.

- **Mean Squared Error (MSE)** (for regression tasks):

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

3.3.2 Random Forest

Random Forest is an ensemble learning method that builds upon the concept of Decision Trees. Instead of relying on a single tree, Random Forest constructs multiple Decision Trees during training and aggregates their outputs to produce a final prediction. Each tree in the forest is built using a random subset of the data and a random subset of features, introducing diversity and reducing the risk of overfitting.

The Random Forest algorithm operates as follows:

- **Bootstrap Aggregating (Bagging)**: Each tree is trained on a random subset of the data, obtained by sampling with replacement. This is known as bootstrapping. For each tree, the data subset is of size n (the number of training samples), but it may contain duplicates, and some samples may be left out.
- **Random Feature Selection**: During the construction of each tree, a random subset of features is selected at each node to make the best split. This reduces correlation among trees and enhances the model's generalization.

For classification tasks, the final output is determined by majority voting across all trees. If there are T trees, and each tree t predicts a class y_t, the final prediction y_{final} is given by:

$$y_{\text{final}} = \text{mode}(y_1, y_2, \ldots, y_T)$$

For regression tasks, the final output is determined by averaging the predictions from all the trees. If each tree t predicts a value $\hat{y}_t$, the final prediction $\hat{y}_{\text{final}}$ is:

$$\hat{y}_{\text{final}} = \frac{1}{T} \sum_{t=1}^{T} \hat{y}_t$$

Random Forest models are highly robust and can handle large datasets with complex interactions between variables. They are less prone to overfitting compared to individual Decision Trees because the model aggregates the predictions of multiple trees, each trained on a different subset of data.We can easy assume that their performance is often superior to that of a single Decision Tree, but they come at the cost of increased computational complexity.

3.3.3 XGBoost

XGBoost (Extreme Gradient Boosting) is an advanced machine learning algorithm that falls under the category of gradient boosting methods. It builds an ensemble of decision trees, where each tree is constructed sequentially, and each subsequent tree tries to correct the errors made by the previous trees. The final prediction is made by combining the outputs of all the trees in the ensemble. XGBoost has become one of the most popular algorithms for structured/tabular data due to its high performance and scalability.

The key idea behind gradient boosting is to minimize the loss function by adding new trees that correct the errors made by the previous trees. The objective of XGBoost can be formulated as:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \mathcal{L}_i(f(\mathbf{x}_i)) + \sum_{k=1}^{K} \Omega(f_k)$$

Where:

- $\mathcal{L}_i(f(\mathbf{x}_i))$ is the loss function for the i-th training example, where $f(\mathbf{x}_i)$ is the predicted value. - $\Omega(f_k)$ is a regularization term that controls the complexity of the model (e.g., preventing overfitting). It is defined as:

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

Here:

- T is the number of leaves in the tree, - γ is a parameter that controls the complexity of the tree (higher values encourage simpler trees), - λ is a regularization parameter that controls the weight of each leaf.

In XGBoost, the learning process is performed by optimizing the objective function via gradient descent. At each iteration, the model computes the gradient of the loss function and builds a tree that minimizes the error (or gradient) for the next prediction:

$$\hat{y}_{\text{final}} = \sum_{t=1}^{T} f_t(\mathbf{x})$$

Where $f_t(\mathbf{x})$ represents the output of the t-th tree, and T is the total number of trees.

XGBoost improves on traditional gradient boosting by introducing several enhancements:

- **Handling Missing Data**: XGBoost automatically handles missing data by learning the best direction (left or right) for missing values during training.
- **Column Subsampling**: During training, it randomly selects a subset of features to construct each tree, which helps reduce overfitting.
- **Weighted Quantile Sketch**: A faster algorithm for finding the best split points when building the trees.
- **Parallelization**: XGBoost can be parallelized, which speeds up training, especially for large datasets.

XGBoost is known for its speed and accuracy and is particularly effective in high-dimensional feature spaces and structured datasets. However, it requires careful tuning of hyperparameters, such as learning rate, max depth, and the number of estimators, to achieve optimal performance.

3.4 Deep Learning Models

Although Machine Learning models provide essential tools for time series prediction, they cannot compare to the performance and capabilities of Deep Learning models. Deep Learning models are considered the most powerful tools available to the scientific community for time series forecasting because of their ability to capture complex patterns,

model long-term dependencies, and handle large-scale datasets with high variability and non-linear relationships.In this work 3 deep learning models were implemented :
LSTM, GRU, TCN.

For the deep learning models, we did not remove the outliers, as the results showed that the deep learning models performed best when the outliers were not removed. This may be because deep learning models are inherently robust to outliers due to their ability to learn complex, non-linear relationships in the data. Outliers can sometimes provide valuable information about extreme cases or rare patterns, which deep learning models may leverage to improve their performance and generalization.

3.4.1 LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) specifically designed to handle long-term dependencies in sequential data. LSTM addresses the vanishing gradient problem found in traditional RNNs by introducing a memory cell and three gates: the forget gate, input gate, and output gate. These gates control the flow of information, allowing the network to selectively remember or forget past data as required.

The architecture of an LSTM is mathematically represented as follows:

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

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

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

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

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

$$h_t = o_t \odot \tanh(C_t)$$

Hyperparameters such as *batch size*, *number of epochs*, and *dropout rates* significantly impact LSTM performance. Batch size determines the number of samples processed before updating model weights, epochs represent the total iterations over the dataset, and dropout prevents overfitting by randomly deactivating neurons during training.

3.4.2 GRU (Gated Recurrent Unit)

Gated Recurrent Units (GRUs) are a simplified alternative to LSTMs that reduce computational complexity while retaining the ability to model sequential dependencies. GRUs achieve this by merging the input and forget gates into a single *update gate* and incorporating a *reset gate* to control how much of the past information influences the current state.

The mathematical representation of GRUs is as follows:

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

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

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

Compared to LSTMs, GRUs require fewer parameters, making them faster to train. However, they may not always perform better, depending on the complexity of the dataset.

3.4.3 Temporal Convolutional Network (TCN)

In this work, a TCN model was preferred over a CNN.The primary difference between Temporal Convolutional Networks (TCNs) and Convolutional Neural Networks (CNNs) lies in their design focus: TCNs are specifically tailored for sequential data, employing causal convolutions to preserve temporal order and dilation to capture long-range dependencies, whereas CNNs, including 1D-CNNs, are more general-purpose feature extractors and lack the inherent structure needed for strict temporal sequence modeling.

Unlike RNNs, which process data sequentially, TCNs use 1D convolutional layers to capture temporal dependencies in parallel. This approach ensures faster computation and avoids the vanishing gradient problem.

Key features of TCNs include:

- **Causal convolutions:** Ensure that predictions at time t depend only on inputs from time t or earlier.
- **Dilated convolutions:** Introduce gaps between filter applications, allowing the network to capture long-term dependencies without increasing model depth.
- **Residual connections:** Help stabilize the training of deep networks.

  The output of a dilated convolution for input x and kernel w is given by:

$$y(t) = \sum_{i=0}^{k-1} w(i) \cdot x(t - r \cdot i)$$

  Where: - k: Kernel size - r: Dilation factor - x: Input sequence - w: Convolutional weights

  TCNs are particularly effective for time series forecasting tasks that require capturing both short- and long-term patterns.

3.4.4 Importance of Deep Learning Models in Time Series Forecasting

LSTM, GRU, and TCN have proven to be highly effective for time series forecasting tasks. While LSTM and GRU excel at capturing temporal dependencies, TCN offers advantages in computational efficiency and parallelism. These models allow for robust predictions in complex datasets with non-linear relationships, making them essential tools for modern forecasting tasks.

3.5 Transformer Models

Transformers, originally introduced in natural language processing (NLP) tasks, have revolutionized sequential data modeling due to their ability to capture long-range dependencies and process sequences in parallel. Initially designed for machine translation and language modeling, transformers have been adapted for time series analysis, where they have demonstrated superior performance in handling complex temporal patterns compared to traditional models and recurrent neural networks (RNNs).

3.5.1 Why Transformers for Time Series?

Traditional time series models and RNN-based approaches often struggle with:

* **Capturing Long-Range Dependencies:** RNNs and LSTMs rely on sequential processing, which can make learning long-term dependencies challenging due to vanishing gradients.
* **Parallel Processing:** RNNs process data sequentially, limiting their scalability for large datasets.

Transformers address these challenges by employing the self-attention mechanism, which allows them to:

* Directly model relationships between all time steps, regardless of their distance in the sequence.
* Process sequences in parallel, significantly improving computational efficiency.

3.5.2 Transformer Architecture for Time Series

The core components of transformers adapted for time series forecasting include:

Input Layer Time series data x_t is embedded into a higher-dimensional space using learnable embeddings. Positional encoding PE is added to retain the order of the time steps:

$$x_t^{\text{embed}} = E(x_t) + PE(t)$$

where $PE(t)$ is computed as:

$$PE(t, 2i) = \sin\left(\frac{t}{10000^{2i/d}}\right), \quad PE(t, 2i+1) = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

Self-Attention Layer The self-attention mechanism computes the relevance of each time step to every other time step in the sequence. The attention scores are:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where:

* Q: Queries,
* K: Keys,
* V: Values,
* d_k: Dimensionality of the key vectors.

Feed-Forward Layer Each time step is independently processed by a position-wise feed-forward network (FFN):

$$\text{FFN}(x) = \text{ReLU}(W_1 x + b_1)W_2 + b_2$$

Encoder The encoder stack consists of multiple layers of self-attention and feed-forward networks, followed by layer normalization. The output of each layer is:

$$\text{LayerNorm}(x + \text{SubLayer}(x))$$

Decoder For forecasting, the decoder generates predictions $\hat{y}_t$ using the encoder outputs and masked self-attention to ensure that predictions for t depend only on previous time steps:

$$\hat{y}_t = f\left(\text{Decoder}(z, \text{mask})\right)$$

3.5.3 Advantages of Transformers for Time Series

- **Scalability:** Parallel processing enables efficient handling of large datasets.
- **Flexibility:** Can be adapted for multivariate time series and irregularly spaced data.
- **Accuracy:** Captures complex temporal dependencies, outperforming traditional models in many cases.

Transformers have set a new benchmark in time series analysis, particularly for tasks requiring long-term forecasts and modeling intricate patterns.

4 Evaluation and Results

4.1 Evaluation Metrics

In most studies, four metrics are used to evaluate the precision of forecasting: RMSE, MAPE, R^2, and MAE. Each metric highlights different aspects of the error. However, in the most cases the most important metric for describing the predictive power of a model is considered R^2. Below are the descriptions of the four most commonly used metrics:

- **Root Mean Squared Error (RMSE)**: This metric measures the average magnitude of the errors between predicted and actual values. It is calculated as the square root of the average squared differences between prediction and actual values:

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

  A lower RMSE value indicates a better fit to the data.
- **Mean Absolute Percentage Error (MAPE)**: This metric expresses the error as a percentage, making it easier to interpret. It is calculated as the average of the absolute percentage differences between actual and predicted values:

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

While MAPE is widely used, it can be sensitive to very small values of y_i, leading to inflated errors.

* **R-Squared (R^2)**: This metric quantifies the proportion of variance in the target variable that is explained by the model. An R^2 value closer to 1 indicates that the model explains most of the variance, while a value closer to 0 means the model is not explaining much of the variance.

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

R^2 is widely regarded as one of the most important metrics for evaluating the predictive power of a model.

* **Mean Absolute Error (MAE)**: This metric calculates the average magnitude of the errors in the predictions, without considering their direction (positive or negative). It is calculated as:

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

MAE is straightforward to interpret and gives a simple measure of the average error.

4.2 Comparative Analysis

A total number of 10 different models have been utilized in the project in order to predict the future consumption of the appliances.The evaluation results are shown below.

Model	**RMSE**	R^2	**MAE**	**MAPE**
Moving Average	37.248	-0.0175	31.811	53.698%
Naive	65.53	-2.1498	59.204	118.45%
Seasonal Naive	48.51	-0.7257	38.788	62.92%
XGBoost	17.502	0.7753	12.375	19%
Random Forest	18.1339	0.7588	12.56	19.4%
Decision Tree	21.639	0.6565	15.3977	23.27%
LSTM	4.5763	0.9973	3.152	3.93%
GRU	5.33	0.9964	3.4173	4.58%
TCN	20.744	0.945	18.3794	29.28%
Transformers	21.963	0.9505	16.0307	20.60%

Table 1: Evaluation Metrics for Different Models on the Testing Dataset

Disclaimer: For the deep learning models, we selected some of the best performance values, as neural networks inherently produce results that can vary within a range due to their stochastic nature. Consequently, the reported performance reflects an upper bound within the observed range rather than a guaranteed or consistent outcome.

According to Table 1, the baseline models produced terrible results, demonstrating weak predictive power. Thus, they are not suitable for this specific dataset. On

the other hand, machine learning models delivered significantly better results, with small errors between the predicted and actual values. We conclude that their ability to fit the dataset is satisfactory. The best-performing machine learning model was XGBoost ($R^2 = 0.7753$).

However, there is no comparison between machine learning models and deep learning models in terms of predictive power. Deep learning models exhibited remarkable accuracy, achieving the most precise results. Among them, the LSTM model emerged as the best performer with $R^2 = 0.9973$, showcasing its superior ability to capture the complexities of the dataset.

The Figure 2 compares the R^2 values for each model. Based on the results, the best-performing models, in order, are: LSTM, GRU, Transformers, TCN, XGBoost, Random Forest, Decision Trees, Seasonal Naive, Moving Average, and Naive.

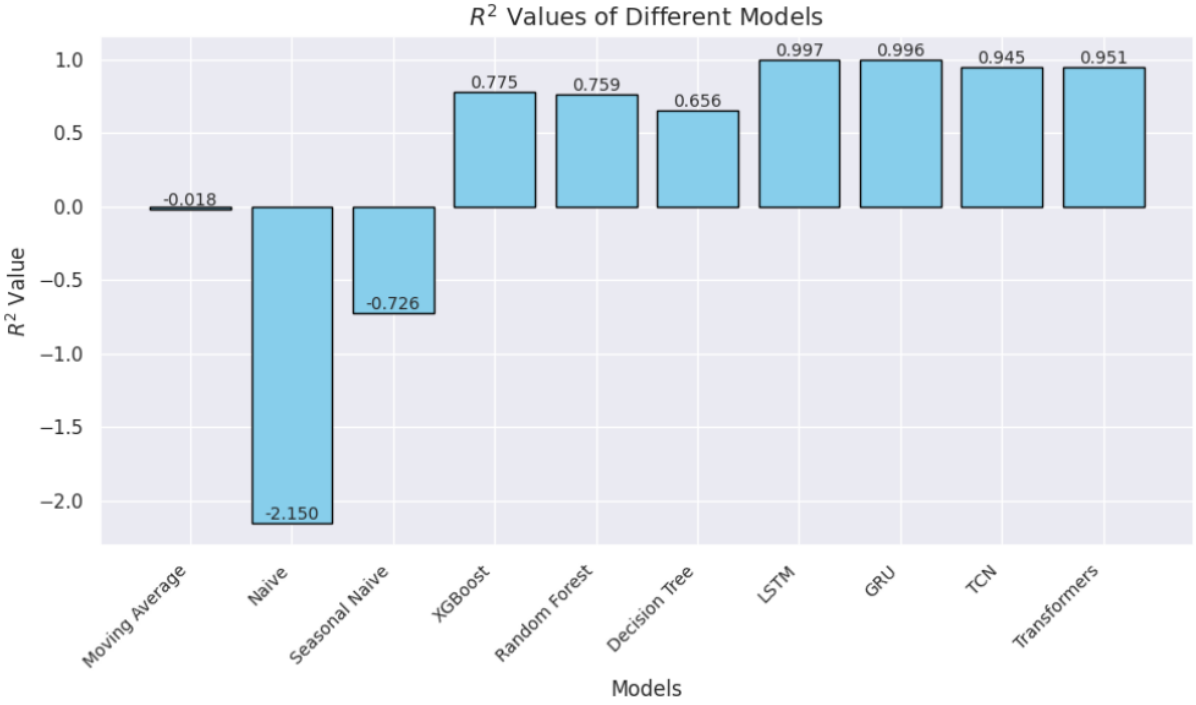

Figure 3: R^2 value compassion for each model.

4.3 Visualization

Below, we present the graphical representations of the actual and predicted values for each model used in this work as well as the residuals plot which show the differences between the actual and predicted values. The residuals plot provides insights into the distribution and magnitude of prediction errors, helping to assess the models' bias and variance. Ideally, residuals should be randomly distributed around zero, indicating that the model captures the underlying patterns without systematic errors or bias. We can see now that the deep learning models fit the actual values almost perfectly, where naive models do not fit the actual values at all and the machine learning models do a very good job to fit the data but fall short compared to the deep learning models.

4.3.1 Naive Visualization

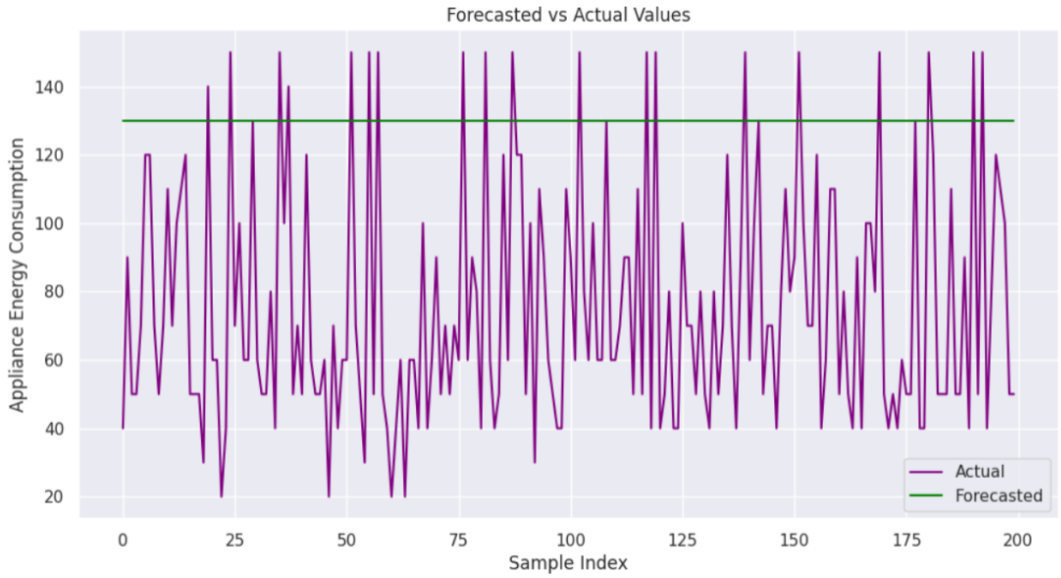

Figure 4: Actual vs Predicted Values of the testing dataset of Naive Model.

Figure 5: Residuals Plot of the Naive Model.

4.3.2 Moving Average Visualization

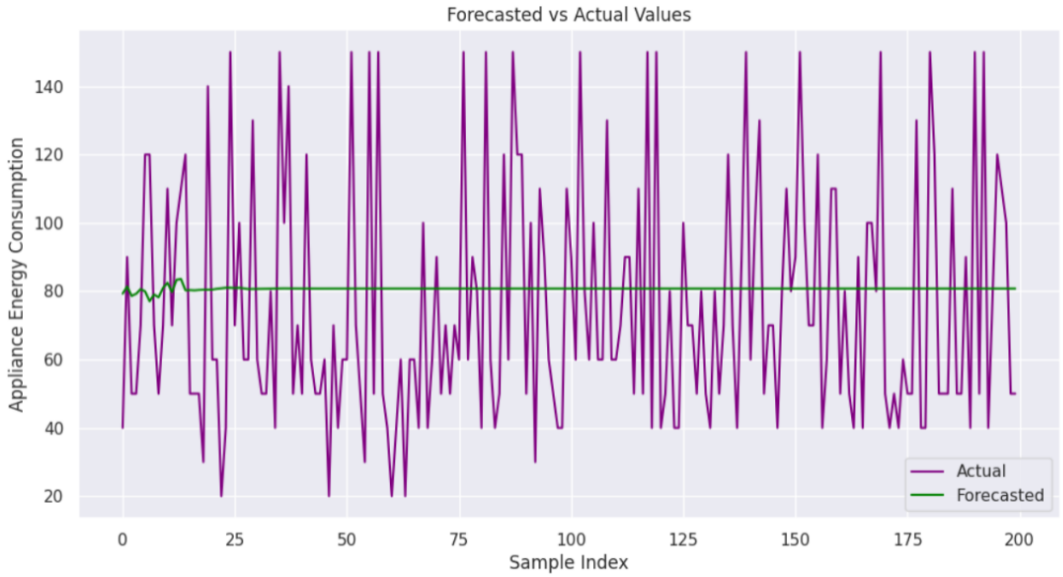

Figure 6: Actual vs Predicted Values of the testing dataset of Moving Average Model.

Figure 7: Residuals Plot of the Moving Average Model.

4.3.3 Seasonal Naive Visualization

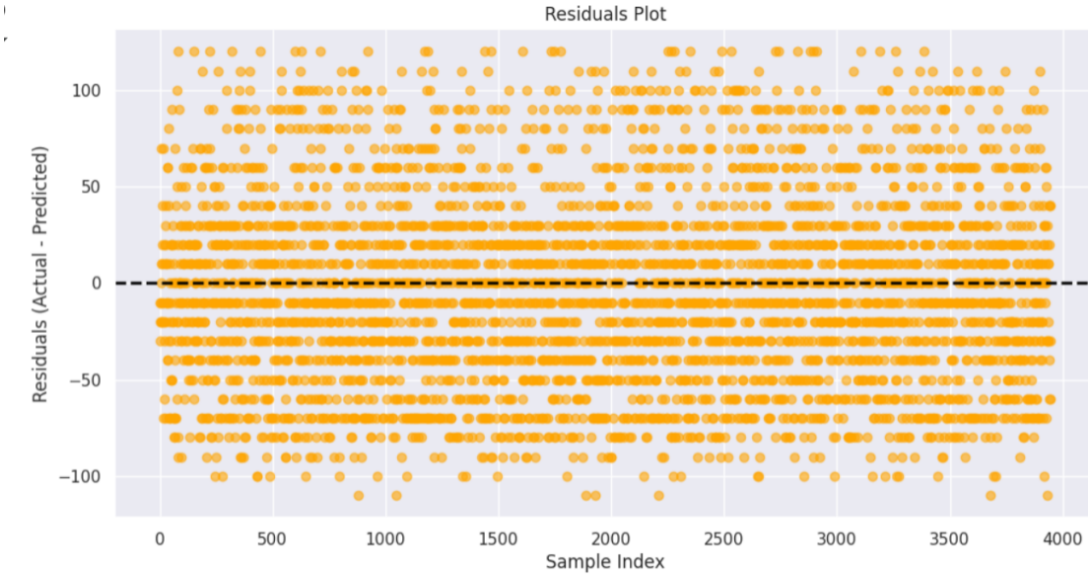

Figure 8: Actual vs Predicted Values of the testing dataset of Seasonal Naive Model.

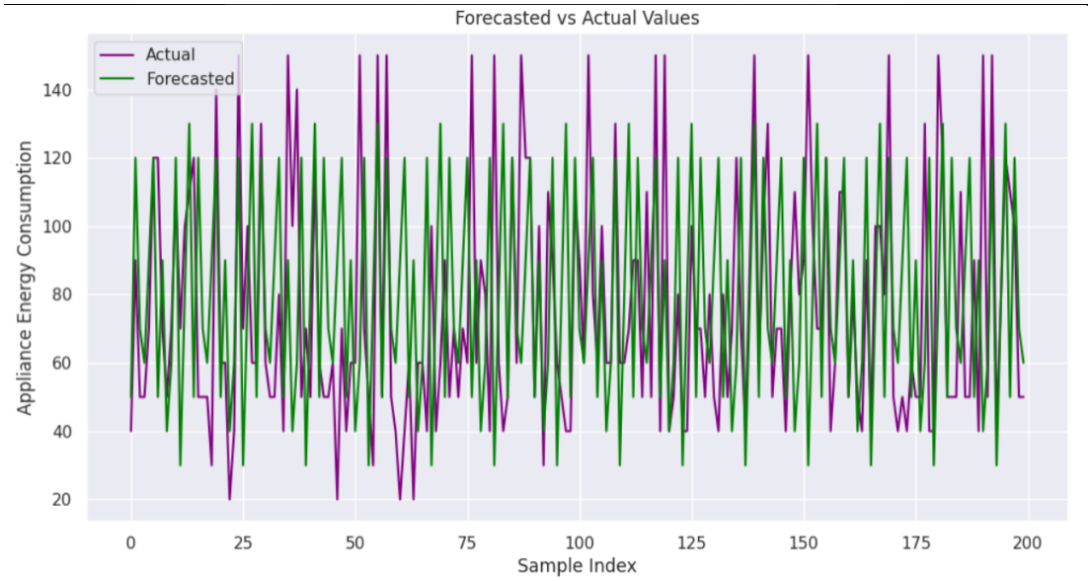

Figure 9: Residuals Plot of the Seasonal Naive Model.

4.3.4 Decision Trees Visualization

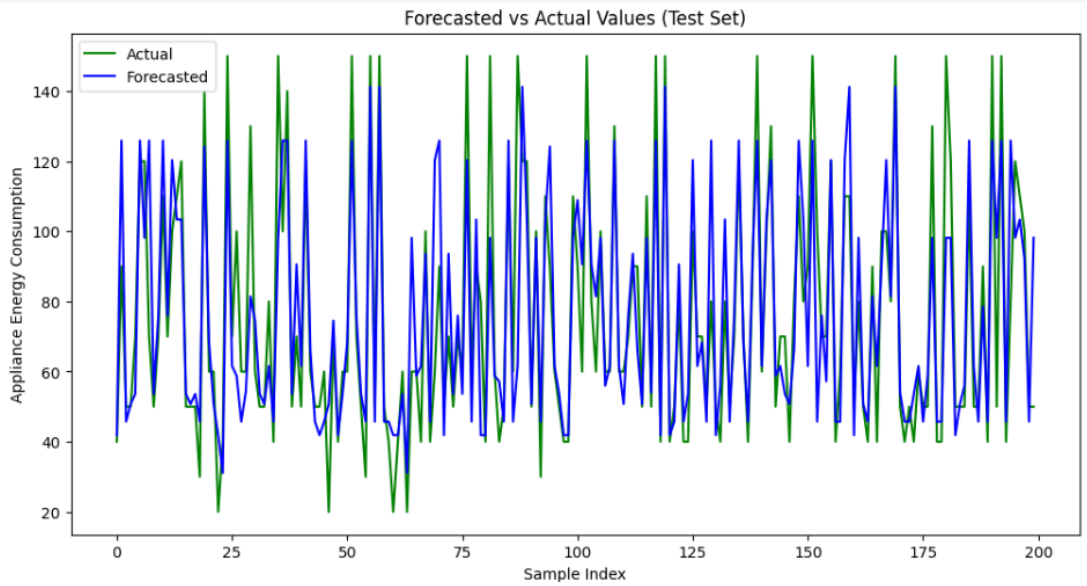

Figure 10: Actual vs Predicted Values of the testing dataset of Decision Trees Model.

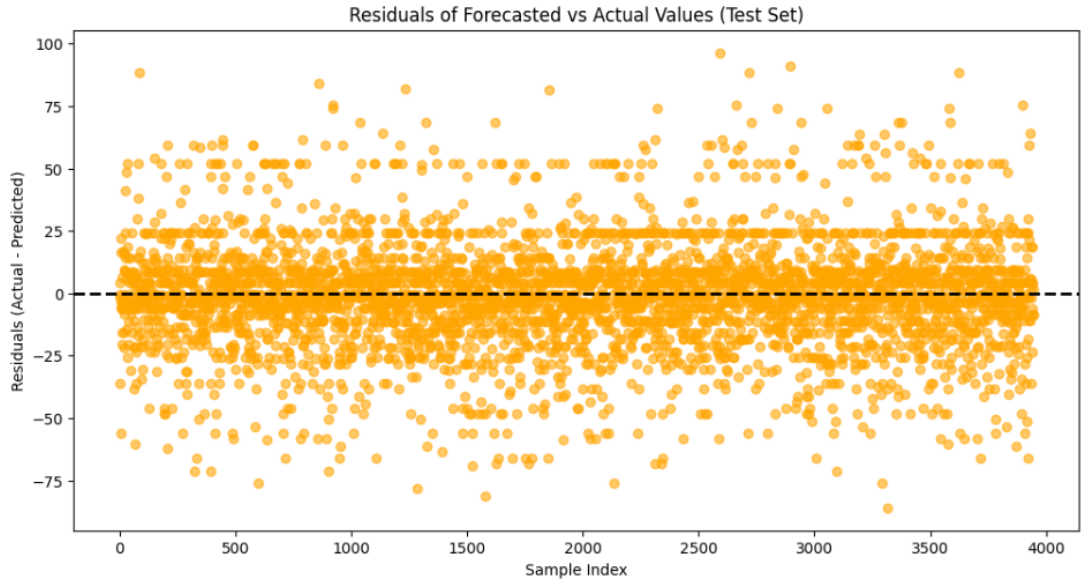

Figure 11: Residuals Plot of the Decision Trees Model.

4.3.5 Random Forest Visualization

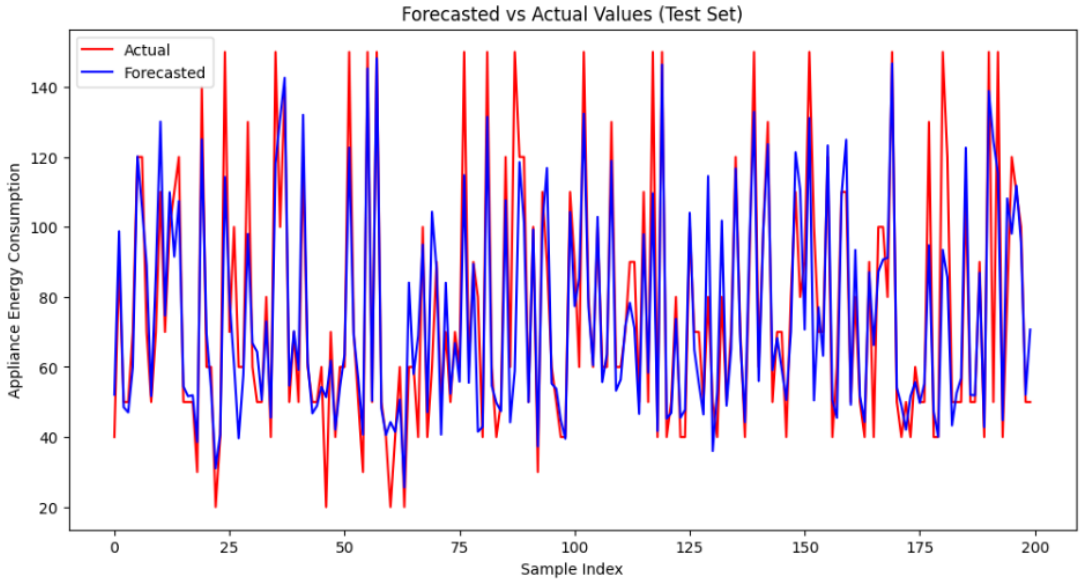

Figure 12: Actual vs Predicted Values of the testing dataset of Random Forest Model.

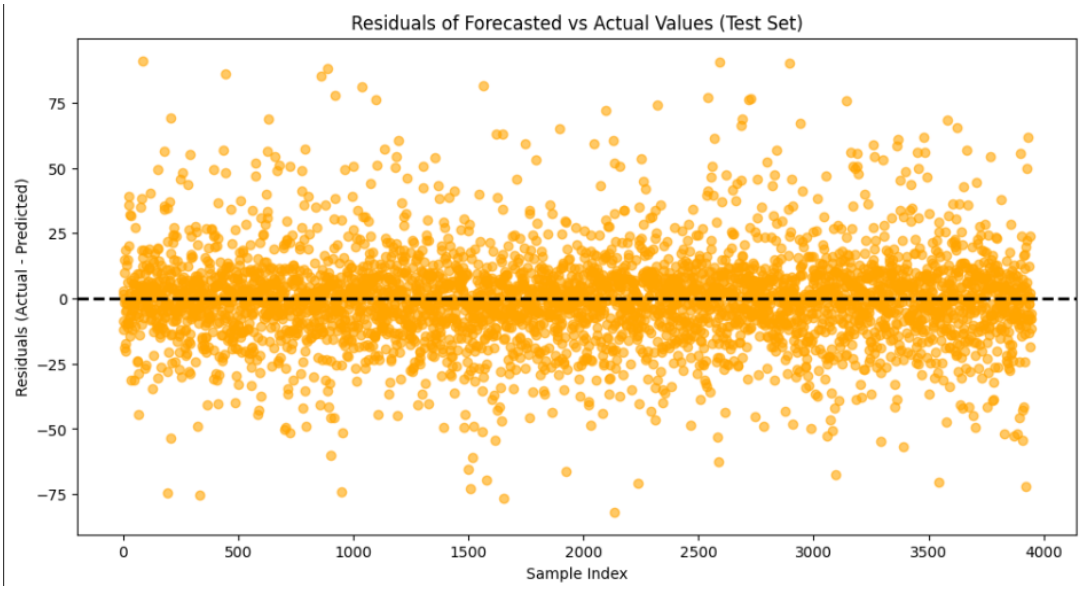

Figure 13: Residuals Plot of the Random Forest Model.

4.3.6 XGBoost Visualization

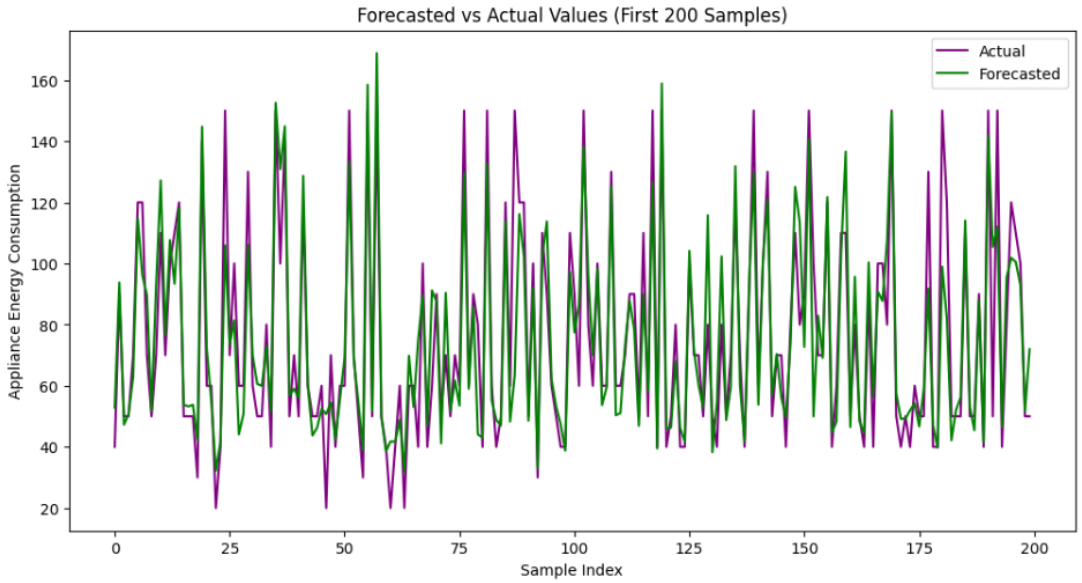

Figure 14: Actual vs Predicted Values of the testing dataset of XGBoost Model.

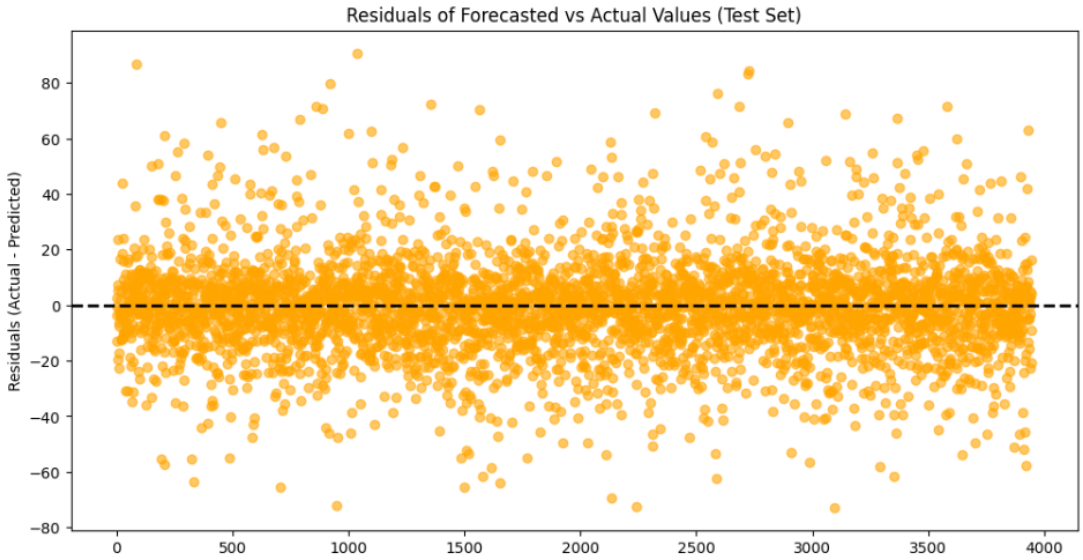

Figure 15: Residuals Plot of the XGBoost Model Model.

4.3.7 LSTM Visualization

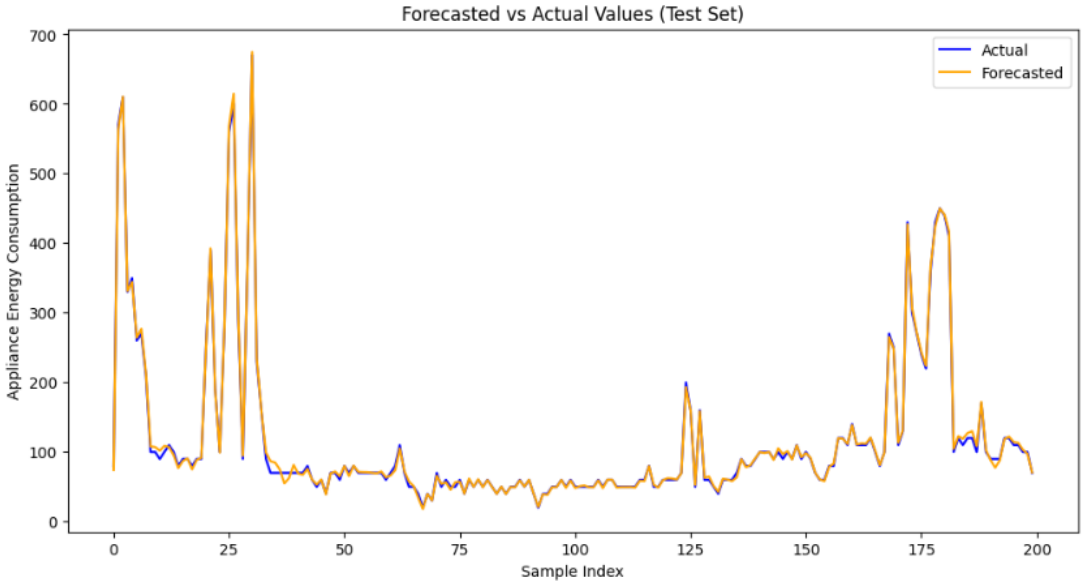

Figure 16: Actual vs Predicted Values of the testing dataset of LSTM Model.

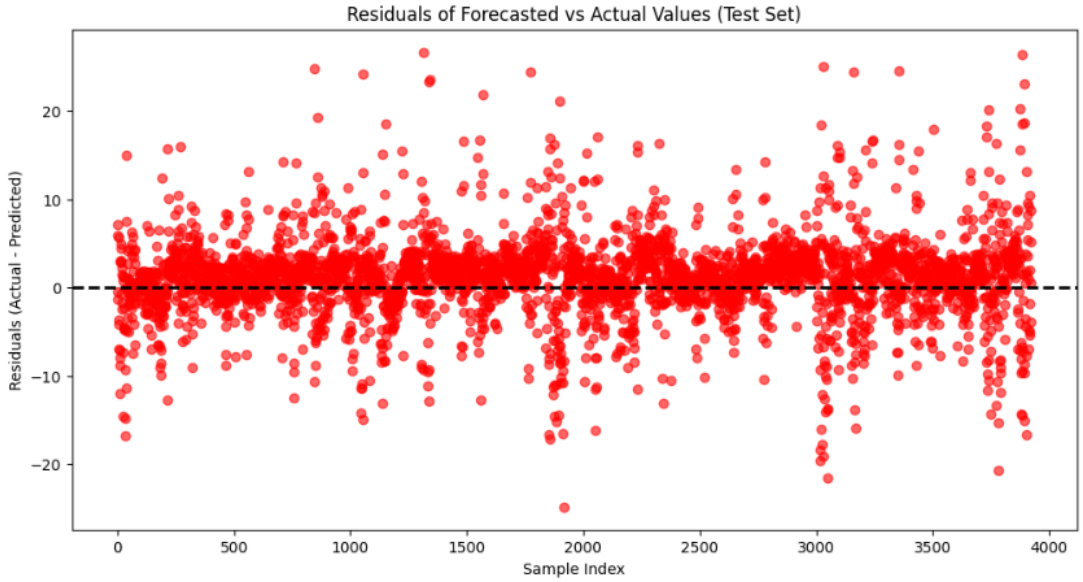

Figure 17: Residuals Plot of the LSTM Model Model.

4.3.8 GRU Visualization

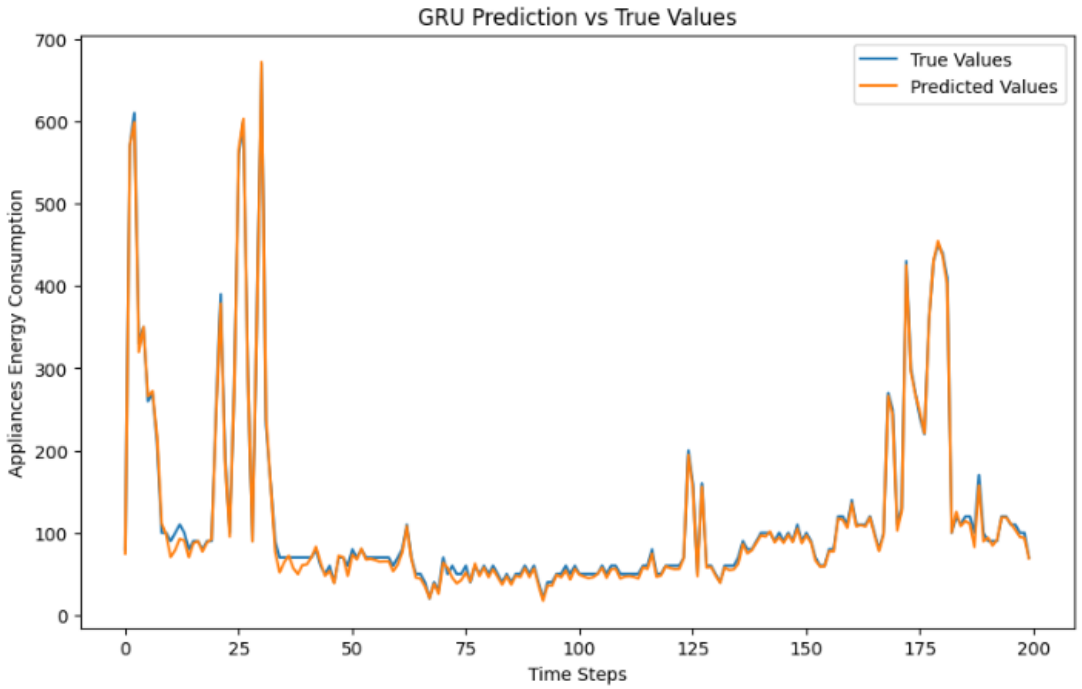

Figure 18: Actual vs Predicted Values of the testing dataset of GRU Model.

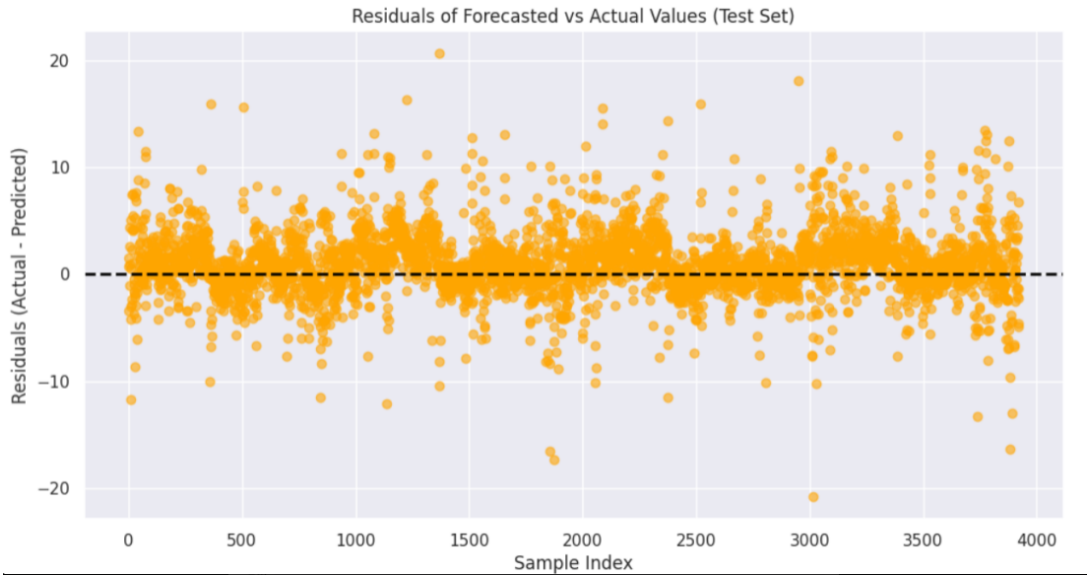

Figure 19: Residuals Plot of the GRU Model Model.

4.3.9 TCN Visualization

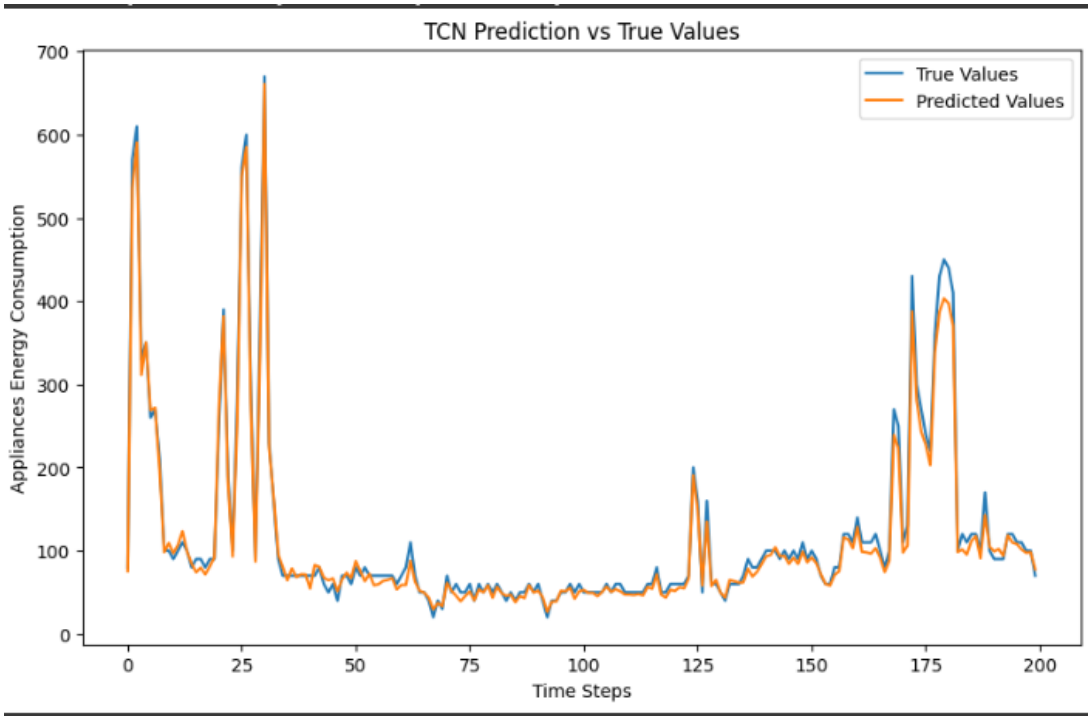

Figure 20: Actual vs Predicted Values of the testing dataset of TCN Model.

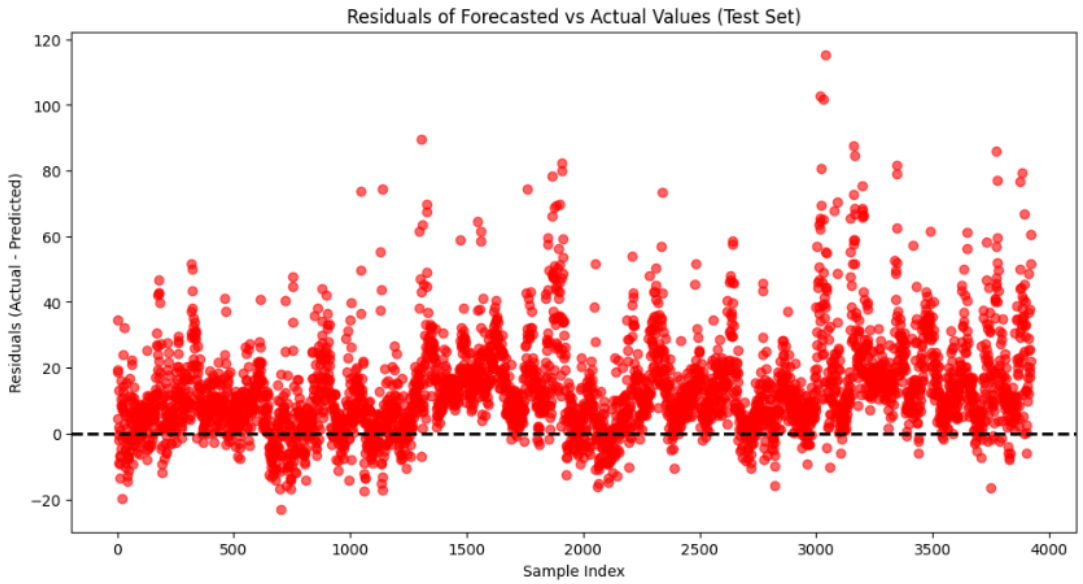

Figure 21: Residuals Plot of the TCN Model Model.

4.3.10 Transformers Visualization

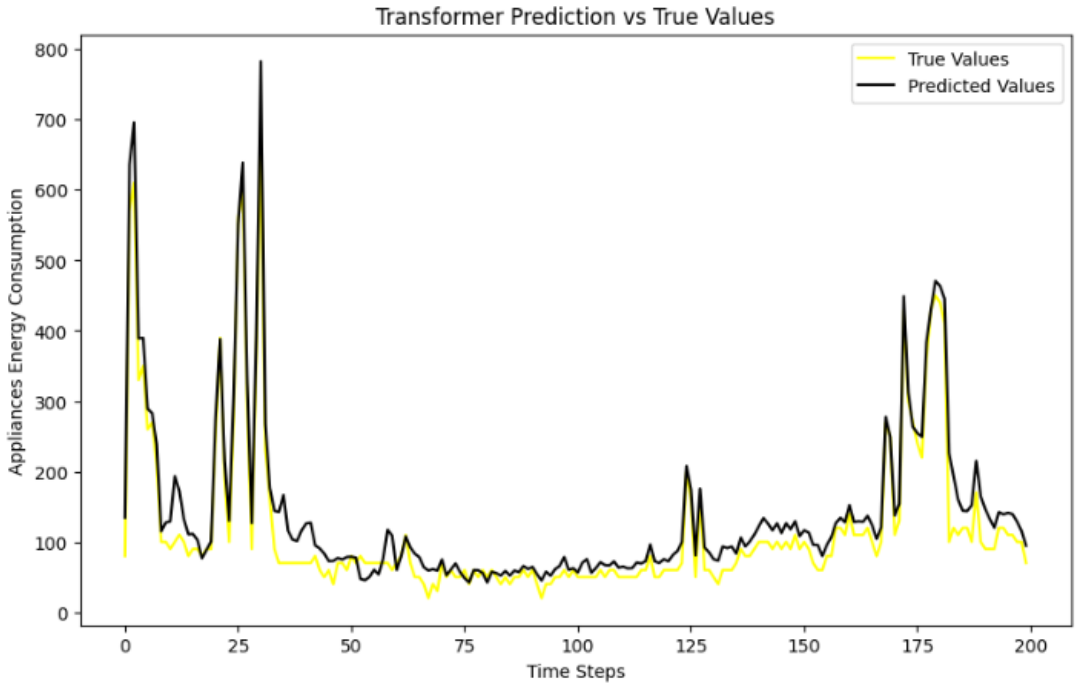

Figure 22: Actual vs Predicted Values of the testing dataset of Transformers Model.

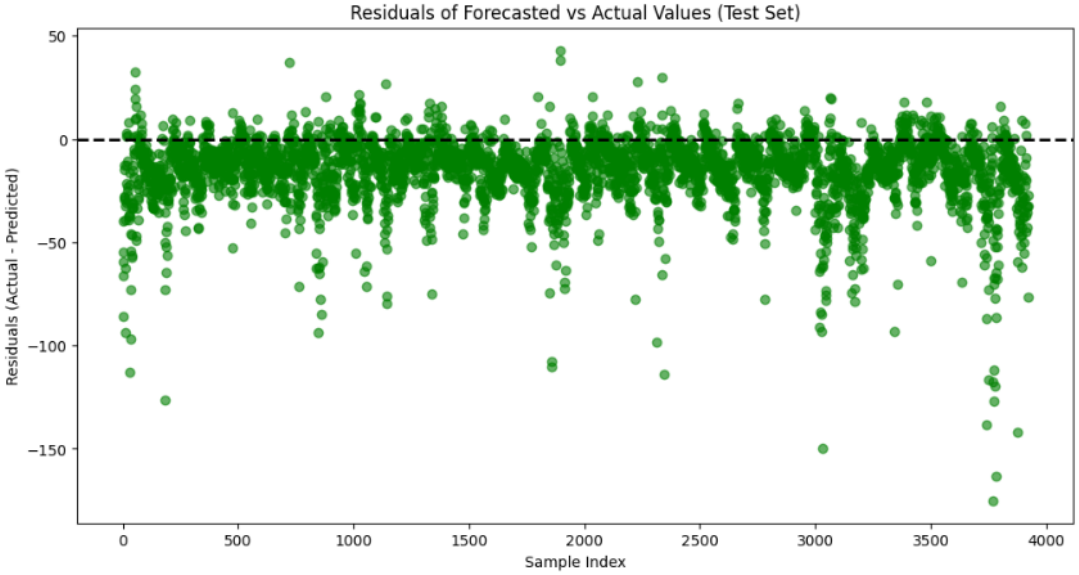

Figure 23: Residuals Plot of the Transformers Model Model.

We use lag values to capture seasonality and periodic patterns, where we use the rolling average values to smooth data and reduce noise while preserving trends across short-term and long-term intervals.

4.4 Execution Runtime

Model	Runtime (seconds)
Moving Average	0.03931
Naive	5.817e-05
Seasonal Naive	0.0007796
XGBoost	21.347
Random Forest	200.4847
Decision Tree	36.1089
LSTM	55.225
GRU	54.1312
TCN	66.3753
Transformers	121.672

Table 2: Training Runtime for each Model

We ran everything on google colab with the T4-GPU to get faster results especially on the deep learning models, which can be especially slow to train on the cpu because of the lack of computational power.

As we observe from the table above the execution time rises as the model complexity increases. We observe that naive models have, because of their minimal complexity exhibit extremely fast times, even with some custom grid search algorithm to find the best parameters for the moving average and seasonal naive models.

Where the machine learning models, which have a more complex structure show higher execution times. The random forest model proves to be the most time consuming of them all, this is due to setting the number of estimators to a high number as well with setting the depth limit to none, becoming unlimited. The decision tree has a high execution time, regarding its complexity. The xgboost seems to be the faster of all the machine learning models.

The deep learning and transformer implementations do not have a high complexity as we chose models which are relatively simple to run because of the huge training time it takes multi-layered, complex networks to run. We ran all these models for a total of 50 epochs. So the times between them can be a comparison of the complexities of each architecture, as the transformer network seems to be the more complex of them all. It comes to no surprise as the transformer network has a multi-head attention mechanism which can be very time consuming. Where the GRU network has the simplest architecture with only two gates and it shows on the table, as it has the fastest time of all the deep learning models.

5 Challenges

5.1 Overfitting

Overfitting is problem created, when a model learns training data too well, including noise leading to poor generalization on new data. To prevent a model from overfitting, techniques like early stopping, dropout, regularization and data augmentation

ensure a proper use of the validation set. We aim to balance capturing the true patterns in the data while maintaining the ability to generalize well to new unseen examples.

5.2 Model Training Time

The model training time did not pose a problem on naive and machine learning models as they did not take much time to train. The naive models have very simple architectures and the training is not practically a time-consuming procedure. The machine learning models Where on deep learning and transformer models we tried to implement small architectures with few layers to speed up the learning process.

6 Future Extensions

1. Optimizing the model architecture by exploring more advanced deep learning techniques and more complex transformer(informers) and recurrent neural networks.
2. Fine-tuning hyperparameters through grid search or random search for improved accuracy.
3. Applying advanced regularization techniques (e.g., dropout, L2 regularization) to prevent overfitting and improve generalization.
4. Exploring hybrid deep learning approaches that combine LSTM, GRU, or CNNs with transformer models to capture both spatial and temporal dependencies.
5. Investigating transfer learning techniques to apply models trained on one region's data to other regions with similar consumption patterns.
6. Developing real-time forecasting capabilities with adaptive models that adjust based on new incoming data.

7 Recommendations

The practical implications of a forecasting model for household energy consumption are significant for various stakeholders. Homeowners can optimize energy usage, reduce costs, and increase sustainability by adjusting appliance usage based on forecasts. Utility companies benefit from better load balancing, demand response programs, and more efficient infrastructure planning. Smart home device manufacturers can enhance their products with predictive features. These models contribute to create an efficient energy management for everyone.

8 Conclusion

From our results we concluded that the deep learning models outperform every other model, the best overall model proved to be the LSTM model. This makes a lot of sense as LSTMs are designed for time-series forecasting tasks, because of their ability to capture both long term and short term data dependencies.

The deep learning models and the transformer model outperformed machine learning and naive models, as evidenced by the earlier plots. These advanced models excel at capturing seasonality and trends, leveraging their ability to model complex, non-linear patterns in the data. By effectively handling long-term dependencies and adapting to irregularities in the dataset, they better capture historical data relationships compared to traditional approaches. This makes them particularly well-suited for tasks requiring accurate modeling of temporal dependencies and complex dynamics in time series data.

References

[1] Kasun Amarasinghe, Daniel L. Marino, and Milos Manic. "Deep neural networks for energy load forecasting." In: *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*. 2017, pp. 1483–1488. DOI: 10.1109/ISIE.2017.8001465.

[2] Saeed M Badran and Ossama B Abouelatta. "Forecasting electrical load using ANN combined with multiple regression method." In: *The research bulletin of Jordan ACM* 2.2 (2011), pp. 152–158.

[3] Luis M. Candanedo, Véronique Feldheim, and Dominique Deramaix. "Data driven prediction models of energy use of appliances in a low-energy house." In: *Energy and Buildings* 140 (2017), pp. 81–97. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2017.01.083. URL: https://www.sciencedirect.com/science/article/pii/S0378778816308970.

[4] Hany Habbak et al. "Load Forecasting Techniques and Their Applications in Smart Grids." In: *Energies* 16.3 (2023). ISSN: 1996-1073. DOI: 10.3390/en16031480. URL: https://www.mdpi.com/1996-1073/16/3/1480.

[5] R. Hariharan and More Kumar. "A Research on Electric Load Forecasting Factors Effecting and Methods Involved." In: *International Journal of Innovative Technology and Exploring Engineering* 8 (Oct. 2019), pp. 1462–1466. DOI: 10.35940/ijitee.L3951.1081219.

[6] Vasileios Laitsos et al. "The State of the Art Electricity Load and Price Forecasting for the Modern Wholesale Electricity Market." In: *Energies* 17.22 (2024). ISSN: 1996-1073. DOI: 10.3390/en17225797. URL: https://www.mdpi.com/1996-1073/17/22/5797.

[7] Junling Luo et al. "Time series prediction of COVID-19 transmission in America using LSTM and XGBoost algorithms." In: *Results in Physics* 27 (June 2021), p. 104462. DOI: 10.1016/j.rinp.2021.104462.

[8] Ashish Vaswani et al. "Attention is all you need." In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.