Machine Learning 2024-25

Final Project

Electrical & Computer engineering
University of Thessally, Volos

Instructor: Elias Houstis

Team 16
Kakepakis Georgios - 03198
Pisxos Vasileios - 03175

Submission Date: 16/1/2025

Abstract

This project explores the application of machine learning algorithms to predict the energy use of appliances based on sensor data. The project is inspired by the study described in the paper "Data driven prediction models of energy use of appliances in a low-energy house" (link), which uses traditional machine learning models (Linear Regression, Support Vector Machines, Random Forest, Gradient Boosting Machines). In this project we implement 2 gradient boosting methods (**Gradient Boosting Machines, XGBoost**), 2 deep learning methods (**LSTM, GRU**), and a **Transformer**, in order to understand the utilization of deep learning and attention models in time series forecasting and to compare it with the models used in the original paper.

Keywords: Time Series Forecasting, Gradient Boosting Machines, XGBoost, LSTM, GRU, Transformers, Deep Learning, Attention

1 Introduction

1.1 Problem Statement

Time series forecasting has been the subject of numerous research studies over the last years. It is the process of predicting future values based on previously observed data points, where the data is collected in a sequential, time-ordered manner (e.g., hourly, daily, monthly) by capturing temporal patterns like trends, seasonality, and cyclic behaviors to make informed predictions.

Time series forecasting is widely used across various domains to predict future trends, optimize operations, and improve decision-making, namely: finance and economics, healthcare, retail and E-commerce, weather and climate, and of cousre, the topic of this project: appliances energy use. Forecasting energy usage is vital for several reasons, like maintaining a demand-supply balance to ensure grid stability, efficient resource allocation, cost management and more. Hence understanding these application is crucial.

1.2 Objectives

The present work is based on the paper "Data driven prediction models of energy use of appliances in a low-energy house". This paper focuses on predicting the energy consumption of appliances of a house. It uses data about the temperature and humidity of the rooms and some extra data provided by a nearby airport weather station. The models used include: a **multiple linear regression** model, a **support vector machine** (SVM) model with radial basis function kernel, a **random forest** (RF) model and a **gradient boosting machine** (GBM) model. The best model proved to be the GBM.

In this project we aim to apply the concepts of **deep learning** and **attention** mechanisms to the same problem. In order to do that we use the **GBM** model, which proved to be the best in the original paper, and we also use: an **XGBoost** model, an **Long Short-Term Memory** (LSTM) model, a **Gated Recurrent Unit** (GRU) model, and finally a **Transformer**. We aim to compare the deep learning models and the transformer with the GBM to see if these concepts improve the performance of time series forecasting.

1.3 Scope and Limitations

Although this project seems promicing about extracting some conclusions about the performance of these different machine learning techniques, there are some contrains that exist and we should highlight.

Firstly, regarding the **dataset** we use, the dates span between about 4.5 months, so we may lose some annual seasonality patterns that would be captured if the dataset included information about multiple years. Also we are not sure that

the extra data captured from the nearby airport weather station accurately represents the information of the house.

In addition, a **computational contrain** is the lack of full-time GPU availability to train the models. We worked on Google Colab and Kaggle which offer part-time available GPUs, so when they expire the training time becomes exceptionally growing. In that case some models like the LSTM and the Transformer can take much longer to train. Also the GBM model is not compatible with the useage the GPU, so its training can not accelerated.

2 Literature Review

This section reviews some articles/papers regarding the application of classical time series models, machine learning models, deep lerning models and transformer-based approaches.

The paper in [1] presented a study of the use of Weather Forecasting Techniques with Time Series data focusing on the ability of the ETS Model and the ARIMA Model to predict different weather parameters such as Rainfall, Air Temperature, Relative Humidity, Gust etc, and it illustrated a good performance and reasonable prediction accuracy. The study in [2] implemented various tree-based models like GMB and Random Forestm and illustrated how they can be utilized to forecast time series values by they presenting promising results. The work presented at [3] presentied the utilization of multiple deep learning models and attention-based models in the domain of time series forecasting, and showed how these technologies can be beneficial.

The motivation behind this work is the study of other projects like the ones mentioned, where we observe that the utilization of deep learning models and attention-based models is promising for forecasting time series, since they can learn complex patterns and store memory. We believe that these features can be used to over-perform the classical time series models, so we aim to compare then and see the results.

3 Data Collection and Preprocessing

3.1 Data Description

The dataset we use is the same that is used in the original paper and can be found in the following link. This dataset uses data about the temperature and humidity of the rooms and some extra data provided by a nearby airport weather station. We can see the features below:

Data variables	Units	Number of features
Appliances energy consumption	Wh	1
Light energy consumption	Wh	2
T1, Temperature in kitchen area	°C	3
RH1, Humidity in kitchen area	%	4
T2, Temperature in living room area	°C	5
RH2, Humidity in living room area	%	6
T3, Temperature in laundry room area	°C	7
RH3, Humidity in laundry room area	%	8
T4, Temperature in office room	°C	9
RH4, Humidity in office room	%	10
T5, Temperature in bathroom	°C	11
RH5, Humidity in bathroom	%	12
T6, Temperature outside the building (north side)	°C	13
RH6, Humidity outside the building (north side)	%	14
T7, Temperature in ironing room	°C	15
RH7, Humidity in ironing room	%	16
T8, Temperature in teenager room 2	°C	17
RH8, Humidity in teenager room 2	%	18
T9, Temperature in parents room	°C	19
RH9, Humidity in parents room	%	20
To, Temperature outside (from Chièvres weather station)	°C	21
Pressure (from Chièvres weather station)	mm Hg	22
RHo, Humidity outside (from Chièvres weather station)	%	23
Windspeed (from Chièvres weather station)	m/s	24
Visibility (from Chièvres weather station)	km	25
Tdewpoint (from Chièvres weather station)	°C	26
Random Variable 1 (RV_1)	Non dimensional	27
Random Variable 2 (RV_2)	Non dimensional	28
Number of seconds from midnight (NSM)	s	29
Week status (weekend (0) or a weekday (1))	Factor/categorical	30
Day of week (Monday, Tuesday... Sunday)	Factor/categorical	31
Date time stamp	year-month-day hour:min:s	–

Figure 1: Data Variables and Description

The features 1-20 are the values captures from inside the rooms of the house with sensors, and the features 21-28 are the extra information from the nearby airport weather station. The features 29-31 are generated from the date just like the original paper methodology suggested, and they are:

- **NSM**: an integer that represents how many seconds have passed since midnight
- **day of week**: an integer that represents the day of the week (0-6 for Monday-Sunday)
- **Week Status**: a booolean that indicates wether the day belongs to the weekend or not

3.2 Data Cleaning

The dataset contained **0 missing values**, so we did not have to remove or handle any of them. Now we want to look for **outliers** in the dataset. Firstly, we want to see the distribution of the appliances values, so we plot their histogram:

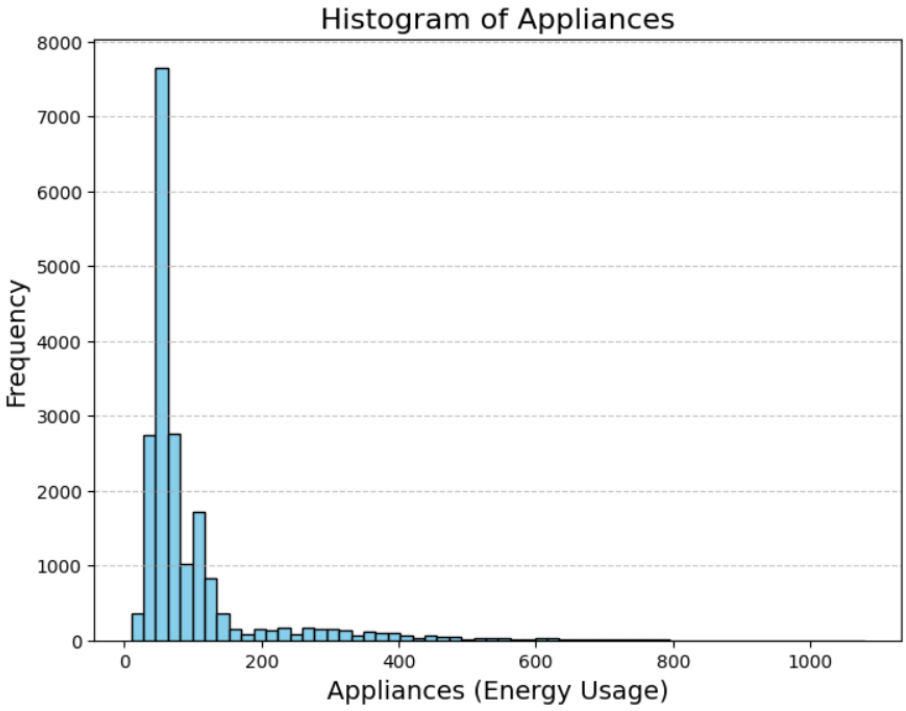

Figure 2: Appliances Histogram

We can see that the distribution of the values is **right-skewed**. This means that while most of the samples have a value at about 0-200, there exist also several samples with much higher values. We can confirm that by plotting the boxplot also:

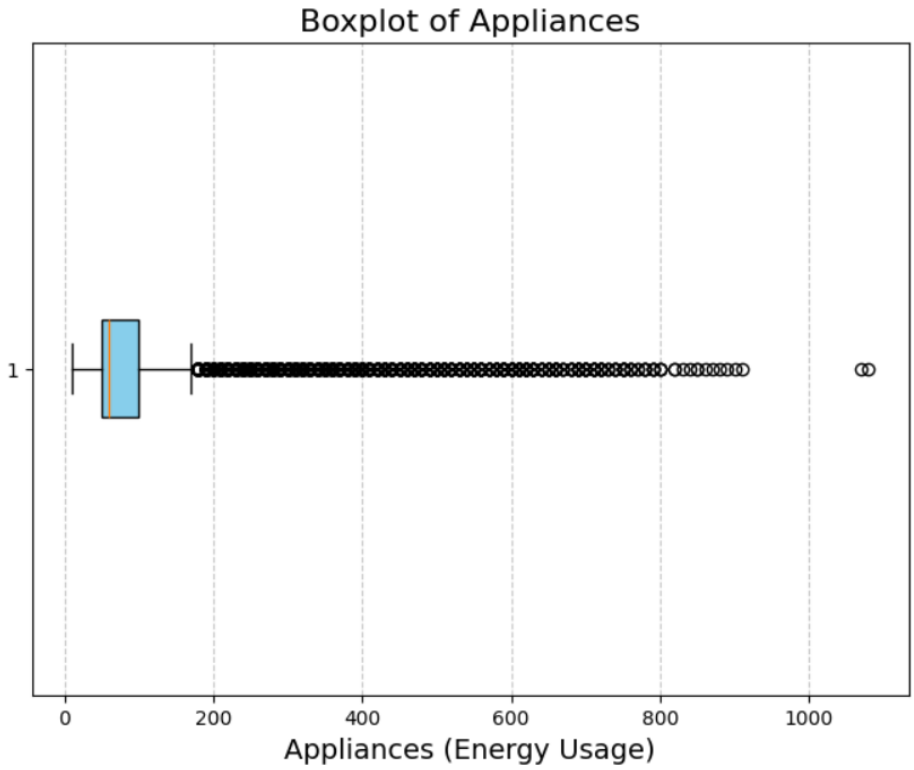

Figure 3: Appliances Boxplot

We see again that there exist outliers with very large values. Since we do not contain a dataset that spans though a large amount of time (like one or multiple years), we decided that we can not make the assumption that these values are wrong, because we do not not the annual pattern and therefore can not dismiss the outliers are wrong mesurements. In addition, simply removing samples from a dataset for time series forecasting can disrupt the sequence of the data, since all samples are 10 minutes apart. Therefore we decided to **keep all the values** for out models.

3.3 Exploratory Data Analysis

3.3.1 Autocorrelation

In this section we analyse the dataset. Firstly, we plot the **autocorrelation** of the appliances to see how the past values relate to the current one. Below we print the autocorrelation of the appliances for **one day**, **one week**, and **two weeks**:

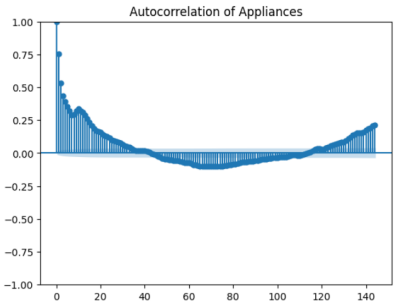

Figure 4: Autocorrelation - 1 Day

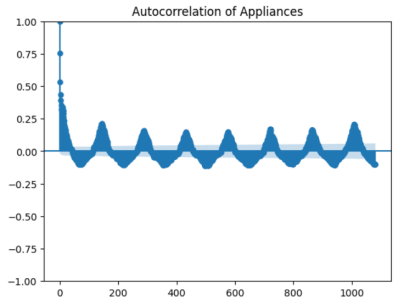

Figure 5: Autocorrelation - 1 Week

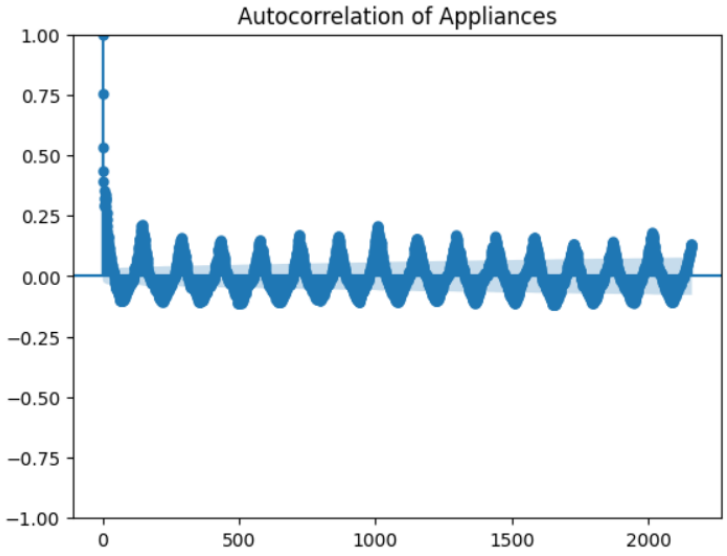

Figure 6: Autocorrelation - 2 Weeks

If we look at the autocorrelation for **1 day**, we can see that there is strong autocorrelation for the first past values, specifically 0.75, 0.50, etc. The autocorrelation is positive for the first part of the previous day, then it is negative, and then positive again. This could be explained by the different usage the house performs, which is assumed to be **periodic** within the day. For example, if it is morning, the energy used is likely similar the latest past moments, but more dissimilar as we look further back. If we looked many hours before, at night, the consumption is different, which is denoted by a **negative** autocorrelation. If we go even further back, we tend to get to a morning time again, which has similar consumption, and is denoted by a **positive** autocorrelation.

Therefore we assume that there is a seasonality between each day, If we look at the plot for **1 week**, we actually see that this pattern appears between **each day**. Since we plot a whole week we expect to see 7 iterations of the period, and we indeed see that this is what happens, we get 7 peaks corresponding to 7 days. This show that there is a strong autocorrelation **for the same hours of the day**

Finally, we want to see if there is a correlation between **the same days of the week**. To do that we extend the autocorrelation plot to include 2 weeks. When we do that we see that the autocorrelation has a larger value at about 1008 and 2016 lags. These correspond to the **same day** of another week, and they are slightly larger that the values at the same hour for different days of the week. So we conclude that there is a seasonality pattern at the same hour every day, and also at the same day every week.

3.3.2 Time Series Decomposition

Next, we **decompose** the values of appliances to obtain the **trend, seasonality** and **residuals**. We assume an **additive** model. In order to find the **optimal seasonal period** we used 2 methods: Fourier Transform and Periodogram. The optimal period on both cases proved to be **144**, which is the lags corresponding to **1 day**. If we apply **STL decomposition** with this period, we get the following visualization:

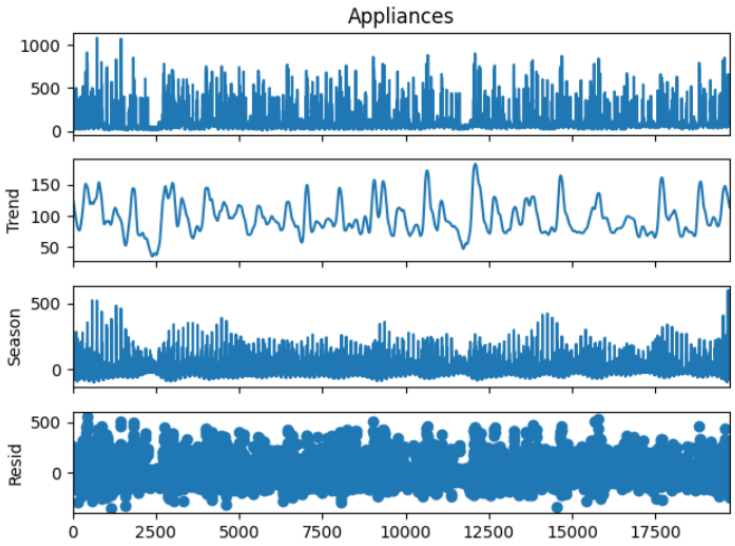

Figure 7: STL Decomposition - Period: 144 lags

We also calculate the **Seasonal Strength** it is equal to **0.5477**. The corresponding Seasonal Strength for a seasonal period of a week (7*144 = 1008 lags) is 0.5234, so we conclude that the seasonality of the appliances is better described by a **period of 144 lags / 1 day**.

3.3.3 Stationarity

In order to perform a **stationarity check**, we compute and plot the **rolling mean** and **rolling standard** deviation and check if they are constant over time or not. We use the optimal seasonality period as the window size and get the following plots:

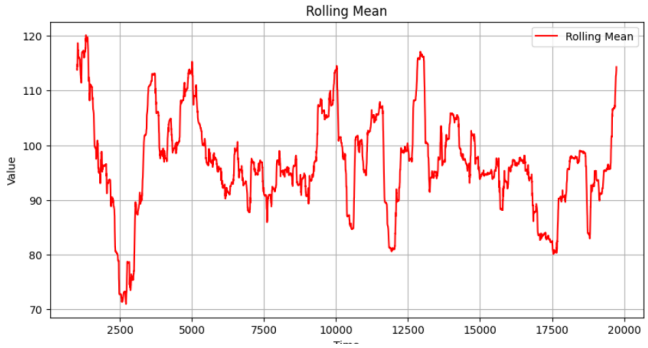

Figure 8: Rolling Mean

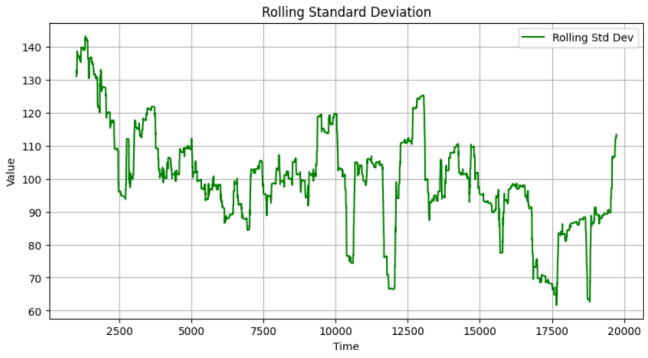

Figure 9: Rolling Standard Deviation

As we can see from the plots, the mean and the standard deviation of the rolling window are not constand over time, in fact they fluctuate a lot. So we conclude that the time series is **not stationary**.

3.3.4 Correlation Matrices

Below we plot the correlation matrices of appliances with all the features. We use 4 matrices since the features are too many just like the original paper:

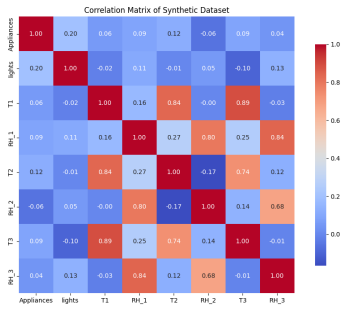

Figure 10: Correlation Matrix (1)

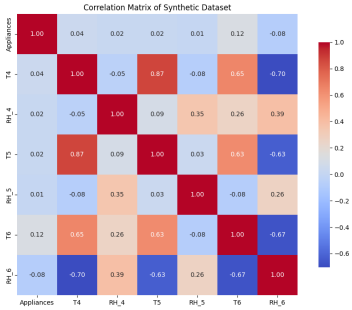

Figure 11: Correlation Matrix (2)

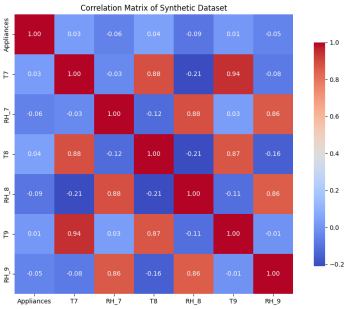

Figure 12: Correlation Matrix (3)

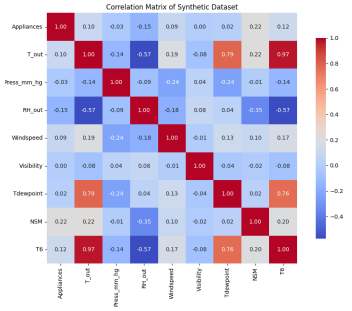

Figure 13: Correlation Matrix (4)

The values we get are almost the same as the ones presented in the correlation matrices of the paper (some differences occur at 2 decimal digits). As we see, there are some features that have almost 0 correlation with the target. Also we see that some features are highly correlated with each other, like T1-T3, T4-T5, T7-T9 and more. In order to find which features are usefull we will perform feature selection.

3.3.5 Feature Selection - Boruta

In order to determine which features to keep we will apply the 2 methods used in the paper. We first apply boruta to find the **importance** of each feature. Internally we use an "ExtraTreeRegressor" ([4]), because it is a lot faster than the Random Forest model and we have too many features. In order to be more sure about the importances, we run the boruta algorithm 10 times and generated boxplots for the importances, similar to the orignal paper. When we plot

them in descending order we get the following plot:

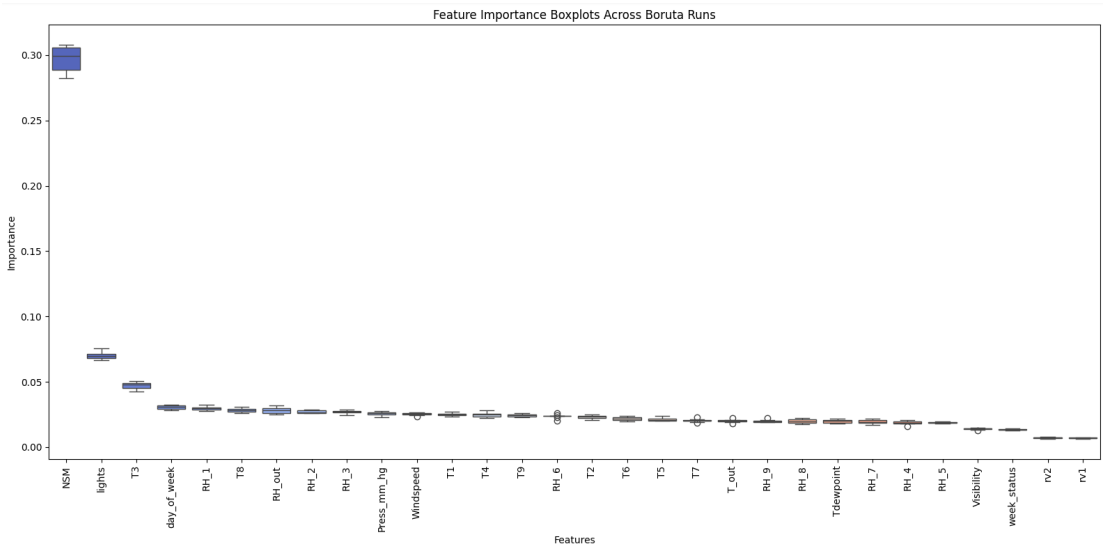

Figure 14: Boruta Importances Boxplot

As we can see, the feature with the biggest importance is by far **NSM**, just like in the original paper, followed by lights, T3 and day_of_the_week. We also see that the boruta algorithm identifies the 2 random variable features and assigns them with the lowest importance. Therefore we decide to **drop "rv1" and "rv2"**, and keep the rest of the features.

3.3.6 Feature Selection - Recursive Feature Elimination

Next we apply Recursive Feature Elimination (RFE) to the rest of the features. This method ranks the features by importance, and then, starting from the most important one, it iteratively adds the next most important feature and evaluates the mean RMSE using corss-validation. We plot the RMSE for each iteration below:

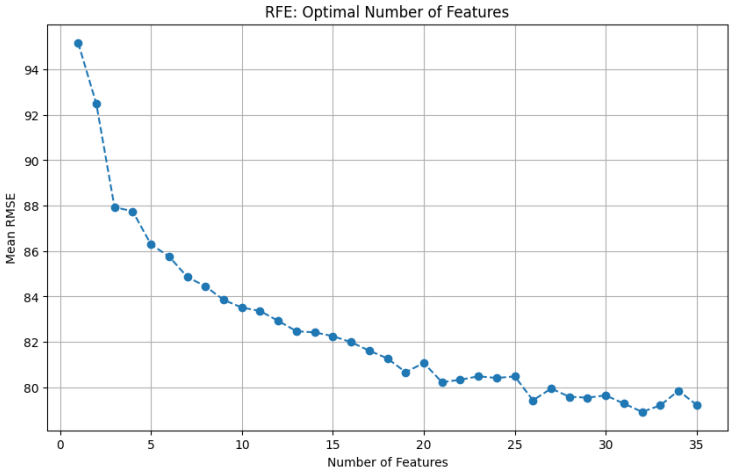

Figure 15: RMSE vs Number of Features

And this is the ranking of the features (we used dummy variables for the RFE algorithm like the paper):

Index	**Feature**	**Ranking**
25	NSM	1
17	T9	2
5	T3	3
0	lights	4
6	RH_3	5
1	T1	6
15	T8	7
21	RH_out	8
7	T4	9
2	RH_1	10
4	RH_2	11
12	RH_6	12
22	Windspeed	13
24	Tdewpoint	14
20	Press_mm_hg	15
3	T2	16
11	T6	17
9	T5	18
32	Day_of_weekFriday	19
14	RH_7	20
33	Day_of_weekSaturday	21
8	RH_4	22

Index	Feature	Ranking
13	T7	23
10	RH_5	24
19	T_out	25
28	Day_of_weekMonday	26
16	RH_8	27
18	RH_9	28
23	Visibility	29
26	WeekStatusWeekend	30
34	Day_of_weekSunday	31
29	Day_of_weekTuesday	32
31	Day_of_weekThursday	33
30	Day_of_weekWednesday	34
27	WeekStatusWeekday	35

The lowest mean PMSE is for **32 features**, but if we look at the last 3 features to be dropped we see that they are the binary values of specific days. So if we would delete just these and keep the other day variables, we believe that this would not make a lot of sense. In addition, we see that the mean RMSE is almost the same for 32 and 35 features, so in order to keep all the information we decided to **keep all the features**, just like the original paper.

4 Methodology

In this section we will mention every model that we use. We will see the normalization/scaling we do to make the data appropriate for each model, how many lags each model uses, and the following metrics: **Root Mean Squared Error** (RMSE), **Mean Absolute Error** (MAE), **Mean Absolute Percentage Error** (MAPE) and R^2.

For the deep learning models (LSTM, GRU) and the Transformer model we used the "**TimeSeriesGenerator**" method from tensorflow which creates dynamically the lagged data saving memory and computational time. This is not compatible howerver with the GBM and XGBoost models, so we experimented with no lags or just a few lags.

4.1 Gradient Boosting Machine (GBM)

Firstly we will explore the model that worked best on the original paper, GBM. As we mentioned above, since the "TimeSeriesGenerator" method is not available for this model, we tested a case where we used **no lags**, and a case where we used only **3 lags**.

Before we pass the data to the GBM we **standarize** them, meaning that they are transformed so they have a mean of 0 and a standard deviation of 1. Below we show the metrics when applied on the testing sets:

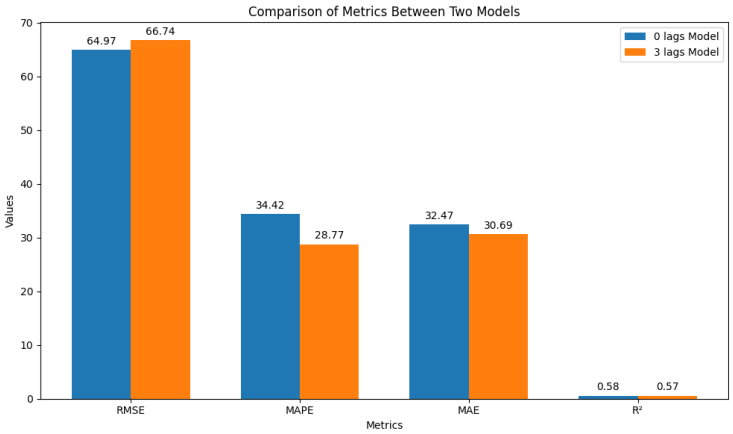

Figure 16: GBM metrics (0 lags VS 3 lags)

As we see the lagged model has a better MAE and MAPE value but a worse RMSE and R^2 value. We also see their residual plots below:

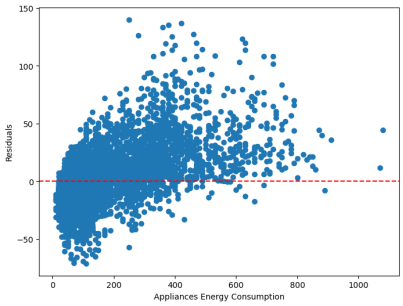

Figure 17: Residuals Plot - 0 lags

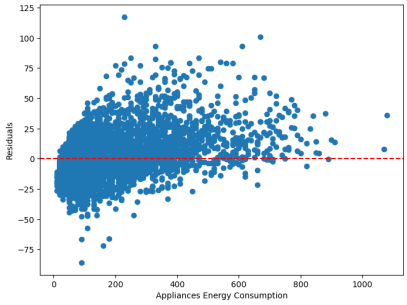

Figure 18: Residuals Plot - 3 lags

We believe that out of these metrics the most indicative of the forecasting quality is RMSE, so therefore we will select the model with **0 lags**. It also present a higher R^2. Nevertheless the lagged model is also a great choice.

4.2 Extreme Gradient Boosting (XGBoost)

Just like we did on GBM, we again use a model with **0 lags** and a model with **3 lags** to see the impact. We again **standarize** the data before feeding them to the model. Below we show the metrics when applied on the testing sets:

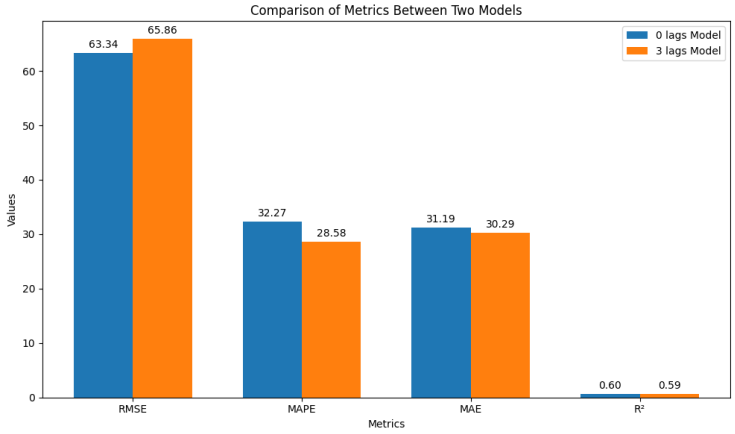

Figure 19: XGBoost metrics (0 lags VS 3 lags)

Just like we saw in the GBM model, the lagged XGBoost model has a better MAE and MAPE value but a worse RMSE and R^2 value. We also see their residual plots below:

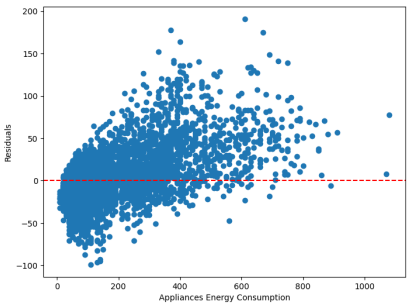

Figure 20: Residuals Plot - 0 lags

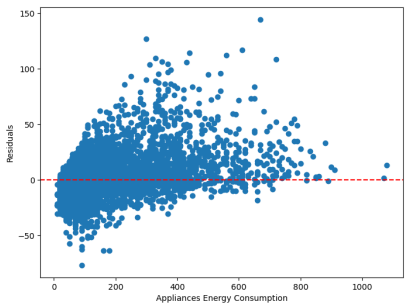

Figure 21: Residuals Plot - 3 lags

Just like we did previously in the GBM model, since we believe that out of these metrics the most indicative of the forecasting quality is RMSE, we will select the model with **0 lags**. It also present a higher R^2. Nevertheless the lagged model is also a great choice.

4.3 Long Short-Term Memory (LSTM)

4.3.1 Standardization

We use MinMaxScaler ([5]) to standardize our data. That means that we transform our data to be in the [0, 1] closed interval. MinMaxScaler does not reduce the effect of outliers, but it linearly scales them down into a fixed range, where the largest occurring data point corresponds to the maximum value and the smallest one corresponds to the minimum value.

4.3.2 Window size

LSTMs are designed to focus on recent information and weigh it more heavily than older data points. In many tasks, especially when working with time series, the most relevant information often comes from a recent window of data. The forget gate allows them to prioritize recent information and discard less relevant past data. With that in mind, we experimented with different window sizes and chose the one that gave us the best results. Here we present a plot with those results for every window size that we experimented with:

Figure 22: LSTM metrics (77 lags VS 144 lags VS 1008 lags)

In all metrics the model with 77 window size performs the best, showing that the LSTM does not capture the long-term dependencies and that the extra windows are not helping it become better. So we chose the model with **77 lags**.

4.3.3 Optimizer

We started experimenting with Adam optimizer([6]) for our model at first, but then changed our optimizer to Nadam ([7]) and observed slightly better results. We decided to change our optimizer because we observed that the dataset contains some noise. We suspected that the noise was causing erratic changes in the gradient direction, which Nadam's look-ahead mechanism helped resolve. We set our initial learning rate at 0.001.

4.3.4 Activation Function

For our activation function we used **LeakyRelu** ([8]). It's strength lies in it's ability to mitigate the dying ReLU problem by allowing a small non-zero gradient for negative inputs, it ensures that neurons remain responsive and continue to learn even when presented with challenging data distributions.

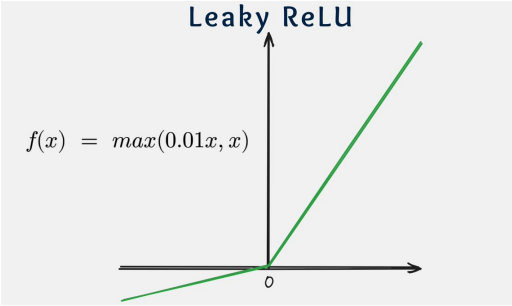

Figure 23: Leaky ReLU

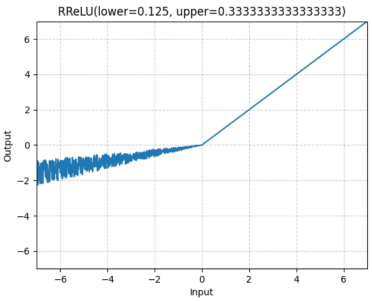

Figure 24: Random ReLU

We used negative slope = 0.3 for our model. In addition, we also tried to use **RReLu** (Random ReLU) ([11]), which is an activation function that introduces randomness to the slope of the LReLu. It does that to better model the noisy data of the dataset. It had about the same results as RLeLu.

4.3.5 Model Architecture

We experimented with many different architectures for our model. To capture the sharp spikes in our data effectively we had to gradually increase our model complexity (up to a point). So starting with one LSTM layer with 128 units and one dense layer with 1 unit we gradually increased the layers in our architecture ending up with 3 LSTM layers and one dense layer with 256, 256, 128 and 1 units accordingly.

We also experimented with more units and more layers but didn't seem to get better results and the training time was increasing exponentially. We also used regular Dropout ([9]) layers and Spatial Dropout1D ([10]) layers in our architecture. The Spatial dropout is similar to the regular dropout but instead of randomly setting individual elements in a layer's output to zero it randomly sets entire feature maps to zero. Here's a picture that shows our entire model architecture for clarity.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 77, 256)	299,008
leaky_re_lu (LeakyReLU)	(None, 77, 256)	0
spatial_dropout1d (SpatialDropout1D)	(None, 77, 256)	0
dropout (Dropout)	(None, 77, 256)	0
lstm_1 (LSTM)	(None, 77, 256)	525,312
leaky_re_lu_1 (LeakyReLU)	(None, 77, 256)	0
spatial_dropout1d_1 (SpatialDropout1D)	(None, 77, 256)	0
dropout_1 (Dropout)	(None, 77, 256)	0
lstm_2 (LSTM)	(None, 128)	197,120
leaky_re_lu_2 (LeakyReLU)	(None, 128)	0
dropout_2 (Dropout)	(None, 128)	0
dense (Dense)	(None, 1)	129

Total params: 1,021,569 (3.90 MB)
Trainable params: 1,021,569 (3.90 MB)
Non-trainable params: 0 (0.00 B)

Figure 25: LSTM parameters

We also experimented with Bidirectional LSTM layers but didn't get better results and the training time and total parameters doubled. So we did not incorporate them in our results. We trained our model using early stopping that monitors the validation MSE and has 10 epochs patience. We gave it 100 epochs to train even though in most of our tests it stops before 50 or a little over 60 epochs.

Below we show prediction of the testing set of the model and the residuals plot:

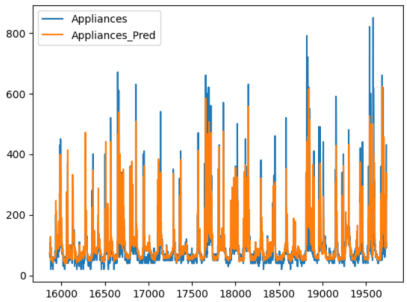

Figure 26: predictions VS targets

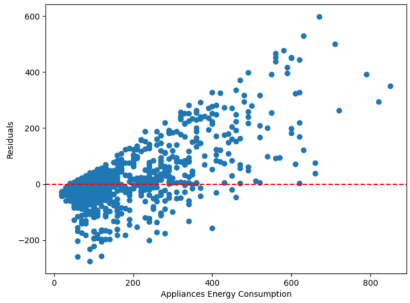

Figure 27: LSTM Residuals Plot

4.4 Gated Recurrent Unit (GRU)

We used the same model architecture here, but didn't achieve the same results. In fact the results were slightly worse when using GRU instead of LSTM layers. We assumed that this is because GRUs have fewer parameters, which, while making them computationally cheaper, can reduce their capacity to model complex relationships in the data. This is especially true for tasks that require a deep network like the one we use in this problem.

Below we show the metrics when applied on the testing sets (we only used 1 GRU model):

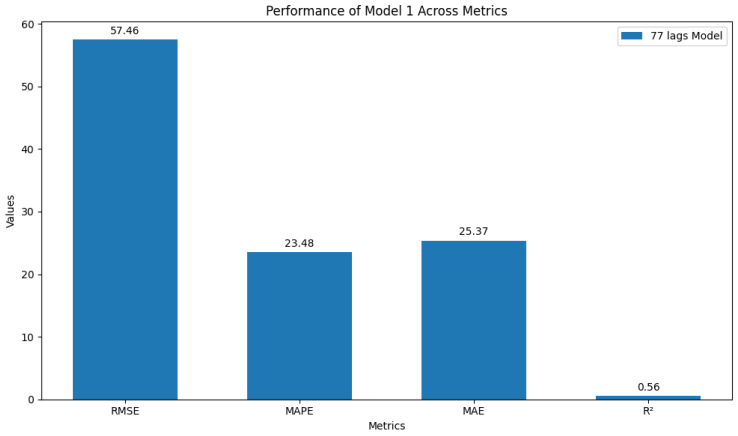

Figure 28: GRU metrics

In addition we show prediction of the testing set of the model and the residuals plot:

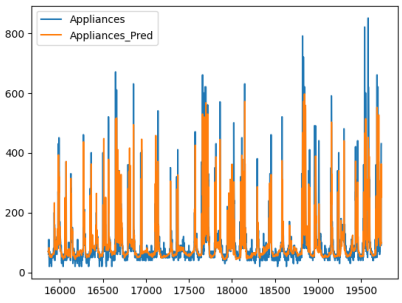

Figure 29: predictions VS targets

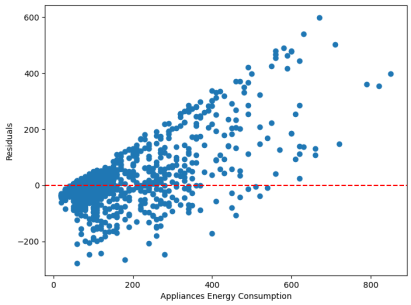

Figure 30: GRU Residuals Plot

4.5 Transformer

In this section we will firstly present the general overview of the transformer and why they are advantageous for capturing long-term dependencies and therefore can be utilized in time series forecasting. Then we will describe the model we used.

4.5.1 Overview on Transformers

A vital part of the transformer functionality is the **attention mechanism**. Self-attention is a mechanism that allows each token in an input sequence to focus on different parts of the sequence when generating its own representation. It calculates attention scores for all token pairs, determining how much influence each token should have on the others. This is done by creating three vectors for each token: **query**, **key**, and **value**. Specifically:

- **Query:** the current token for which the attention score is being calculated
- **Key:** the context for each token in the sequence
- **Value:** the information that is passed on to the output of the self-attention

The attention score for a token 'i' is computed by taking the dot product of its query vector (Q_i) with the key vectors of all other tokens (K_j). The result is scaled (divided by the square root of the dimension) and passed through a softmax function to obtain normalized attention weights. These attention weights are then used to weight the corresponding value vectors (V_j), and the weighted sum of values forms the output for the token 'i'. This process can be summarized mathematically by:

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

The Transformer model is built with an **encoder-decoder structure**. The **encoder** processes the input sequence and generates a set of representations, which are passed to the decoder. The **decoder** then generates the output sequence using these representations. The encoder and decoder both use multiple layers of self-attention and feed-forward networks. The encoder transforms the input into a context-rich representation, while the decoder generates the output sequence step by step, using both previously generated tokens and the encoder's outputs. This process can be visualized as:

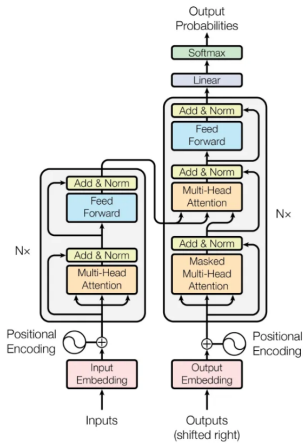

Figure 31: Transformer Architecture (encoder left, decoder right)

Transformers are also highly **scalable** due to their parallelizable structure. Unlike RNNs or LSTMs, which process sequences sequentially, the self-attention mechanism enables Transformers to process all tokens in parallel, leading to faster training times. Additionally, the model's architecture allows for easy scaling in terms of model size and the amount of data it can handle. As the model size increases, it can capture more complex relationships and patterns, making Transformers particularly effective for tasks with large datasets.

One task like these is **time series forecasting**, because of their ability at capturing long-term dependencies in sequences. In traditional models like RNNs or LSTMs, each token can only directly depend on its previous tokens. However, in Transformers, self-attention allows each token to directly interact with every other token in the sequence, regardless of their position. This means that every token can attend to distant tokens, giving the model a direct way to capture long-term dependencies without any inherent position-based limitations.

4.5.2 Transformer-Based Models

While we saw that the LSTM model does not benefit from too many past values, we expect that the transformer will because of its ability to better capture long-term dependencies. We experimented for **144 lags** and **1008 lags** which correspond to 1 day and 1 week respectfully. For both cases we use the following parameters:

- Adam optimizer with initial learning rate 10^{-6}
- Attention size of 64
- 8 attention heads

- 64 neurons on the MLP
- 1 layer of attention-MLP stacked

These parameters were found to be the best through fine-tuning. For both models we used **early stopping** with patience = 5, and a maximum of 100 epochs. Below we show the metrics when applied on the testing sets:

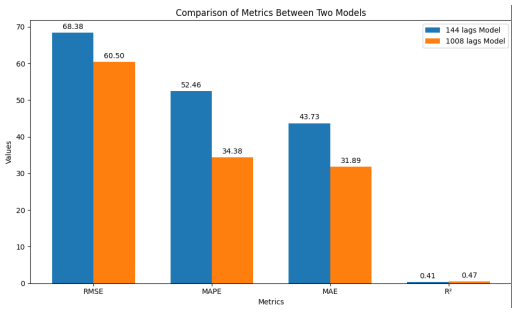

Figure 32: Transformer Metrics (144 lags VS 1008 lags)

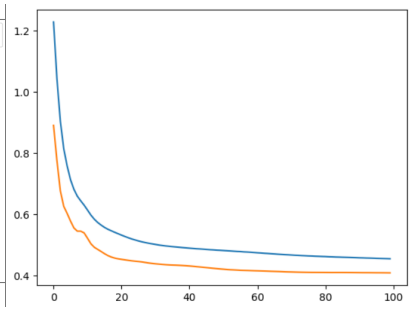

Figure 33: Transformer train - validation loss

As we see the transformer benefits severely from the inclusion of more past values, showing that it indeed captures the sequential context. We believe that it could be further improved if we included even more lags, but the computational time already reached over 1 hour using GPUs so we did not try it. Below we see the approximation of a test sample of the 1008-lag transformer:

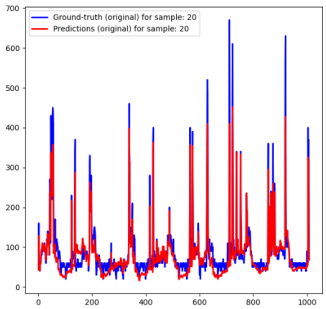

Figure 34: predictions VS targets

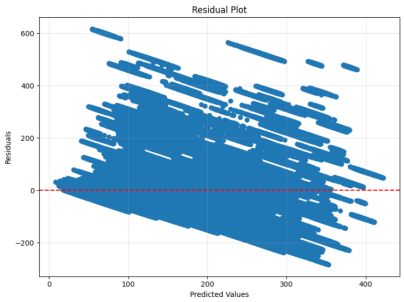

Figure 35: Transformer Residuals Plot

5 Results and Discussion

5.1 Model Comparison

In this section we will compare the models we presented. We will keep 1 model of each type, so we end up with: a **GBM** model (no lags), an **XGBoost** model (no lags), an **LSTM** model (window size: 77), a **GRU** model (window size: 77), and finally a **Transformer** model (window size: 1008).

The metrics we will use are the same metrics we use throughout this project: **Root Mean Squared Error** (RMSE), **Mean Abosolute Error** (MAE), **Mean Abosolute Percentage Error** (MAPE) and finally R^2.

Below we see the see the corresponding values for each model and each metric:

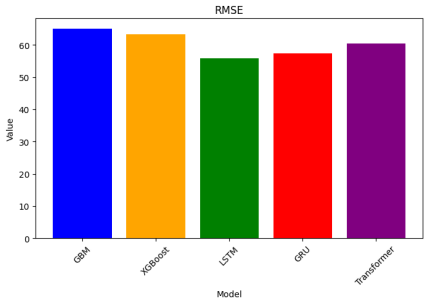

Figure 36: RMSE values

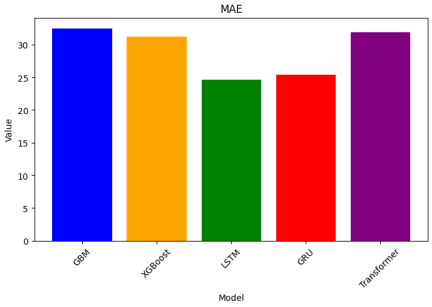

Figure 37: MAE values

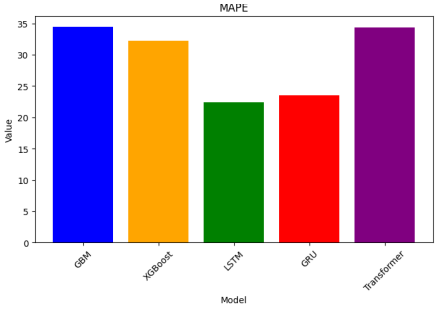

Figure 38: MAPE values

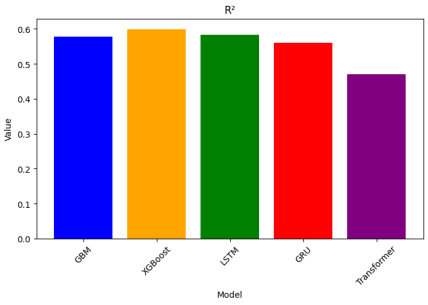

Figure 39: R^2 values

In addition, we see below the residual plots of all the models together:

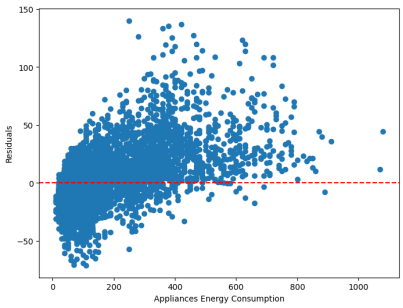

Figure 40: GBM residual plots

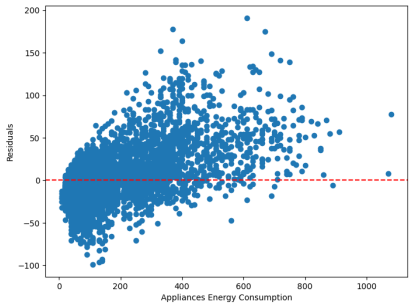

Figure 41: XGBoost residual plot

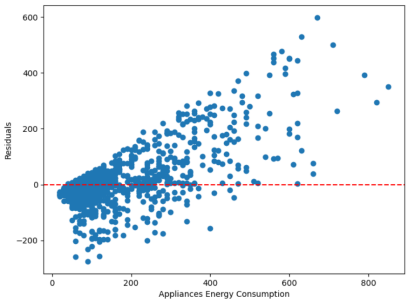

Figure 42: LSTM residual plot

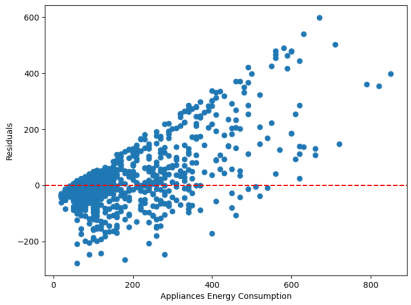

Figure 43: GRU residual plot

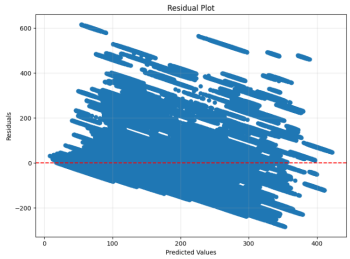

Figure 44: Sample prediction VS true value

5.2 Conclusions

The **LSTM** model proved to have the best RMSE, MAE, and MAPE scores, and the **XGBoost** model had the best R^2 score. If we look at the residual plots, we see that the gradient boosting methods concentrate the residual values closer to 0 than the deep learning approaches.

Overall we believe that out of these models the one that is best fitted for this task is the **LSTM**, due to its constantly good metric values. This is the model we fine-tuned the most. We also believe that the Transformer model could become even better through more fine-tuning, because it can capture the long-term dependencies better than all other models, but because of the copmutational time needed we did not manage to fine-tune it as much as we did with the LSTM.

Overall we conclude that the deep learning models and the transformer-based models **outperform** the traditional machine learning models for time series forecasting. Since GBM was the best model on the original paper, and all of the deep learning and transformer models were better, we understand that these methods indeed capture the past relationships better and use them to their benefit.

5.3 Limitations

Some limitations and challenges we encountered and believe that limit our models from becoming even better is the **data** we use. Specifically, if we had a larger dataset that contained information about multiple years, we could extract more accurate seasonal patterns (months/seasons/...). For example we may have identified a drop in the usage of energy at the times the family left the house for some days, like when they go on vacation. Also we are not sure that the extra data captured from the nearby airport weather station accurately represents the information of the house.

In addition, it would be beneficial if we would have had information about multiple houses and predicted the average, so we would see the general picture and the dataset would be less noisy and accurate.

Finally, as we mentioned in the Introduction, a **computational constrain** is the lack of full-time GPU availability to train the models, because when they expire the training time becomes exceptionally larger. And even when they are available, the Transformer model still takes a large amount of time to train for many lags. This is the reason that the fine-tuning was compromised for this model.

6 Running Instructions

We have implemented the following Jupyter files:

- ExploratoryDataAnalysis.ipynb
- GBM_XGBoost.ipynb
- LSTM_GRU.ipynb
- Transformer.ipynb

We include them in the zip folder we sent, and you can also find them is this link ([12]), along with some other files (datasets).

The 1st file contains the EDA analysis, when we run it it creates the file "final_data_no_PCA.csv", which also exists in the google drive link in case you want to use it straight away without waiting for the 1st file execution. The other files then use this dataset to run.

We implemented the models GBM and XGBoost in the same file because their functionalities are similar. This is also true for the LSTM and GRU models.

References

[1] "A Study of Time Series Models ARIMA and ETS", available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2898968.

[2] "Modelling time series trend with tree based models", available at: https://cienciadedatos.net/documentos/py49-modelling-time-series-trend-with-tree-based-models.

[3] "A Survey of Deep Learning and Foundation Models for Time Series Forecasting", available at: https://arxiv.org/abs/2401.13912.

[4] ExtraTreesRegressor sklearn documentation, available at: https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html

[5] MinMaxScaler sklearn documentation, available at: https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[6] Adam sklearn documentation, available at: https://keras.io/api/optimizers/adam/

[7] Leaky ReLU documentation, available at: https://keras.io/api/optimizers/Nadam/

[8] NAdam sklearn documentation, available at: https://keras.io/api/layers/activation_layers/leaky_relu/

[9] Dropout sklearn documentation, available at: https://keras.io/api/layers/regularization_layers/dropout/

[10] Spatial dropout sklearn documentation, available at: https://keras.io/api/layers/regularization_layers/spatial_dropout1d/

[11] Random ReLU documentation, available at: https://pytorch.org/docs/stable/generated/torch.nn.RReLU.html

[12] Google Drive link that contains all the files, available at: https://drive.google.com/drive/folders/1X5LhsC2PC3pGS1Debb8enZTvGZsyf726?usp=sharing