ADAML 2025 Project Work - Predicting House Energy Consumption

Toni Koskinen
Veeti Rajaniemi
Amanda Valtanen

December 15, 2025

1 Introduction

In this work, the aim is to forecast electric power consumption of a house for next 24 hours using previous energy consumption data.

2 Materials and methods

Next, the materials and methods used in the work are described.

2.1 Data description

The original time-series dataset contains approximately 2,000,000 measurements from a household between December 2006 and November 2010, with one measurement recorded per minute. Thus, all the variables are synchronized. In addition to date and timestamp, it includes 6 variables as follows:

- Global active power - minute-averaged active power (the target variable)
- Global reactive power - minute-averaged reactive power
- Voltage - minute-averaged voltage
- Global intensity - minute-averaged current intensity
- Sub-metering 1 - energy sub-metering corresponding to kitchen appliances
- Sub-metering 2 - energy sub-metering corresponding to the laundry room
- Sub-metering 3 - energy sub-metering corresponding to an electric water heater and an air-conditioner.

The dataset contains individual timestamps with missing values. Also, there are a few time periods when the measurements are missing for several straight days.

2.2 Time-series decomposition

Time-series decomposition refers to breaking down a time series into its different components in order to understand what causes variation of a specific variable. The components are trend, seasonality, and residual. The trend indicates long-term variation, seasonality indicates repeating seasonal variation, and residual indicates random variation that cannot be explained by trend or seasonality. Time-series decomposition can be represented as follows

$$y_t = S_t + T_t + R_t, \tag{1}$$

where y_t is the observed data, S_t is the component describing seasonality, T_t is the trend component, and R_t is the residual component, all at the time t (Hyndman and Athanasopoulos, 2018).

2.3 Dividing a long time-series into sub-samplings

In time-series analysis, dividing a long series into shorter sub-samplings is a common strategy to capture local patterns, improve model performance, and handle non-stationarity. According to Hyndman and Athanasopoulos (2018), many real-world time series exhibit evolving dynamics where model parameters, trends, or seasonal effects change over time. Sub-sampling allows models to adapt to these local structures rather than forcing a single global fit.

Fixed-length rolling windows are a common method for dividing a long time-series into shorter, overlapping segments that capture local patterns. In this approach, the series is split into

consecutive windows of equal length, and the window is then shifted forward one time step at a time. Rolling windows are particularly useful when the goal is to model short-term dependencies or extract features that describe recent trends. (Institute (2023))

The time series can also be divided into sub-samplings based on seasonal characteristics or other detected sources of variation. According to Sharma (2022), season-based segmentation divides a time series into segments that correspond to different seasonal periods, such as weekdays, weekends, or distinct seasonal cycles. Another segmentation strategy is change-point detection, which identifies points where the statistical properties of the series change, allowing the data to be split into segments with different behaviours (Aminikhanghahi and Cook (2017)).

Seasonality is an important factor in deciding how a time series should be split into sub-samplings. When seasonal patterns are strong and repeat regularly, each sub-sampling should include a whole number of seasonal cycles so that the periodic structure is preserved. (Hyndman and Athanasopoulos (2018))

When seasonality changes over time, sub-sampling can help by separating the series into periods where the seasonal pattern is more stable. If seasonality is weak or inconsistent, it may be better to use flexible windowing or change-point detection instead of strict season-based segmentation Cleveland et al. (1990).

2.4 Seasonal naïve predictor

The model considered as the baseline for predicting future consumption is seasonal naïve predictor. With this model, the predictions for future values are calculated using the previous observation of the same season. The predictions can be calculated using

$$\hat{y}_{t+T} = y_{t+T-h}, \tag{2}$$

where T is the prediction horizon and h is the length of the season. As the electricity consumption has seasonality in daily and weekly values this model can be considered as good model for benchmarking.

2.5 LSTMs and Data Normalization

Long short-term memory (LSTM) model is a recurrent neural network (RNN) model trying to overcome the vanishing gradient problem. LSTM includes memory cells consisting of gates deciding which input values effect the memory states and the model output.

Even though LSTM models have multiple strengths in powerful learning, nonlinear mapping and sequence modeling, they often struggle at finding trends and seasonality in time-series data. One way to overcome this is using hybrid models compounding a machine learning model, data pre-processing methods and an optimization algorithm. Some approaches do the pre-processing to learn seasonality and trends before using LSTM-models with the data while others include it to the model. (Wu et al. (2024))

Multiple different standardization methods can be used with LSTMs, but min-max normalization and z-score normalization seem to be the most used techniques. In their study, Tawakuli et al. (2025) compared different standardization methods for LSTM's by predicting carbon monoxide concentration using air quality data. Best results were achieved using z-score standardization and robust standardization, but also other methods such as min-max normalization and Box-Cox led to promising results, highlighting the possibility of using different methods.

In another study, Pranolo et al. (2024) compared min-max and z-score normalization techniques with LSTM models predicting energy load with different energy generation attributes. The results state that min-max normalization outperformed z-score in multiple scenarios.

These examples show that for different problems, different normalization methods can be the optimal ones with LSTMs.

2.6 Seq2Seq LSTM

To address multi-step forecasting problem with LSTM-models, we aim to implement Seq2Seq LSTM model. This model has been proposed previously by Masood et al. (2022) for forecasting energy consumption of households. In this architecture, the model consists two LSTM layers: encoder and decoder. The encoder uses value from input sequence x_t and previous state h_{t-1} to predict the states h_t and c_t. These values are passed to decoder, which calculates output sequence y_t using the previous state and weighted combination of the encoder outputs. The model's benefit is that it can learn to produce multi-step predictions. (Masood et al., 2022)

2.7 Transformer model

Our Transformer is an encoder-only architecture implemented with the TransformerEncoder module from the PyTorch library (PyTorch, 2025). Each time step in the input sequence is projected into an embedding space and supplemented with positional information so that the model can represent temporal order. The Transformer encoder then processes the entire sequence in parallel using multi-head self-attention and feedforward layers, so that the model learns dependencies between distant time steps. After encoding, the sequence representation is aggregated and passed through a fully connected layer to produce the 24-hour forecast.

2.8 Data partition plan for cross-validation

The data will be divided into several folds for cross-validation. We will start with a basic approach doing cross-validation on a rolling basis with expanding window. We will use 5 folds, and the size of each test set will be a fifth of one year (73 days). In practice, it means that we first use all data but the last year (meaning last 365 days of the dataset) as a training set, and the first fifth of the last year of the data is the first test set. During the second fold, we add the previous test set into our training data, and try to predict the second fifth of the last year. This process is repeated so that across five folds, the test periods together span the entire last year of the data.

With this approach, the validation process includes the full seasonality of one year. Of course, we can try to change the number of folds and the size of test sets later. After cross-validation is done, we can also try to predict values for the entire last year to see if the model performs well for a longer period. One option is to leave some data as fully unseen for our model so we can test it. These details will be decided later when the model type is known.

3 Results

This section shows the results of the work.

3.1 Data preprocessing and visualization

If the missing values were individual measurements or occurred over a short gap, maximum of 12 hours, they were approximated by simple linear interpolation. If the values were missing for a longer period, we used the value from one week before, as it correlates most with the missing value based on autocorrelation, which will be shown later.

As the goal of the work is to predict daily consumption, we decided to modify the original minute-level data into hourly measured values. Thus, the mean value of each variable during each hour is used.

The dataset after these modifications can be seen in Figure 1. Based only on these visualizations, we can see some kind of seasonality in almost each variable.

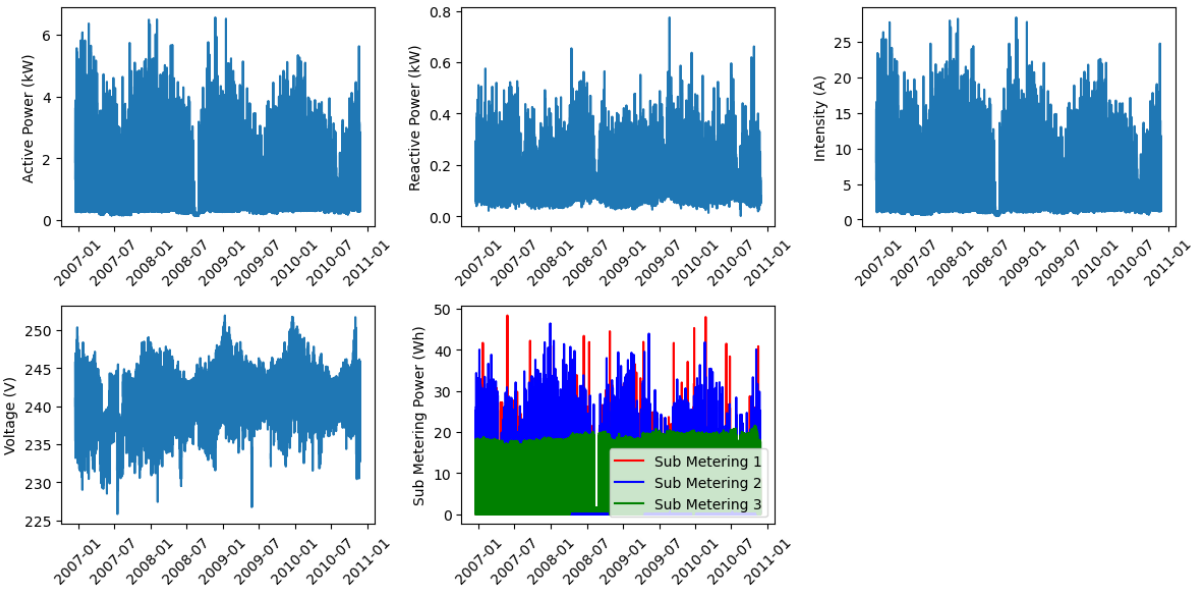

Figure 1: Variables after handling missing values and modifying data into hourly averages.

3.2 Time-series decomposition

The time-series data of the target variable is used to analyze and understand its temporal behavior. Time-series decomposition was first applied on data containing hourly averages of target variable, using a season length of one day (24 hours). The result of this is shown in the Figure 2. The figure shows that the trend still includes seasonal variability, which could be interpreted as annual, because at the same time of year, the trend behaves repeatedly in the same way. For this reason, time-series decomposition is not satisfactory, and we must try to define the length of the season so that the trend does not include seasonal variation.

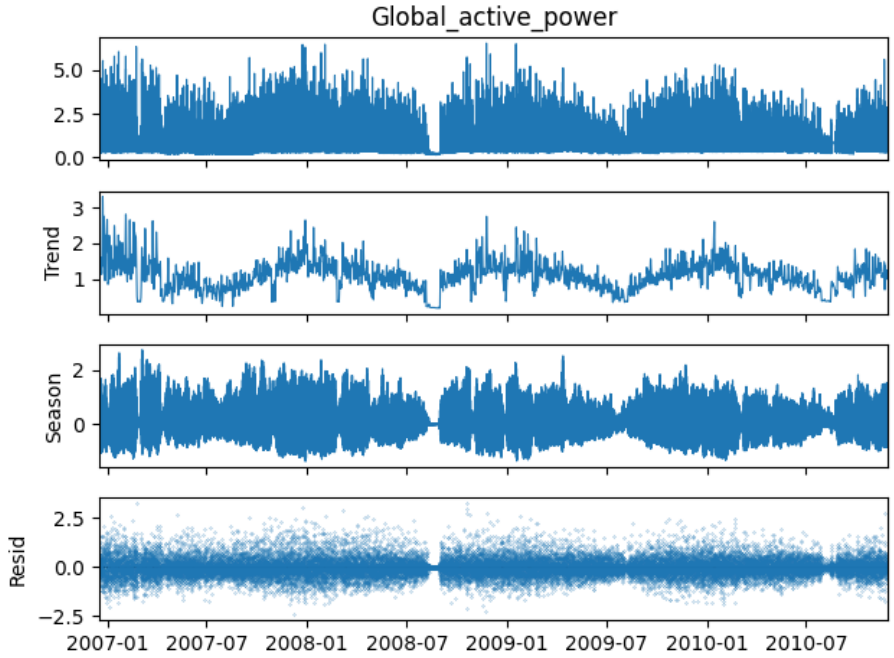

Figure 2: Time-series decomposition of the Global Active Power variable using hourly average values, and a day as a season length.

In the Figure 3, time-series decomposition is created for hourly average data, with the season length being one year (24 hours * 365). In Figure 4, daily averages have been taken from the data for the explained variable and a time-series decomposition has been performed using as a season length a year (365 days). Both annual season length distributions show a similar, slightly decreasing trend. Seasonality shows annual variation, with values of the target variable falling in summer and peaking in winter.

In the figure, the residuals appear to be quite evenly distributed around zero, although in the middle of the time series the residuals clearly show greater variance compared to the beginning and end. If the figure showed a single large residual that did not repeat itself seasonally, that sample could be considered a possible outlier. Trend and seasonal components should appear smooth without any distinct outliers, as such outliers are expected to appear in the residuals. At this moment, there are no clear potential outliers visible in the figure, but they may still appear in later analysis.

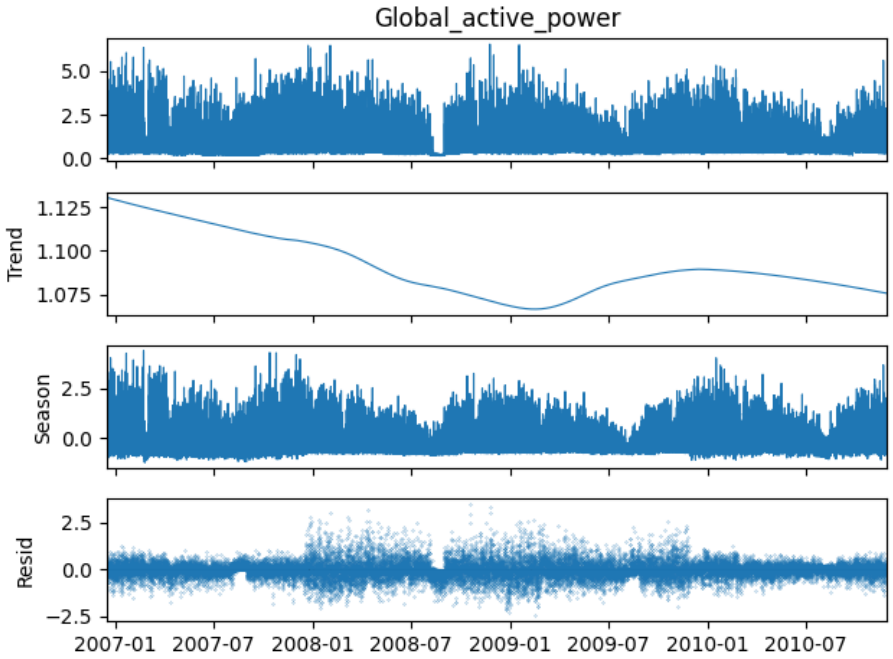

Figure 3: Time-series decomposition of the Global Active Power variable using hourly average values, and a year as a season length.

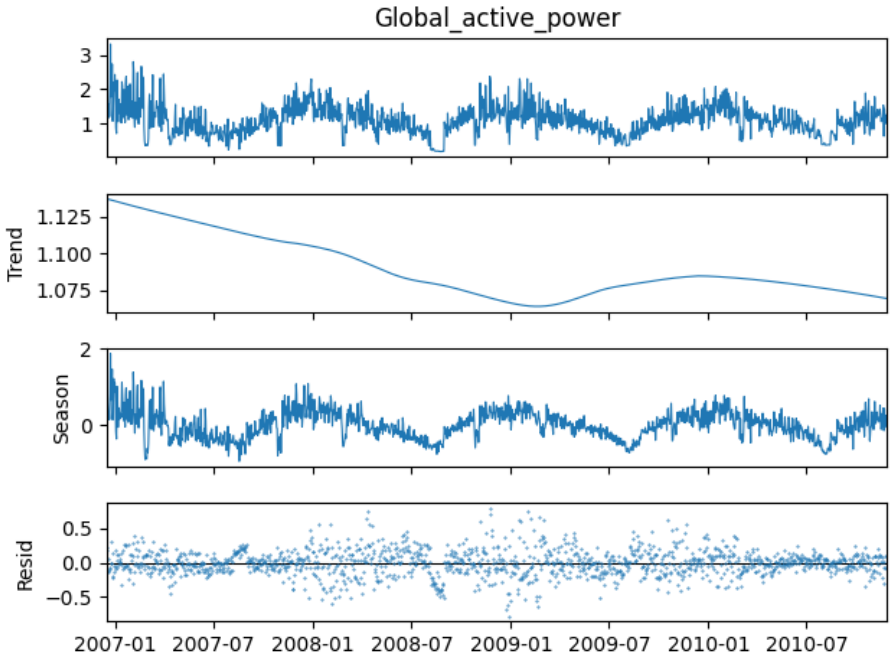

Figure 4: Time-series decomposition of the Global Active Power variable using daily average values, and a year as a season length.

3.3 Autocorrelation analysis

Autocorrelation function can be used to evaluate relationships between past and present observations. Significant correlation between these indicate that the time series is not a random walk.

Before using autocorrelation function, the stationarity of the observations was evaluated using Augmented Dickey-Fuller (ADF) test. For Global active power, the ADF Statistic of -11.12 and p-value of $3 \cdot 10^{-20}$ indicated stationary observations, so no time series transformations were needed before applying the autocorrelation function.

The autocorrelation function plots for Global active power, using 48 and 168 lags are shown in Figure 5. Each lag represents delay of 1 hour, which means that 24 lags correspond to a one-day delay and 168 lags a one-week delay.

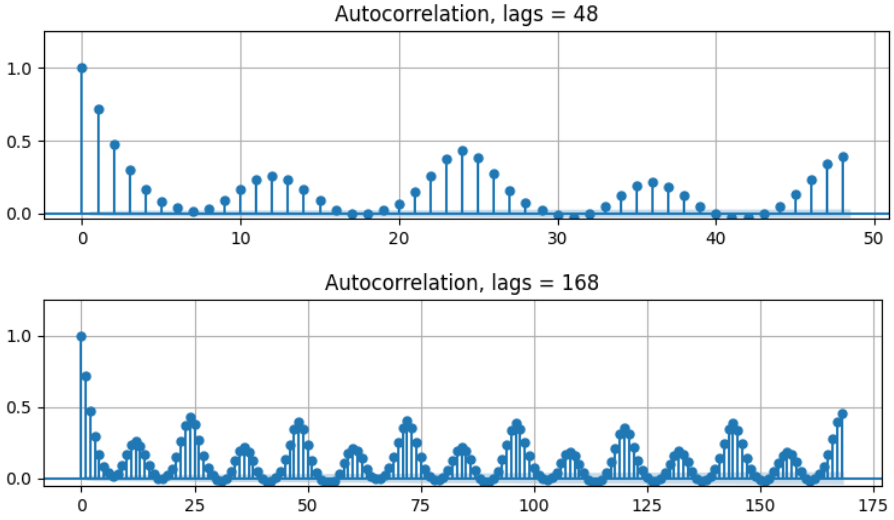

Figure 5: Autocorrelation function plots for Global active power, using 48 and 168 lags.

From the figure we could see that there were significant relationship between the past and the present observations, especially in daily and weekly periods. These significant relationships occurred also in explanatory variables.

3.4 Baseline model

For the seasonal naïve predictor we used one week as the length of the season which in the used dataset corresponds $h = 168$ hours. This means that, for the predictions the previous value from same weekday and hour was used.

The predictions were calculated for a time horizon of one week with a resolution of one hour. An example of the model's predictions is shown in Figure 6.

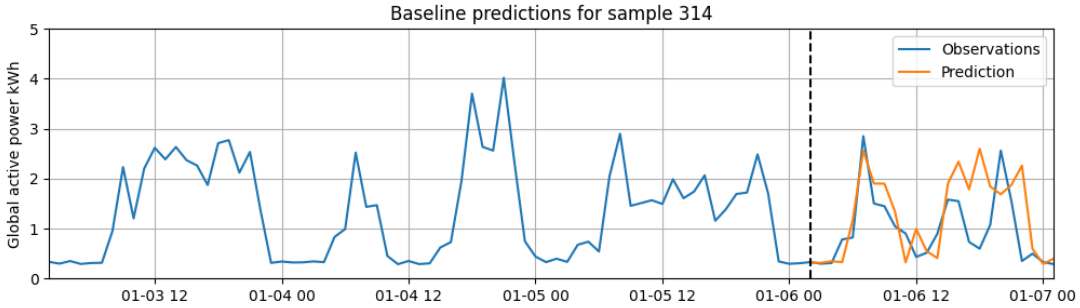

Figure 6: Observations and predictions using seasonal naïve predictor. The beginning of the predictions has been marked using a dashed line.

In addition to predictions, the root-mean-squared errors (RMSE) and mean arctangent absolute percentage errors (MAAPE) were calculated for each prediction horizon. The mean value for RMSE-values was 0.81 kWh and for MAAPE-values 44 %.

3.5 RNN model

A recurrent neural network was the first trained model. The model consists of two RNN layers, one for encoding the input features and the other for decoding and calculating the prediction for the horizon of 24 hours. To select optimal the optimal lookback window, we used cross-validation as described in the Chapter 2.8. The results from the cross-validation are shown in Table 1.

Lookback (hours)	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
24	0.890 ± 0.230	0.800 ± 0.206	0.624 ± 0.140	0.511 ± 0.135	0.801 ± 0.180
48	0.890 ± 0.208	0.788 ± 0.190	0.635 ± 0.139	0.553 ± 0.127	0.845 ± 0.194
72	0.855 ± 0.245	0.797 ± 0.191	0.559 ± 0.152	0.544 ± 0.154	0.744 ± 0.236
96	0.872 ± 0.220	0.809 ± 0.204	0.644 ± 0.140	0.540 ± 0.155	0.768 ± 0.215
120	0.921 ± 0.204	0.713 ± 0.212	0.630 ± 0.123	0.531 ± 0.135	0.721 ± 0.193
144	0.898 ± 0.201	0.723 ± 0.209	0.667 ± 0.118	0.486 ± 0.156	0.801 ± 0.190
168	0.820 ± 0.237	0.726 ± 0.206	0.562 ± 0.156	0.503 ± 0.160	0.723 ± 0.211

Table 1: Average root-mean-squared errors and standard deviations for each fold used in training of the RNN model with different lookback windows. The results are in kWhs and have been rounded to three decimals.

Based on the results from the cross-validation, the lookback window selected for the RNN model was 168 hours. First 4 years of the dataset were used for training the model and last year for testing. The data was scaled using min-max scaling and in addition sine and cosine tranformations about the hour, weekday and the day of the year were added as features. With

this setup, we trained the RNN model using learning rate of $5 \cdot 10^{-5}$, batch size of 128, hidden layer size of 32, and 150 epochs. The training and validation losses calculated during the training process are shown in Figure 7.

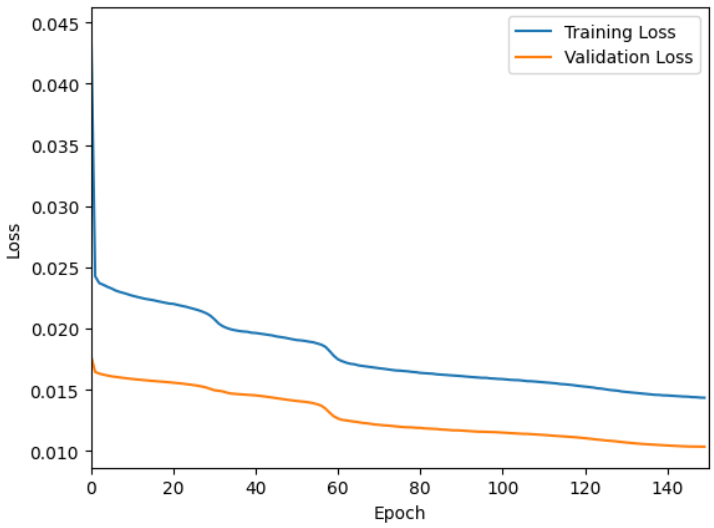

Figure 7: Training and validation losses for RNN model.

An example of RNN predictions for the test data is shown in Figure 8.

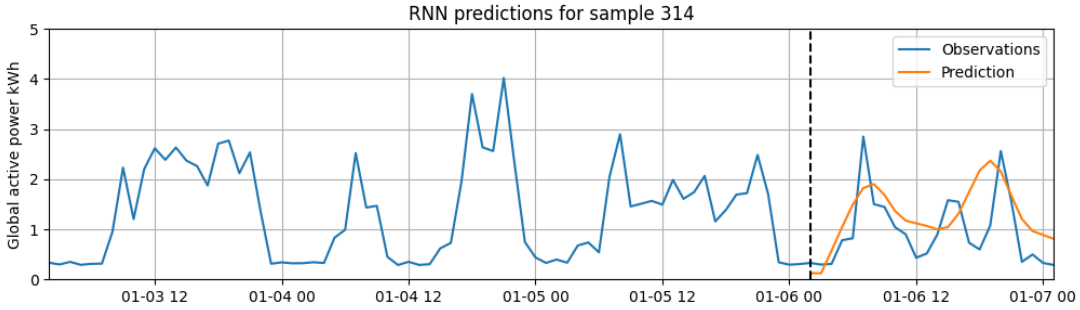

Figure 8: Observations and predictions using a RNN model.

When examining multiple test samples it was noticed that the model was able to capture daily seasonalities in predictions but not sudden spikes in the electricity consumption.

3.6 LSTM model

To find the optimal lookback window for the LSTM-model, the model was trained using five different folds as described in Chapter 2.8. For each proposed lookback window, the model was trained and its performance was evaluated using root-mean-squared errors for the test samples of the next fold. The results from cross-validation are showed in Table 2.

Lookback (hours)	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
24	0.804 ± 0.211	0.672 ± 0.217	0.561 ± 0.132	0.512 ± 0.135	0.654 ± 0.181
48	0.826 ± 0.204	0.715 ± 0.212	0.554 ± 0.128	0.487 ± 0.136	0.650 ± 0.175
72	0.798 ± 0.224	0.675 ± 0.222	0.578 ± 0.126	0.502 ± 0.129	0.667 ± 0.169
96	0.835 ± 0.190	0.683 ± 0.225	0.558 ± 0.129	0.479 ± 0.143	0.659 ± 0.163
120	0.794 ± 0.237	0.665 ± 0.223	0.544 ± 0.129	0.493 ± 0.138	0.658 ± 0.172
144	0.864 ± 0.191	0.686 ± 0.237	0.564 ± 0.124	0.499 ± 0.142	0.656 ± 0.176
168	0.835 ± 0.192	0.712 ± 0.195	0.544 ± 0.128	0.503 ± 0.125	0.662 ± 0.175

Table 2: Average root-mean-squared errors and standard deviations for each fold used in training of the LSTM model with different lookback windows. The results are in kWhs and have been rounded to three decimals.

From the table we could see that the differences between the outcomes of the lookback windows were small. As the model had the smallest average RMSE-value over all folds when the lookback window was 120 hours, it was chosen as the lookback window size.

In addition to the min-max scaled previous observations, information about the weekday, hour and day of the year were passed to the model using sin and cosine transforms. The model was trained using 100 epochs, with batch size of 128 and learning rate of $5 \cdot 10^{-5}$. Utilizing the validation data, each iteration when the validation loss improved the model weights were saved. The training and validation losses from the training are visualized in the Figure 9.

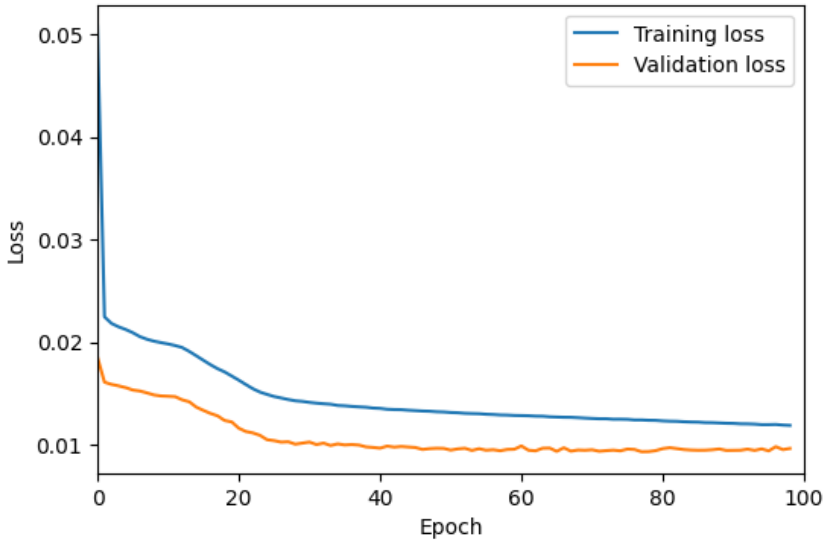

Figure 9: Training and validation losses for LSTM model.

After the training, the predictions for the test data were calculated for horizon of 24 hours. An example of the model's predictions is shown in Figure 10.

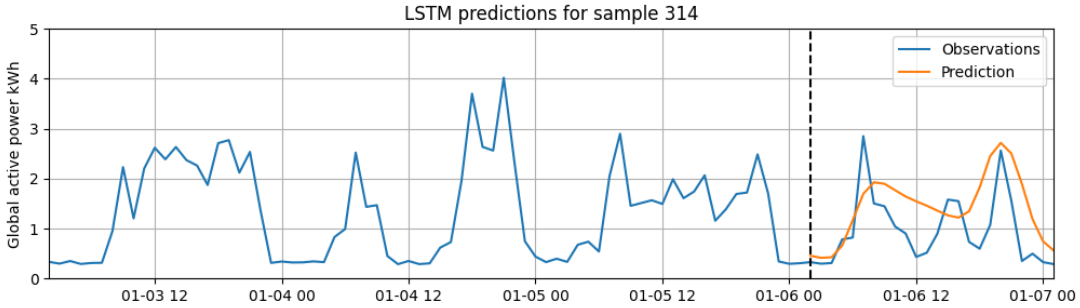

Figure 10: Observations and predictions using Seq2Seq LSTM model. The beginning of predictions has been marked using a dashed line.

When plotting samples of the model's predictions it could be seen that the trained model was able to capture average daily consumption but not predict sudden increases or drops well in the consumption.

3.7 Transformer model

Before training the actual Transformer model for forecasting electricity consumption, the optimal lookback window was determined using cross-validation as described in Chapter 2.8. The results from the cross-validation are shown in Table 3.

Lookback (hours)	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
24	0.764 ± 0.212	0.667 ± 0.215	0.547 ± 0.134	0.482 ± 0.129	0.652 ± 0.172
48	0.753 ± 0.207	0.674 ± 0.212	0.548 ± 0.133	0.476 ± 0.126	0.647 ± 0.183
72	0.766 ± 0.200	0.680 ± 0.212	0.553 ± 0.131	0.472 ± 0.132	0.652 ± 0.181
96	0.761 ± 0.207	0.679 ± 0.210	0.545 ± 0.134	0.473 ± 0.127	0.641 ± 0.178
120	0.768 ± 0.206	0.679 ± 0.210	0.545 ± 0.135	0.480 ± 0.129	0.655 ± 0.172
144	0.770 ± 0.207	0.691 ± 0.200	0.542 ± 0.134	0.480 ± 0.133	0.660 ± 0.182
168	0.765 ± 0.215	0.694 ± 0.206	0.545 ± 0.134	0.474 ± 0.139	0.649 ± 0.181

Table 3: Average root-mean-squared errors and standard deviations for each fold used in training of the Transformer model with different lookback windows. The results are in kWhs and have been rounded to three decimals.

Based on the cross-validation results, the lookback window selected for the Transformer model was 96 hours. This means that a Transformer encoder, which uses a 96-hour history window of past measurements to predict the next 24 hourly values, was created. Information about the weekday, hour and day of the year were included using sin and cosine transforms, following the same approach as in the LSTM model. The Transformer model was trained using mean squared error (MSE) loss on MinMax-scaled data, with separate training, validation and test sets. The model was trained using 100 epochs, with batch size of 128 and learning rate of $5 \cdot 10^{-5}$. The training and validation losses calculated during the training of the model are shown in Figure 11.

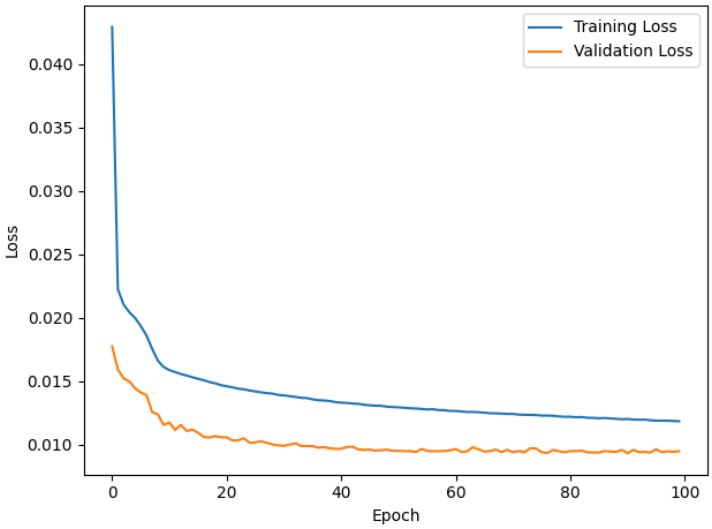

Figure 11: Training and validation losses for Transformer model.

The trained Transformer model achieved an average test RMSE of 0.604, representing the average error per 24-hour prediction vector. Figure 12 shows one representative 24-hour forecasts from test set. For a single 24-hour forecast horizon, the predicted curve captures the main daily pattern and timing of peaks, although some local peaks and troughs are slightly smoothed compared to the true values.

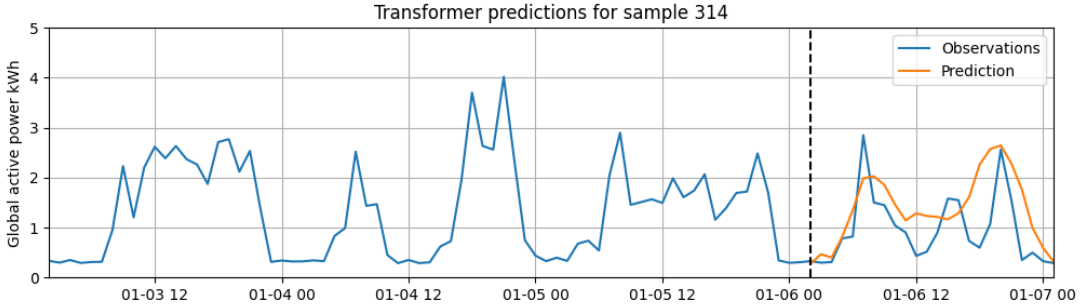

Figure 12: Observations and predictions using Transformer model. The beginning of predictions has been marked using a dashed line.

3.8 Model performance comparison

The performance between the models was compared by calculating root-mean-squared-errors and mean arctangent absolute percentage errors for the test data. The root-mean-squared errors for the predictions of the test data are shown in Figure 13.

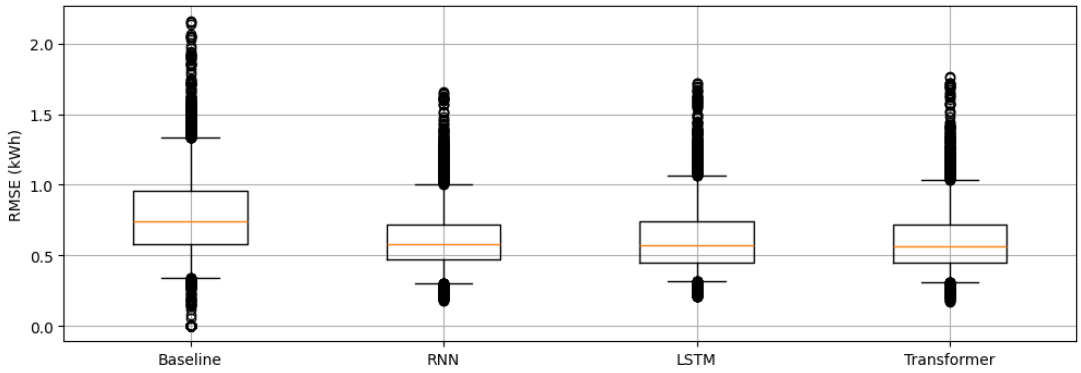

Figure 13: Root-mean-squared-errors for the predictions of the test data for each model. The medians of the values have been marked with orange lines and the whiskers indicate 5th and 95th percentiles of the values.

Based on the RMSE-values we could see that each of the proposed models produced on average more accurate results than the baseline model. The proposed models had similar performance. The mean arctangent absolute percentage errors for the predictions of the test data are shown in Figure 14.

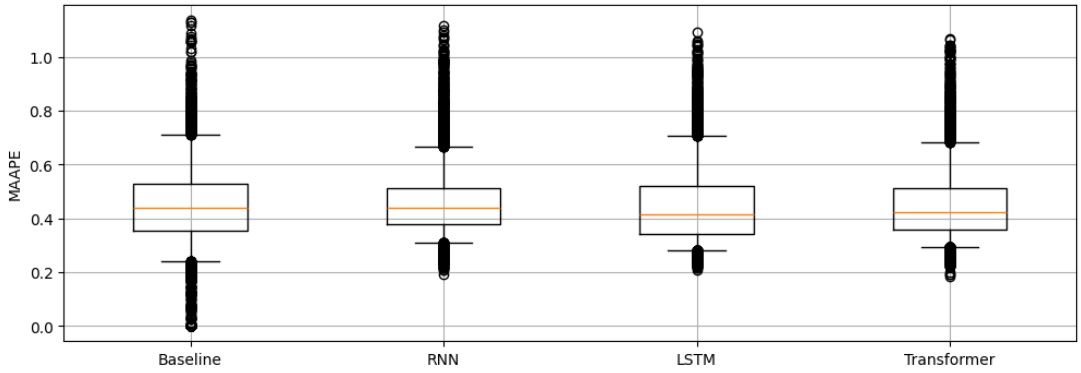

Figure 14: Mean arctangent absolute percentage errors for the predictions of the test data for each model. The medians of the values have been marked with orange lines and the whiskers indicate 5th and 95th percentiles of the values.

From the MAAPE-values we could see that the models had similar performance, but the baseline model could also perform some very accurate results even though they were not in the 90% of the values. However some of these accurate predictions might have been caused by the interpolation of the missing values in the dataset.

4 Discussion

Based on the results all of the proposed neural network architectures were able to produce more accurate results than the naïve seasonal predictor. From the results it could be also notified that the even though the baseline model performed some very accurate forecasts, these might have been samples where the interpolation methods have been used to fill missing observations. All of the proposed models were able to capture seasonal patterns, but not predict sudden spikes. The lookback windows for each model were selected based on the cross-validation results, but there were not significant differences between the performances so it can be possible that the most suitable lookback windows were not reached. For the used forecast horizon of 24 hours no significant differences between the performance of the proposed models were reached.

References

Aminikhanghahi, S. and Cook, D. J. (2017). A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51(2):339–367.

Cleveland, R. B., Cleveland, W. S., McRae, J. E., and Terpenning, I. (1990). Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–73.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: Principles and Practice (2nd ed)*. OTexts, Melbourne, Australia. Accessed: 13 November 2025.

Institute, B. A. (2023). Rolling window features for time series. Accessed: 15 November 2025.

Masood, Z., Gantassi, R., and Choi, Y. (2022). A multi-step time-series clustering-based seq2seq lstm learning for a single household electricity load forecasting. *Energies*, 15(7):2623.

Pranolo, A., Setyaputri, F., Paramarta, A., Triono, A., Fadhilla, A., Akbari, A., Utama, A., Wibawa, A., and Uriu, W. (2024). Enhanced multivariate time series analysis using lstm: A comparative study of min-max and z-score normalization techniques. *ILKOM Jurnal Ilmiah*, 16(2).

PyTorch (2025). Pytorch documentation. `https://docs.pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html`. Accessed: 2025-02-10.

Sharma, A. (2022). A novel seasonal segmentation approach for day-ahead load forecasting. *Applied Energy*. Segmenting time series based on seasonality for forecasting.

Tawakuli, A., Havers, B., Gulisano, V., Kaiser, D., and Engel, T. (2025). Survey:time-series data preprocessing: A survey and an empirical analysis. *Journal of Engineering Research*, 13(2):674–711.

Wu, Y., Meng, X., Zhang, J., He, Y., Romo, J. A., Dong, Y., and Lu, D. (2024). Effective lstms with seasonal-trend decomposition and adaptive learning and niching-based backtracking search algorithm for time series forecasting. *Expert Systems with Applications*, 236:121202.