POLITECNICO
MILANO 1863

# Financial Engineering: Final Project

## ENERGY PRICE FORECASTING

**Irene Ferrari, Mattia Fioravanti, Giorgio Enrico Maria Serva**

Group: 15
Professor: Prof. Roberto Baviera, Dott. Alessandro Brusaferri
Academic Year: 2023-24

# Contents

# 1 | Introduction

Forecasting energy prices is a complex yet essential practice for maintaining the stability and efficiency of energy markets, demanding robust and innovative methodologies. This project employs various probabilistic forecasting techniques, leveraging historical data to build models that can accurately predict energy prices.

The dataset under study includes a range of critical features for energy forecasting, such as target energy market prices, future forecasts for energy load, solar and wind energy, and cyclic transformations of temporal features. Spanning multiple years from 2015 onward, this dataset provides a comprehensive basis for training and validating forecasting models effectively.

By applying best practices in machine learning, the models developed predict future values based on previously observed data points. This approach captures trends, seasonal patterns, and irregular fluctuations. Given the volatility and complexity of energy markets, probabilistic forecasting techniques are particularly valuable as they offer not only point estimates but also a measure of uncertainty around the predictions.

Throughout this project, a variety of forecasting models were explored, including deep learning and econometric models such as Autoregressive with Exogenous Inputs (ARX), Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and customized networks, also experimenting with embedded models. These models were rigorously tested to achieve optimal performance. Advanced optimization techniques enhanced model accuracy and reliability, while fine-tuning strategies further refined the models and improved their predictive capabilities. Additionally, selecting the best loss function was crucial for minimizing prediction errors and ensuring robust model performance.

Finally, the Delta Coverage metric was used to evaluate the models' capabilities, ranking them alongside other metrics discussed throughout this report.

# 2 | Statistical Analysis

Before developing any model, it is crucial to analyze the data that will be used for training. In probabilistic forecasting, it is particularly important to examine the shape of the distribution to be forecasted. Our statistical analysis is divided into two parts:

**Distribution Analysis**: This part evaluates whether the price distribution can be fitted to a Normal distribution or a Johnson distribution.
**Correlation and Autocorrelation Analysis**: This part examines the correlation and autocorrelation within the data, which will be provided as input to the model.

## 2.1. Distribution Analysis

In this section, we examine the distribution of spot prices across different seasons. Our analysis focuses on the year 2017, which serves as the primary dataset for training and validating most of our models.

We have opted to designate January as representative of the winter season, April for spring, August for summer, and October for autumn.

Initially, we decomposed the price time series into three components: trend, seasonality, and residuals.
To decompose having seasonality with multiple periods the *multiseasonal_ decomposition* function was implemented. In our case the seasonal part was chosen to have both a daily period and a 12 hours period. Subsequently, we subjected the data to the Jarque Bera test to assess its adherence to Gaussian distribution and the Kolmogorov Smirnov test for the Johnson distribution (JSU), both conducted at a significance level of $\alpha = 5\%$.

The decomposition plots are the following:

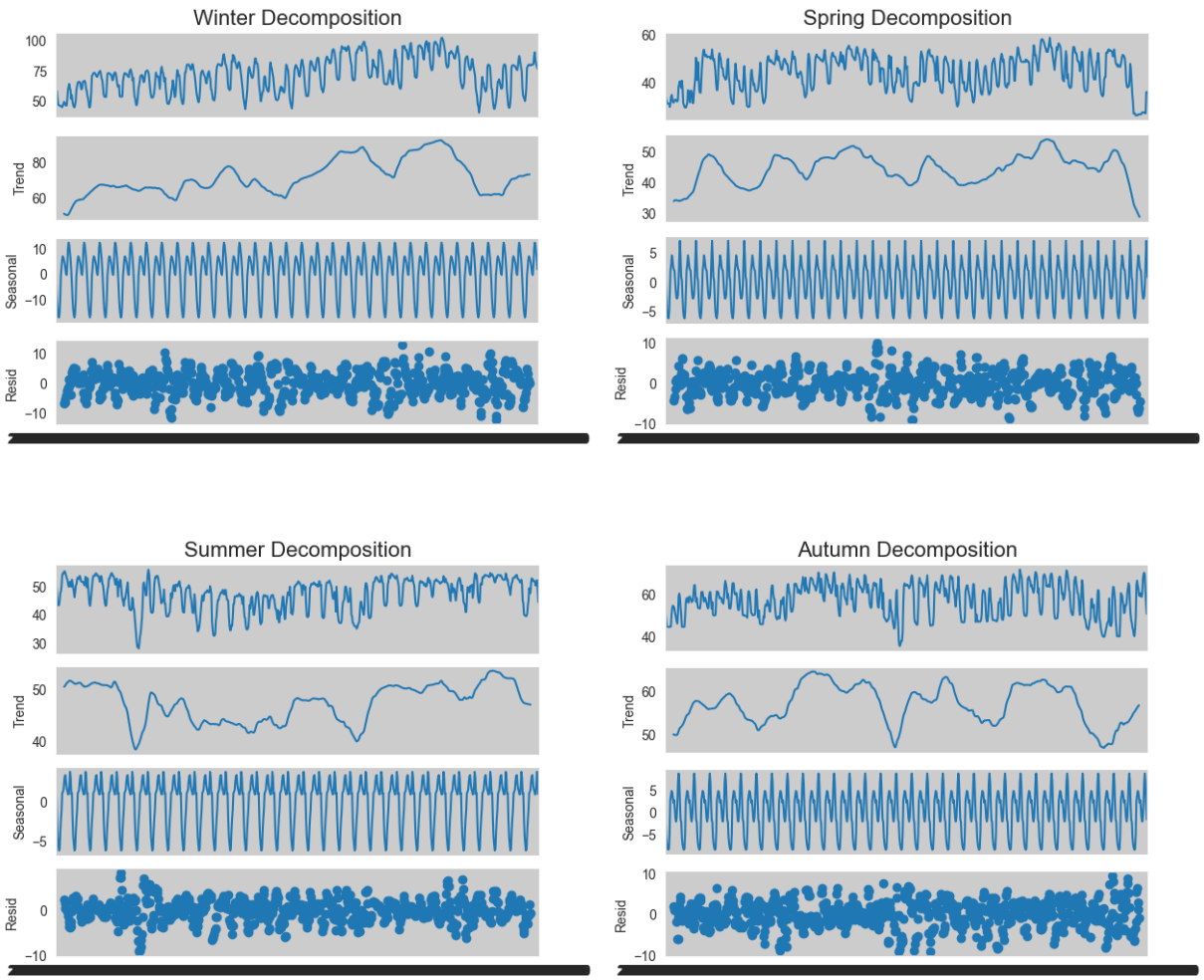


Figure 2.1: Decomposition plots

In these plots, we can observe distinct daily seasonal patterns for each season. Additionally, the residuals are significantly lower during the summer compared to the other seasons. Specifically, we calculated the average and standard deviation for both observed prices and residuals across all seasons, as presented in the table below:

| Winter | Average | Std |
|---|---|---|
| Price | 71.524 | 13.898 |
| Residuals | -0.024 | 4.066 |

| Spring | Average | Std |
|---|---|---|
| Price | 44.178 | 7.336 |
| Residuals | 0.001 | 3.098 |

| Summer | Average | Std |
|---|---|---|
| Price | 47.434 | 5.501 |
| Residuals | -0.017 | 2.442 |

| Autumn | Average | Std |
|---|---|---|
| Price | 56.992 | 7.681 |
| Residuals | -0.007 | 3.238 |

Table 2.1: Prices and Residuals Average and Standard deviation

From these tables, we can observe that the average price, as expected, is higher during the colder seasons, such as winter and autumn. Furthermore, we can confirm that the season with the highest volatility is winter, while summer is the least volatile.

Given these decompositions, we proceeded to apply statistical tests on the residuals. The tests were first applied to the residuals divided by each hour of the day and then to all the residuals combined. This approach helps to determine whether the residuals belong to the same distribution or if each follows a different distribution.

In general, the null hypothesis for both the Jarque-Bera and Kolmogorov-Smirnov tests is accepted at almost every hour, except during the summer period. In the summer, the Jarque-Bera test rejects the null hypothesis four times when considering the hourly divisions, and it completely rejects the null hypothesis in the general case (i.e., the p-value is 0%).

The Kolmogorov-Smirnov test is always accepted, even in the general case, consistently achieving a p-value greater than 50%, with the lowest p-value observed being in the winter period.

## 2.2. Correlation and Autocorrelation Analysis

The correlation between all the features in the dataset was computed and is presented in the following heatmap for examination.

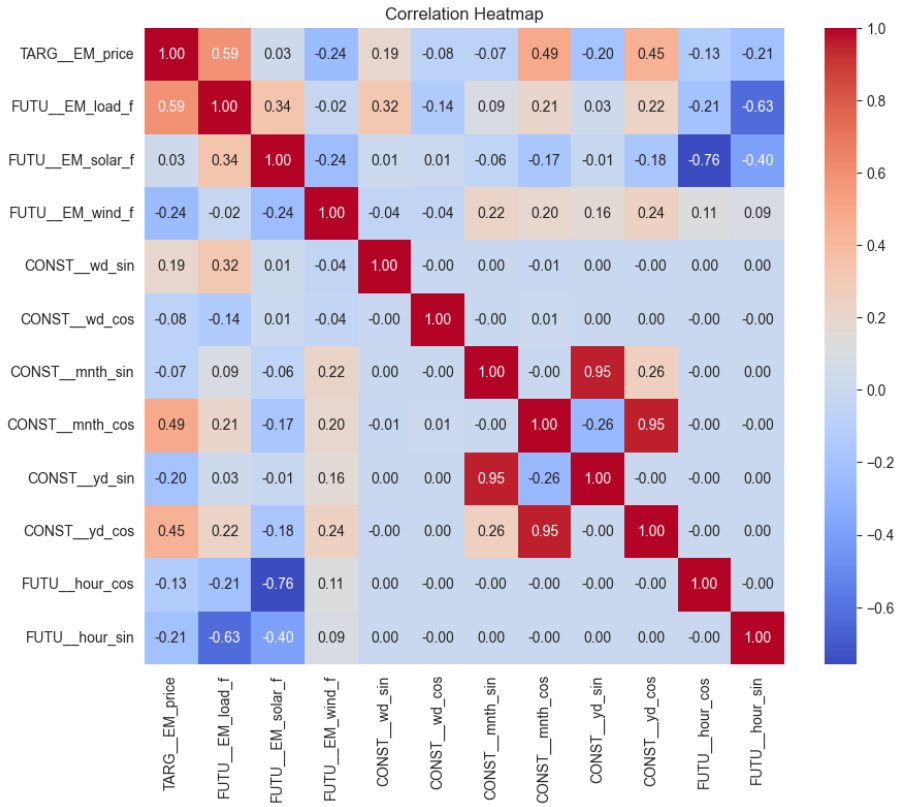


Figure 2.2: Correlation Heatmap

As expected, the feature with the highest correlation with the price is the load forecast. Surprisingly, the solar forecast shows very little correlation with the price, likely due to the presence of zeros in the observations during night time.

We can also affirm that the generated temporal features show a significant correlation with the price, indicating their usefulness for inclusion as inputs in our models.

Another important factor to consider is autocorrelation. We have implemented the function *cov_ autoreg* to analyze this, and subsequently, the autocorrelation was plotted.

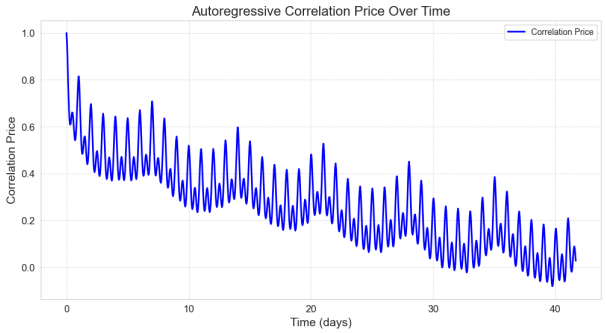


Figure 2.3: Autocorrelation plot

In this plot, we can observe a surprising shape for the autocorrelation function. The seasonal peaks correspond to increments that are multiples of 24 hours. It is evident from the plot that prices within approximately 7 days of each other exhibit significant correlation with future prices, while beyond this threshold, the correlation diminishes.

The just presented statistical analysis and its results were then utilized in the following models development in order to analyse more in depth the structural possibilities and justify Johnson distribution of residuals and time series decomposition.

# 3 | Optimization

A first need in the training process was to identify strategies to enhance model performance. The primary scope was to optimize the training procedure, allowing experimentation with different techniques in a reasonable timeframe.

As known from the theory, each network must be validated using a loss function that quantifies the network performance, which provides feedback on the validity of weight selection. In order to minimize the error function, the gradient descent technique is used, based on the fact that, for a given function, the direction of steepest descent at any given point corresponds to the negative gradient at that point.

Optimization algorithms like Adam and Learning rate schedulers like Cyclic Learning Rate (CLR) can significantly speed up the training process adjusting the learning rate used to determine the size of the steps the algorithm takes towards the minimum.

Therefore, we decided to explore the aforementioned techniques to accelerate convergence and, additionally, modified the code to enable customized selection for the frequency of days for recalibration.

## 3.1. Adaptive Moment Estimation

Adam (Adaptive Moment Estimation) adjusts the learning rate for each parameter dynamically based on estimates of first and second moment, combining the advantages of AdaGrad and RMSProp, two other extensions of the stochastic gradient descent. On the other hand, AdamW ( Adaptive Moment Estimation Weighted ) modifies the original Adam by decoupling weight decay from gradient descent.

Practically, Adam and AdamW optimizers are integrated during the model compilation step, specifying both the learning rate and weight decay.

We observed improvements in models' efficiency, stability, and generalization using both methods, with the weighted method proving to be slightly more effective.

## 3.2. Cyclic Learning Rate

Cyclic Learning Rate (CLR) is a learning rate scheduler where the learning rate is varied cyclically, increasing and decreasing it over iterations, which can help in escaping local minima and saddle points in the loss landscape.

The cyclic learning rate is implemented in the fit method of the model training class. The initial learning rate and maximum learning rate are defined, along with a scaling function to adjust the learning rate in each cycle, and the function calculates the rate for each epoch using a triangular scaled function to oscillate the learning rate between the initial and maximum value.

We noticed an important boost leading to faster convergence by allowing larger updates during the high learning rate phases.

Additionally, it reduces the need for extensive manual tuning of the learning rate, allowing for more robust training across different model architectures.

## 3.3. Recalibration days

The recalibration days are determined by a customized parameter that dictates the frequency of model recalibration during the ensemble prediction process. During these specific iterations, a new model is initialized, trained, and validated using the recalibration data. This approach allows us to assess model performance without frequent recalibration, typically every seven or thirty days for more complex models. By adopting this method, we maintained model performance while achieving greater efficiency and convenience.

# 4 | Point Metrics

Point Metrics are essential tools for evaluating the performance of predictive models. They quantify the accuracy of predictions, providing easily interpretable measures of error.

Thus, we relied on point metrics to gain insight into how well the model under study was performing in terms of predicting exact values.

Commonly used point metrics that we have exploited include Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Squared Error (RMSE). Each of these metrics offers a different perspective on model accuracy, capturing various aspects of prediction error.

## 4.1. Mean Absolute Error

MAE provides a straightforward measure of average magnitude of errors between predicted and actual values in absolute value.

$$MAE = \frac{1}{N}\sum_{n=1}^{N}|y_n - \hat{y}_n|$$

Where $y_n$ indicates the exact value and $\hat{y}_n$ the predicted one.

We decided to consider this metric as it provides a clear and direct measure of how well a model performs in predicting precise values.

## 4.2. Mean Absolute Percentage Error

MAPE provides a measure of prediction accuracy as a percentage, quantifying the average magnitude of error between predicted and actual values.

$$MAPE = \frac{1}{N}\sum_{n=1}^{N}|\frac{y_n - \hat{y}_n}{y_n}|$$

Where $y_n$ represents the actual value and $\hat{y}_n$ the predicted value.

We decided to include this metric to ease interpretation, allowing for a clear picture of relative performance, ensuring that the model aims to reduce errors proportionally across all prediction ranges. However, it is important to notice that MAPE can be sensitive to small actual values, potentially leading to large percentage errors.

## 4.3. Root Mean Squared Error

RMSE measures the square root of the average squared differences between predicted and actual values.

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2}$$

Where $y_n$ represents the actual value and $\hat{y}_n$ the predicted value.

We decided to consider this metric as it provides a sensitive measure of point prediction, giving more weight to larger errors. Indeed, the advantage of using RMSE is its ability to penalize larger errors more severely, making it a robust choice for detecting significant prediction errors and ensuring the model performs well across various error magnitudes.

# 5 | Quantile Metrics

Alongside point metrics it is useful to evaluate the model ability to predict the different quantiles according to the levels of interest. A new set of metrics is then considered, leading to measures such as: Pinball loss, Winkler Score, Continuous Ranked Probability Score and Delta Coverage.

## 5.1. Pinball loss

The first quantile metric we consider is the Pinball loss, which being a measure of fit for quantiles is suitable to be used as loss function.
Given the forecast for the $\alpha$-quantile $\hat{q}_{\alpha,n}$ of the target variable for the n-th sample and the corresponding realisation $y_n$, the associated Pinball Loss is:

$$P(\alpha, \hat{q}_{\alpha,n}, y_n) = \alpha \cdot (y_n - \hat{q}_{\alpha,n}) \mathbb{1}_{\{y_n \geq \hat{q}_{\alpha,n}\}} + (1-\alpha) \cdot (\hat{q}_{\alpha,n} - y_n) \mathbb{1}_{\{y_n < \hat{q}_{\alpha,n}\}}$$

Then we can compute the mean over all the predicted samples and get the pinball loss for every confidence level $\alpha \in (0, 1)$:

$$Pinball(\alpha) = \frac{1}{N} \sum_{n=1}^{N} P(\alpha, \hat{q}_{\alpha,n}, y_n)$$

In order to adapt it to a loss function, we can define the Average Pinball Loss (APL) as the average with respect to the predicted quantile levels. In our case, this is:

$$APL = \frac{1}{21} \sum_{\alpha=1}^{21} Pinball(\alpha)$$

where 21 is the number of target quantiles we predicted, i.e the ones for $\alpha$ from 0.5% to 5%, the median, and the ones for $\alpha$ from 95% to 99.5%.
The APL is a great indicator of the quality of our forecast and a valuable loss function, as it tends to shrink the quantiles towards the realized prices. This happens because the APL penalizes the predicted quantiles according to their confidence level and their distance to the realization. In fact, for point prediction, we use its particular case previously discussed, the MAE, which corresponds to the pinball loss for $\alpha = 0.5$.

## 5.2. Winkler score

Another metrics that we consider in probabilistic forecasting is the Winkler Score, defined as follows for prediction intervals $PI = [\hat{L}_n, \hat{U}_n]$ with nominal coverage level $1 - \alpha$:

$$Winkler_n = \hat{U}_n - \hat{L}_n + \frac{2}{\alpha} \cdot (\hat{L}_n - y_n)\mathbb{1}_{\{y_n < \hat{L}_n\}} + \frac{2}{\alpha} \cdot (y_n - \hat{U}_n)\mathbb{1}_{\{y_n > \hat{U}_n\}}$$

Then we take the average of the Winkler scores computed for the predicted intervals and get the Average Winkler Score, which can also be used as a loss function, which in our case is:

$$AWS = \frac{1}{10} \sum_{n=1}^{10} Winkler_n$$

In our case we compute 10 PIs for coverage levels from 90% to 99% and we get the AWS. The Winkler score is an important metrics since it has a constant penalty, which is the size of the PI, rewarding sharp intervals and another penalty if the realized price falls outside the PI, adjusting this term for the nominal coverage level of the PI. This is because it has to be high for high coverage levels, i.e if a realized price falls outside a 90% PI, then $\frac{2}{\alpha} = 20$, resulting in a huge sanction.

## 5.3. CRPS

Another metrics which can also be used as a loss functions is the Continuous Ranked Probability Score. Formally, given a forecast $\hat{F}_n$ for the cumulative distribution function (CDF) of the n-th sample, the associated CRPS is defined as follows:

$$CRPS(\hat{F}_n, y_n) = \int_{-\infty}^{+\infty} \left[\hat{F}_n - \mathbb{1}_{\{z \geq y_n\}}\right]^2 dz$$

In particular, we implemented its empirical approximation in order to compute it over the predicted quantiles:

$$CRPS_N = \frac{1}{N} \sum_{n=1}^{N} \left[ \frac{1}{m} \sum_{i=1}^{m} |\hat{y}_n^{(i)} - y_n| - \frac{1}{2m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} |\hat{y}_n^{(i)} - \hat{y}_n^{(j)}| \right]$$

Here m is the number of target quantiles and N the number of samples we predicted. This metrics has several peculiarities, first of all it is robust and sensitive to distances. In fact, the first mean inside the parenthesis is similar to the Pinball score, but there is an additional term referred to the distance between the quantiles which we aim to minimize in order to reward densities around the realizations and maximize the sharpness of the PIs.

## 5.4. Delta Coverage

Finally we explore the metrics which we aim to minimize in the competition, the Delta Coverage. The empirical coverage comes from the backtesting of VaR and is defined as

follows: for all the prediction intervals: $[\hat{L}_n, \hat{U}_n]$ of nominal coverage $\alpha$ we compute:

$$I_n = \begin{cases} 1 & \text{if } y_n \in [\hat{L}_n, \hat{U}_n] \qquad \text{"hit"} \\ 0 & \text{if } y_n \notin [\hat{L}_n, \hat{U}_n] \quad \text{"violation"} \end{cases}$$

Then we retrieved the empirical coverage $EC_\alpha$ by averaging this indicator, which counts how many times the realization falls in the PIs, over the number of samples we predicted, so: $EC_\alpha = \frac{1}{N} \sum_{n=1}^{N} I_n$. Finally the Delta Coverage is computed as:

$$\Delta C = \frac{1}{100 \cdot (\alpha_{max} - \alpha_{min})} \sum_{a=100 \cdot \alpha min}^{100 \cdot \alpha max} |EC_a - a|$$

Where $\alpha_{min}$ and $\alpha_{max}$ are percentiles, respectively of 90% and 99%.
The aim of the Delta Coverage is to verify the statistical consistency between the probabilistic forecasts and the realized observations OS in the test set which translates into verifying the reliability of the PIs, since if they have a nominal coverage of $\alpha$%, then their empirical coverage should be close to it.
Despite its aim, the Delta Coverage does not take into account the size of the intervals, which is a big drawback for the interpretability of the metrics, since as it is defined, if we choose as PIs $\mathbb{R}$, i.e all the real axis, for all the confidence levels from $\alpha_{min}$ to $\alpha_{max}$, we get $\Delta C = 6.\overline{1}$. If we have shorter Prediction Intervals (PIs), they may perform worse than this value.

# 6 | Data prepocessing

In order to improve the robustness and accuracy of our forecasting models, it was necessary to consider preprocessing methodologies. In this project, we implemented a comprehensive preprocessing pipeline to handle data, leveraging the already implemented preprocessing library in scikit-learn.

Additionally, one key option introduced is the removal of spikes, which helps in mitigating the impact of outliers caused by unexpected market events.

## 6.1. Types of Preprocessing

We proceeded by sequentially considering the different types of preprocessing and testing their performance.

Below a detailed description of preprocessing available.

| **Selected Technique** | **Usage** | **Limitations** |
|---|---|---|
| StandardScaler | - Used when features have different units or variances, scales features to have zero mean and unit variance | - Sensitive to outliers which can skew the mean and variance |
| MinMaxScaler | - Transforms features by scaling each feature to a given range, typically [0, 1]. Useful for algorithms requiring bounded feature spaces, like neural networks | - Sensitive to outliers as it uses the min and max values for scaling |
| RobustScaler | - Scales features using statistics that are robust to outliers (median and IQR) | - Less effective for data without significant outliers |
| PowerTransformer | - Applies a power transformation to make data more Gaussian-like, useful for stabilizing variance and minimizing skewness. Used to handle heteroscedasticity. | - Can be computationally intensive and may not be effective if data is already close to Gaussian distribution |

Through our trials and models, the StandardScaler and RobustScaler emerged as the most effective preprocessing techniques. The specific preprocessing procedures for each model will be detailed when the models are presented.

## 6.2. Remove spikes option

A common problem in machine learning is the tendency of models to overfit to outliers during the training period. In financial markets, outliers can result from shocks in prices due to unexpected financial news.

To tackle this issue, we introduced the "remove spikes" option, which excludes market windows outside a 95% confidence interval from the training and validation sets. This option proves especially beneficial for complex models with a large number of parameters, as it aids in preventing the models from learning from noise. However, this type of option does not fare well with data scarcity and can lead to the model generalizing less effectively.

## 6.3. Transforming temporal features

Transforming features is a commonly used technique to enhance the dataset capabilities. In particular, transforming temporal features using cosine and sine functions is a powerful technique to handle cyclic or periodic data. In this case, all the information about days, weeks, hours and years have been transformed. By applying sine and cosine transformations, these features are mapped onto a unit circle, which helps maintain their periodic properties. For instance, the hour of the day, ranging from 0 to 23, can be converted into $\cos(2\pi * \frac{hour}{24})$ and $\sin(2\pi * \frac{hour}{24})$, creating two continuous variables that effectively represent the cyclical pattern of a day. This transformation allows models to understand that midnight (0 hours) is closer to 1 AM (1 hour) and 11 PM (23 hours) than to 12 PM (12 hours). Consequently, this approach enhances the model's ability to learn and predict patterns in time-based data, making it particularly useful in applications like demand forecasting, time series analysis, and scheduling tasks.

# 7 | Models tested

During our journey into probabilistic forecasting we have encountered and implemented several models that we bring here.

## 7.1. ARX

The simplest model we have seen is the AutoRegressive model with eXogenous input, which forecasts the future values of the price series as a linear combination of the inputs (i.e $\hat{y}_n = Ax_n$, where $x_n$ contains both the past values of the price series and the eXogenous features: load, weather-related and temporal ones). The architecture is simple and only composed by an input layer, a dense layer with 24 neurons as the hours we predict and with an l1 regularizer which we set to $10^{-7}$. Then we create the model and compile it through the optimizer AdamW together with a cyclic lr which starts from 0.001.
Despite the simplicity and the limits of this simple linear model, as we will see, it performs very well together with conformal prediction even outperforming some more complex models.

## 7.2. DNN

An evolution of the ARX model is the Deep Neural Network (DNN), which can also capture non-linear relationships between the inputs and the output prices. The architecture is more complex and consists of the input layer, to which we apply batch normalization. This technique normalizes the inputs by the batch mean and batch standard deviation, improving the stability and speed of training. The core of the architecture is represented by a fine-tuned number of hidden dense layers, all with the same specifications that we fine-tuned: the number of neurons, the activation function, and the L1-L2 regularization to prevent overfitting.

In the appendix we will specify all the finetunned hyperparameters, while the performances of this model which resulted in many cases to be the best could be found in the results chapter. As we expected, it outperforms the ARX given its more general imprinting and beats also more complex models.

## 7.3. CNN

The Convolutional Neural Network (CNN) Regressor is a classical machine learning model that leverages the power of convolutional layers to capture spatial hierarchies in the data.

Here we report how the CNN Regressor works:

1. **Input Layer and Preprocessing**: The model begins with an input layer that accepts data shaped according to the specified input size. This input is initially processed through a masking layer to handle the null values, followed by batch normalization to standardize the input data. This ensures that the model training is stable and efficient.
2. **Time Series and feature separation**: The input data is divided into two components: the time series data and the feature data. The time series data represents the historical data points, while the feature data include the exogenous load predictions, weather and time features.
3. **Convolutional Layers**: The time series component is reshaped to fit the convolutional layers, which are specifically adept at capturing patterns over time. A series of 1D convolutional layers with suitably tuned activation functions are applied. Each convolutional layer is followed by a dropout layer to prevent overfitting by randomly setting a fraction of input units to zero during training. This series of convolutional operations helps in extracting high-level features from the raw time series data. Similarly, the feature data is also passed through its own set of convolutional layers. These layers work independently of the time series convolutional layers to extract relevant patterns from the feature set.
4. **Pooling and Concatenation**: After convolution, max pooling layers are applied to both the time series and feature convolutions. This operation reduces the dimensionality of the data and helps in capturing the most significant features. The outputs of these pooling layers are then concatenated along the feature axis, combining the high-level representations of both the time series and features.
5. **Fully Connected Layers**: The concatenated output is passed through several dense (fully connected) layers. These layers are designed to further capture the intricate patterns whithin the combined feature space. Each dense layer uses an activation function and is regularized with L1 and L2 regularizers to prevent overfitting.
6. **Output Layer**: Depending on the specified probabilistic forecasting methods, the final dense layer configuration and loss function are set. The model is then compiled using Adam optimizer.

## 7.4. RNN

The Recurrent Neural Network (RNN) Regressor is an advanced machine learning model designed to handle time series data, capturing temporal dependencies and patterns through

recurrent layers.

Here we report how the RNN Regressor is organized:

1. **Input Layer and Preprocessing**: The model starts with an input layer that takes in shaped data. This input undergoes initial preprocessing through a masking layer on null values, followed by batch normalization to standardize the input data, ensuring stability during the training.
2. **Time series and Feature Separation**: Again, the input data is split into two main components: the time series data and the feature data.
3. **Feature Transformation**: The feature component is processed through multiple dense layers. Each dense layer applies an activation function and is regularized using L1 and L2 regularizers to prevent overfitting. Dropout layers are interpersed between dense layers to further mitigate overfitting. The transformed features are then batch normalized.
4. **Recurrent Layers**: The time series, on the other hand, is reshaped and fed into convolutional layers to initially extract local patterns. Following this, the data is processed through Long Short-Term Memory (LSTM) layers. LSTM layers are crucial for capturing long-term dependencies and temporal dynamics within the time series. Each LSTM layer is configured with activation functions and regularizers for both the kernel and recurrent weights, enhancing robustness. Dropout layers are applied between LSTM layers to prevent overfitting.
5. **Flattening and Batch Normalization**: The output from the LSTM layers is flattened and batch normalized. This step ensures that the data is in a suitable format for the subsequent dense layers and helps stabilize the training.
6. **Concatenation of features and Output Layer**: The processed time series and feature data are concatenated along the feature axis. This concatenation combines the high-level temporal patterns captured by the LSTM layers with the transformed feature data, providing a comprehensive representation for the final prediction layer. Then, the final dense layer configuration varies depending on the specific probabilistic forecasting method used. The model is compiled using the AdamW optimizer, the loss is chosen based on the selected probabilistic forecasting method, ensuring the model is trained to minimize the appropriate error metrics.

## 7.5. Recursive Decomposition Regressor (RDR)

The Recursive Decomposition Regressor (RDR) model is the most sophisticated model presented in our study. Its primary objective is to incorporate the statistical decomposition conducted during the statistical analysis phase as an input for the forecasting process. By leveraging this decomposition, the RDR model aims to enhance the accuracy and robustness of the forecast.

Here is a detailed explanation of how the RDR model operates:

1. **Statistical Decomposition**: Initially, the time series data is decomposed into its

fundamental components: trend, seasonality, and residuals. This decomposition is crucial as it allows the model to handle different aspects of the time series separately, improving the overall forecasting capability.

2. **Deep Neural Network (DNN) on Trend and Features**: The trend component, which captures the long-term progression of the time series, is processed using a Deep Neural Network (DNN). This DNN is designed to learn complex patterns and relationships within the trend data, allowing it to accurately model the underlying trajectory of the time series. Simultaneously, the features derived from the data (such as exogenous variables or additional relevant information) are also fed into a separate DNN to extract meaningful patterns and relationships.

3. **Convolution and LSTM on Residuals**: The residuals, which represent the irregular and unpredictable fluctuations in the time series, are processed through a combination of convolutional layers and Long Short-Term Memory (LSTM) networks. The convolutional layers help in identifying local patterns and anomalies in the residuals, while the LSTM networks capture the temporal dependencies and dynamics. Specifically, two LSTM layers are employed in sequence to ensure a deep and nuanced understanding of the residual behavior.

4. **Concatenation with Seasonality**: In the final layer, the outputs from the processed trend, features, and residuals are concatenated. Additionally, the seasonality component, which represents the repeating patterns or cycles within the time series, is integrated into this concatenated layer. By combining all these components, the model ensures that it captures the complete spectrum of the time series' behavior.

5. **Output Layer**: The concatenated layer, now enriched with information from the trend, features, residuals, and seasonality, is passed to the chosen output layer. This layer generates the final forecast, providing a comprehensive prediction that takes into account all the different facets of the time series.

# 8 | Output methodologies

In the wide world of probabilistic forecasting there are numerous output methodologies, which we have implemented for every architecture we have just described, which can vary from the target quantiles that we want to predict such as QR, or the parameters of the PDF that we suppose represents our process despite we only observe one realization of those quantiles, the median. Last but not least, there is the conformal prediction which is quite different from the others since it relies on a distribution-free predictive inference task, which may be also integrated with an adaptive technique.

## 8.1. QR

The first output methodology we dive into is the Quantile Regression, it is easy to implement by specifying the number of target quantiles to predict, then adding a dense layer with a number of neurons equal to the prediction horizon, 24 hours, times the number of target quantiles, in our case 21 and as activation function we set it to linear. Then we reshaped the output into a matrix 24x21 and ordered the matrix along the rows such that the quantiles are in increasing order and they do not overlap each other.
It is highly customizable, in fact we can directly choose the quantiles to predict and the output will change accordingly. In addition it adapts perfectly to be evaluated with the quantile metrics previously described, since all of them evaluate the performance of the forecast directly from the quantiles.

## 8.2. Normal

The first distributional output analyzed is the Normal distribution. Here, we assume that the probability density function (PDF) of the price follows a Gaussian distribution. We observe two parameters: the mean and the scale (standard deviation) of the prices. To ensure the scale is always positive in our predictions, we apply the softplus function and add 1e-3. This adjustment guarantees that the standard deviation is strictly greater than 0, preventing the distribution from collapsing into a delta function centered at the mean. As we will see the normal delivers good performances and guarantees a well-behaved forecast of the quantiles which as consequence of the distribution are symmetrical.

## 8.3. JSU

Another distributional output we tested is the JohnsonSU which is a transformation through the $arcsinh(.)$ transformation of the gaussian distribution we have just described and extends it, since has four parameters to describe the distribution: the mean (loc) and the standard deviation (scale) as the Normal distribution, and the skewness and tailweight. The skewness depicts the asimmetry of the distribution while the tailweight is important to describe how fat the tails of the distribution are. The latter has to be positive as the standard deviation, so that we apply the same exact procedure for its output, while the asimmetry coefficient can be both positive or negative in order to describe a right or left asimmetry.
In principle, the JSU is a generalization of the normal, therefore it is able to capture a wider range of patterns, however it is more difficult to calibrate. It is often precise, but sometimes it can produce a huge quantile both in positive and negative prices in presence of spikes in the prices, as consequence of a fat tail which can be not very explicable and can result in a worse performance on the quantile metrics.

## 8.4. Conformal Prediction

Another interesting methodology that we explore is the conformal prediction. Differently from the two previous methods, which rely on the estimation of parameters for a fixed type of distribution, here we remove this assumption and construct marginal prediction intervals at test points $x_n$, such that $\mathbb{P}(y_n \in \mathcal{C}(x_n)) \geq 1 - \alpha$ where our dataset is $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$, $y_i$ is our target price and $x_i$ is a vector containing the time series of past prices and both past and forecasted features. In our case they are: load, solar, wind and the time variables.

First of all, to build these intervals we compute the conformity scores:

$$\mathcal{S}(x_i, y_i) = |y_i - f(x_i)| = \mathcal{S}_i$$

and we leverage on the exchangeability assumption, i.e that all the scores are i.i.d, so that we can order them increasingly and get their empirical distribution:

$$\mathbb{P}(\mathcal{S}_i \leq \mathcal{S}_{(k)}) = \mathbb{P}(|y_i - f(x_i)| \leq \mathcal{S}_{(k)}) = \frac{k}{n}$$

where n is the number of scores and $1 \leq k \leq n$.

We define the $1 - \alpha$ prediction intervals as $\mathcal{C}_{1-\alpha}(x) = \{y : \mathcal{S}(x, y) \leq \hat{q}_{1-\alpha}\}$.
The quantiles derived from the empirical distribution of the scores are:

$$\hat{q}_{1-\alpha} = \begin{cases} \mathcal{S}_{(\lceil (n+1)(1-\alpha) \rceil)} & \text{if } \lceil (n+1)(1-\alpha) \rceil \leq n \\ \mathcal{S}_n & \text{otherwise} \end{cases}$$

There are great advantages using this method, the first is the simple implementation, trivial without efficiency requirements and the other is that the aim with this method is to build sharp intervals close to their nominal coverage $1-\alpha$, which is the ideal in this competition since our main goal is to minimize the Delta Coverage which relies exactly on this. However there are also some gaps to fill: the first is that we build marginal intervals, the coverage is over $y$ and not on $y|x$ as in the conditional coverage therefore what happens is that we build average PIs, but the uncertainty should be adapted to the actual condition. The other problem is due to the exchangeability assumption, which is quite weak and not verified in reality.

## 8.5. Adaptive conformal inference

Here comes the Adaptive Conformal Inference (ACI) which aims at filling the gaps we have just presented. The ACI is designed to adapt the conformal prediction to temporal distribution shifts, this is achieved by a simple online update of $\alpha_t$, the target quantile, by examining the empirical miscoverage frequency of the previous prediction sets and then decreasing (resp. increasing) $\alpha_t$ if the prediction sets were historically under-covering (resp. over-covering) the target price $y_t$. The sequence of miscoverage events is tracked by:

$$err_t = \begin{cases} 1 & \text{if } y_t \notin \hat{\mathcal{C}}_t(\alpha_t) \\ 0 & \text{otherwise} \end{cases}$$

where $\hat{\mathcal{C}}_t(\alpha_t)$ is the confidence interval of level $\alpha_t$.
The online update algorithm in our case is the following:

$$\alpha_{t+1} = \alpha_t + \gamma\Big(\alpha - \frac{1}{24}\sum_{t=1}^{24} err_t\Big)$$

Here we consider the mean over the previous 24 hours, all equally weighted, to compute the miscoverage frequency and update the confidence level accordingly.
We have noticed that a specificity of ACI's algorithm is thus to often produce infinite intervals. We have solved this problem by inserting a check every time in order $\alpha_t$ not to fall off 100% and in this case capping the quantile to the maximum score.
The only one parameter to select for the ACI is the $\gamma$. This parameter improves the adaptability of the quantiles and was crucial in the delta coverage optimization. As we will see in the results the ACI consistently improves the performances in terms of Delta Coverage with respect to CP and effectively enables to adapt to the local uncertainty of the prices.

# 9 | Finetuning methodologies

A crucial aspect of machine learning model development is the selection of hyperparameters. The performance of complex models can vary significantly depending on this choice.

In our study, we implemented three different fine-tuning methodologies using the Optuna library to support the optimal selection of hyperparameters.

The objective function to minimize during the fine-tuning process is the Mean Absolute Error (MAE) for point prediction models and the delta coverage for quantile prediction models.

Differently from the model fit, we decided to not fix the random seed in order to have always different trials.

You could find all the resulting tuned hyperparameters for the different models in the dedicated Appendix A.

## 9.1. Grid search

This methodology is the simplest one. Given a grid of hyperparameters, it tests all possible combinations and selects the best one.

Grid search can be useful for simple models with few hyperparameters, such as ARX. However, it becomes challenging to use as the number of parameters increases, due to the exponential growth in the number of combinations to evaluate.

## 9.2. Random

This methodology selects hyperparameters randomly from a given range. It can be very helpful in obtaining a general idea of a smaller range for the optimal parameters.

For models such as Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN), this methodology is very effective. However, it requires a very large number of trials to ensure the selection of the right hyperparameters.

## 9.3. Gaussian Process Sampler

This methodology is still in an experimental phase in the Optuna library. It uses the Bayesian optimization fitting a Gaussian process (GP) to the objective function and

optimizes the acquisition function to suggest the next hyperparameters.

The choice of this methodology for fine-tuning was crucial for the hyperparameters selection of the RDR model, given its large number of hyperparameters. Unlike the random sampler, this approach uses a criterion for selecting the next trials, making it more efficient and effective for complex models.

On the other hand, this type of sampler exhibits significant oscillations in the objective function value. Therefore, we opted to pair it with a Wilcoxon Pruner. This pruner performs the Wilcoxon signed-rank test between the current trial and the best trial so far, and halts the trial whenever it is confident, up to a given p-value, that the current trial is worse than the best one. The p-value threshold was set to 0.1. This feature is particularly crucial when fine-tuning heavy models, where a single trial can take several minutes.

# 10 | Loss functions

Loss function selection plays a pivotal role in shaping the performance and accuracy of predictive models. They consist in mathematical tools used to quantify the error between predicted and actual values. They guide the optimization process during model training, driving adjustments to minimize errors and enhance predictive accuracy.

We have both utilized functions found in literature and customized losses inherent to the peculiar task.

## 10.1. MAE

We have already presented MAE as a general metric for point evaluation. It could be employed as loss function for point prediction, according to which the objective of the training algorithm is to minimize the MAE value.

## 10.2. Average Pinball Loss

We have already presented Pinball Loss when dealing with Quantile Metrics in Chapter 5. However, it is not only a useful metric for evalutation but also a powerful loss function for model training. By employing Average Pinball Loss directly as a loss function, models are incentivized to produce accurate quantile forecasts rather than point predictions. This approach align the training with the goal of quantile accuracy, fostering better calibration and sharper predictions.

Building on the foundation of Average Pinball Loss, the Normal Pinball loss introduces an extension that incorporates probabilistic elements into the quantile forecasting framework. Unlike standard Average Pinball Loss, which directly operates on quantile predictions, Average Normal Pinball Loss is adapted to the fact that the output layer uses the gaussian distribution.

Practically, it involves constructing a normal distribution based on the predicted mean and variance and then sampling from this distribution to estimate the quantiles. Then, those are used to compute the pinball loss.

## 10.3. Average Winkler Loss

Also introduced within Quantile Metrics, Average Winkler Loss serves as a loss function. It covers two purposes: it rewards narrow prediction intervals that successfully capture the observed values, while penalizing intervals that fail to include the actual values. This balance makes it particularly useful for training models that generate reliable and precise prediction intervals.

## 10.4. CRPS Loss

The already introduced Continous Ranked Probabilistic Score is another important metric that compares the entire predicted cumulative distribution function to the observed value. As a loss function, CRPS provides a comprehensive measure that reflects both the accuracy and sharpness of the probabilistic forecasts. It rewards distributions that concentrate their probability mass near the observed values and penalizes those that spread out their mass or deviate significantly from the actual outcomes.

## 10.5. Pinball Delta Loss

Pinball Delta Loss is a customized loss function designed to combine aspects of traditional Pinball Loss and interval coverage accuracy. This dual focus makes Pinball Delta Loss particularly valuable for applications where both quantile and interval forecasts are crucial.

The calculation begins by iterating over pairs of quantiles, evaluating the loss for both lower and upper quantiles and incorporating a hit percentage term to reflect interval accuracy.

Let $\hat{y}_l$ and $\hat{y}_u$ be the predictions for the lower $q_l$ and upper $q_u$ quantiles respectively. The Pinball Delta Loss assesses the precision of these quantiles and their alignment with the actual value $y$.

First, we compute the Pinball Loss for both the lower and upper quantiles:

$$\text{Pinball}_l = q_l \cdot (y - \hat{y}_l) \cdot \mathbf{1}\{y \geq \hat{y}_l\} + (1 - q_l) \cdot (\hat{y}_l - y) \cdot \mathbf{1}\{y < \hat{y}_l\}$$

$$\text{Pinball}_u = q_u \cdot (y - \hat{y}_u) \cdot \mathbf{1}\{y \geq \hat{y}_u\} + (1 - q_u) \cdot (\hat{y}_u - y) \cdot \mathbf{1}\{y < \hat{y}_u\}$$

Next, we introduce a term to evaluate the hit percentage of the true value $y$ within the prediction interval $[\hat{y}_l, \hat{y}_u]$. Define the hit percentage HitPerc as:

$$\text{HitPerc} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\{\hat{y}_{l_n} \leq y_n \leq \hat{y}_{u_n}\}$$

This term ensures that the combined Pinball Losses for the lower and upper quantiles are adjusted according to whether $y$ falls within the prediction interval. The final Pinball Delta Loss is:

$$\text{PinballDeltaLoss} = \frac{(\text{Pinball}_l + \text{Pinball}_u) \cdot |\text{HitPerc} - (q_u - q_l)|}{q_u - q_l}$$

By focusing on both quantile accuracy and interval reliability, it helps improve the robustness and utility of probabilistic forecasts, providing more comprehensive insights into future energy prices.

# 11 | Delta Coverage Criticality

One of the most significant challenges encountered during the development of this project was optimizing based on delta coverage performance. This metrics does not account for the size of the prediction interval or the accuracy of the point prediction. The only consideration is that the proportion of errors made by the confidence interval matches its confidence level.

The first issue with this metric is its unreliability over short time intervals. For instance, if we test our model over a 7-day period, it is unlikely that the proportion of errors will align with the expected confidence level. Therefore, we chose to test all models over a period of one year, similar to the Financial VaR backtesting approach.

The second issue is mathematical in nature. Unlike the APL and Winkler scores, this metric has an infinite number of global minima that do not coincide with the optimal solution for our problem. To demonstrate this, we implemented a function, *zero_ delta_ cov*, which computes the quantiles based on the point prediction of any model. This function sets the confidence intervals very large for a proportion of the predictions equal to the confidence level and very small (with a length of zero) for the remainder.

For example, with a 90% confidence interval for 100 observations, we can set the first 90 quantiles by adding a large number to the point prediction to ensure almost 100% inclusion of the real price, and for the remaining 10 observations, we set the quantiles to match the point prediction. On average, the number of 'hits' will be almost exactly 90%. Applying this procedure to all the quantiles can achieve a delta coverage close to zero without affecting the accuracy of the point predictions.

Since there are infinitely many possible large numbers to select for this procedure, the optimal solutions for this metric are also infinite.

Here an example of the results obtained with the function applied on point-ARX predictions:

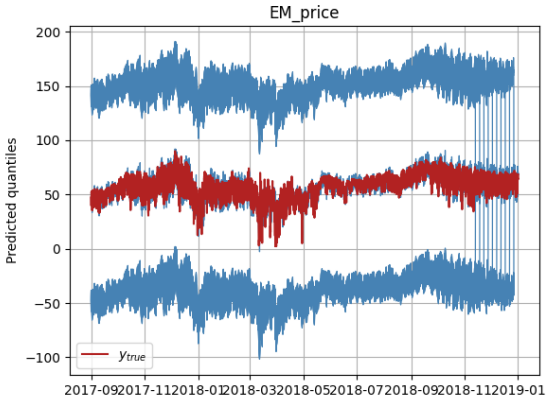


Figure 11.1: Zero Delta Coverage example

The metrics obtained for this prediction are:

- Mean Absolute Error: 3.989
- MAPE: 0.098
- RMSE: 5.318
- Average Pinball Score: 2.448
- Average Winkler Score: 185.319
- CRPS: 39.483
- Delta Coverage: 0.004

By exclusively examining the metrics, it is evident that both the Pinball loss and, notably, the average Winkler score and the CRPS classify it as a subpar solution. Conversely, the delta coverage appears to closely align with the value expected for an ideal model.

Indeed, while this model may be entirely impractical in real-world applications, it serves as a reminder to consistently monitor our predictions and consider all metrics collectively when validating a model.

# 12 | Results

Here we present the performances for all the models described, with different output methodologies. The metrics that we show here are: MAE, MAPE and RMSE for point forecast and APL, AWS, CRPS and Delta Coverage for probabilistic forecasting. All the results we display have been computed with a fixed seed of 420 ensuring reproducibilty and allowing us to compare the models. Moreover for point-forecast models we compare the results in terms of Delta Coverage for CP and ACI, the last one for several values of $\gamma$ on both 2017 and 2018 as test sets. All the models have been finetunned for every output and the hyperparameters have been reported in the Appendix A.
All the models were trained and validated on the 2 years before. This choice permitted us to have 2 different test set (2017 and 2018) to study the consistency of the models.

## 12.1. ARX

Here are the performances of the ARX-l1 model, for which we adopt the MAE as loss function for the point-forecasts together with CP and ACI with $\gamma = \mathbf{0.005}$ for probabilistic-forecasts, with 122 calibration samples, training starting from 01-09-2015 and as prediction horizon all the year 2018.

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| **ARX-CP** | 4.092 | 0.109 | 5.487 | 0.474 | 32.013 | 5.454 | 0.994 |
| **ARX-ACI** | 4.092 | 0.109 | 5.487 | 0.472 | 31.849 | 5.353 | 0.592 |

Clearly the point-metrics are the same as they refer to the point-forecast and are quite good. We can observe that all the forecast metrics remain very similar between CP and ACI, except for the delta coverage, which improves significantly as we expect.

It was chosen for this reason to prove many different values of $\gamma$ and try to identify a good range for an optimal delta coverage.

| $\gamma$ | $\Delta C$ |
|---|---|
| 0.01 | 0.3675 |
| 0.02 | 0.24 |
| 0.03 | 0.23 |
| 0.04 | 0.228 |
| 0.05 | 0.218 |
| 0.06 | 0.184 |
| 0.07 | 0.173 |
| **0.08** | **0.154** |
| 0.09 | 0.175 |
| 0.1 | 0.193 |

| $\gamma$ | $\Delta C$ |
|---|---|
| 0.01 | 0.616 |
| 0.02 | 0.62 |
| 0.03 | 0.538 |
| 0.04 | 0.498 |
| 0.05 | 0.475 |
| 0.06 | 0.469 |
| 0.07 | 0.459 |
| **0.08** | **0.454** |
| 0.09 | 0.462 |
| 0.1 | 0.466 |

Table 12.1: Tables of Delta Coverage for different values of $\gamma$ on the left for 2018 and on the right for 2017

## 12.2. DNN

Here are the performances of the point-DNN for different loss functions and test sets (2017 with training starting from 03-01-2015 and 2018 with training starting from 01-09-2015), weekly recalibration, all the probabilistic forecasts have been performed through ACI with $\gamma = \mathbf{0.005}$.

| Model | loss | test set | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|---|---|
| point-DNN | MAE | 2018 | **3.768** | 0.103 | **5.099** | 0.441 | 30.15 | 4.884 | 0.276 |
| point-DNN | MSE | 2018 | 3.8 | 0.105 | 5.137 | 0.446 | 30.462 | 4.99 | 0.375 |
| point-DNN | MAE | 2017 | 3.853 | 0.077 | 5.124 | **0.424** | **27.74** | **4.786** | 0.233 |
| point-DNN | MSE | 2017 | 3.833 | **0.077** | 5.195 | 0.431 | 28.469 | 4.859 | **0.213** |

We highlighted the best metrics and we can observe that generally the same model performs better on 2017 and it is interesting to see that the model with the best quantile metrics does not achieve the lowest Delta Coverage.
As for the ARX, we try different values of $\gamma$ in order to identify a good range for an optimal delta coverage.

| $\gamma$ | $\Delta C$ |
|---|---|
| 0.01 | 0.171 |
| 0.015 | 0.111 |
| **0.016** | **0.107** |
| 0.017 | 0.1139 |
| 0.02 | 0.124 |
| 0.03 | 0.159 |

| $\gamma$ | $\Delta C$ |
|---|---|
| 0.002 | 0.191 |
| **0.003** | **0.089** |
| 0.004 | 0.1311 |
| 0.005 | 0.232 |
| 0.006 | 0.308 |
| 0.01 | 0.518 |
| 0.02 | 0.587 |

Table 12.2: Tables of Delta Coverage for different values of $\gamma$ on the left for 2018 and on the right for 2017

## 12.3. CNN

Now we present the results for the point-CNN with MAE as loss function and objective of the finetuning, test set 2018 with training starting from 01-09-2015, monthly recalibration since it is computationally heavy, all the probabilistic forecasts have been performed through ACI with $\gamma = \mathbf{0.005}$.

Here are the metrics:

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| point-CNN | 4.712 | 0.125 | 6.204 | 0.525 | 35.528 | 5.877 | 0.616 |

We can clearly notice that despite the CNN is a complex architecture, it performs worse than the 2 models we have seen so far. This is the reason why we did not performed too many trials because of the high computational time, but it can be interesting to run it with a more frequent recalibration.

## 12.4. RNN

Now we present the results for the point-RNN with MAE as loss function and objective of the finetuning, test set 2018 with training starting from 01-09-2015, monthly recalibration since it is computationally heavy, all the probabilistic forecasts have been performed through ACI with $\gamma = \mathbf{0.005}$.

Here are the metrics:

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| point-RNN | 7.901 | 0.197 | 9.567 | 0.739 | 46.024 | 8.228 | 0.265 |

As for the CNN, also the RNN is a complex architecture, the point metrics are not optimal, we did not performed too many trials because of the high computational time, but it can be interesting to run it with a more frequent recalibration.

## 12.5. RDR

We conclude with the RDR, a model we have stressed for all the possible output methodologies which have all been finetunned through a gaussian process sampler.
We start with the Normal output. We have tested it with a 2 year training set and as test set all the 2018, here are the metrics:

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| **Normal-RDR** | 4.661 | 0.132 | 6.426 | 0.531 | 36.666 | 5.145 | 3.177 |

Then we pass to the second output methodology, the JSU, here are the metrics:

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| **JSU-RDR** | 4.752 | 0.1499 | 6.863 | 0.588 | 40.791 | 5.855 | 1.599 |

We examine the QR output with the same configuration, here are the metrics:

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| **QR-RDR** | 5.996 | 0.1625 | 8.052 | 0.753 | 54.554 | 7.367 | 3.31 |

As for all the other models, we adapted the RDR to perform point-forecasts with MAE as loss function and objective of the finetuning, monthly recalibration, 2 years of training set and implemented the ACI with $\gamma = 0.005$ with 122 calibration samples and test set all the 2018 for probabilistic forecasts. Here are the metrics:

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| **point-RDR** | 4.025 | 0.119 | 5.697 | 0.509 | 35.427 | 5.742 | 0.424 |

Also for this model we tried with several values of $\gamma$ to achieve the best Delta Coverage:

| $\gamma$ | $\Delta C$ |
|---|---|
| 0.01 | 0.135 |
| 0.02 | 0.113 |
| 0.021 | 0.113 |
| **0.022** | **0.1095** |
| 0.023 | 0.121 |
| 0.024 | 0.123 |
| 0.025 | 0.1311 |
| 0.03 | 0.146 |

| $\gamma$ | $\Delta C$ |
|---|---|
| 0.02 | 0.662 |
| 0.021 | 0.657 |
| 0.022 | 0.652 |
| 0.023 | 0.640 |
| 0.024 | 0.645 |
| 0.025 | 0.647 |
| **0.03** | **0.637** |
| 0.04 | 0.649 |

Table 12.3: Tables of Delta Coverage for different values of $\gamma$ on the left for 2018 and on the right for 2017

## 12.6. Ensemble

A final trial was made by attempting to ensemble the point-DNN and the point-RDR models with the ACI algorithm, using a $\gamma$ value of 0.022. The resulting metrics were as follows:

| Model | MAE | MAPE | RMSE | APL | AWS | CRPS | $\Delta C$ |
|---|---|---|---|---|---|---|---|
| **DNN-RDR** | 3.683 | 0.107 | 5.131 | 0.452 | 31.309 | 5.083 | 0.526 |

## 12.7. Considerations

There are many considerations to make about these results.
First of all is astonishing to see that all the results coming from different architectures are comparable between each other, not only this, the ARX performs better than the CNN and the RNN and similar to the best models which are the DNN and the RDR.
We can compare the different outputs thanks to the results of the RDR, the QR is undoubtedly the worst under every aspect, both for point and probabilistic forecasts, the quantiles are not only very large as suggested by the high values of the APL and AWS but produces also confidence intervals with little consistency given the huge Delta Coverage. The Normal and the JSU are similar considering the point metrics but we notice that both present a spike during spring, a period with high volatility which messes a little the quantile metrics which are good but produce a little too narrow intervals resulting in a higher Delta Coverage if compared to the point-RDR which produces both good point-forecasts and, as assessed by the theory performs, well also on the probabilistic hand thanks to the ACI methodology.
There are some uncomfortable conclusions coming from these results, the fact that similar performances are obtained with neural networks of such varying complexity is only possible if all the different NNs are able to adapt with equal precision to the objective of minimizing the loss function, the MAE. Consequently, the complexity of more structured networks is unnecessary and perhaps even counterproductive given the higher computational effort needed. We would not be surprised if an advanced Transformer performs worse than the ARX. Moreover the fact that the ARX performs so well highlights that there may be a linear relation between inputs and output.
Moreover, we can state that model ensembling, often used in machine learning and neural networks, helps to achieve better results in all the point and quantile metrics except for delta coverage. This may be due to the fact that higher accuracy of the point forecasts can cause the model to include the real prices more frequently within the confidence interval, leading to a larger number of hits than the expected confidence level.

With the ARX we have verified the improvements between the Conformal Prediction and the ACI, which enables us to achieve really low values of Delta Coverage. This puts us closer to the goal of building statistically consistent confidence intervals, since a Delta coverage of 1 tells us that there is a miscoverage of only 1% on average for 90% to 99% confidence intervals. Here we are even under 0.5 consistently with different values of $\gamma$ with the ACI. Our goal comparing the ACI for different values of $\gamma$ between 2017 and 2018 test sets, despite not having two full years of training for 2017, is to verify the consistency

of the method and the importance of this hyperparameter which is what we observe in the ARX. Conversely, the DNN is highly affected by slight changes of $\gamma$ despite always achieving great results. Finally the RDR performs worse than the other two on 2017, this can be due to the shorter training set, while shows their best attributes on 2018 where it both performs well and is robust to changes in $\gamma$. This will be important for our choice for the competition.
We conclude this section with some observations about the role of $\gamma$ in the ACI. As explained in the introduction of ACI, the value of $\gamma$ is a tradeoff between adaptability and stability and this is what we found in our analysis. For all the three models for which we tested various values of gamma we retrieved the same curve trend, finding a unique minimum in the Delta Coverage w.r to $\gamma$, in fact higher values force too much the quantile adjustment, sometimes resulting in that quantile to be capped, while lower values of $\gamma$ tend to adapt slowly the quantile level but giving it a stable trajectory. Our analysis agrees with the one carried out by papers about the ACI but we improved the procedure by capping the quantile and not setting it to infinity as they do, this enables us both to achieve better quantile metrics and to try higher $\gamma$ without the risk to produce unrealistic quantiles.

# 13 | Competition choice

The final important decision to make as a team was the choice of the model to use in the competition.
We decided not to use the *zero_ delta_ coverage* procedure in order to select a model that can be applied in a real-world framework.
The best-performing models in our results were the point-DNN and the point-RDR, both with the quantiles computed using the ACI algorithm. After careful consideration, we selected the point-RDR model for its superior performance and robustness in our evaluations.

The point-RDR model is an original approach that has not been previously seen in academic papers, as it was developed by our team. It demonstrated exceptional accuracy and reliability, making it a suitable choice for practical applications. By leveraging the ACI algorithm for quantile computation, it consistently delivered high-quality predictions with the necessary coverage and precision. This model aligns well with our objectives of achieving both competitive performance in the competition and practical utility in real-world scenarios.

For this model, we will use the hyperparameters fine-tuned with the Gaussian Process Sampler in the *recalib_ opt_ grid_ 1_ 2*, and a $\gamma$ value of 0.022, as it was the best parameter for the year 2018. We expect similar behavior in 2019. We have commented out the seed settings in the code to aim for a better local minimum in 2019. Additionally, we retrained the model to forecast for 2018 with a weekly recalibration to verify the consistency of our model's results.

Here the final plots for our forecasts:

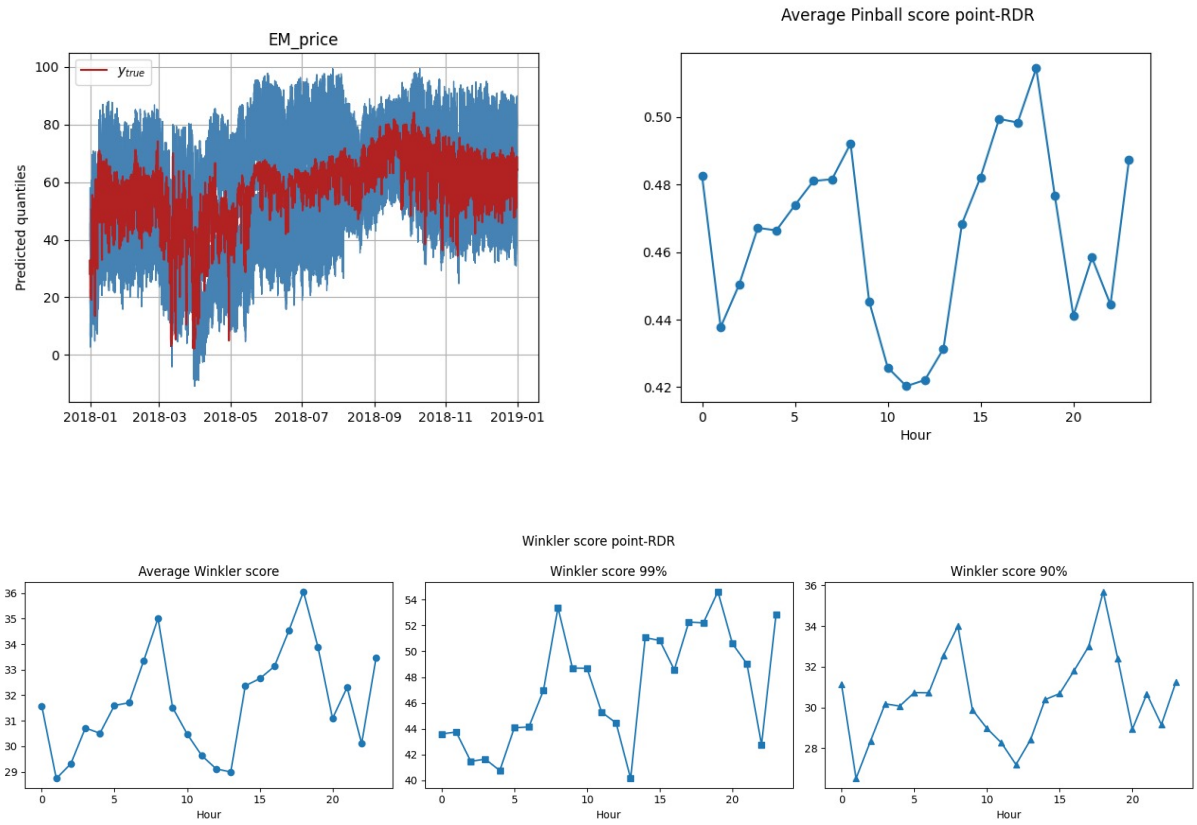


Figure 13.1: Final competition model results

The results presented the following metrics:

- Mean Absolute Error: 3.9577493340348546
- MAPE: 0.11563656924164817
- RMSE: 5.4587469322292606
- Average Pinball Score: 0.46454259700503303
- Average Winkler Score: 31.745766824492264
- CRPS: 5.162936715532948
- Delta Coverage: 0.12785388127854386

We can conclude that the model is consistent and can be a great selection for the 2019 forecast. All this decisions were imported in a last and final script *main_competition*.

# 14 | Further developments

In this section we propose some developments and make some considerations about the models we have implemented and the accuracy of the forecasts. For what concerns the architectures we used, they are indeed quite complex and advanced, but not the state of art of the NNs, which are the Transformers, despite the great results achieved by simple architectures, this new frontier, if fed with a complex dataset, is even better at extracting patterns since for what we stated in the results, whatever the model, the goodness of the results is strongly dependent on the dataset.
This introduces the second point which can lead to better performances: the dataset. It can be, indeed, enlarged, according to the area we are examinating. In fact, there are numerous information that can be aggregated, such as the energy policy of the country: does it rely on nuclear or coal power plant? Info about raw materials prices: gas, oil,coal. How much of its energy demand is self-produced and how much is imported? These and many other informations about the climate or other energy production methods regarding our dataset can be very useful.
Finally we also have to consider the limitations of our forecasts, in fact they can even be performed by the best architecture with the most complete dataset, but as we experienced in the last years there are some exceptional events which can't be predicted and may severely influence the energy price such as: pandemics, wars or climate disasters. These extreme events can deeply influence the forecasts of prices and make it a hard task, much harder than predicting the load of the same areas which is more stable and less affected by these factors. This indeed makes load forecasts more realistic to perform.

# Bibliography


- Lago, J., De Ridder, F., De Schutter, B. (2018). Forecasting Spot Electricity Prices: Deep Learning Approaches and Empirical Comparison of Traditional Algorithms. Applied Energy, 221, 386-405. https://doi.org/10.1016/j.apenergy.2018.02.069
- Brusaferri, A., Baviera, R. (2024). Electricity Price and Load Forecasting. EPFL Financial Engineering Course Notes, Politecnico di Milano.
- Gibbs, I., Candès, E. J. (2021). Adaptive conformal inference under distribution shift. Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Stanford University.
- Lago, J., Marcjasz, G., De Schutter, B., Weron, R. (2021). Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms. Applied Energy, 293, 116983. https://doi.org/10.1016/j.apenergy.2021.116983
- Nowotarski, J., Weron, R. (2018). Recent Advances in Electricity Price Forecasting: A Review of Probabilistic Forecasting. Renewable and Sustainable Energy Reviews, 81, 1548-1568. DOI: 10.1016/j.rser.2017.05.234.
- Hong, T., Fan, S. (2016). Probabilistic electric load forecasting: A tutorial review. International Journal of Forecasting, 32(3), 914-938. doi:10.1016/j.ijforecast.2015.11.011
- Baviera, R., Messuti, G. (2023). Daily Middle-Term Probabilistic Forecasting of Power Consumption in North-East England. Energy Systems, 14, 654-679. https://doi.org/10.1007/s12667-023-00577-0

# A | Appendix

In this appendix we report the finetunned hyperparameters of all the models for all the output methodologies.

We finetunned the ARX through the grid-search, these are the hyperparameters:

**ARX**
$lr : 0.001 \;\; l1 : 1e-07$

The finetuning for the Point-DNN was performed through a random search, these are the hyperparameters:

**Point DNN**
$hidden\_size : 960 \;,\; n\_hidden\_layers : 2 \;,\; lr : 0.00154 \;\; l1 : 1.239e-19 \;,\; l2 : 2.1206e-17$
$activation\ function : "softplus"$

We finetunned the CNN through the gaussian process sampler (GP), these are the hyperparameters:

**point CNN**
$hidden\_size : 448 \;,\; n\_hidden\_layers : 2 \;,\; lr : 1e-06 \;\; activation\ function : softplus$
$activation\_conv : relu \;\; l1 : 7.039e-06 \;,\; l2 : 3.817e-06$

We finetunned the RNN through the gaussian process sampler (GP), these are the hyperparameters:

**point RNN**
$hidden\_size : 64 \;,\; n\_hidden\_layers : 4 \;,\; lr : 1.8116e-05 \;,\; conv\_units : 128 \;,$
$lstm\_units : 18 \;\; l1\_feat : 3.0732e-05 \;,\; l2\_feat : 4.4051e-05 \;,\; l1\_lstm : 2.3095e-06$
$l2\_lstm : 3.0260e-05 \;,\; l1\_lstm\_rec : 8.874e-08 \;,\; l2\_lstm\_rec : 2.7779e-05$
$l1\_out : 1.0000e-06$
$l2\_out : 0.000659 \;,\; drop\_lstm : 0.2085 \;,\; activation : relu \;\; activation\_lstm : tanh$
$activation\_conv : softmax \;\; drop\_rate : 0.4063$

We finetunned the RDR through the gaussian process sampler (GP), these are the hyper-parameters for all the output methodologies:

**Normal RDR**

$hidden\_size1 : 256$ , $n\_hidden\_layers1 : 0$ , $hidden\_size2 : 384$ $n\_hidden\_layers2 : 1$
$lr : 2.552e-06$ , $conv\_units : 64$ , $lstm\_units : 4$ $l1\_feat : 10.0$
$l2\_feat : 2.204e-07$ , $l1\_trend : 1e-15$ , $l2\_trend : 1e-10$ $l1\_lstm : 0.0001$
$l2\_lstm : 1.0$ , $l1\_lstm\_rec : 0.0099$ , $l2\_lstm\_rec : 1e-20$ $l1\_out : 4.306e-10$
$l2\_out : 1.2318e-15$ , $drop\_lstm : 0.5$ , $activation : softplus$ $activation\_lstm : relu$
$activation\_conv : sigmoid$ $drop\_rate : 0.4538$

**JSU RDR**

$hidden\_size1 : 960$ , $n\_hidden\_layers1 : 4$ , $hidden\_size2 : 704$ $n\_hidden\_layers2 : 4$
$lr : 1e-06$ , $conv\_units : 64$ , $lstm\_units : 22$ $l1\_feat : 1e-07$
$l2\_feat : 10.0$ , $l1\_trend : 1e-15$ , $l2\_trend : 1.882e-06$ $l1\_lstm : 0.0001$
$l2\_lstm : 1e-15$ , $l1\_lstm\_rec : 0.0099$ , $l2\_lstm\_rec : 2.71e-13$ $l1\_out : 1e-15$
$l2\_out : 4.017e-10$ , $drop\_lstm : 0.5$ , $activation : softplus$ $activation\_lstm : tanh$
$activation\_conv : sigmoid$ $drop\_rate : 0.0$

**QR RDR**

$hidden\_size1 : 576$ , $n\_hidden\_layers1 : 3$ , $hidden\_size2 : 640$ $n\_hidden\_layers2 : 1$
$lr : 2.5e-05$ , $conv\_units : 192$ , $lstm\_units : 6$ $l1\_feat : 0.000199$
$l2\_feat : 3.954$ , $l1\_trend : 2.522e-07$ , $l2\_trend : 1.154e-07$ $l1\_lstm : 6.37e-09$
$l2\_lstm : 3.6e-11$ , $l1\_lstm\_rec : 0.001$ , $l2\_lstm\_rec : 3.4647e-18$ $l1\_out : 0.000142$
$l2\_out : 5.168e-06$ , $drop\_lstm : 0.3349$ , $activation : softplus$
$activation\_lstm : softmaxactivation\_conv : sigmoid$ $drop\_rate : 0.0798$

**Point RDR**

$hidden\_size1 : 384$ , $n\_hidden\_layers1 : 1$ , $hidden\_size2 : 384$ $n\_hidden\_layers2 : 2$
$lr : 0.000458$ , $conv\_units : 64$ , $lstm\_units : 10$ $l1\_feat : 1.609e-05$
$l2\_feat : 1.981e-05$ , $l1\_trend : 4.116e-12$ , $l2\_trend : 1.417e-09$ $l1\_lstm : 0.0001$
$l2\_lstm : 0.00932$ , $l1\_lstm\_rec : 1e-15$ , $l2\_lstm\_rec : 1e-15$ $l1\_out : 3.076e-11$
$l2\_out : 2.104e-07$ , $drop\_lstm : 0.5$ , $activation : relu$ $activation\_lstm : tanh$
$activation\_conv : relu$ $drop\_rate : 0.47$