POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Forecasting Energy Prices in Zonal Electricity Markets

Tesi di Laurea Magistrale in
Energy Engineering - Ingegneria Energetica

Author: **Michele Maria Mainoli**

Student ID: 232724
Advisor: Filippo Bovera
Co-advisors: Guillaume Pierre Arnauld Koechlin, Andrea Scrocca
Academic Year: 2024-25

Abstract

This thesis addresses this challenge of electricity price forecasting for the Italian Day-Ahead Market (MGP), focusing on zonal prices. The thesis provides a comparative analysis of state-of-the-art forecasting models for its seven distinct price zones, evaluating three distinct models: a hybrid SARIMAX-LSTM model, a Lasso-VARX statistical model that employs regularization for feature selection and a daily recalibration strategy to adapt to market dynamics, and finally, a Deep Neural Network (DNN) model. The DNN exploits EPFToolbox, a specialized framework that automates model architecture design and hyperparameter optimization through Bayesian methods. All models are trained on hourly data from Italy's seven market zones, incorporating a feature set of exogenous variables, including load and renewable generation forecasts, market concentration indices, and technical indicators. Key data preprocessing steps are also applied to handle the statistical properties of electricity prices. The aim of the project is to identify strengths and weaknesses of these statistical and machine learning based models, helping to understand their performance in the multi-zonal forecasting environment.

Keywords: Electricity Price Forecasting, Zonal Markets, Italian Day-Ahead Market, Deep Neural Networks (DNN), Lasso-VARX, SARIMAX-LSTM.

Sommario

Questa tesi affronta la sfida del forecasting dei prezzi dell'energia elettrica sul mercato italiano del giorno prima (MGP), concentrandosi sui prezzi zonali. La tesi fornisce un'analisi comparativa dei modelli di previsione state-of-the-art per le sue sette zone di prezzo, valutando tre modelli distinti: un modello ibrido SARIMAX-LSTM, un modello statistico Lasso-VARX che sfrutta la regolarizzazione per effettuare la selezione delle features e una strategia di ricalibrazione giornaliera per adattarsi alle dinamiche di mercato e, infine, un modello Deep Neural Network (DNN). La rete neurale sfrutta EPFToolbox, un framework specializzato che automatizza la progettazione dell'architettura del modello e l'ottimizzazione degli iperparametri attraverso metodi bayesiani. Tutti i modelli sono addestrati su dati orari provenienti dalle sette zone di mercato italiane, incorporando un insieme di variabili esogene, tra cui previsioni di carico e di produzione da fonti rinnovabili, indici di concentrazione del mercato e indicatori tecnici. Vengono inoltre applicate delle tecniche di preprocessing dei dati per gestire le proprietà statistiche dei prezzi dell'energia elettrica. L'obiettivo del progetto è identificare i punti di forza e di debolezza di questi modelli basati su statistica e machine learning, contribuendo a comprenderne le prestazioni in un ambiente di previsione multizonale.

Parole chiave: Previsione Prezzi Elettricità, Mercati Zonali, Mercato del Giorno Prima Italiano, Reti Neurali, Lasso-VARX, SARIMAX-LSTM.

Contents

List of Figures

List of Tables

Listings

List of Symbols

Variable	Description	SI unit
y_t	Observed value of a time series at time t	(Varies)
$\hat{y}_t$	Forecasted value of a time series at time t	(Varies)
y'_t	Transformed value of a time series at time t	-
$y_{d,h}$	Price for a single zone at day d and hour h	€/MWh
$\boldsymbol{Y}_{d,h}$	Vector of zonal prices for day d and hour h	€/MWh
L_t	Linear component of a time series	(Varies)
$\hat{L}_t$	Forecasted linear component of a time series	(Varies)
N_t	Non-linear component of a time series	(Varies)
y_t^{in}	In-sample observed value at time t	(Varies)
$\boldsymbol{X}_{d,h}$	Vector of exogenous variables for day d and hour h	(Varies)
x	Original value of a feature	(Varies)
x'	Standardized value of a feature	-
$\bar{y}^{\text{ewm7}}$	7-day exponential moving average	(Varies)
σ^{ewm7}	7-day exponential moving standard deviation	(Varies)
EMA_t	Exponential Moving Average at time t	(Varies)
EMSD_t	Exponential Moving Standard Deviation at time t	(Varies)
e_t	Residuals of a model at time t	(Varies)
ϵ_t	Error term of a model at time t	(Varies)
ϕ, α, β	Autoregressive coefficients (scalar)	-
$\boldsymbol{\Phi}$	Autoregressive coefficient matrix	-
θ	Exogenous variable coefficient (vector)	-
$\boldsymbol{\Theta}$	Exogenous variable coefficient matrix	-
$\mathbf{x}$	Input vector to a neuron	-
$\mathbf{w}$	Weight vector of a neuron	-

Continued on next page

b	Bias term of a neuron	-
z	Linear combination of inputs in a neuron	-
$\sigma(\cdot)$	Non-linear activation function	-
s	Seasonal period length	h
N	Number of observations in a sample	-
T	Number of observations in a training sample	-
$L(\epsilon)$	Loss function of a forecast error	-
d_t	Loss differential series at time t	-
μ_x	Mean of a feature x	(Varies)
σ_x	Standard deviation of a feature x	(Varies)
α	Smoothing factor for exponential moving average	-
σ_t^2	Exponentially weighted variance at time t	(Varies)^2

1 | Introduction

1.1. Electricity Markets and their Evolution

The electricity sector is currently going through a major shift, driven by two key goals: reducing carbon emissions and opening up markets to competition through liberalization [25]. Moving from a system based on a few large, fossil-fuel-powered plants to one with many decentralized Renewable Energy Sources (RES) systems is changing the way electricity markets work [2], since these renewable power plants (which use sources like solar and wind) are unpredictable and cannot be turned on or off easily or produce electricity on demand, and will therefore bring uncertainty and volatility in the electricity markets [15]. Liberalizing and opening up electricity markets to competition has also turned electricity into a commodity, which has created markets where prices change according to supply and demand rules [1]: this is why being able to forecast electricity prices is a critical need for everyone involved in the markets, from energy producers to wholesalers to ESPs (Energy Service Providers)[28].

1.2. Focus on DAM Price Forecasting

Among the various electricity markets, the day-ahead market (which is the focus of the thesis) is where most of the electricity is traded. Here participants submit bids and offers for each hour of the following day, and a single market clearing price based on supply and demand is determined for each of these hours [28]. For any company involved in this market, being able to accurately forecast these prices is extremely important, since a good price forecast is the foundation for a successful bidding strategy [31] both for power producers, as it helps them decide how much energy to offer and at what price, with the goal of maximizing their profits and for large consumers or utility companies, because it guides their purchasing decisions, helping them buy energy when it is cheapest, thus avoiding high costs. The high volatility of electricity prices can make forecasting quite challenging: prices can change significantly hour over hour, influenced by factors like weather conditions, unexpected power plant outages, and unforeseen changes in demand

[1]; these factors may lead to inaccurate forecasts which in turn can lead to significant financial losses. This is why developing reliable and accurate price forecasting models is a key area of research and a necessity for players who need to remain competitive in the energy sector.

1.3. Overview of Forecasting Methods

In the past the most used models in the field of electricity price forecasting were statistical models, but now a shift to more sophisticated, machine learning/artificial intelligence based model is in place [16]. Initially, models like ARIMA were widely adopted, even though they fall short in capturing complex dynamics, such as non-linearity and high volatility, which define modern electricity markets [1]: this is why the research community has been pushed towards more sophisticated methods. This thesis will focus on two models that are considered state-of-the-art, each representing a leading approach from the statistical and machine learning families: the Lasso Estimated AutoRegressive (LEAR) model and the Deep Neural Network (DNN). The LEAR model is an advanced statistical method used to enhance traditional autoregressive (AR) models by using the LASSO (Least Absolute Shrinkage and Selection Operator) regularization method [27], which allows the model to process a large number of exogenous variables and automatically perform feature selection by shrinking the coefficients of less relevant variables to zero [20, 27]. This method is extremely useful since it combines a strong statistical foundation with an automated way to handle high-dimensional data (which is typical in the field of EPF), often resulting in highly accurate and robust forecasts [20]. From the ML/AI domain, Deep Neural Networks (DNNs) have also been used with great success: they are designed to capture complex, non-linear patterns and interactions within the data [20] and even simple DNN architectures, often with just two hidden layers, have been shown to achieve state-of-the-art performance, outperforming many traditional statistical and simpler machine learning models [20, 30]. The problem with these more complex models is the computational cost, because DNNs require significantly more time and data for training: so, even though they represent the best solution for maximum accuracy, for some applications where computational speed is a critical factor, faster models like LEAR are the state-of-the-art [20].

1.4. Thesis Goals and Structure

The focus of this thesis is the comparison between these 2 state of the art models and their performance, along with their comparison with a SARIMAX-LSTM hybrid model

developed by a former MSc thesis student; we will focus on comparing these different forecasting methods and to understand their strengths and weaknesses. This will be the organization of the following chapters

- **Chapter 2** provides an overview of how the Italian Day-Ahead Market works, its zonal organization, and the impact of congestion.
- **Chapter 3** discusses the theory behind the different forecasting models used.
- **Chapter 4** describes the data, the setup and the methods used to implement and test the models.
- **Chapter 5** describes the evaluation criteria used to compare the results.
- **Chapter 6** discusses the results obtained and compares them, highlighting differences across the models.

2 | Italian Day-Ahead Zonal Market

2.1. Market Structure and Price Formation

The Italian DAM, or *MGP - Mercato del Giorno Prima*, is the main platform for wholesale electricity trading in Italy and is a key component of the Italian Power Exchange (IPEX), operated by the *GME - Gestore dei Mercati Energetici* [11]. The Day Ahead Market is an auction-based market where participants[24] trade blocks of electricity for each hour of the following day. To take part in the auction, participants will have to submit their bids (to buy) and offers (to sell) up to a deadline at 9:00 AM on the day before actual delivery [11]. After this point, the GME aggregates all submissions and applies the economic merit order principle to clear the market: this principle is standard in most liberalized electricity markets and ensures that generation resources are dispatched in order of their marginal costs, from the cheapest to the most expensive [7, 22], encouraging maximum efficiency in the market. Once all offers are in place, the GME constructs an aggregated supply curve by stacking all sales offers in ascending price order. Usually, renewable sources with null or very low marginal costs (such as renewable-based power plants) are at the bottom, followed by thermal plants in order of efficiency and fuel costs [5]. A corresponding demand curve is built by aggregating buy bids in descending order and the intersection of demand and offer curves determines the *MCP - Market Clearing Price* for each hour, which is a uniform price, meaning that every unit of electricity bought and sold in that hour has this single price [24].

2.1.1. Italian Zonal Market Structure

The Italian electricity market is divided into seven different market zones [11], which follow the geography of the Italian territory; they are: North, Central-North, Central-South, South, Sicily, Calabria and Sardinia, as represented in Figure 2.1. In addition, there are virtual foreign zones that represent interconnections with neighboring countries, which facilitate international electricity flows [7].

Figure 2.1: Italian Electricity Market Zones (Source: TERNA S.p.A.)

The main idea behind this zonal configuration is the management of bottlenecks in the power grid in a market-based way [4], since in an ideal grid with infinite transmission capacity, electricity could flow freely from the cheapest generation sources to the load centers, and a single price would be sufficient, but, in reality, the transmission lines con-

necting these zones have finite capacities. The zonal model recognizes these physical constraints by dividing the national market into smaller areas that are internally unconstrained. Within each zone, the assumption is that there are no significant transmission limits and that all transactions are cleared at a single zonal price [11].

2.1.2. Congestions in the Zonal Market

Network congestion is a critical concept in zonal markets: it occurs when the economically optimal flow of electricity, which is the flow from low-cost generation areas to high-demand areas, determined by merit order, exceeds the net transfer capacity of the transmission lines connecting these zones [4, 11]; if this is the case, the TSO (Terna) has to intervene to ensure that the network remains stable. In order to manage these congestions, the main tool in the MGP is zonal price separation: if the transmission capacity between zones is sufficient to accommodate all energy exchanges, a single national price, the *PUN - Prezzo Unico Nazionale* is established. The PUN is the weighted average of all zonal prices, weighted by consumption in each zone, and is the reference price for end consumers [11]. However, if congestion occurs, the market will effectively split at the point where the congestion is located and the GME then calculates separate clearing prices for each zone (or group of zones) involved [7]. As a consequence, the zone with a surplus of cheap generation that cannot be fully exported will have a lower price; while the zone with high demand that cannot import enough cheap electricity will need to call more expensive generators to produce, driving up the price [11]. This price differential has two consequences: ensuring the physical security of the grid by preventing lines from being overloaded and creating important price signals. If prices in a specific zone are constantly higher, there must be a structural deficit in local generation, or inadequate transmission capacity, highlighting the need for new investments in power plants/grid reinforcements in the area [4].

3 | Theoretical Framework

In this chapter we will describe the different forecasting models that will be used and compared in this thesis. We will describe four different approaches: a simple Naive model which serves as a baseline, a hybrid SARIMAX-LSTM model, and state-of-the-art[31] Lasso-VARX and Deep Neural Network (DNN) models.

3.1. Naive Model

This method is the simplest one to perform time series forecasting, since it assumes that future values will be a continuation of past values: it is ideal to use as a benchmark when dealing with data with strong seasonality; which is exactly the case of day-ahead electricity price forecasting, where we have strong weekly patterns. So, the Naive model will predict the price for a specific hour of the day using the price observed at the same hour seven days prior, this can be formulated as:

$$\hat{y}_t = y_{t-s} \tag{3.1}$$

where $\hat{y}_t$ is the forecast for the current hour t, y_{t-s} is the observed price from the same hour in the previous cycle, and s is the seasonal period, which in this case is $24 \times 7 = 168$ hours [23]. Since we need to forecast well over the seasonal period of 168 hours, the model runs out of actual historical data to reference, leading to NaN values for any forecast beyond the first week. So, to overcome this issue, we will use a recursive naive model: it works by appending each forecast to the old dataset, always allowing to have a previous seasonal period. Of course a big drawback will be that Any inaccuracy in an early forecast will be repeated and carried forward every week for the entire forecast period, which can lead to a significant decline in accuracy over time.

3.2. SARIMAX-LSTM Hybrid Model

The hybrid model combines a statistical approach (SARIMAX) with a machine learning model (LSTM)[23], a key assumption behind this framework is that a time series is composed of a linear and a non-linear component: $y_t = L_t + N_t + \epsilon_t$ [23]. The first stage involves an iterative SARIMAX (Seasonal AutoRegressive Integrated Moving Average with eXogenous variables) model, which is a powerful statistical method for time series forecasting extending the ARIMA model [6]. Its structure is defined by two sets of parameters: the non-seasonal order (p, d, q) and the seasonal order $(P, D, Q)_s$. These parameters represent:

- **Autoregressive (AR - p, P):** use of past values of the time series to predict future values, p is the non-seasonal order, and P is the seasonal order.
- **Integrated (I - d, D):** number of times the data is differenced to make it stationary, d is the non-seasonal differencing order, and D is the seasonal differencing order.
- **Moving Average (MA - q, Q):** use of past forecast errors to improve the current forecast, q is the non-seasonal order, and Q is the seasonal order.

The X in SARIMAX indicates its ability to include exogenous variables that can influence the target series. When making predictions for multiple steps into the future, and when the model uses recent lagged values as a feature (like the price of the previous hour), it must operate recursively, therefore it's an **Iterative SARIMAX**: with this configuration, the model first predicts one step ahead. This prediction is then added to the input data and used to predict the second step, and this process repeats for the entire forecast horizon [23]: the problem with this approach is that, even though it allows for long-term predictions, it can also lead to an accumulation of errors, as inaccuracies in early forecasts are propagated through following predictions. This iterative SARIMAX is used to capture the linear patterns L_t in the data. To avoid these long term inaccuracies we take the residuals from the SARIMAX model ($e_t = y_t - \hat{L}_t$) and we model them using a Long Short-Term Memory (LSTM) network, which is a special type of Recurrent Neural Network (RNN) designed to overcome the vanishing gradient problem: this allows the Neural Network to effectively learn long-term dependencies in sequential data [13]. The core of the LSTM is its structure, which contains a cell state and three gates (input, forget, and output) that regulate the flow of information and prevent the vanishing gradient:

- The **forget gate** decides which information from the previous cell state should be discarded.
- The **input gate** determines which new information should be stored in the cell state.
- The **output gate** controls which information from the current cell state is used to generate the output [13].

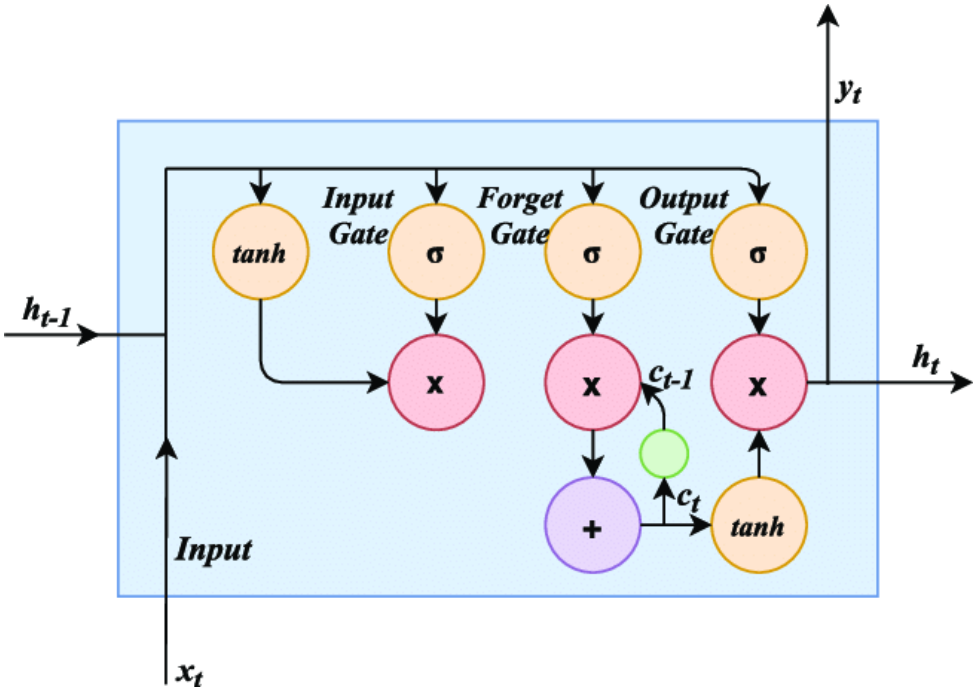

Figure 3.1: LSTM RNN Structure

This architecture enables the LSTM to effectively model the complex, non-linear patterns that may remain in the residuals of the SARIMAX model [23]. The final forecast is the sum of the SARIMAX prediction and the LSTM's prediction of the residual. The parameters for the SARIMAX model and the hyperparameters and Neural Network architecture used in the hybrid model are taken from previous research [23] and are reported in the following tables:

Table 3.1: SARIMAX Model Parameters.

Parameter	Value
Non-seasonal component (p,d,q)	
p (AR Order)	3
d (Differencing)	0
q (MA Order)	2
Seasonal component (P,D,Q,s)	
P (Seasonal AR Order)	2
D (Seasonal Differencing)	1
Q (Seasonal MA Order)	0
s (Seasonal Period)	24

Table 3.2: LSTM Model Architecture and Hyperparameters.

Network Architecture

Layer	Units / Rate	Output Shape
Input Layer	-	(None, 24, 47)
LSTM	32	(None, 24, 32)
Dropout	0.1	(None, 24, 32)
LSTM	64	(None, 64)
Dropout	0.1	(None, 64)
Dense (Output)	1	(None, 1)

Optimal Hyperparameters

Parameter	Value
Activation Function	Tanh
Batch Size	16
Epochs	100
Optimizer	Adam
L1 factor	0.001
Dropout rate	0.1

The dataset was used as follows to perform SARIMAX training, LSTM training on residuals and testing (there was no need for a validation set, as the LSTM was already tuned):

Table 3.3: Dataset Split for Hybrid Model

Set	Date Range	Duration
SARIMAX Training set	2022-01-01 – 2023-06-30	1.5 years
LSTM Training set	2023-07-01 – 2024-06-30	1 year
Test set	2024-07-01 – 2025-06-30	1 year

3.3. Lasso-VARX Model

The LEAR model, as specified in the literature [30] is not used in a VAR configuration, since it is not used to forecast multiple zonal prices which could influence one another. Instead, the Lasso-VARX model is a multivariate statistical framework designed to forecast multiple time series simultaneously, making it a great solution for zonal price forecasting, since it combines a VAR structure with exogenous variables (X) and uses LASSO regularization for parameter estimation. During training we can select whether we want to model interactions between different hours and/or different zones, therefore switching from an AR to a VAR configuration.

3.3.1. Model Specifications and Structures

The core of the model is to forecast a K-dimensional vector of target variables $\boldsymbol{Y}_{d,h}$ (e.g., the prices in K different zones) for a given day d and hour h. The general approach can be specified in different ways to account for dependencies within each time series (autoregressive) and across different time series (vector autoregressive) [18]. During training 168 (24 hours x 7 zones) models are built, one for each hour of the day (to highlight the daily seasonality of electricity prices) and one for each zone.

Univariate ARX Models

As a baseline, we can consider each zonal price series $y_{d,h}$ as independent.

- **Concurrent ARX:** This model assumes the price at hour h depends only on its past values at the same hour. The equation for a single component is:

$$y_{d,h} = \phi_{1,h} y_{d-1,h} + \phi_{7,h} y_{d-7,h} + \theta_h \boldsymbol{X}_{d,h} + \epsilon_{d,h} \tag{3.2}$$

  where only same-hour lags are used [18].

- **Full ARX (fARX):** This model allows the price at hour h to depend on the prices from all 24 hours of previous days:

$$y_{d,h} = \sum_{j=1}^{24} \phi_{1,h}^{(j)} y_{d-1,j} + \sum_{j=1}^{24} \phi_{7,h}^{(j)} y_{d-7,j} + \theta_h \boldsymbol{X}_{d,h} + \epsilon_{d,h} \tag{3.3}$$

  This captures intraday dependencies within a single zone's price series [18].

Multivariate VARX Models

To model the interdependencies between zones, a vector approach is used.

- **Concurrent VARX:** This model forecasts the vector of all zonal prices $\boldsymbol{Y}_{d,h}$ using only the vectors from the same hour on previous days. It models the simultaneous relationships between zones at a specific hour:

$$\boldsymbol{Y}_{d,h} = \boldsymbol{\Phi}_{1,h}\boldsymbol{Y}_{d-1,h} + \boldsymbol{\Phi}_{7,h}\boldsymbol{Y}_{d-7,h} + \boldsymbol{\Theta}_h\boldsymbol{X}_{d,h} + \boldsymbol{\epsilon}_{d,h} \tag{3.4}$$

  Here, $\boldsymbol{Y}_{d,h}$ is a vector of prices for all zones, and $\boldsymbol{\Phi}$ and $\boldsymbol{\Theta}$ are matrices of coefficients capturing cross-zonal effects [18].

- **Full VARX:** This is the most comprehensive structure, where the vector of prices at hour h depends on the price vectors from all 24 hours of the preceding days:

$$\boldsymbol{Y}_{d,h} = \sum_{j=1}^{24} \boldsymbol{\Phi}_{1,h}^{(j)}\boldsymbol{Y}_{d-1,j} + \sum_{j=1}^{24} \boldsymbol{\Phi}_{7,h}^{(j)}\boldsymbol{Y}_{d-7,j} + \boldsymbol{\Theta}_h\boldsymbol{X}_{d,h} + \boldsymbol{\epsilon}_{d,h} \tag{3.5}$$

  This formulation can capture complex, time-lagged spillovers between different market zones across different hours of the day [18]. This configuration hasn't actually been implemented in the thesis due to the extremely long training time, which would defeat the purpose of using a LEAR model.

The strength of this algorithm lies in the presence of LASSO (Least Absolute Shrinkage and Selection Operator) regularization, given the high number of potential predictors, especially in the structures which employ a VAR: this regularization with L1 penalty automatically selects the most relevant variables by shrinking the coefficients of unimportant features to zero, which helps prevent overfitting and improves the model's forecasting accuracy [18, 20].

3.3.2. Model Recalibration Strategy

To ensure the model remains accurate over time and adapts to changing market conditions, a daily recalibration strategy is employed: this approach avoids training the model only once on a fixed historical dataset, instead, it continuously updates the model with the most recent information available. As implemented in the model's forecasting process, this is handled through a sliding calibration window. For each day being forecast, the model is re-trained using a fixed period of the most recent historical data (in our case this is the last 358 days, as set by the calibration window parameter); this way, the

model's parameters are always estimated based on the latest market dynamics, capturing shifts in volatility, seasonality, or the impact of exogenous variables. Once the model is recalibrated on the data for day d-1 and earlier, it is used to produce the forecast for day d; so, daily recalibration is critical to ensure forecast accuracy in a dynamic environment like the electricity market.

Model Training and Testing

The dataset was used as follows to perform training and testing:

Table 3.4: Dataset Split for LassoVARX

Set	Date Range	Duration
Training set	2022-01-01 – 2024-06-30	2.5 years
Test set	2024-07-01 – 2025-06-30	1 year

3.4. Deep Neural Network Model

This chapter will provide a deep dive into the Deep Neural Network (DNN) model used in this thesis, the discussion starts from the foundational theory to the implementation. We will begin with an introduction to the DNN's application in the domain of electricity price forecasting, then, we study the theory of neural networks, explaining their components and learning mechanisms. The most important part of this chapter is a deep dive into the EPFToolbox, a sophisticated framework that automates the most critical aspects of DNN modeling: architecture design and hyperparameter optimization. To conclude, we will discuss the specific modifications made to adapt this framework to the task of multizone price forecasting.

3.4.1. Introduction to DNNs for Electricity Price Forecasting

As we have said before, forecasting electricity prices is one of the most challenging tasks in time series analysis, since, unlike other financial or commodity markets, electricity cannot be economically stored in large quantities, meaning that supply and demand must be balanced in real-time. This imposes a set of statistical properties that make traditional forecasting models often struggle to provide accurate predictions: these properties will be discussed in depth in the following chapter 4. Traditional statistical models are based on assumptions of linearity and stationarity, and even though they can be adapted to handle seasonality, they still struggle to capture the non-linear interactions in electricity markets [1]. This is why Deep Neural Networks are a state-of-the-art solution: they are a class of machine learning models that are considered **universal function approximators**: meaning that a DNN can model any continuous function to any desired degree of accuracy, given enough complexity: this is why they are particularly suited for EPF. The features used to train the DNN are the exact same ones used with the other methods mentioned in this thesis: the DNN takes these features as input and is trained to produce the 24 hourly prices for the following day as its output, its main advantage is the ability to automatically discover complex patterns and interactions. However, this power comes at a cost: the performance of a DNN is very sensitive to its design, architecture and a vast number of hyperparameters that determine its structure and training process. Manually designing and tuning these models is a complex and most importantly time-consuming task, and it is a challenge that the EPFToolbox framework solves through automation.

3.4.2. The Theory behind Deep Neural Networks

To understand how the EPFToolbox works, first we need to understand the theoretical principles behind Deep Neural Networks: in this section we deconstruct a DNN into its main components, from the single neuron to the full network architecture and the mechanisms by which it learns from data.

The Neuron

The neuron is the base of a neural network, it is a simple computational unit that transforms multiple inputs into a single output [12]. This transformation is a two-step process.

In the first step, the neuron will perform a linear combination of its inputs, then it receives an input vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and it will calculate a weighted sum. Each input x_i is multiplied by a corresponding weight w_i and the weights are learned by the network through backpropagation: a larger weight signifies that its corresponding input has a higher influence on the neuron's output. A **bias** term, b, is then added to this sum: it is another learnable parameter that allows the neuron to shift its activation function. This aggregation is a dot product between the weight vector $\mathbf{w}$ and the input vector $\mathbf{x}$, plus the bias:

$$z = \mathbf{w} \cdot \mathbf{x} + b = \sum_{i=1}^{n} (w_i x_i) + b \tag{3.6}$$

Second, to model the complex patterns, the linear result z is passed through a non-linear **activation function**, denoted by $\sigma(\cdot)$. The final output of the neuron, y, is therefore:

$$y = \sigma(z) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{3.7}$$

The choice of activation function is critical and it is one of the main hyperparameters: without a non-linear activation, a network with any number of layers would be equivalent to a simple linear model, and it wouldn't be capable of capturing non-linear relationships. Common activation functions include:

- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$. This function maps any input to the range (0, 1). It is being used less now in hidden layers, due to the *vanishing gradient* problem, where the function's derivative becomes very close to zero for large positive or negative inputs, which can pose problems during the learning process.

- **Hyperbolic Tangent (Tanh):** $\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. This function maps inputs to the range (-1, 1). Being zero-centered, it often helps models converge faster than the sigmoid function, but it also suffers from the vanishing gradient problem.
- **Rectified Linear Unit (ReLU):** This has become the default activation function for most applications, since it is computationally very efficient and helps to alleviate the vanishing gradient problem. Its formula is simply $\sigma(z) = \max(0, z)$.

The Multi-Layer Perceptron

To best exploit the abilities of neural networks the neurons are organized into **layers**, which are stacked to form a **MLP, Multi-Layer Perceptron**, the most common type of feedforward DNN. An MLP has three types of layers:

1. **Input Layer:** represents the input features of the dataset, the number of neurons in this layer is equal to the number of features.
2. **Hidden Layers:** are the layers responsible for capturing non-linearity. In a fully-connected (or **Dense**) layer, every neuron is connected to every neuron in the previous layer, and the network is considered deep when it has multiple hidden layers. The depth of the network allows it to learn a hierarchy of features, since early layers might learn simple patterns, while deeper layers combine these patterns to recognize more complex ones.
3. **Output Layer:** produces the final prediction. Its size and activation function depend on the specific task, for our EPF task, which involves predicting 24 hourly prices for 7 zones, the output layer would have $24 \times 7 = 168$ neurons with a linear activation function to allow for unconstrained price predictions.

The process of generating a prediction is called **forward propagation**, it works by passing the input data to the first hidden layer, its outputs are calculated, these outputs become the input for the second hidden layer, and so on, until the final output layer produces the forecast.

Backpropagation and Gradient Descent

Training a DNN means finding the optimal values for all the weights and biases that minimize the model's error, and it is done through an iterative optimization process. The process for each step is as follows:

1. **Calculate the Error:** a batch of data is passed through the network (forward propagation) to get predictions, which are compared to the true values using a loss function, such as the MAE, which quantifies the overall error.
2. **Calculate the Gradients:** the backpropagation algorithm is used to calculate the gradient of the loss function with respect to every weight and bias in the network; the gradient for a specific weight tells us how a small weight change affects the total loss.
3. **Update the Parameters:** an optimizer algorithm uses these gradients to update the weights and biases; an example is **Gradient Descent**, which updates each parameter by taking a small step in the opposite direction of its gradient, the size of this step is controlled by a hyperparameter called the **learning rate**.

This cycle is repeated for many epochs until the model's performance on a separate validation set stops improving.

3.4.3. The EPFToolbox Framework

The EPFToolbox is a specialized framework designed to automate the process of choosing the best hyperparameters and architecture of a neural network, transforming it from a manual process into a data-driven one [19]. It uses the Keras API of TensorFlow for model building and the `hyperopt` library for intelligent optimization.

DNN Implementation in EPFToolbox

The toolbox contains a set of functions that can dynamically build, compile, and train a Keras DNN model based on a given set of hyperparameters, which is the key to its automated approach.

Dynamic Model Construction The core of the implementation is a function, called `build-model`, that takes a dictionary of hyperparameters (`params`) as input and returns a compiled Keras model, which allows the optimization script to generate thousands of different architectures on the fly. It is also of course used to build the final DNN model, once the hyperparameters have been chosen. A simplified but representative example of such a function is shown below, this function can build a model with a variable number of hidden layers, each with its own number of neurons and dropout rate.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop

def build_model(params, input_shape, n_outputs):
    """
    Builds a Keras Sequential model based on a dictionary of
   hyperparameters.

    Args:
        params (dict): Dictionary containing hyperparameters like '
   n_layers',
                       'neurons_l1', 'activation', etc.
        input_shape (tuple): The shape of the input data.
        n_outputs (int): The number of neurons in the output layer.

    Returns:
        A compiled Keras model.
    """
    model = Sequential()

```

```
    # Input Layer and First Hidden Layer
    model.add(Dense(
        units=int(params['neurons_l1']),
        activation=params['activation'],
        input_shape=input_shape
    ))

    model.add(Dropout(rate=params['dropout_l1']))

    # Additional Hidden Layers (if any)
    for i in range(2, params['n_layers'] + 1):
        model.add(Dense(
            units=int(params[f'neurons_l{i}']),
            activation=params['activation']
        ))
        model.add(Dropout(rate=params[f'dropout_l{i}']))

    # Output Layer
    model.add(Dense(units=n_outputs, activation='linear'))

    # Select and configure the optimizer
    if params['optimizer'] == 'adam':
        optimizer = Adam(learning_rate=params['learning_rate'])
    else:
        optimizer = RMSprop(learning_rate=params['learning_rate'])

    # Compile the model
    model.compile(optimizer=optimizer, loss='mean_absolute_error')

    return model
```

Listing 3.1: Dynamic Keras Model Construction Function.

Here are the key components seen in the code above:

- `Sequential`: the simplest Keras model, representing a linear stack of layers.
- `Dense`: the fully connected layer, where each neuron is connected to all neurons in the previous layer: its most important arguments are `units`, the number of neurons in the layer, and `activation`, the activation function to use.
- `Dropout`: a crucial **regularization** layer used to prevent overfitting, it works by randomly setting a fraction of neurons to 0 at each update step during training (the fraction is determined by the `rate`). This way, the network is forced to learn features that are not too dependent on any single neuron.

Model Compilation After defining the architecture, the model is compiled using the `.compile()` method. This step configures the model for training and requires two main arguments:

1. `optimizer`: This argument specifies the algorithm used to update the model's weights during training, in our model we can dynamically selects an optimizer from four possible options: `Adam`, `RMSprop`, `Adagrad`, or `Adadelta`. The chosen optimizer is then instantiated with a specific **learning rate** (lr) provided by the hyperparameter set. Additionally, gradient clipping is implemented by setting clipvalue=10000 to prevent the gradients from becoming too large, which enhances training stability, especially when dealing with volatile data like electricity prices. In the code, if no learning rate is provided or an invalid optimizer name is passed, the configuration defaults to the standard `Adam` optimizer.

The Training Loop and Callbacks The actual training is initiated by calling the `.fit()` method on the compiled model. The EPFToolbox enhances this process by using Keras Callbacks, which are utilities that can be applied at various stages of the training process to perform actions like stopping the training early, which is what the callback `EarlyStopping` does (it has also been implemented in the code). This Callback prevents overfitting this by monitoring a specified metric (typically the loss on a separate **validation set**, `val-loss`) and stopping the training automatically if that metric does not improve for a specified number of consecutive epochs (the `patience`).

```
from tensorflow.keras.callbacks import EarlyStopping

# Configure the EarlyStopping callback
# - monitor: The metric to watch. 'val_loss' is the loss on the
    validation data.
# - patience: Number of epochs with no improvement after which training
    will be stopped.
# - restore_best_weights: If True, the model weights from the epoch with
    the
#   best value of the monitored metric will be restored at the end of
    training.
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=25,
    restore_best_weights=True
)

# The fit method now includes the callback
```

```
# The number of epochs is set to a high value (e.g., 500), as EarlyStopping
# will find the optimal number of epochs automatically and stop the training.
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=500,
    batch_size=int(params['batch_size']),
    callbacks=[early_stopping],
    verbose=0 # Suppress the output for each epoch during optimization
)
```

Listing 3.2: Setting Up and Using the ‘EarlyStopping‘ Callback.

Another big advantage of using `EarlyStopping` is that the number of training epochs becomes an automatically tuned parameter rather than a fixed hyperparameter that needs to be optimized separately, this way the optimization process is more efficient and more robust against overfitting.

Hyperparameter Optimization in EPFToolbox

With the ability to dynamically build and train any DNN configuration, the final step is to automate the search for the best configuration, we do this using the `hyperopt` library to perform an intelligent search.

Bayesian Optimization When performing parameter tuning there is an almost infinite number of combinations that can be tried, there are some methods, like **Grid Search**, try all combinations on a predefined grid, becoming computationally infeasible very quickly. Other methods, like **Random Search** are more efficient, but it does not learn from its past evaluations. EPFToolbox employs an intelligent strategy: **Bayesian Optimization** [3], which treats the search for optimal hyperparameters as a statistical problem. The algorithm:

1. Builds a probabilistic surrogate model of the objective function.
2. Uses the surrogate model to decide which set of hyperparameters to evaluate next.
3. Updates the surrogate model with the new result and repeats the process.

This way the algorithm can learn which regions of the hyperparameter space are most promising, leading to a much more efficient search: the algorithm used by `hyperopt` in the EPFToolbox is the **Tree-structured Parzen Estimator (TPE)**.

Defining the Search Space with `hyperopt` The first step is to define the search space using `hyperopt`'s probabilistic distribution functions, which provides a way to describe the range of values for each hyperparameter.

```
from hyperopt import hp
import numpy as np

# A detailed search space for a DNN with up to 3 hidden layers
space = {
    # Architecture Hyperparameters
    'n_layers': hp.choice('n_layers', [1, 2, 3]),
    'neurons_l1': hp.quniform('neurons_l1', 50, 500, 25),
    'neurons_l2': hp.quniform('neurons_l2', 50, 500, 25),
    'neurons_l3': hp.quniform('neurons_l3', 50, 500, 25),
    'activation': hp.choice('activation', ['relu', 'tanh']),

    # Regularization Hyperparameters
    'dropout_l1': hp.uniform('dropout_l1', 0.0, 0.5),
    'dropout_l2': hp.uniform('dropout_l2', 0.0, 0.5),
```

```
    'dropout_l3': hp.uniform('dropout_l3', 0.0, 0.5),

    # Training Hyperparameters
    'optimizer': hp.choice('optimizer', ['adam', 'rmsprop']),
    'learning_rate': hp.loguniform('learning_rate', np.log(0.0001), np.log(0.01)),
    'batch_size': hp.choice('batch_size', [32, 64, 128, 256]),
}
```

Listing 3.3: A Detailed Hyperparameter Search Space for 'hyperopt'.

It is crucial to use the correct distribution for each parameter type:

- `hp.choice`: Used for selecting from a list of categorical options like the number of layers, activation function, or optimizer.
- `hp.quniform`: Used for discrete, uniformly distributed numerical values, such as the number of neurons.
- `hp.loguniform`: This is the correct choice for parameters that should be optimized on a logarithmic scale, like the learning rate: this distribution ensures the search explores different orders of magnitude effectively.

The Objective Function It is the most important part of the optimization process and it is a function that `hyperopt` calls for each trial and it takes one argument, which is a dictionary of hyperparameter values sampled from the space, and must return the value to be minimized (the validation loss).

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

def objective_function(params):
    """
    This function is called by hyperopt for each trial.
    It builds, trains, and evaluates a model with the given 'params'.
    """
    print(f"--- Testing hyperparameters: {params} ---")

    # STEP 1: PREPARE DATA
    # The full training dataset (from the calibration window) is split into
    # a new training set and a validation set for this specific trial.
    # This ensures an unbiased evaluation of this hyperparameter set.
    X_train, X_val, y_train, y_val = train_test_split(
        features, labels, test_size=0.2, random_state=42
```

```
    )

    # STEP 2: BUILD THE MODEL
    # The dynamic model-building function is called with the sampled params.
    model = build_model(
        params,
        input_shape=(X_train.shape[1],),
        n_outputs=y_train.shape[1]
    )

    # STEP 3: TRAIN THE MODEL
    # The model is trained using the EarlyStopping callback to find the
    # optimal number of epochs and prevent overfitting.
    early_stopping = EarlyStopping(monitor='val_loss', patience=25, restore_best_weights=True)

    model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=500,
        batch_size=int(params['batch_size']),
        callbacks=[early_stopping],
        verbose=0
    )

    # STEP 4: EVALUATE AND RETURN THE LOSS
    # The final performance is measured on the validation set.
    predictions = model.predict(X_val)
    validation_mae = mean_absolute_error(y_val, predictions)
    print(f"Validation MAE for this trial: {validation_mae:.4f}")

    # Hyperopt requires a dictionary containing the loss to minimize.
    # The 'status' key is also required.
    return {'loss': validation_mae, 'status': 'ok'}
```

Listing 3.4: The Conceptual Structure of the ‘hyperopt‘ Objective Function.

Starting the Search With the search space and objective function defined, the entire optimization process is launched with a single call to the *fmin* function.

```
from hyperopt import fmin, tpe, Trials

# The Trials object is used to store the history of the search,
# including the hyperparameters and loss for every single trial.
```

```
trials = Trials()

# 'fmin' starts the optimization process. It will call the
# objective_function 'max_evals' times.
best_hyperparameters = fmin(
    fn=objective_function,
    space=space,
    algo=tpe.suggest,
    max_evals=3000,
    trials=trials
)

print("--- Optimization Finished ---")
print("Best hyperparameters found:", best_hyperparameters)
```

Listing 3.5: Launching the Bayesian Optimization Search with 'fmin'.

At the end of the max-evals trials, the *fmin* function returns a dictionary containing the set of hyperparameters that yielded the lowest loss on the validation set across all trials: this will be the best configuration and it is saved and used to train a final model on the entire calibration window dataset, which is then used to make predictions on the test set.

3.4.4. Modifications for Zonal Price Forecasting

The EPFToolbox framework, was originally designed for single-output time series forecasting, to use it in this thesis it was adapted to the multi-output problem of forecasting prices for multiple electricity zones simultaneously. This required a few modifications, which focused more on the DNN model rather than on the optimization process, since the primary changes are concentrated in the model's architecture and how the loss is calculated.

Multi-Output Model Architecture

The biggest change is in the output layer of the DNN: instead of having a single neuron to predict one price, the output layer is constructed with a number of neurons equal to the number of zones being forecast (`N-ZONES`). In the `build-model` function shown previously, this is handled by the `n-outputs` argument:

```
# Inside the build_model function...

# The final layer's 'units' argument is set to the number of zones.
model.add(Dense(units=n_outputs, activation='linear'))
```

Listing 3.6: Modifying the Output Layer for Zonal Forecasting.

Each of these output neurons is responsible for predicting the price of a single, specific zone at a specific hour. All of these neurons receive their inputs from the same final hidden layer and this information sharing allows the model to learn high-level features that are relevant to the entire electricity system, exchanging information between hours and zones. The final output layer then maps these shared features to the specific price of its designated zone. This approach is more powerful than training a separate, independent model for each zone and it allows us to leverage the cross-zonal dependencies present in the data.

Aggregated Loss Function

The second adaptation concerns the loss function, when the model produces a vector of K predictions (one for each zone) and is compared against a vector of K true prices, the Keras framework handles the loss calculation. The loss function specified in `model.compile()` (e.g., `mean-absolute-error`) is applied element-wise. That is, the absolute error is calculated for each zone independently:

$$\text{error}_k = |y_k - \hat{y}_k| \quad \text{for } k = 1, \ldots, K \text{ zones}$$

These individual errors are then averaged across all zones to produce a single, scalar loss value for that data point (or for the entire batch).

$$\text{Loss} = \frac{1}{K} \sum_{k=1}^{K} |y_k - \hat{y}_k|$$

This single scalar loss is what is used by the backpropagation algorithm to calculate the gradients and update the network's weights: the optimizer will then be forced to learn a set of weights that performs well, on average, across all zones by minimizing this aggregated loss. The single training objective is what enables the model to learn the shared information mentioned earlier, making it a multi-variate forecasting system.

3.4.5. DNN Architecture and Hyperparameters

To have a better understanding of how a different number of hidden layers affects the performance, we tried 2 different DNN configurations, one with 2 hidden layers and one with 3 hidden layers. After performing the optimization through the modified EPFToolbox framework the best hyperparameters and architecture found for the 2 different DNN configurations are:

Table 3.5: Comparison of DNN Model Configurations and Performance

Parameter	**DNN 3 hidden layers**	**DNN 2 hidden layers**
Performance Metrics		
Test MAE	0.2227	0.2334
Test sMAPE	4.91%	5.12%
Validation Loss	0.1767	0.1711
Network Architecture		
Layer Neurons	121 / 230 / 285	92 / 400
Batch Normalization	Yes	Yes
Model Hyperparameters		
Learning Rate	0.00075	0.00109
Dropout Rate	0.2116	0.2496
Initialization	glorot_normal	he_uniform
Regularization	L_2	L_2
Input Scaler (X)	RobustScaler	StandardScaler
Target Scaler (Y)	RobustScaler	MinMaxScaler

These values have been found after 1500 evals, as suggested by the author of the framework[20]. Please note that these error metrics are much lower than the final ones, since the tuning took place on transformed data through the arcsinh transform; therefore, the data's scale was much smaller. Also, we can already observe that the 3 layer model performed slightly better than the two layer one.

The dataset was used as follows to perform training, validation and testing:

Table 3.6: Dataset Split for DNN

Set	Date Range	Duration
Training set	2022-01-01 – 2023-06-30	1.5 years
Validation set	2023-07-01 – 2024-06-30	1 year
Test set	2024-07-01 – 2025-06-30	1 year

4 | Data Description and Feature Engineering

This chapter describes the data used in this thesis, performs an exploratory analysis to observe its statistical properties, and details the preprocessing and feature engineering steps taken to prepare it for the forecasting models: these steps are essential for building robust and accurate models, especially given the complex nature of electricity market data.

4.1. Data Sources and Characteristics

The primary data set for this thesis consists of the hourly zonal electricity prices of the Italian Day-Ahead Market (MGP), sourced directly from the market operator, GME [10], and the data cover the seven Italian market zones. We have selected data from 2022 onward, therefore, excluding periods heavily influenced by prior geopolitical events (that are not the focus of this analysis), to build robust models that reflect current market conditions. Electricity price time series are known to exhibit typical characteristics, including multiple seasonalities (daily, weekly, and yearly), high volatility, and the presence of sudden and sharp price spikes [11]; these characteristics can be observed in the next figures.

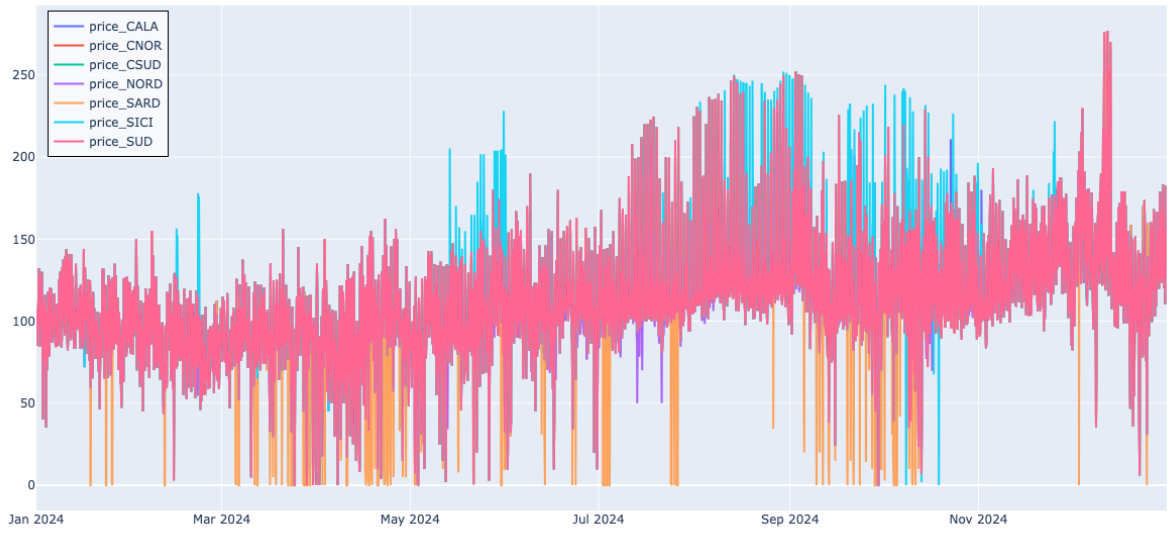

Figure 4.1: Price Series for All Zones.

Figure 4.1 clearly shows the volatile nature of the price series in all zones. It is also evident that prices can occasionally drop to zero, which is a key reason for choosing sMAPE over the standard MAPE as a percentage error metric, as discussed in Chapter 5.

4.2. Exploratory Data Analysis (EDA)

It is common practice to perform an exploratory analysis of our dataset to understand the data's underlying properties before building a model, in this section we verify the presence of the typical features mentioned previously like seasonality, non-normality, and non-stationarity in the Italian zonal price data.

4.2.1. Seasonality Analysis

To investigate seasonal patterns in the data, we can analyze the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots.

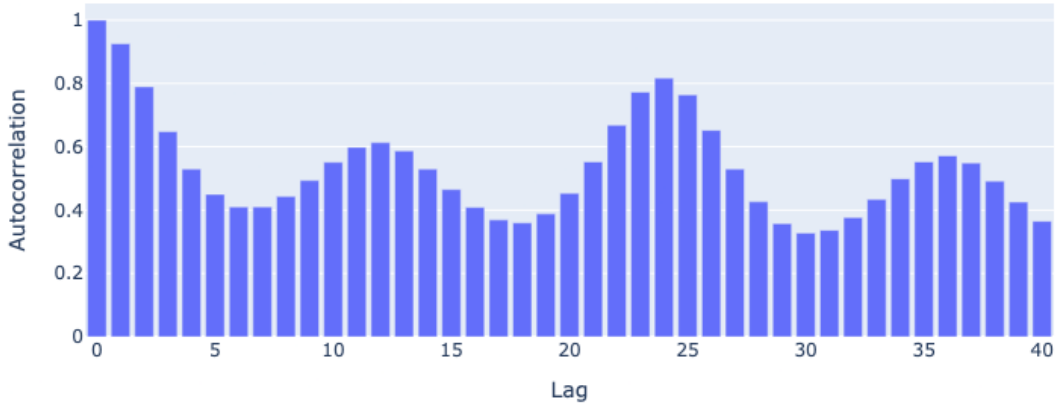

Figure 4.2: Autocorrelation Function Plot for Zone NORD (40 lags).

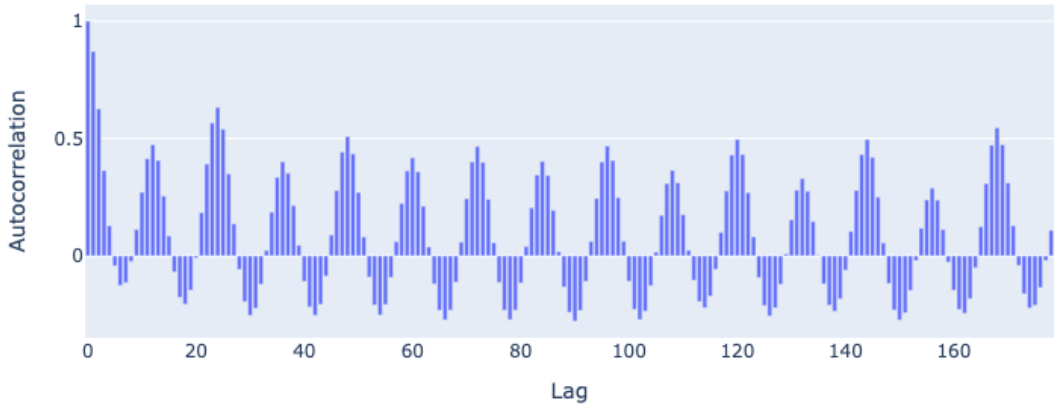

Figure 4.3: Autocorrelation Function Plot for Zone NORD (168 lags).

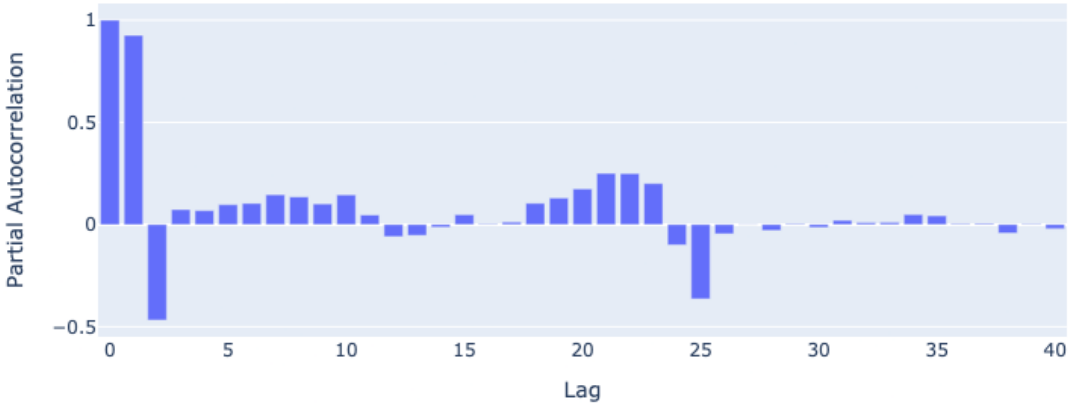

Figure 4.4: Partial Autocorrelation Function Plot for Zone NORD (40 lags).

As seen in Figure 4.2 and Figure 4.4, the significant peaks at lags that are multiples of 24 indicate a strong daily seasonality. Furthermore, Figure 4.3 shows a significant spike at lag 168 (which corresponds to 7 days), confirming a strong weekly seasonality as well. To exploit these patterns and increase the prediction ability of the Lasso-VARX model, the lagged prices of the previous day and the previous week (for the same hour) are included as autoregressive features in the model [18].

4.2.2. Distributional Properties

A primary assumption in statistical models is that the data follows a normal distribution, we can investigate this by examining the skewness and kurtosis of the price series.

Table 4.1: Skewness and Kurtosis for Zonal Prices.

Zone	Skewness	Kurtosis
price_CALA	-0.538	0.901
price_CNOR	-0.476	0.902
price_CSUD	-0.488	0.911
price_NORD	-0.467	1.014
price_SARD	-0.633	-0.313
price_SICI	-0.538	0.901
price_SUD	-0.528	0.887

The results in Table 4.1 show a consistent negative skewness across all zones, indicating a distribution with a longer left tail, which suggests that sharp price drops are a prominent feature in this dataset. Also, the positive kurtosis values for most zones point towards leptokurtic distributions, meaning the series have fat tails with more extreme values than a normal distribution: these are classic features of electricity prices, caused by market dynamics like sudden changes in renewable energy generation [21]. To formally test for normality, the Jarque-Bera (JB) test is employed, this test evaluates whether the sample data has skewness and kurtosis that match a normal distribution, under the null hypothesis that the data is normally distributed.

Table 4.2: Jarque-Bera Test for Normality.

Zone	JB Statistic	p-value
price_CALA	59.084	1.48e-13
price_CNOR	51.590	6.27e-12
price_CSUD	53.456	2.47e-12
price_NORD	57.023	4.15e-13
price_SARD	50.967	8.56e-12
price_SICI	59.084	1.48e-13
price_SUD	57.108	3.97e-13

The extremely small p-values in Table 4.2 lead to a strong rejection of the null hypothesis for all zones. This confirms that the price data is not normally distributed, a fundamental characteristic that requires the use of robust models or data transformations [32].

4.2.3. Stationarity and Long Memory

Another important property of a time series is stationarity, which we can test for using two different, but complimentary, tests: the Augmented Dickey-Fuller (ADF) test (null hypothesis: the series is non-stationary) and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test (null hypothesis: the series is trend-stationary).

Table 4.3: Stationarity Tests (ADF and KPSS).

Zone	ADF Stat	ADF p-value	KPSS Stat	KPSS p-value
price_CALA	-2.883	0.047	1.913	0.010
price_CNOR	-2.711	0.072	1.750	0.010
price_CSUD	-2.787	0.060	1.747	0.010
price_NORD	-2.797	0.059	1.614	0.010
price_SARD	-4.390	0.000	0.448	0.057
price_SICI	-2.883	0.047	1.913	0.010
price_SUD	-2.902	0.045	1.796	0.010

The results in Table 4.3 are conflicting: the ADF test suggests rejecting non-stationarity for most zones, while the KPSS test strongly rejects stationarity: this combination of results is a classic indicator of long memory, because the effect of past shocks on prices decays very slowly [29]. This long-range dependence is another classic feature of electricity prices that forecasting models must account for [21].

4.3. Data Preprocessing and Feature Engineering

Based on the data analysis, several steps are taken to prepare the data and construct features for the forecasting models.

4.3.1. Data Preprocessing

Two main preprocessing steps are applied: standardization of exogenous features and variance stabilization of the price data.

Standardization of Exogenous Variables

Exogenous variables like load and renewable generation forecasts have different scales. To ensure they contribute appropriately to the model, they are standardized using z-score normalization. The formula for a standardized variable x' is:

$$x' = \frac{x - \mu_x}{\sigma_x} \tag{4.1}$$

where x is the original feature value, μ_x is its mean, and σ_x is its standard deviation.

Variance Stabilizing Transformations

As mentioned before, electricity prices exhibit non-constant variance (heteroscedasticity) and extreme spikes, hence the application of a Variance Stabilizing Transformation (VST); since prices can be zero or negative, the standard logarithm cannot be used [26].

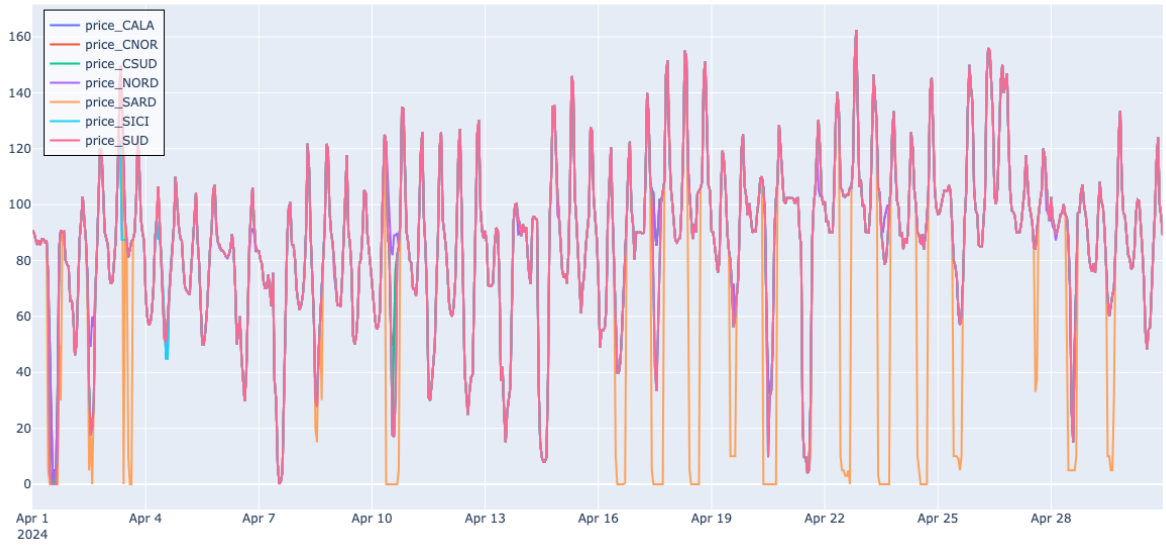

Figure 4.5: Monthly Price Series for All Zones.

As shown in Figure 4.5, price spikes and zero-price events are common, for this reason the robust inverse hyperbolic sine (asinh) transformation is used [20, 26]. The asinh transformation is defined as:

$$y'_t = \text{asinh}(y_t) = \ln(y_t + \sqrt{y_t^2 + 1}) \tag{4.2}$$

This transformation stabilizes the variance while correctly handling zero and negative values, leading to significant improvements in forecast accuracy [20].

4.3.2. Feature Engineering

This stage involves creating the final set of predictors for the models.

Exogenous Features

The selection of relevant exogenous variables is crucial. This thesis uses a set of predictors commonly chosen in state-of-the-art models [18, 20], and used in previous forecasting

models for Italian zonal prices [11], which are available at the time of the auction. The exogenous variables used in the models are:

Variable	Description
EF_Solar_IT_ZONE	Energy production from solar power plants from ENTSO-E.
EF_Wind_IT_ZONE	Energy production from wind power plants from ENTSO-E.
Load_IT_ZONE	Electric load (electricity demand) forecast from ENTSO-E.
Import_IT_ZONE	Electricity fluxes imported from abroad and from adjacent from ENSTO-E.
Export_IT_ZONE	Electricity fluxes exported abroad and to adjacent zones from ENSTO-E.
purchases_ZONE	Purchased electricity volumes.
sales_ZONE	Sold electricity volumes.
hhi_ZONE	Herfindahl–Hirschman market concentration Index.
MTI_Coal	Marginal technology index (COAL).
MTI_Wind	Marginal technology index (WIND).
MTI_Combined_Cycles	Marginal technology index (CC).
MTI_Other	Marginal technology index (OTHER).
congestion	Dummy variable to flag congestion.

Table 4.4: Exogenous Variables Used in the Models.

Some specifications regarding these exogenous variables:

- **MTI - Marginal Technology Index**, is an index provided by the GME that indicates which production technology is setting the price in a specific zone at a given time. The data was clustered into 4 groups to avoid creating dummy variables for all 12 categories, which would have added too many features. [11].
- **HHI - Hirschmann-Herfindahl Index**, represents market concentration and it is calculated as the sum of the shares of the volumes sold in the market by each market participant.
- **congestion**, is a dummy variable that is 1 when the zonal price differs from the *PUN*, therefore flagging the presence of a congestion. It is 0 if the two prices are equal.

datetime	price_CSUD	EF_Solar_IT_CSUD	EF_Wind_IT_CSUD
2023-01-01 00:00:00	195.90	0.0	3.0
2023-01-01 01:00:00	191.09	0.0	8.0
2023-01-01 02:00:00	187.95	0.0	3.0
2023-01-01 03:00:00	187.82	0.0	5.0
2023-01-01 04:00:00	187.74	0.0	7.0

datetime	Load_IT_CSUD	IT_CSUD > IT_SUD	IT_SUD > IT_CSUD
2023-01-01 00:00:00	4221.0	3163.0	5594.0
2023-01-01 01:00:00	3915.0	3163.0	5594.0
2023-01-01 02:00:00	3804.0	3163.0	5594.0
2023-01-01 03:00:00	3587.0	3163.0	5594.0
2023-01-01 04:00:00	3401.0	3163.0	5594.0

datetime	IT_SARD > IT_CSUD	IT_CSUD > IT_SARD
2023-01-01 00:00:00	870.0	720.0
2023-01-01 01:00:00	870.0	720.0
2023-01-01 02:00:00	870.0	720.0
2023-01-01 03:00:00	870.0	720.0
2023-01-01 04:00:00	870.0	720.0

datetime	purchases_CSUD	sales_CSUD	hhi_CSUD
2023-01-01 00:00:00	4.070	1.462	3.073
2023-01-01 01:00:00	3.844	1.457	3.090
2023-01-01 02:00:00	3.584	1.456	3.097
2023-01-01 03:00:00	3.408	1.425	3.217
2023-01-01 04:00:00	3.325	1.398	3.335

datetime	MTI_Coal	MTI_Wind	MTI_Combined_Cycles
2023-01-01 00:00:00	0	0	1.0
2023-01-01 01:00:00	0	0	1.0
2023-01-01 02:00:00	0	0	1.0
2023-01-01 03:00:00	0	0	1.0
2023-01-01 04:00:00	0	0	1.0

datetime	MTI_Other	congestion
2023-01-01 00:00:00	0.0	0.0
2023-01-01 01:00:00	0.0	0.0
2023-01-01 02:00:00	0.0	0.0
2023-01-01 03:00:00	0.0	0.0
2023-01-01 04:00:00	0.0	0.0

Table 4.5: Example of Exogenous Variables for Zone CSUD.

Technical Indicators for the Lasso-VARX Model

In addition to standard lagged prices and the previously mentioned exogenous variables, technical indicators are included as features to capture trends and momentum [8]: among the different technical indicators we decided to incorporate the *EMA - Exponential Moving Average* and the *EMSD - Exponentially Weighted Moving Standard Deviation*, their formulas are as follows:

The EMA at time t is defined as:

$$\text{EMA}_t = \begin{cases} x_0, & \text{if } t = 0 \\ \alpha x_t + (1-\alpha)\text{EMA}_{t-1}, & \text{if } t \geq 1 \end{cases} \tag{4.3}$$

where x_t is the price at time t and α is the smoothing factor. The exponentially weighted variance σ_t^2 is calculated as:

$$\sigma_t^2 = \begin{cases} 0, & \text{if } t = 0 \\ \alpha(x_t - \text{EMA}_t)^2 + (1-\alpha)\sigma_{t-1}^2, & \text{if } t \geq 1 \end{cases} \tag{4.4}$$

And the EMSD is the square root of the variance:

$$\text{EMSD}_t = \sqrt{\sigma_t^2} \tag{4.5}$$

Augmented Model Formulations

The inclusion of these technical indicators enhances the VARX models from Chapter 3. The new formulations explicitly include terms for the rolling mean ($\bar{y}_{d,h}^{\text{ewm7}}$) and rolling standard deviation ($\sigma_{d,h}^{\text{ewm7}}$), with a typical span of 7 days.

- **Concurrent ARX with Technical Indicators:**

$$y_{d,h} = \phi_{1,h} y_{d-1,h} + \phi_{7,h} y_{d-7,h} + \alpha_h \bar{y}_{d,h}^{\text{ewm7}} + \beta_h \sigma_{d,h}^{\text{ewm7}} + \theta_h \boldsymbol{X}_{d,h} + \epsilon_{d,h} \tag{4.6}$$

- **Full ARX (fARX) with Technical Indicators:**

$$y_{d,h} = \sum_{j=1}^{24} \left(\phi_{1,h}^{(j)} y_{d-1,j} + \phi_{7,h}^{(j)} y_{d-7,j} + \alpha_h^{(j)} \bar{y}_{d,j}^{\text{ewm7}} + \beta_h^{(j)} \sigma_{d,j}^{\text{ewm7}} \right) + \theta_h \boldsymbol{X}_{d,h} + \epsilon_{d,h} \tag{4.7}$$

- **Full ARX with Concurrent VAR component (sfVARX):**

$$
\begin{aligned}
y_{d,h} = & \sum_{j=1}^{24} \left(\phi_{1,h}^{(j)} y_{d-1,j} + \phi_{7,h}^{(j)} y_{d-7,j} + \alpha_h^{(j)} \bar{y}_{d,j}^{\text{ewm7}} + \beta_h^{(j)} \sigma_{d,j}^{\text{ewm7}} \right) \\
& + \sum_{k \neq i} \left(\phi_{1,h}^{(k)} y_{d-1,h}^{(k)} + \phi_{7,h}^{(k)} y_{d-7,h}^{(k)} + \alpha_h^{(k)} \bar{y}_{d,h}^{(k),\text{ewm7}} + \beta_h^{(k)} \sigma_{d,h}^{(k),\text{ewm7}} \right) \\
& + \boldsymbol{\Theta}_h \boldsymbol{X}_{d,h} + \epsilon_{d,h}
\end{aligned} \tag{4.8}
$$

where i refers to the target zone and k iterates over the other zones. Once again, there's no Full ARX with Full VAR component (fVARX), as mentioned before in chapter 3.

5 | Evaluation Criteria

Choosing the right metrics to evaluate and compare forecasting models is crucial, especially in a field like electricity price forecasting: due to the different complexities linked to the unique characteristics of electricity prices (such as high volatility, seasonality, and the presence of sudden spikes) no single metric can provide a complete picture of a model's performance [20]. This chapter outlines the set of evaluation criteria used in this thesis, including standard error metrics, scaled and percentage-based errors, and a statistical test for forecast comparison.

5.1. Point Forecast Accuracy Metrics

5.1.1. Standard Error Metrics (MAE and RMSE)

Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) are the most common metrics for evaluating point forecasts. The **Mean Absolute Error (MAE)** is the average of the absolute differences between the forecasted and actual values. It provides a clear and direct measure of the average forecast error magnitude. Its formula is:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{5.1}$$

where N is the number of observations, y_i is the actual value, and $\hat{y}_i$ is the forecasted value. A big advantage of MAE is that its measurement unit is the same as the original data (€/MWh), thus it's easier to interpret, but, as it is scale-dependent, it is difficult to use it to compare forecasts across datasets with different scales [20]. The **Root Mean Square Error (RMSE)** is the square root of the average of the squared forecast errors:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{5.2}$$

Due to the squaring of the error terms, the RMSE gives a much higher weight to large errors. This makes it a useful metric when large forecast errors are particularly undesirable; but, on the other hand, it will be much more sensitive to outliers than the MAE and its value is more difficult to interpret in terms of real-world economic costs [17, 20].

5.1.2. Percentage and Scaled Errors (sMAPE and MASE)

These percentage and scaled error metrics are often used to overcome the scale-dependency of MAE and RMSE. The **symmetric Mean Absolute Percentage Error (sMAPE)** is an alternative to the standard MAPE, designed to be more robust when actual values are close to zero, which is exactly the case of electricity prices, which could even go in the negatives. Its formula is:

$$\text{sMAPE} = \frac{100\%}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2} \tag{5.3}$$

While sMAPE provides a percentage error that is bounded between 0% and 200%, it is not without its problems, since statistically its distribution has an undefined mean and infinite variance, which can make it an unreliable metric in some contexts [20]. The **Mean Absolute Scaled Error (MASE)** was proposed to provide a scale-free and easily interpretable metric [14]. It is calculated by scaling the MAE of the forecast against the in-sample MAE of a simple naive forecast (usually a one-step forecast from the previous period):

$$\text{MASE} = \frac{\frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|}{\frac{1}{T-1}\sum_{t=2}^{T} |y_t^{in} - y_{t-1}^{in}|} \tag{5.4}$$

where the numerator is the MAE on the test set and the denominator is the MAE of the naive forecast on the training set, therefore a MASE value below 1 indicates that the forecast is better than the average naive forecast, while a value above 1 indicates it is worse. However, MASE is problematic for comparing models that use different training sets, such as those with rolling calibration windows, because the scaling factor in the denominator changes for each forecast [20].

5.2. Statistical Significance Testing

Along these error metrics we also need to use statistical significance tests to validate whether the difference in performance is statistically significant or simply due to chance, since error metrics can rank only models based on their average performance: the EPF

literature has been criticized for often neglecting this crucial step [20] and, to address this, we use the Diebold-Mariano test.

5.2.1. The Diebold-Mariano (DM) Test

The Diebold-Mariano (DM) test is a method used to compare the predictive accuracy of two different forecasts [9]: it works by testing the null hypothesis and checking that there is no difference in the accuracy of the two models. The test is based on the loss differential series, d_t, which is the difference between the loss functions of the two forecasts for each time step:

$$d_t = L(\epsilon_t^A) - L(\epsilon_t^B) \tag{5.5}$$

where L is a loss function (e.g., absolute error $|\epsilon|$ for MAE, or squared error ϵ^2 for RMSE) and ϵ_t^A and ϵ_t^B are the forecast errors of model A and model B, respectively. The DM test statistic is then calculated based on the mean and standard deviation of this loss differential series and a significant p-value allows us to reject the null hypothesis and conclude that one model is statistically superior to the other. When working with day-ahead electricity markets, where forecasts are generated for all 24 hours at once, it is recommended to use either a univariate test for each hour separately or a multivariate test that considers the entire 24-hour period as a single vector [20].

6 | Results and Model Comparison

6.1. LassoVARX Model Coefficients

As we have mentioned before, we have 3 possible configurations for the LassoVARX Model:

1. **Semi-Full ARX**: with 'concurrent' AR structure and no VAR structure.
2. **Full ARX**: with 'full' AR structure and no VAR structure.
3. **Semi-Full VARX**: with a 'full' AR structure and 'concurrent' VAR structure.

To test if the interactions between zones, different hours and the exogenous variables were actually providing an improvement in the models we compared these 3 configurations with and without exogenous variables, for a total of 6 models. Below are 6 heatmaps that show for each model which coefficients were used and explain quite well how the model is working. The dates from the coefficients are not the same as the ones used in the final test set since this study on the coefficients was conducted during the development of the model, but the heatmaps are still useful to understand the different models.

Firstly, we will look at the coefficients for the 3 models with exogenous variables.

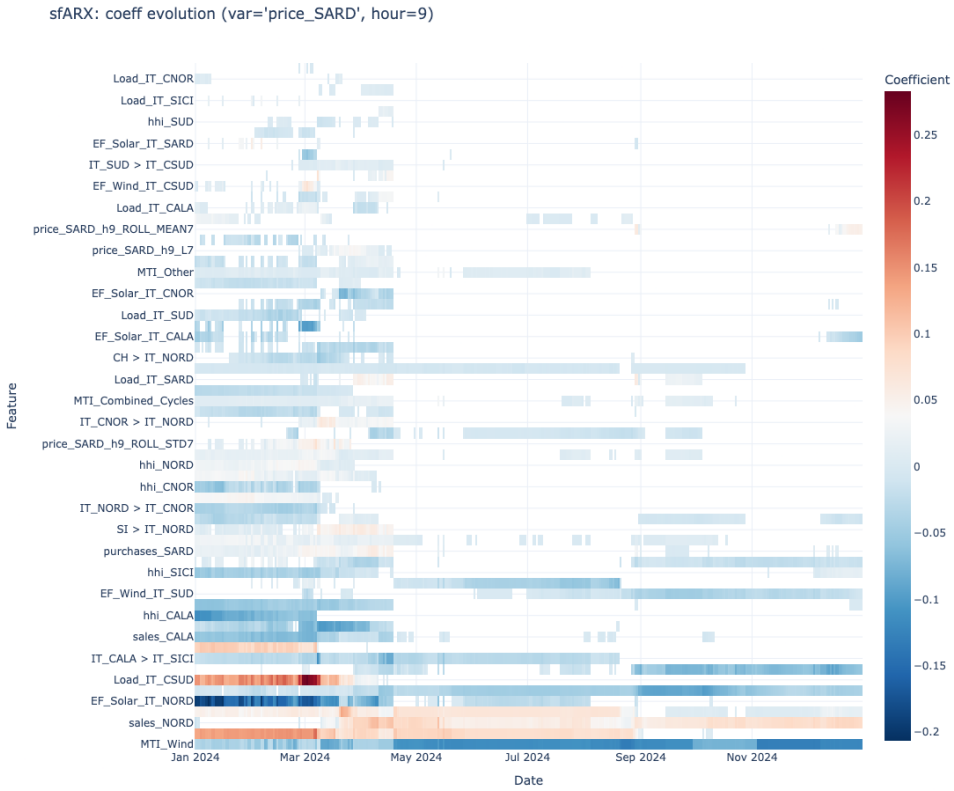

Figure 6.1: Coefficients sfARX with Exogenous Variables, zone: SARD, hour: 9

Figure 6.2: Coefficients fARX with Exogenous Variables, zone: SARD, hour: 9

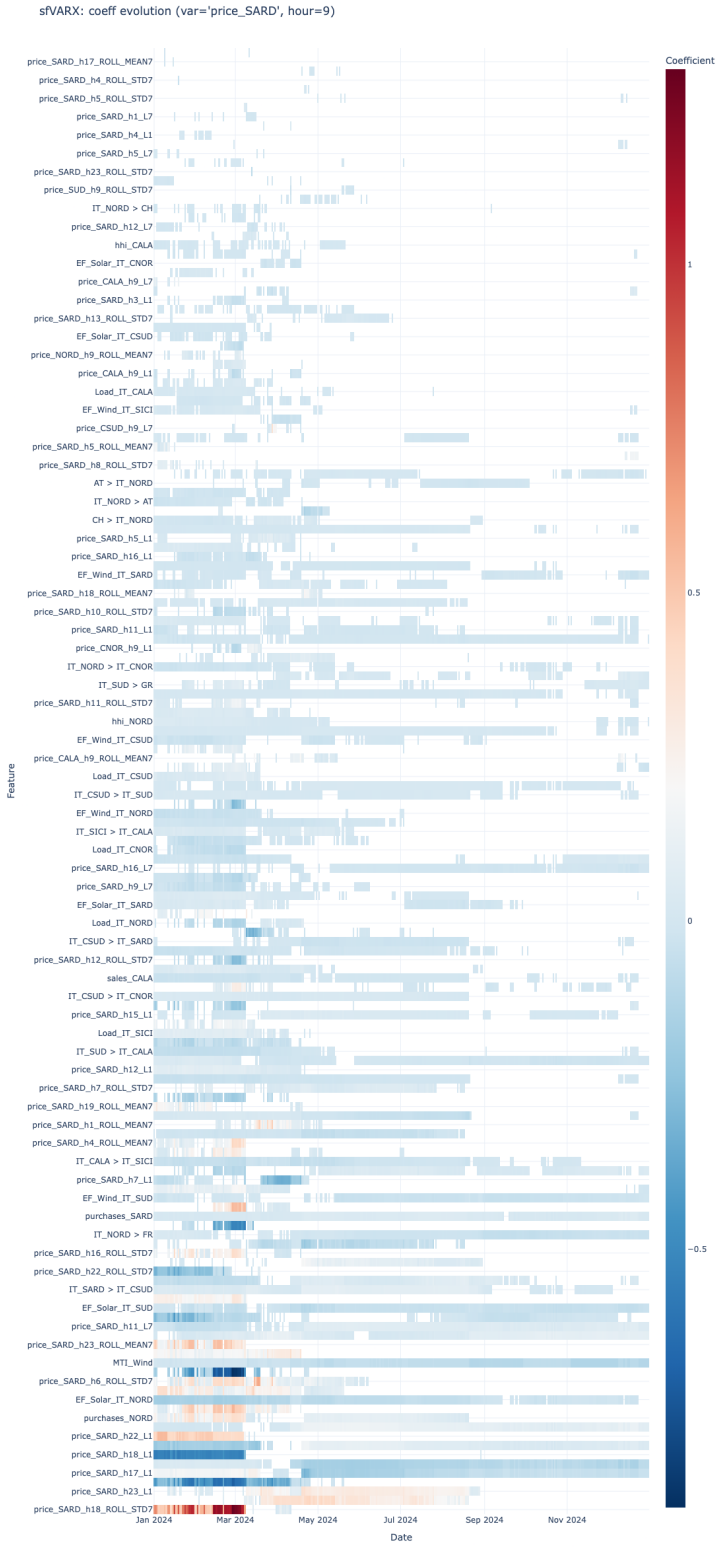

Figure 6.3: Coefficients sfVARX with Exogenous Variables, zone: SARD, hour: 9

Now, we will look at the coefficients for the 3 models without exogenous variables.

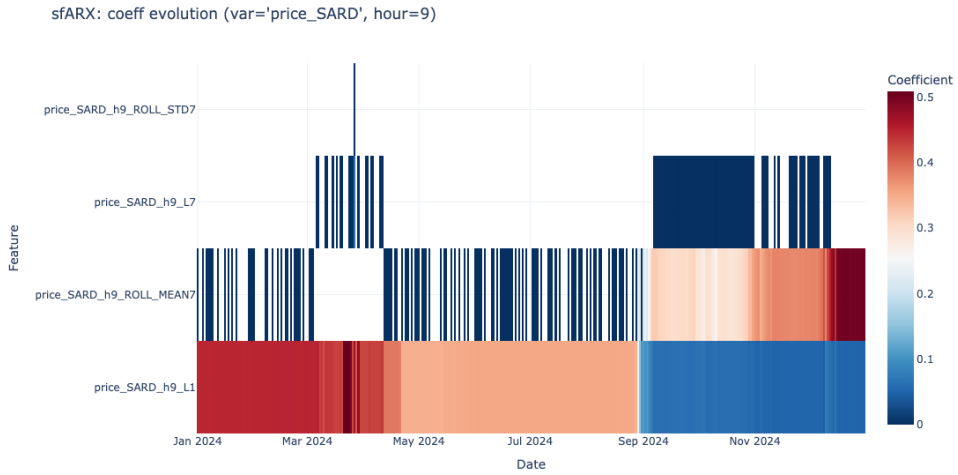

Figure 6.4: Coefficients sfARX without Exogenous Variables, zone: SARD, hour: 9

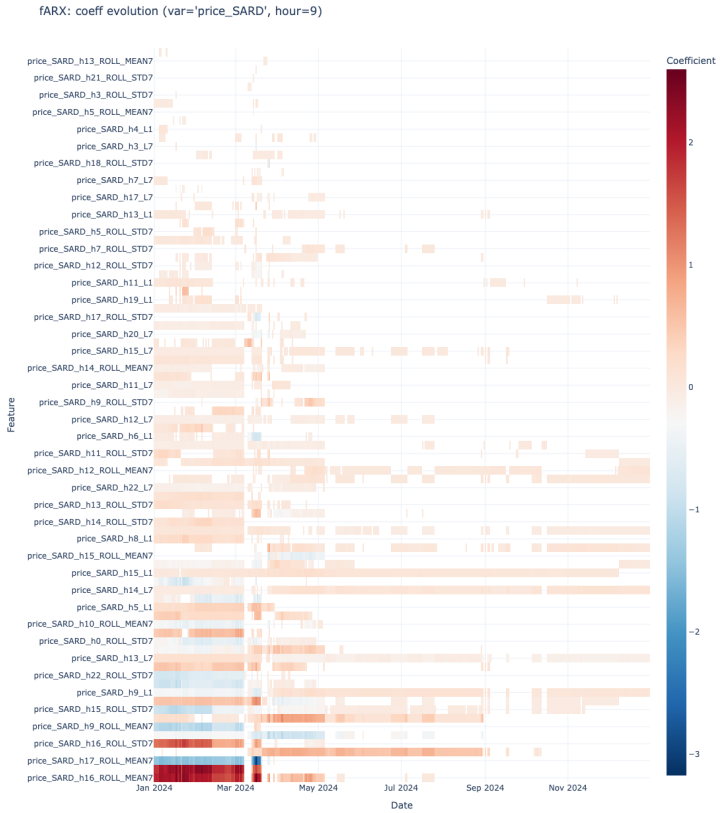

Figure 6.5: Coefficients fARX without Exogenous Variables, zone: SARD, hour: 9

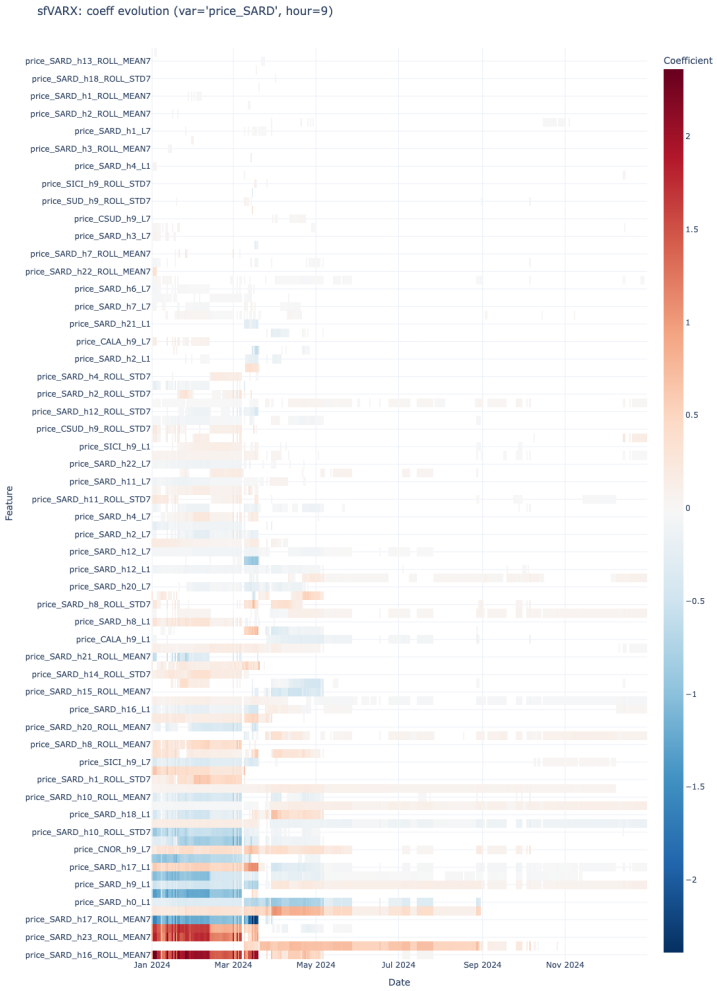

Figure 6.6: Coefficients sfVARX without Exogenous Variables, zone: SARD, hour: 9

More coefficient plots are available in the Appendix A, with different hours and zones. As we can see from these coefficient plots, the coefficients are chosen quite erratically by the LassoLARS algorithm, and there is no explainable logic behind it. This is why the DNN approach was also taken, because if the model is not really explicable it is losing its biggest advantage over a DNN, which will act as a black box, but may lead to better results.

6.2. Error Metrics and Performance Comparison

In the following page, table 6.1, which compares all the error metrics described in Chapter 5 is available. This is the table with the error metrics for each model (as a reference for the DM test, the recursive naive forecast model was chosen).

Since training time is also a crucial parameter to evaluate a model's performance [20], here are the different training times for the different models in their various configurations:

Table 6.2: Model Training/Validation times

Model	**Training Time**
LassoVARX sfARX with exog	35m 24s
LassoVARX fARX with exog	1h 41m 38s
LassoVARX sfVARX with exog	1h 50m 3s
LassoVARX sfARX without exog	5m 51s
LassoVARX fARX without exog	30m 45s
LassoVARX sfVARX without exog	44m 24s
DNN 2 hidden layers tuning	7h 41m 53s
DNN 2 hidden layers training	1h 33m 03s
DNN 3 hidden layers tuning	9h 42m 30s
DNN 3 hidden layers training	1h 45m 21s

Table 6.1: Comparison of Forecasting Model Performance Metrics

Model	**MAE**	**sMAPE**	**RMSE**	**MASE**	R^2	**DM_stat**	**DM_p_value**
Naive Model	26.1329	26.4879	35.9356	0.7869	-0.0667	0.0000	1.0000
Hybrid Model	39.3129	42.9871	49.5761	1.1838	-1.0555	-40.4702	0.0000
LassoVARX sfARX without exog	16.0704	17.0494	23.3098	0.4839	0.5695	30.8104	0.0000
LassoVARX fARX without exog	16.0523	16.6944	26.9172	0.4834	0.4063	4.1525	0.0000
LassoVARX sfVARX without exog	16.1361	16.7905	27.7022	0.4859	0.3748	2.3882	0.0169
LassoVARX sfARX with exog	15.9292	16.5968	22.8171	0.4797	0.5862	17.2712	0.0000
LassoVARX fARX with exog	15.2820	15.9083	22.4251	0.4602	0.6010	22.8638	0.0000
LassoVARX sfVARX with exog	15.2849	15.9032	22.4456	0.4603	0.6001	22.4613	0.0000
DNN model 2 hidden layers	15.9616	15.7343	22.7732	0.4806	0.5858	31.6152	0.0000
DNN model 3 hidden layers	15.0826	15.0179	22.2713	0.4542	0.6008	35.7204	0.0000

All algorithms were trained on an M2 Macbook Air, here are its specifications:

Table 6.3: Specifications of the Apple M2 Chip - MacBook Air

Feature	Specification
Chip	Apple M2
Manufacturing Process	5nm (second-generation)
Transistor Count	20 billion
CPU	8-core
- Performance Cores	4
- Efficiency Cores	4
GPU	8-core
Neural Engine	16-core
Memory	
- Unified Memory (RAM)	8 GB
- Memory Bandwidth	100 GB/s

6.3. Daily and Monthly Forecast Plots

To better visualize the performance of the models, below are some plots featuring the comparison of the forecasting accuracy for a certain day or month. To avoid having a plot that is too cluttered for the LassoVARX and DNN we chose the 2 best performing models, which are the sfVARX model and the 3 layer DNN, as reported in table 6.1. More plots are available in the appendix A.

From the daily plots we are able to observe how the more sophisticated models, LassoVARX and DNN, are able to capture more accurately the price movements and the volatility, whereas from the monthly plots we can see how the Hybrid, and at times the Naive model, struggle to correctly follow the price movements and follow more of a daily cycle.

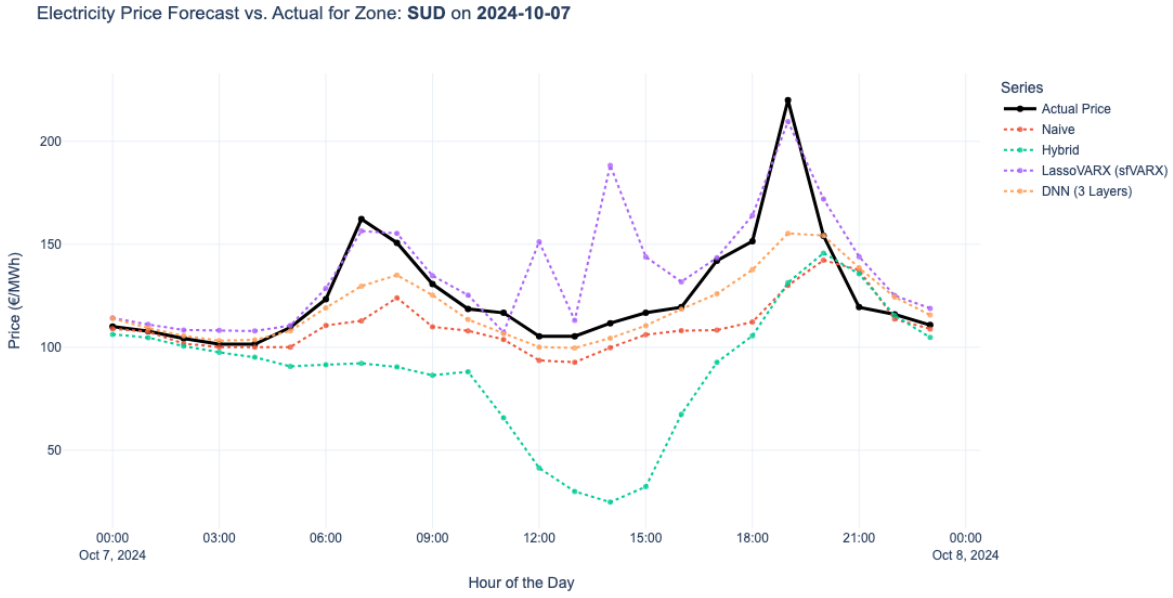

Figure 6.7: Electricity Price Forecasts Comparison for Zone: NORD on 2024-09-04

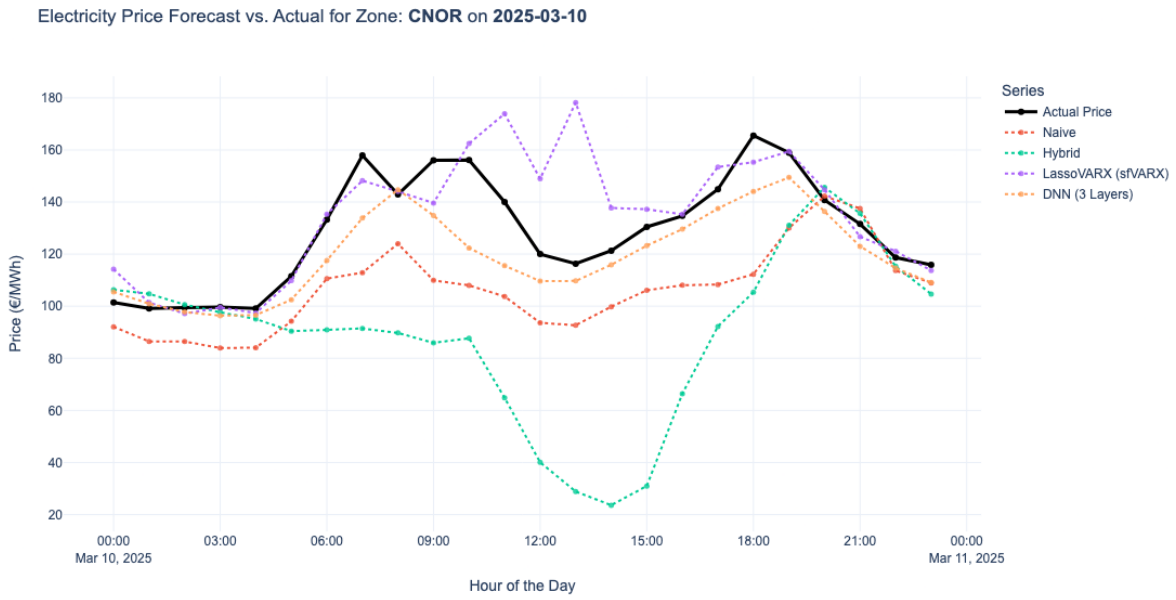

Figure 6.8: Electricity Price Forecasts Comparison for Zone: SUD on 2024-07-16

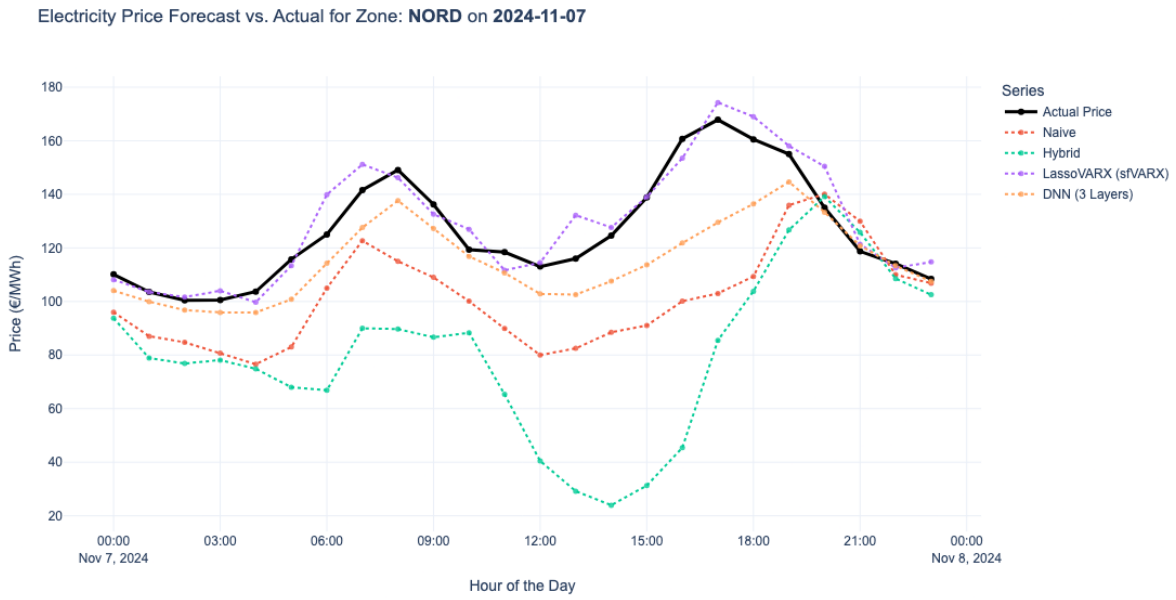

Figure 6.9: Electricity Price Forecasts Comparison for Zone: NORD on 2024-10-02

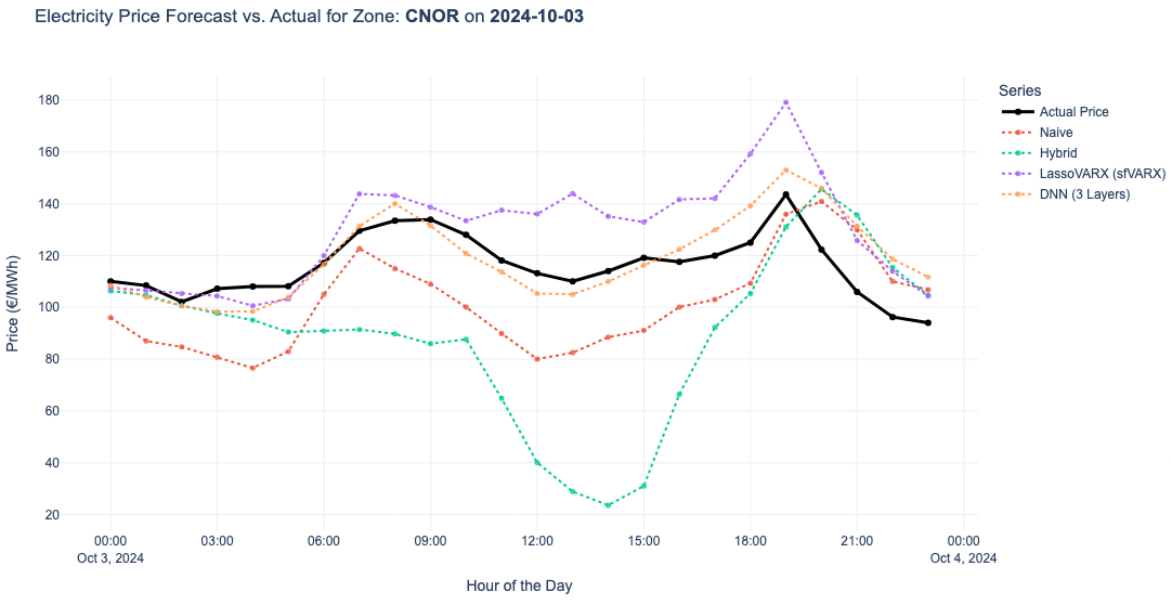

Figure 6.10: Electricity Price Forecasts Comparison for Zone: CNOR on 2025-03-18

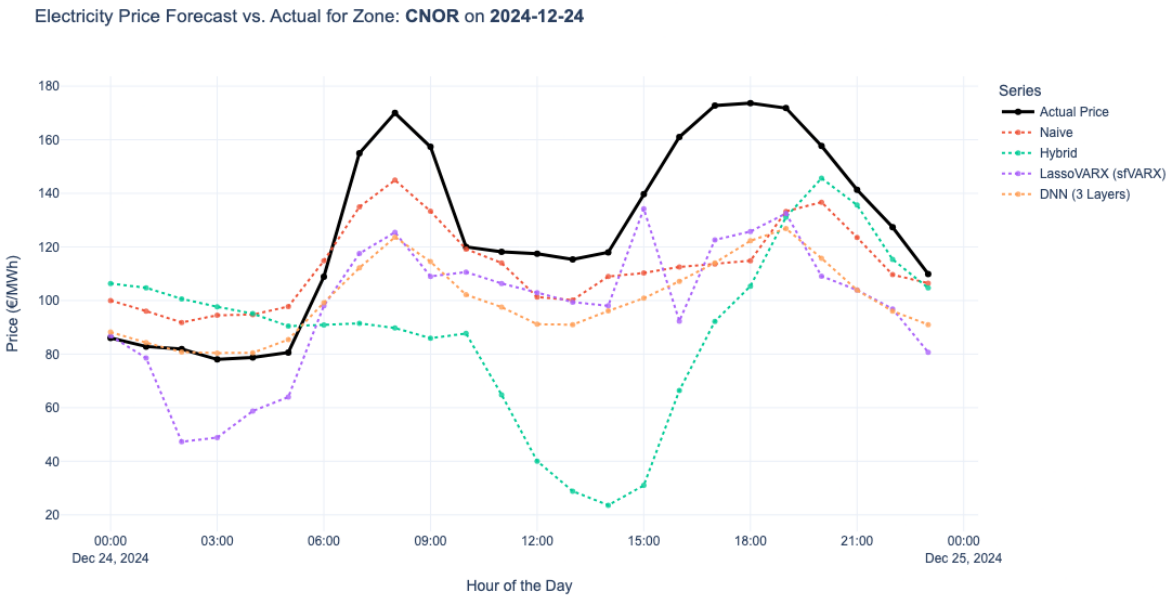

Figure 6.11: Electricity Price Forecasts Comparison for Zone: SUD on 2025-05-05

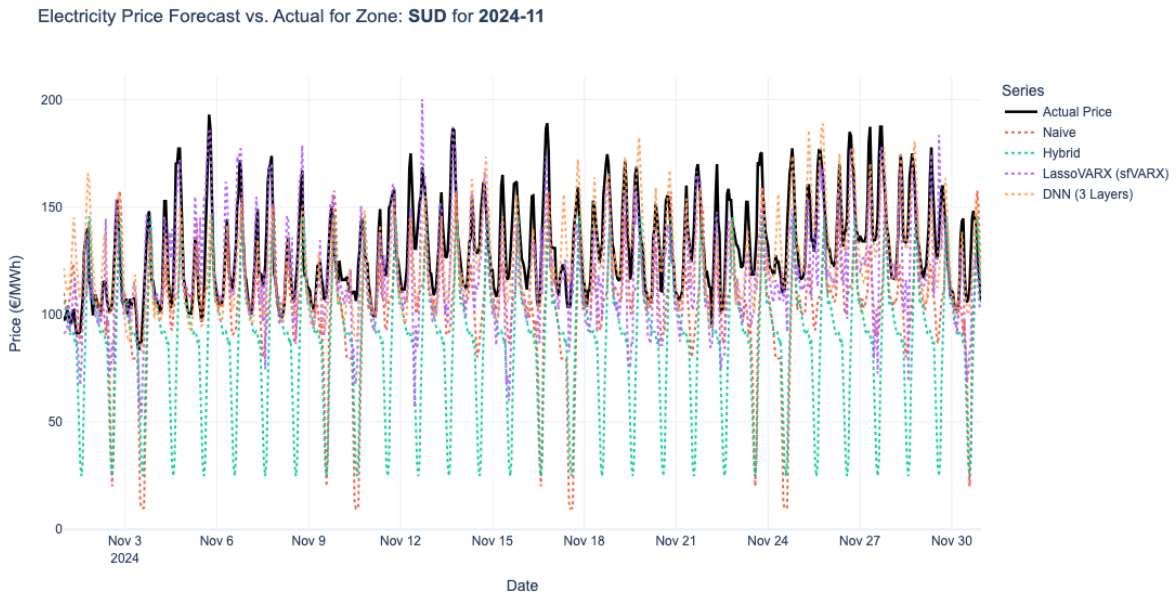

Figure 6.12: Monthly Electricity Price Forecasts Comparison for Zone: SUD on 2024-11

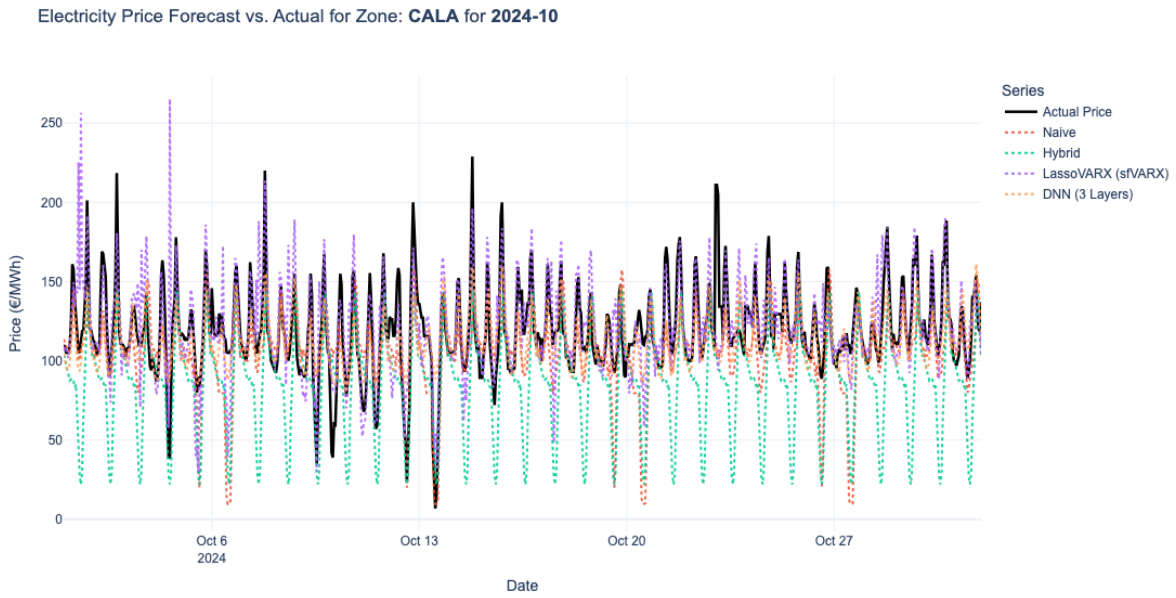

Figure 6.13: Monthly Electricity Price Forecasts Comparison for Zone: CALA on 2024-10

6.4. Conclusions

This research was aimed at finding the best methods for forecasting zonal electricity prices in the Italian Day-Ahead Market. We tested a variety of models against each other, from a simple Naive benchmark to an advanced Deep Neural Network (DNN) and Lasso-VARX frameworks.Our goal was to build on existing research and get a clear picture of each model's strengths and weaknesses in this environment.

The results show that the more complex 3 layer Deep Neural Network was the standout performer, achieving the lowest error across all our metrics (MAE of 15.08 €/MWh): this confirms what many recent studies have found, which is that deep learning is particularly skilled in finding the complicated, nonlinear dynamics of volatile electricity markets [20, 30]. Using EPFToolbox to automatically handle the architecture and tuning was critical to reaching this level of accuracy, although the high computational demand for both training and tuning is a practical drawback for real-world use [20].

Interestingly, the Lasso-VARX models were not far behind, proving to be highly competitive alternatives, in some configurations even better than a simpler 2 layer Deep Neural Network. The configurations that included a full autoregressive structure and data from other zones (fARX and sfVARX) performed nearly as well as the top DNN. This really highlights how useful LASSO regularization is for automatically selecting important features when you have a lot of data to work with [27].

Following the results of previous work on the Italian market, we confirmed that including external variables is essential for getting better results [11]. These statistical models represent an interesting trade-off, offering robust performance that is much less computationally intensive than a DNN [20]. On the other hand, the hybrid SARIMAX-LSTM model performed quite poorly, and failed to do better than even our most basic Naive benchmark. It is likely that its iterative forecasting process is susceptible to compounding errors over time, making this particular hybrid design a poor fit for the high volatility and long forecast period of our problem.

In the end, our findings, validated as statistically significant using the Diebold-Mariano test [9], align well with the broader conversation in electricity price forecasting. While Deep Neural Networks remain the benchmark for pure accuracy, sophisticated statistical models like Lasso-VARX are still powerful and relevant, especially when processing power is a constraint.

For future work, it would be interesting to test more accurate deep learning architectures, developed through more powerful calculators, which would allow a better tuning (possibily with a higher number of evaluations) and the training of more complex, even deeper, Neural Networks. Another interesting research topic could be the further exploration of cross-zonal dependencies that the sfVARX and DNN models captured to better understand the dynamics of the Italian electricity market.

A | Appendix A

A.0.1. Additional LassoVARX Coefficient Plots

Following are more coefficient plots from the LassoVARX model:

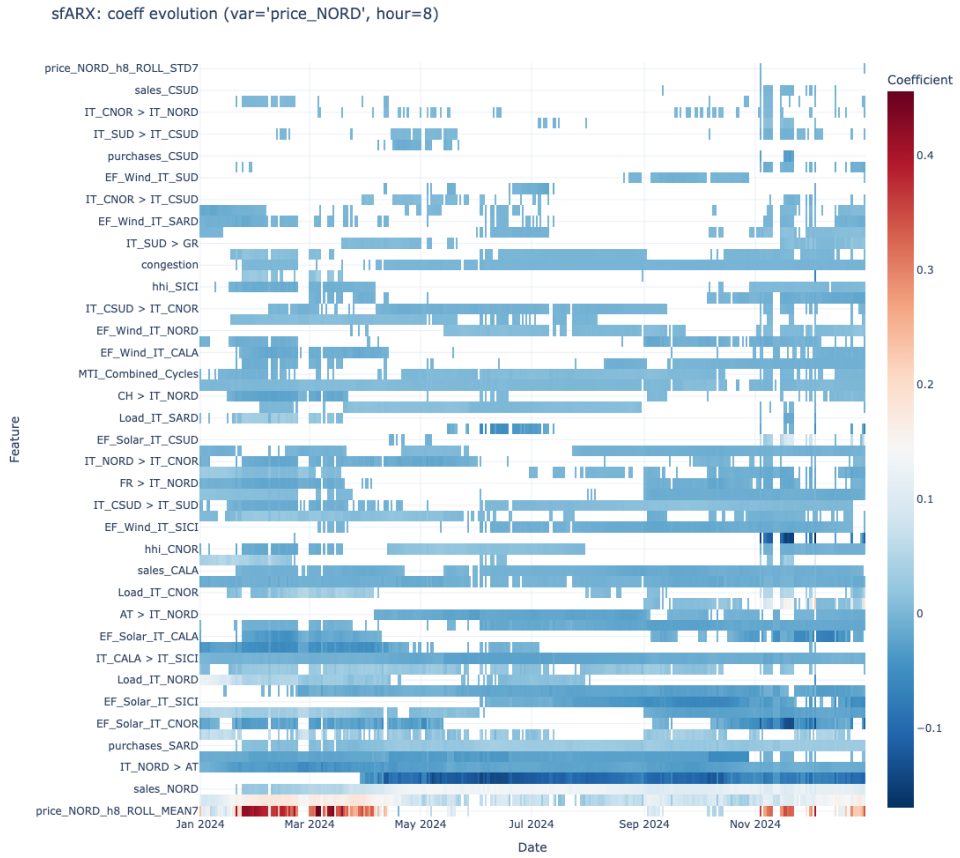

Figure A.1: Coefficients sfARX with Exogenous Variables, zone: CNOR, hour: 16

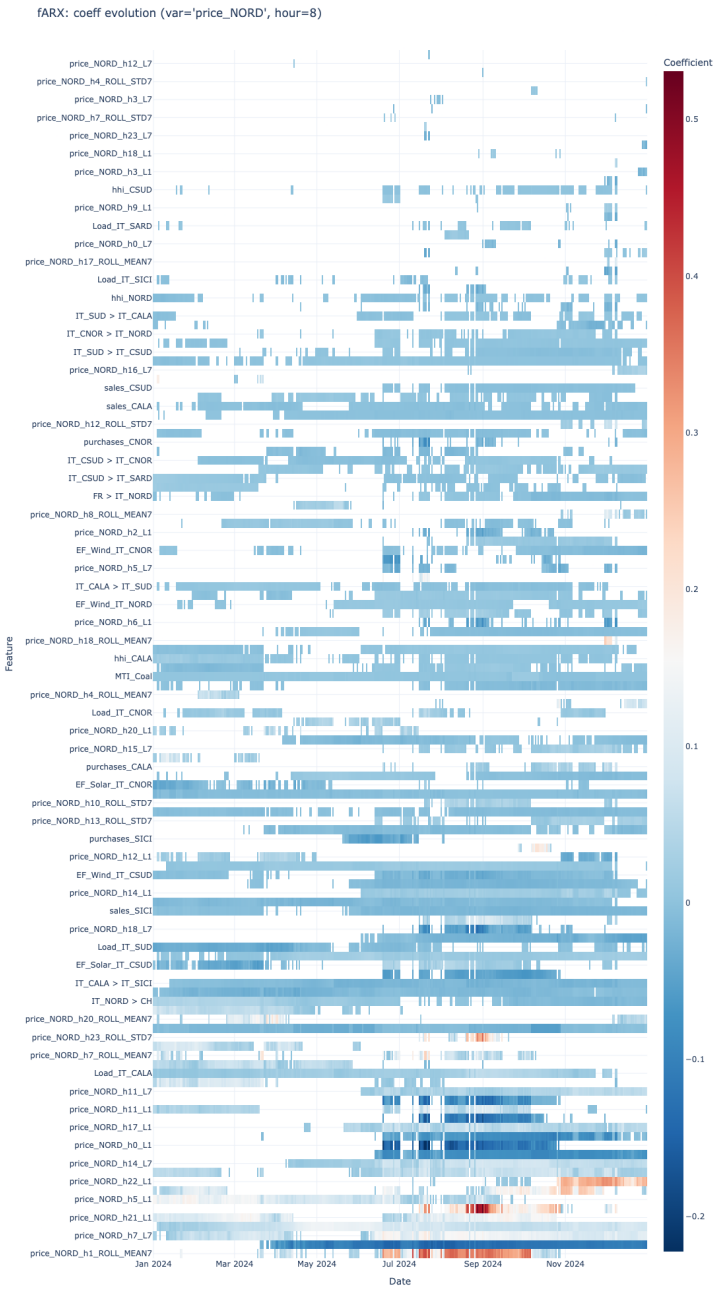

Figure A.2: Coefficients fARX with Exogenous Variables, zone: CNOR, hour: 16

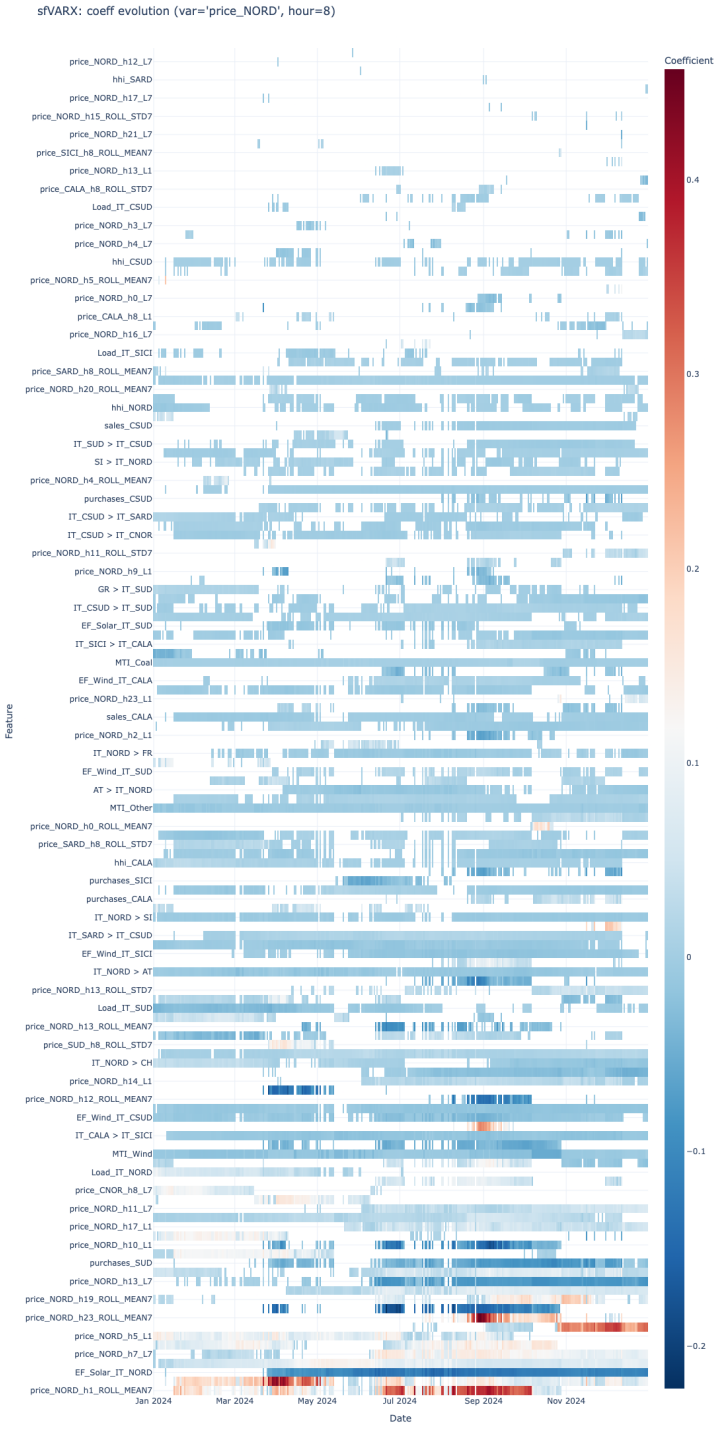

Figure A.3: Coefficients sfVARX with Exogenous Variables, zone: CNOR, hour: 16

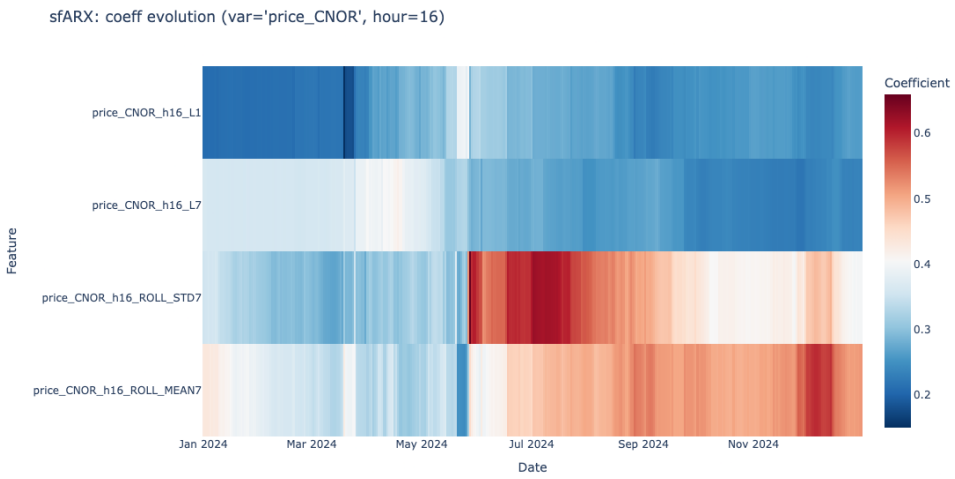

Figure A.4: Coefficients sfARX without Exogenous Variables, zone: CNOR, hour: 16

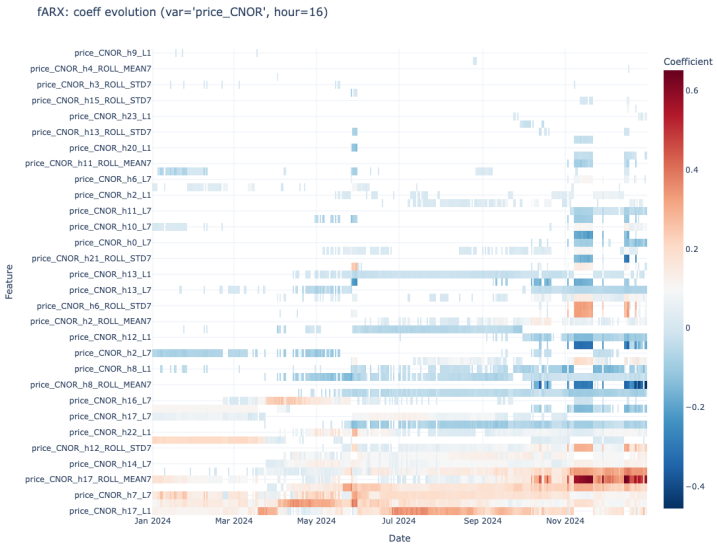

Figure A.5: Coefficients fARX without Exogenous Variables, zone: CNOR, hour: 16

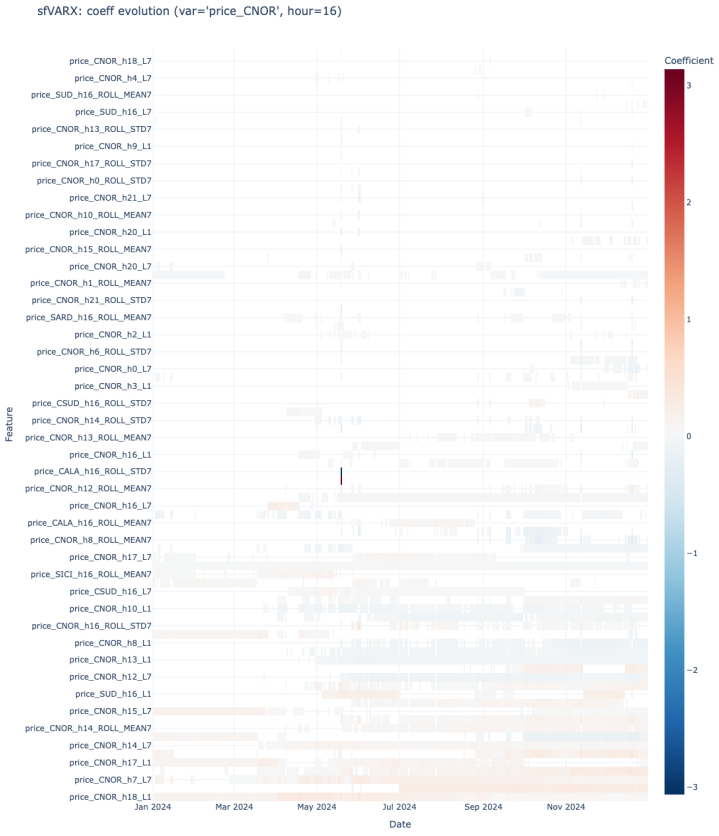

Figure A.6: Coefficients sfVARX without Exogenous Variables, zone: CNOR, hour: 16

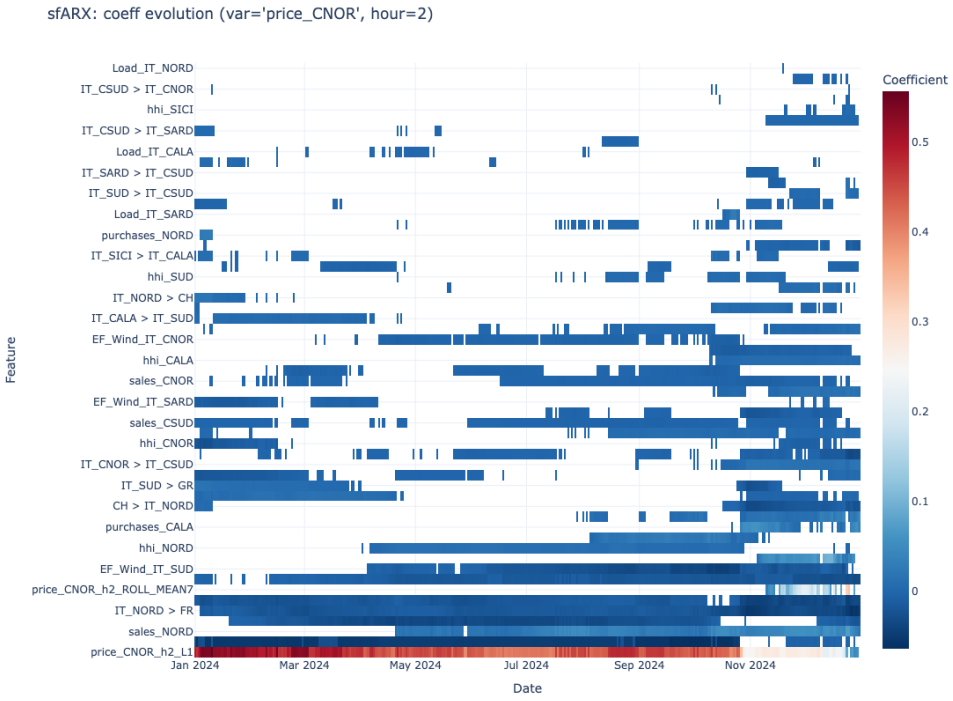

Figure A.7: Coefficients sfARX with Exogenous Variables, zone: NORD, hour: 8

Figure A.8: Coefficients fARX with Exogenous Variables, zone: NORD, hour: 8

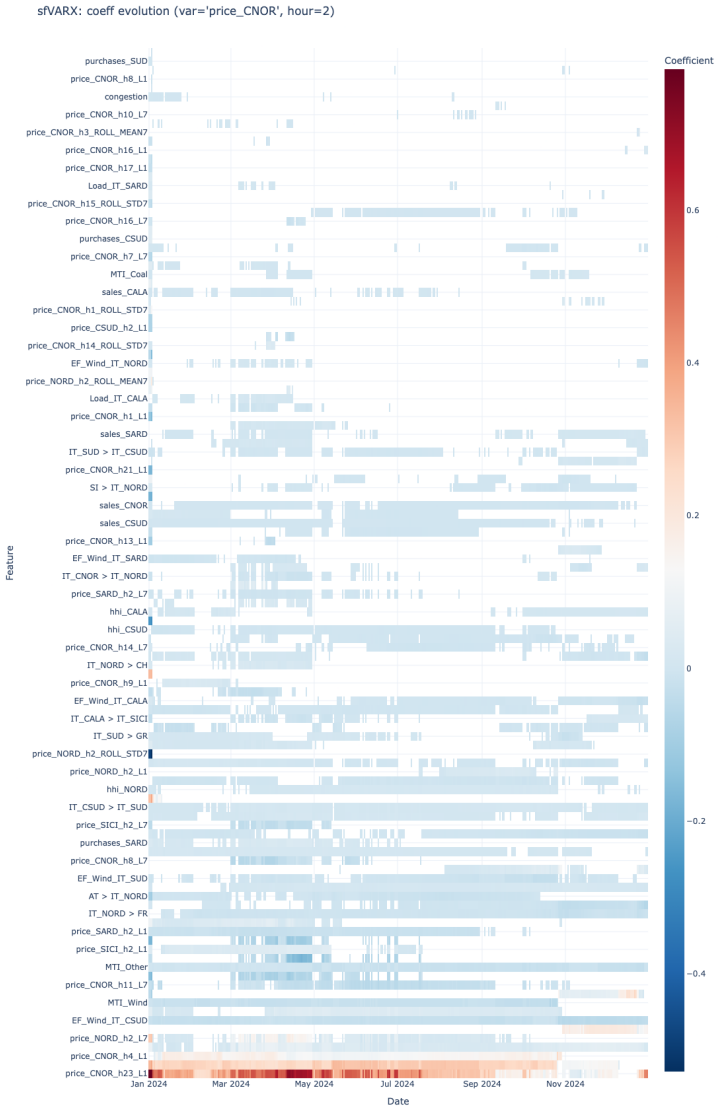

Figure A.9: Coefficients sfVARX with Exogenous Variables, zone: NORD, hour: 8

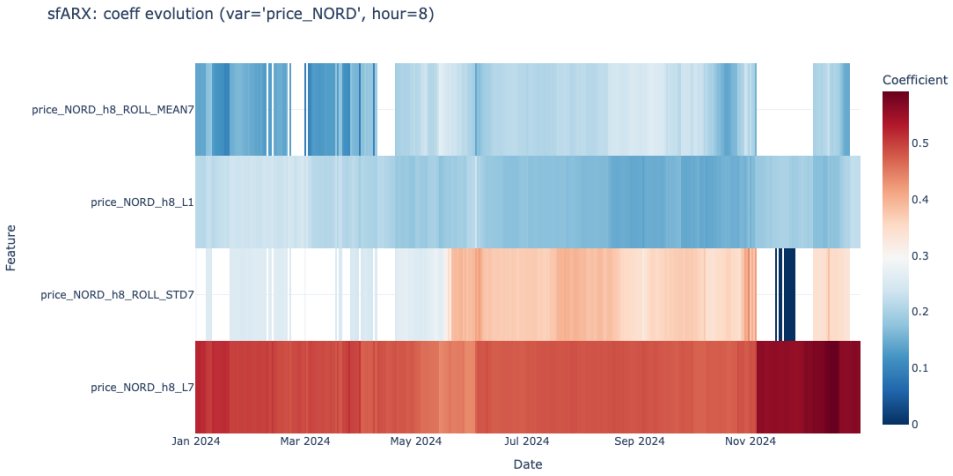

Figure A.10: Coefficients sfARX without Exogenous Variables, zone: NORD, hour: 8

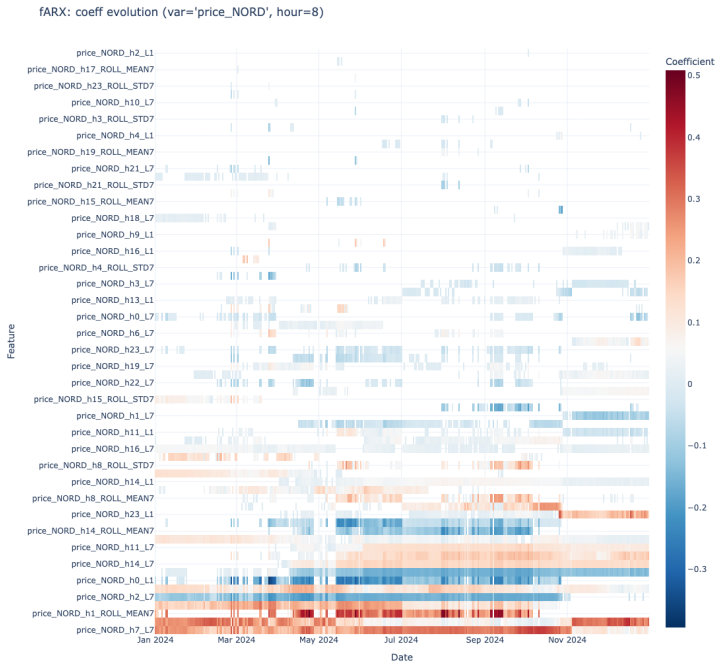

Figure A.11: Coefficients fARX without Exogenous Variables, zone: NORD, hour: 8

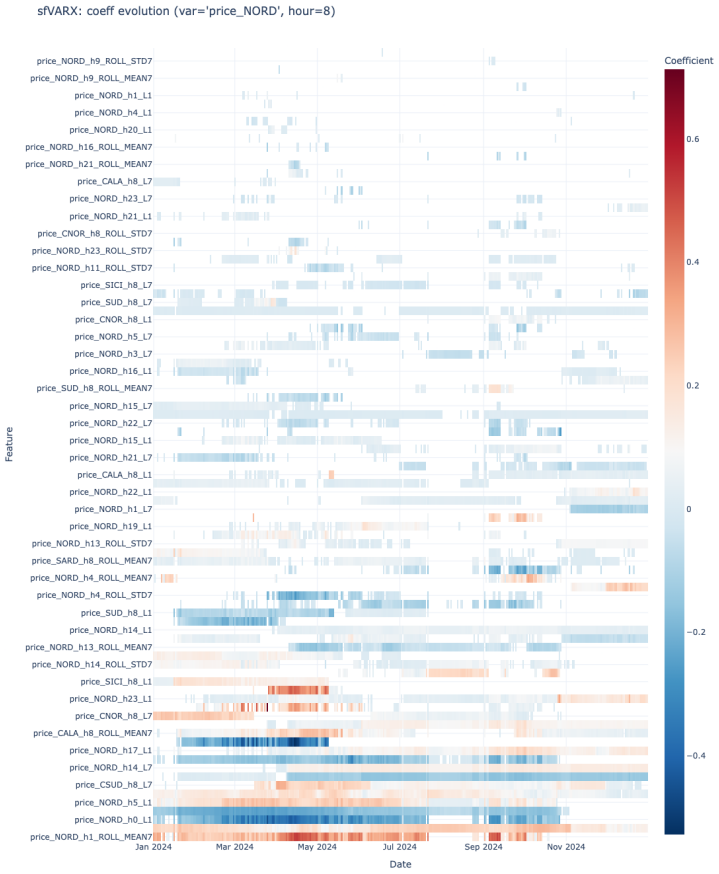

Figure A.12: Coefficients sfVARX without Exogenous Variables, zone: NORD, hour: 8

A.0.2. Additional Forecasts Comparison Plots

Following are more comparison plots for the forecasts from different models:

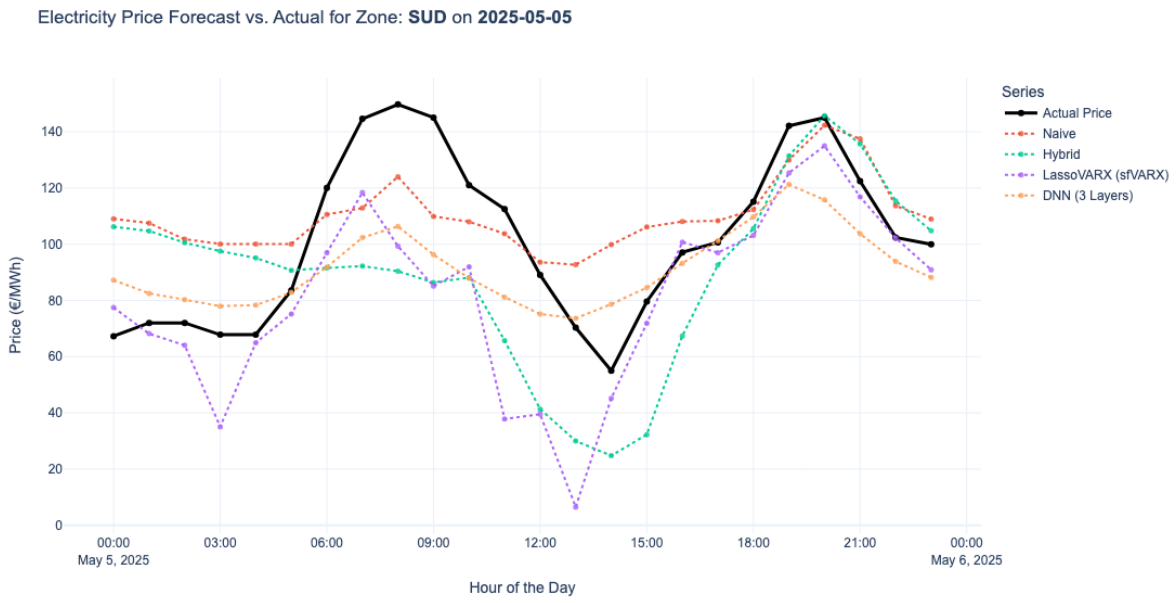

Figure A.13: Electricity Price Forecasts Comparison for Zone: CNOR on 2024-12-24

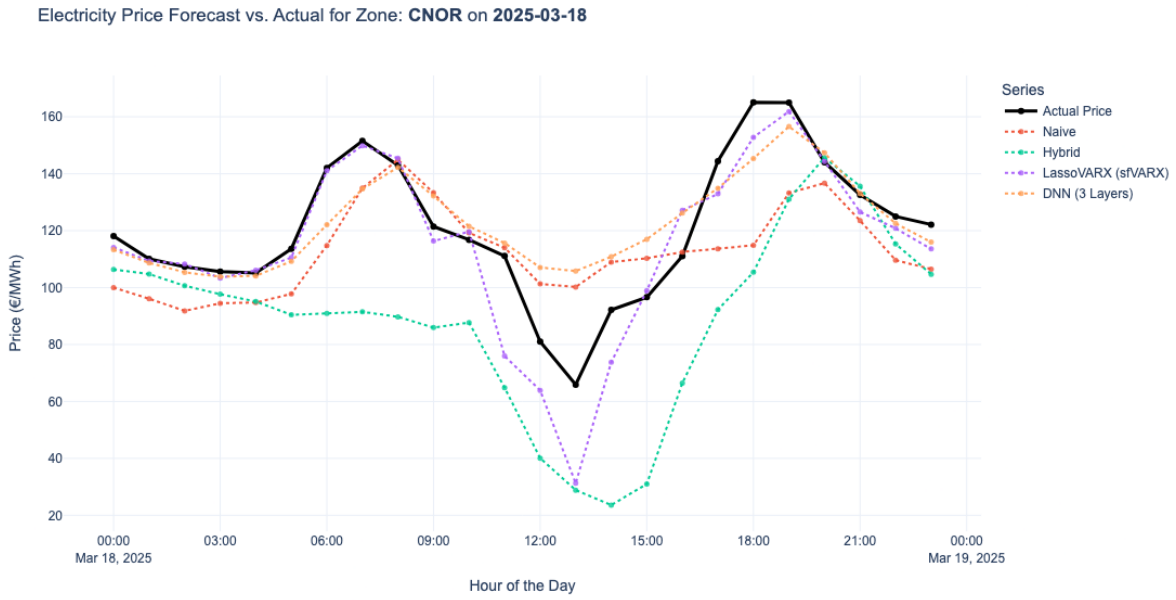

Figure A.14: Electricity Price Forecasts Comparison for Zone: CNOR on 2024-10-03

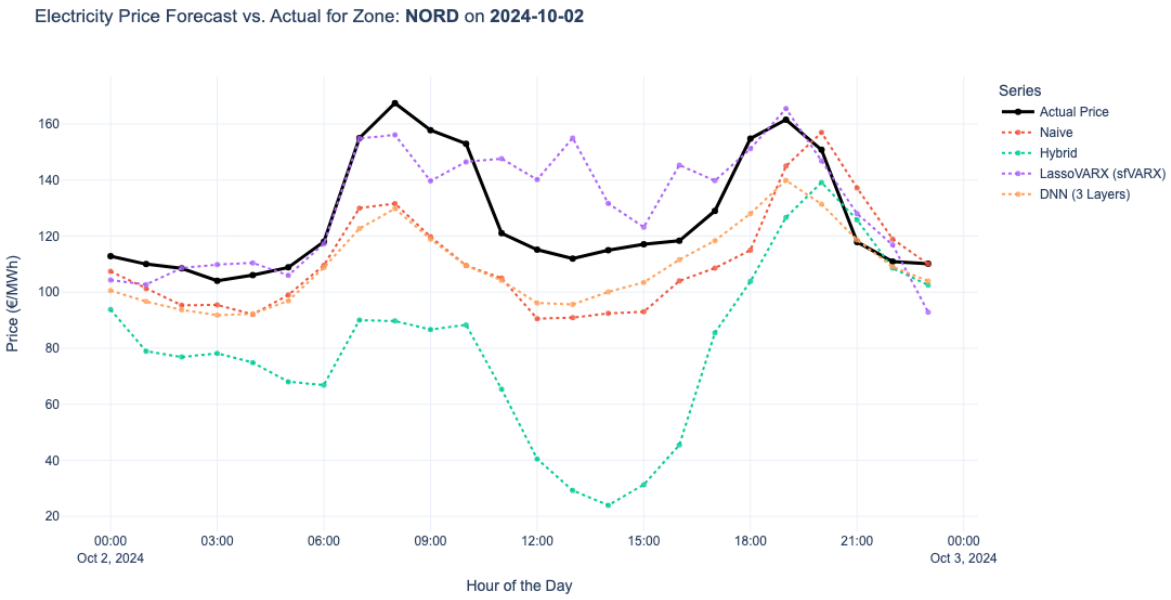

Figure A.15: Electricity Price Forecasts Comparison for Zone: NORD on 2024-11-07

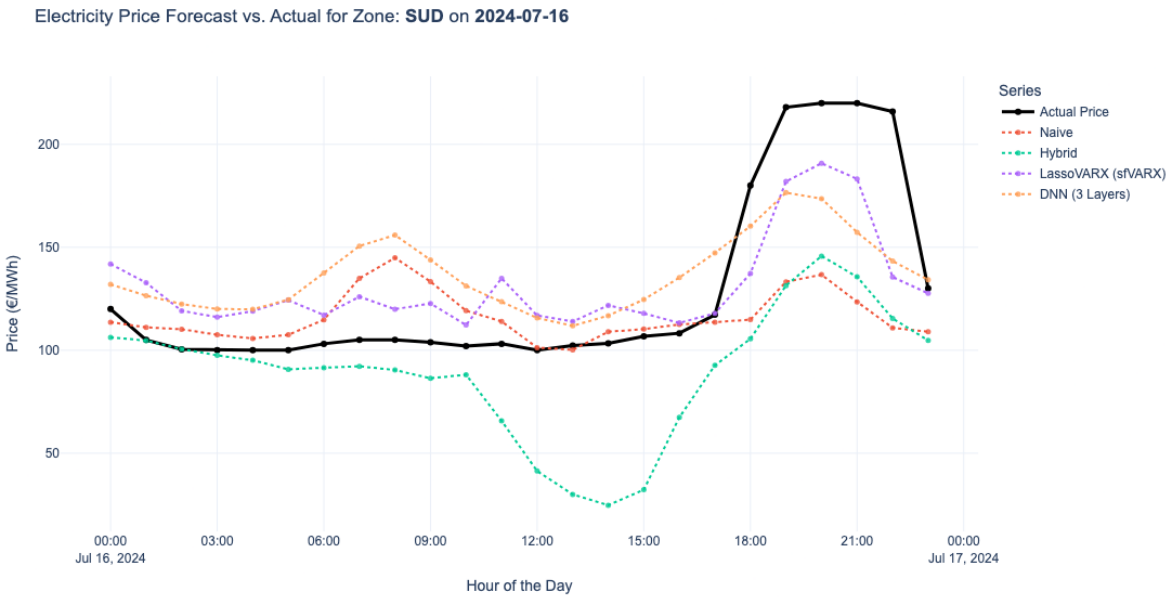

Figure A.16: Electricity Price Forecasts Comparison for Zone: CNOR on 2025-03-10

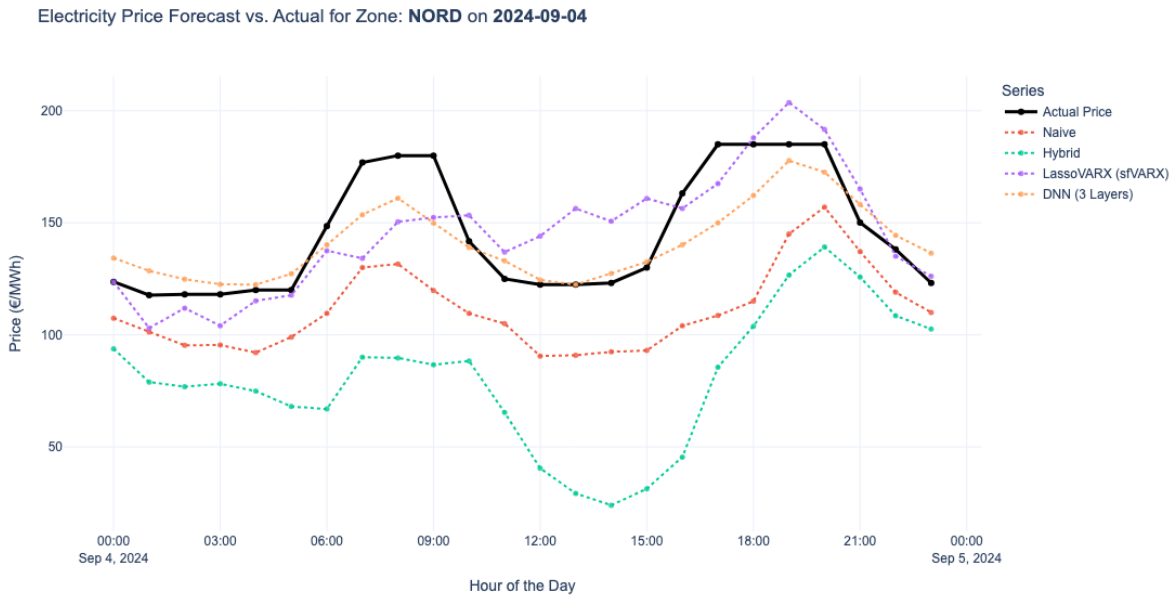

Figure A.17: Electricity Price Forecasts Comparison for Zone: SUD on 2024-10-07

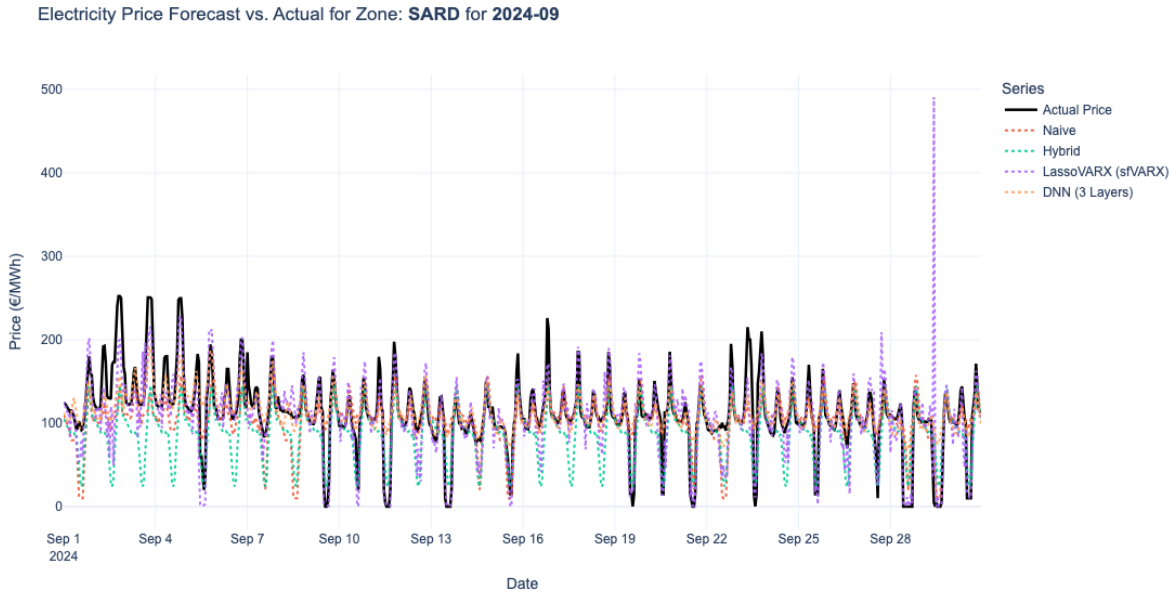

Figure A.18: Monthly Electricity Price Forecasts Comparison for Zone: SARD for 2024-09

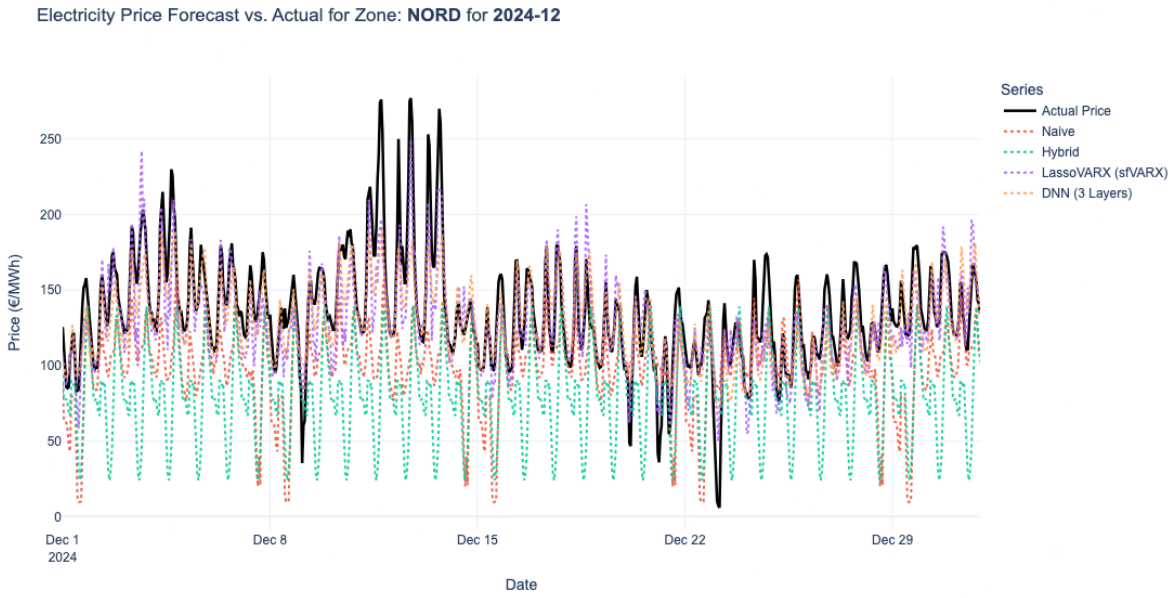

Figure A.19: Monthly Electricity Price Forecasts Comparison for Zone: NORD for 2024-12

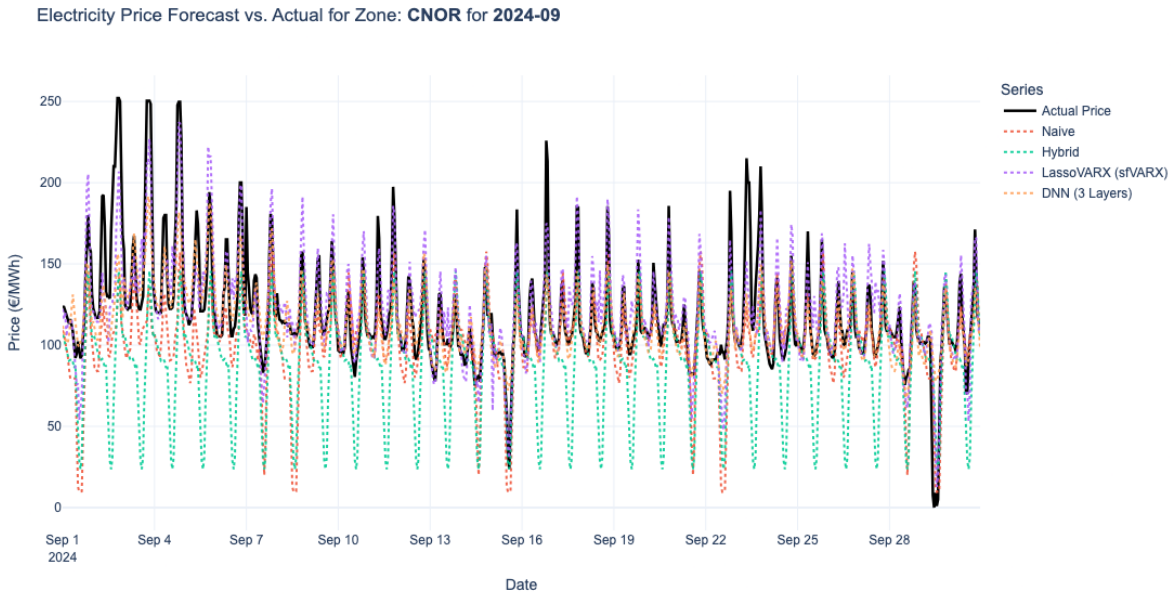

Figure A.20: Monthly Electricity Price Forecasts Comparison for Zone: CNOR for 2024-09

Bibliography

[1] S. K. Aggarwal, L. M. Saini, and A. Kumar. Electricity price forecasting in deregulated markets: A review and evaluation. *International Journal of Electrical Power & Energy Systems*, 31(1):13–22, 2009.

[2] Z. Akyurek and T. Tanyeri. Impact of renewable energy sources on electricity price and volatility: A comprehensive review. *Energy Reports*, 6:592–602, 2020.

[3] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, pages 115–123. PMLR, 2013.

[4] S. Bigerna, M. C. D'Errico, and P. Polinori. Wholesale electricity prices in italy: an empirical analysis. *Journal of Policy Modeling*, 38(4):743–758, 2016.

[5] B. Bosco, L. Parisio, and M. Pelagatti. Deregulated wholesale electricity prices in italy: an empirical analysis. *International Advances in Economic Research*, 13:415–432, 2007.

[6] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[7] N. Clabastini, S. Giansante, and S. Ruggiero. The italian electricity market: a survey of the literature. *Annals of Public and Cooperative Economics*, 93(3):685–714, 2022.

[8] S. Demir, K. Mincev, K. Kok, and N. G. Paterakis. Introducing technical indicators to electricity price forecasting: A feature engineering study for linear, ensemble, and deep machine learning models. *Applied Sciences*, 10(1):255, 2019.

[9] F. X. Diebold and R. S. Mariano. Comparing predictive accuracy. *Journal of Business & economic statistics*, 13(3):253–263, 1995.

[10] Gestore dei Mercati Energetici (GME). Zonal prices on the day-ahead market (mgp). `https://gme.mercatoelettrico.org/it-it/Home/Esiti/Elettricita/MGP/Esiti/PrezziZonali#IntestazioneGrafico`, 2024. Accessed: 2024-05-16.

[11] A. Gianfreda and L. Grossi. Forecasting italian electricity zonal prices with exogenous variables. *Energy Economics*, 34(6):2228–2239, 2012.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[14] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[15] E. G. Kardakos, T. L. Alexopoulos, and A. C. Vlachos. A review of short-term electricity price forecasting models and their applications. *Renewable and Sustainable Energy Reviews*, 62:1122–1143, 2016.

[16] D. Keles, J. Scelle, F. Paraschiv, and W. Fichtner. A review on the applications of computational intelligence in electricity price forecasting. *Artificial Intelligence Review*, 46:313–337, 2016.

[17] M. Khairalla, J. Catalao, and D. Pires. A comparative study of machine learning and deep learning models for electricity price forecasting. *IEEE Access*, 9:133221–133233, 2021.

[18] G. Koechlin, F. Bovera, and P. Secchi. Forecasting electricity spot market merit-order curves with functional time series modeling. In *2025 21st International Conference on the European Energy Market (EEM)*. IEEE, 2025.

[19] J. Lago. EPFToolbox: A toolbox for electricity price forecasting. GitHub repository, 2018. URL `https://github.com/jeslago/epftoolbox`.

[20] J. Lago, G. Marcjasz, B. De Schutter, and R. Weron. Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark. *Applied Energy*, 293:116983, 2021.

[21] P. Nowotarski and R. Weron. Recent advances in electricity price forecasting. *Applied Energy*, 288:116637, 2021.

[22] G. Oggioni and Y. Smeers. The italian electricity market: an overview. *In "Regulation of the Power Sector", edited by R. Cunha, I.J. Pérez-Arriaga, Springer-Verlag, London*, 2012.

[23] C. Santi. Forecasting electricity price and demand with a hybrid of sarima model

and lstm network in different time horizons. Technical report, Politecnico di Milano, 2024. Tesi di Laurea Magistrale in Mathematical Engineering.

[24] S. Sapio. The italian electricity market: a survey. *Available at SSRN 1133374*, 2008.

[25] B. K. Sovacool. *The global energy challenge: Environment, development and security*. Routledge, 2016.

[26] B. Uniejewski and R. Weron. Efficient forecasting of electricity spot prices with expert and lasso models. *Energies*, 11(8):2039, 2018.

[27] B. Uniejewski, J. Nowotarski, and R. Weron. Automated variable selection and shrinkage for day-ahead electricity price forecasting. *Energies*, 9(8):621, 2016.

[28] R. Weron. Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting*, 30(4):1030–1081, 2014.

[29] R. Weron. Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting*, 38:1169–1198, 2022.

[30] Z. Zhang and A. Willing. A review of deep learning based electricity price forecasting. *iScience*, 24(12):103344, 2021.

[31] Y. Zhou, Y. Zhang, and Y. Chen. A review on electricity price forecasting. *Energy Economics*, 94:105072, 2021.

[32] F. Ziel. Forecasting electricity prices with statistical and machine learning methods. *Energy Economics*, 115:106412, 2023.

Acknowledgements

Professor Filippo Bovera, Guillaume, and Andrea, I am grateful for the opportunity to work with you and for your guidance and availability throughout this experience.

Most importantly, to my mom and dad, my grandparents, my sister, my girlfriend and my friends: I want to thank you for standing beside me through every step of this journey. Your presence and understanding have been an important source of motivation, and I am truly grateful for all you have done for me.