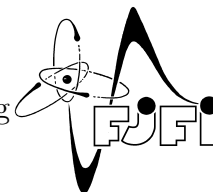




CZECH TECHNICAL UNIVERSITY IN PRAGUE  
Faculty of Nuclear Sciences and Physical Engineering



# Application of transformers for system imbalance prediction in electric power transmission system

## Aplikace transformérů pro predikci systémové odchylky v elektrické přenosové soustavě

Master's Thesis

Author: **Bc. Vojtěch Obhlídal**  
Supervisor: **Ing. Jiří Franc, Ph.D.**  
Academic year: 2023/2024



## ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Vojtěch Obhlídal
Studijní program:	Aplikované matematicko-stochastické metody
Název práce (česky):	Aplikace transformérů pro predikci systémové odchylky v elektrické přenosové soustavě
Název práce (anglicky):	Application of transformers for system imbalance prediction in electric power transmission system
Jazyk práce:	angličtina

### Pokyny pro vypracování:

1. Seznamte se s fungováním elektrické přenosové soustavy a zejména pak problematikou regulační energie a systémové odchylky.
2. Nastudujte možnosti predikce časových řad pomocí metod hlubokého učení a transformérů.
3. Implementujte vhodný model pro predikci systémové odchylky pomocí transformérů.
4. S použitím veřejně dostupných zdrojů připravte vhodný dataset obsahující proměnné s významným vlivem na systémovou odchylku.
5. Aplikujte model na reálná data z elektrické přenosové soustavy a porovnejte dosažené výsledky s používanými modely strojového učení.
6. Navrhněte možnosti zlepšení predikce časové řady pomocí ensemble metod.

Doporučená literatura:

1. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning: Adaptive Computation and Machine Learning series. The MIT Press, 2016.
2. B. Lim, S.O. Arik, N. Loeff, T. Pfister: Temporal fusion transformers for interpretable multi-horizon time series forecasting. arXiv preprint, 2019. arXiv:1912.09363
3. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin: Attention is all you need. In Advances in Neural Information Processing Systems, 2017, 6000–6010.
4. A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, 2019.
5. M. López Santos, X. García-Santiago, F. Echevarría Camarero, G. Blázquez Gil, P. Carrasco Ortega: Application of Temporal Fusion Transformer for Day-Ahead PV Power Forecasting. Energies 15, 2022.
6. J. Browell, C. Gilbert: Predicting Electricity Imbalance Prices and Volumes: Capabilities and Opportunities. Energies 15, 2022.

Jméno a pracoviště vedoucí diplomové práce:

Ing. Jiří Franc, Ph.D.

Katedra matematiky FJFI ČVUT v Praze

Trojanova 13

120 00 Praha 2

Jméno a pracoviště konzultanta:

Datum zadání diplomové práce: 31.10.2022

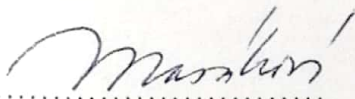
Datum odevzdání diplomové práce: 3.5.2023

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 31.10.2022

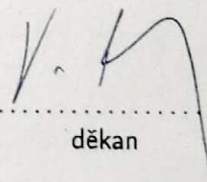


garant oboru



vedoucí katedry





děkan

*Acknowledgment:*

I would like to thank my supervisor, Ing. Jiří Franc, Ph.D., for his expert guidance and patient mentorship. His thoughtful insights and profound understanding have been instrumental in shaping this thesis.

*Author's declaration:*

I declare that this Master's Thesis is entirely my own work and I have listed all the used sources in the bibliography.

Prague, January 8, 2024

Vojtěch Obhlídal





*Název práce:*

**Aplikace transformerů pro predikci systémové odchylky v elektrické přenosové soustavě**

*Autor:* Bc. Vojtěch Obhlídal

*Studijní program:* Aplikované matematicko-stochastické metody

*Druh práce:* Diplomová práce

*Vedoucí práce:* Ing. Jiří Franc, Ph.D.  
Katedra matematiky, Fakulta jaderná a fyzikálně inženýrská,  
České vysoké učení technické v Praze

*Abstrakt:* Tato práce se zabývá použitím modelů založených na transformerech pro predikci systémové odchylky v elektrické síti. Na začátku je stanoven kontext trhu s elektřinou a přenosové soustavy. Jsou zdůrazněny výzvy, které přináší integrace obnovitelných zdrojů energie, a z toho vyplývající potřeba přesné předpovědi systémové odchylky. Poté je zkoumána architektura a schopnosti modelů transformerů s důrazem na použití attention mechanismu. Dále je podrobně popsáno předzpracování a aplikace dat z belgického trhu s elektřinou pro trénování a testování modelů. Následně je představena implementace modelů transformerů s důrazem na konkrétní úpravy architektury pro predikci časových řad. Nakonec je provedena komparativní analýza s metodami strojového učení, jako je vícevrstvý perceptron a XGBoost.

*Klíčová slova:* energetický trh, predikce na více časových horizontů, systémová odchylka, Transformer

*Title:*

**Application of transformers for system imbalance prediction in electric power transmission system**

*Author:* Bc. Vojtěch Obhlídal

*Abstract:* This thesis investigates the application of transformer-based models for predicting system imbalance in the electrical grid. Initially, the study establishes the context of the electricity market and transmission system, highlighting the challenges posed by the integration of variable energy sources and the consequent need for accurate forecasting of system imbalance. It then explores the architecture and capabilities of transformer models, highlighting the use of the attention mechanism. The research meticulously details the preprocessing and use of Belgian electricity market data for model training and evaluation. Furthermore, the implementation of transformer-based models is examined, with an emphasis on specific architectural modifications suitable for time series forecasting. Finally, a comparative analysis is conducted with other machine learning forecasting methods, such as multilayer perceptron and XGBoost.

*Key words:* electricity market, multi-horizon forecasting, system imbalance, Transformer





# Contents

<b>Introduction</b>	<b>11</b>
<b>1 Motivation</b>	<b>13</b>
1.1 Electricity . . . . .	13
1.2 System Imbalance . . . . .	15
<b>2 Time Series Forecasting</b>	<b>17</b>
2.1 XGBoost . . . . .	19
2.2 MLP . . . . .	20
2.3 Transformer . . . . .	22
2.4 Temporal Fusion Transformer . . . . .	31
<b>3 Data</b>	<b>33</b>
3.1 Temporal causality . . . . .	33
3.2 Data preprocessing . . . . .	35
3.3 Data processing . . . . .	42
<b>4 Transformer for Time Series</b>	<b>45</b>
4.1 Input Data . . . . .	45
4.2 Vanilla Transformer . . . . .	46
4.3 Encoder model . . . . .	48
4.4 Transformer with future lags . . . . .	49
<b>5 Results</b>	<b>51</b>
5.1 Conformal prediction intervals . . . . .	53
5.2 Model evaluation and comparison . . . . .	54
5.3 Ensemble . . . . .	61
5.4 Prediction intervals . . . . .	62
<b>Conclusion</b>	<b>64</b>



# Introduction

Electricity is essential to the operation of modern civilization, serving as a key component that energizes daily life and fuels technological progress. In the quest for sustainability and reduced carbon emissions, the integration of renewable energy sources like wind and solar is crucial. However, these green energy solutions, dependent on variable weather conditions, introduce an element of uncertainty and volatility into the power grid. In order to maintain the stability of the grid, its operator is forced to implement strategies to compensate for these imbalances. This necessity paves the way for developing advanced models to accurately predict and understand the evolving variations over time, thereby enhancing grid stability. This thesis delves into various modeling techniques, exploring their potential in managing the dynamic nature of today's energy landscape.

In this thesis, we focus on predicting system imbalances using transformer-based models. Transformers, renowned for their effectiveness in processing sequential data, are particularly suitable for this application. Their ability to capture long-term dependencies and patterns in data sets them apart from traditional models. Unlike conventional approaches, Transformers can manage large volumes of data and yield more accurate forecasts by considering the complete history of data inputs. Despite these advantages, the application of Transformers in time series prediction remains under-explored. Commonly, statistical modeling methods such as autoregressive moving average models are employed. Additionally, prevalent techniques include leveraging machine learning models like decision trees and their variants, as well as neural networks, for time series forecasting. In this thesis, we provide insights into system imbalance prediction using transformer-based models, detailing their necessary modifications for this specific task and offering a comparative analysis with popularly used models.

The first chapter establishes the motivation of the thesis, introducing the electricity market and defining key terms essential for understanding its mechanisms. Subsequently, the system imbalance, a variable indicating the difference between electricity production and consumption, which is the focus of the study, is defined.

The second chapter provides an introduction to time series forecasting. Next, benchmark models from the machine learning world, such as the Multilayer Perceptron [49] and XGBoost [13], are briefly introduced. The Transformer [57], a model prevalent in natural language processing, is described. Emphasis is placed on a detailed description of the Transformer, its internal structure, and its components. Special focus is given to the attention mechanism, a distinctive feature of Transformers. At the end of the chapter, the Temporal Fusion Transformer [39], one of the most advanced time series models built upon the Transformer architecture, is mentioned.

The third chapter examines the Belgian data utilized for training, testing, and comparing the proposed models. The chapter begins by addressing temporal causality, which describes the problem associated with delayed data availability. Subsequently, all the transformations performed on the data are described, from variable selection and feature engineering to data normalization,

division into training, validation, and test sets, and the actual construction of the input data. Three implemented transformer-based models, specifically tailored for time series analysis, are detailed in the fourth chapter. The necessary modifications made to adapt these models to time series forecasting are described. Furthermore, the chapter provides a comprehensive discussion on each model, elucidating their respective advantages and disadvantages. The final chapter presents the results achieved in this thesis. It includes a detailed comparison of the implemented transformer-based models against benchmark models in the context of predicting system imbalance. This comparative analysis highlights the effectiveness and performance variations among the different models. Furthermore, conformal prediction intervals [41] are incorporated with the point predictions to indicate the quality of the forecast. Additionally, an ensemble comprising the implemented models is constructed to enhance the overall prediction accuracy.

# Chapter 1

## Motivation

Safe and reliable access to the electricity grid is one of the essential features of modern society. It is a crucial element in promoting sustainable development and forms the foundation of social and economic well-being. Improving distribution networks and coverage is key to global progress and to improving living standards in less developed regions. A shortage in electricity supply could result in disruptions in critical sectors such as health and industry.

### 1.1 Electricity

Along with natural gas, coal, or oil, electricity is classified as an energy commodity and is traded on electricity markets. However, its specific characteristics make it very different from other physical commodities.

Electricity must be produced and consumed simultaneously, and the power grid must be balanced in real time. This requirement implies that supply must exactly meet demand. The challenge is to ensure that this balance is maintained. Other energy commodities are relatively easy to store, and any shortages or surpluses can be dealt with by adding or storing reserves. In the case of electricity, a complication arises because of the complexity of electricity storage, and there are only a limited number of electrical energy storage (EES) options that humanity has invented so far. Currently available EES technologies do not meet all the requirements such as long life, affordability, high efficiency, and environmental friendliness. However, promising new technologies are being further investigated [12].

Electricity generation is the process of producing electricity from different primary energy sources. In practice, it consists of thermal and renewable power plants. Thermal power plants are a stable source of electricity and account for the majority of electricity generation worldwide. These are mainly fossil fuel and nuclear power plants.

Renewable power plants typically represent a minority of the energy mix and are a less stable source of energy. Yet, they produce minimal carbon emissions and are a sustainable source of energy. Renewable energy sources (RES) fall into two categories: variable, such as wind and solar power, which are not easily dispatchable, and controllable, such as hydroenergy or biomass, which allow their production to be more easily regulated. In some regions, the former account for tens of percent of total electricity production [25]. Wind and photovoltaic energy production is sensitive to weather conditions, and a sudden change can drastically affect production. In extreme cases, it can lead to complete shutdown of the power generation.

In contrast, controllable RES produce electricity on demand, as do thermal power plants. In the electricity grid, therefore, generation can be alternated to balance the grid and compensate for the intermittency of variable RES.

To reduce dependence on fossil fuels and the production of carbon emissions, the use of low-carbon electricity is increasing. After the Fukushima nuclear disaster in 2011, the debate has intensified, and nuclear power generation has been the subject of global controversy due to concerns about safety and waste disposal [8]. In Europe, there has been a noticeable decline in nuclear energy usage over the past two decades. This trend is characterized by the decommissioning of several nuclear reactors and the phased-out operation of entire nuclear power plants, largely influenced by a strategic shift toward other energy sources and significant policy decisions aimed at reducing the dependence on nuclear energy [8, 59]. Wind and photovoltaic energy utilization is simultaneously increasing and is expected to increase further in the coming years [30, 53].

The electrical grid is a complex system designed to deliver electricity from its generation source to end-users for their hourly requirements. Evolving from modest local setups, these systems now span thousands of kilometers, interlinking millions of homes and commercial establishments today. It comprises three essential components: electricity generation, transmission, and distribution, each integral and indispensable within the system.

After electricity is generated in power plants, it is delivered to the grid. The transmission system then employs high-voltage power lines to bridge the distance between power plants and local distribution networks. Last, but not least, electricity distribution is the final part of the electricity network that ensures the conversion to the appropriate voltage level and further distribution to consumers.

As described in Section 1.1, the increasing trend towards integrating variable RES, such as wind and photovoltaic power, leads to greater unpredictability in general electricity production and disturbances in the balance of the grid between consumption and production. In addition, the grid is also affected by various other factors. Electricity demand exhibits seasonal variations on daily, weekly, and monthly scales. Furthermore, changes in delivery requirements from large consumers can significantly affect the grid, as can various repairs, outages, and failures of both power plants and the transmission system.

These challenges are primarily addressed by the Transmission System Operator (TSO), which is responsible for operating the transmission system and keeping balance between electricity supply and demand. The utility frequency of the power grid is a key indicator of this balance. Deviations from the standard frequency (50 or 60 Hz) signal an imbalance between electricity generation and consumption. Close cooperation between the TSO and the Distribution System Operator (DSO) is also necessary to enable the energy transition. In Europe, TSOs form the European Network of Transmission System Operators for Electricity (ENTSO-E), an association that now consists of 39 members from 35 countries [26]. The main objectives of ENTSO-E are currently to integrate renewable energy sources into the energy mix of member countries and to support the creation of a single European energy market.

Electricity trading, like other tradable commodities, is fundamentally driven by the dynamics of supply and demand. In addition to producers and buyers, there are several other players in the market. To ensure balance on the grid, the TSO delegates responsibility for maintaining balance to private entities called Balance Responsible Parties (BRPs). Each market participant, whether a producer or an offtaker, must be under the management of a BRP or be its own BRP. The primary function of a BRP is to balance the gap between electricity production and consumption. Additionally, the Market Operator plays a pivotal role in setting prices and

facilitating transactions. It is responsible for organizing day-ahead and intraday markets, as well as for aligning supply and demand through the matching of bids.

There are several types of contracts that can be traded on European markets. The vast majority of electricity trading takes place in advance of physical delivery on the forward and futures markets [20]. These types of contracts can be concluded for different volumes and durations and can be traded days, months, or even years in advance. Their main advantage is that they reduce risk and give both producers and buyers certainty that they will be able to sell or receive the electricity. Additionally, both parties can hedge against price fluctuations, allowing them to negotiate a fixed price that is mutually beneficial.

Forward production and consumption forecasts are not very accurate, so market participants have to adjust their position. This takes place on the spot markets, where contracts are traded less than a day in advance on the so-called day-ahead and intraday markets.

As implied by its name, the day-ahead market involves trading electricity one day prior to its delivery. Participants submit bids and offers, each specifying a desired quantity at a particular price. Bids and offers are matched afterward, and the final price is established by the market operator. In European countries, trading usually terminates at 12:00 local time the day before delivery. If a participant is interested in further adjusting its market position, the intraday market comes into play for the final settlement of trades. The market opens at 15:00 local time on the day preceding delivery, providing participants an opportunity to settle any imbalances, offering the possibility of trading up to 5 minutes before the delivery period.

For more information on how the markets work, see [2].

## 1.2 System Imbalance

Deviations from absolute balance in the grid are omnipresent. To resolve any imbalance in the grid, the TSO is equipped with several ancillary services used to maintain stability. Apart from monitoring frequency in the system, power plants are also scheduled by the TSO to start up at times when electricity demand is expected to be high, referred to as peaks. Furthermore, the TSO has operating reserves in case of an outage or imbalance in electricity demand and supply.

This reserve energy ensures sufficient power is available. We distinguish between primary, secondary, and tertiary reserves, which differ in the speed of their activation.

The primary reserve, known as Frequency Containment Reserve (FCR), is used for immediate network stabilization. It comprises a combination of resources, including battery generators, pumped-storage hydroelectricity, and often gas-fired power plants, and is activated automatically within 30 seconds. Its purpose is to keep the network running until additional reserve resources are available.

The secondary reserve, called the Automatic Frequency Restoration Reserve (aFRR), is also triggered automatically. It must be activated within 5 or 7.5 minutes, depending on the country, and is provided by resources that can be activated quickly and sustained for a longer period. These include not only conventional power plants but also battery systems in combination with biogas stations and hydroelectric power. Additionally, international cooperation, facilitated between some European countries on the PICASSO platform (Platform for the International Coordination of Automated Frequency Restoration and Stable System Operation), makes it possible to deliver aFRR across borders.

If aFRR does not compensate for the deviation in the network, a tertiary reserve called manual Frequency Restoration Reserve (mFRR) can be activated. It must be triggered within 12.5 or 15 minutes, depending on the country. Inter-European sharing is again possible thanks to the MARI (Manually Activated Reserve Initiative) platform.

Additionally, in some regions, a contingency reserve, also called Replacement Reserves (RR), is subsequently activated. It takes longer to activate and is used to restore the operating reserves to readiness. Although this mechanism is not used in all countries, it is possible to trade RR across borders using TERRE (Trans European Replacement Reserves Exchange).

In addition to these ancillary services, which activate reserve resources as a counteraction to a growing imbalance in the grid, there is a European platform IGCC (International Grid Control Cooperation). Unlike the aforementioned platforms designed for trading reserves, IGCC is designed for cross-border balancing so that the energy from reserve resources does not need to be activated at all. All these platforms are designed and managed by ENTSO-E.

Moreover, in the case of all reserves, a distinction can be made as to whether the regulation is positive or negative. Positive regulation is used to offset shortages of electricity, while negative regulation is utilized in instances of surplus electricity within the network. This mechanism is essential to prevent power outages and ensure a consistent and stable energy supply.

The process of balancing the grid is known as the balancing mechanism. As mentioned in Section 1.1, market participants first trade contracts in the futures and forward markets, where they can secure long-term supply and demand of electricity in advance. Prior to actual physical delivery, they can adjust their position as their supply and demand forecasts become more precise.

However, at the time of physical delivery, the actual value of production and offtake may differ from the contracted value. The system may then remain out of balance due to over-consumption or over-production, for which the BRPs are responsible. The sum of the imbalances of all BRPs is called the **system imbalance**. Correspondingly, the total amount of balancing energy, i.e., the difference between the sum of positive and negative reserve energy that the TSO has to supply to the system, is called net regulation volume (NRV).

This imbalance needs to be regulated by the TSO to maintain a stable frequency in the power system through the ancillary services. Balancing energy, whether domestic or obtained through cross-border capacities, is paid for by the TSO. The price depends on various factors, but particularly on the origin of the reserve energy that had to be activated. Once the price is set by the TSO, BRPs in deviation must settle their position. They are thus forced to buy or sell the imbalance amount at a fixed price determined by the TSO. If a BRP can accurately forecast market trends, this practice of settlement can be advantageous and profitable. However, if their predictions are incorrect, it can lead to losses, such as the need to purchase electricity at a price higher than the market rate.



## Chapter 2

# Time Series Forecasting

This thesis is devoted to time series analysis and forecasting. Using historical data points of a time series, the goal is to predict its future values. Time series forecasting, positioned at the intersection of statistical analysis and predictive modeling, offers a range of methods for application. The main goal, however, is to leverage historical data, indexed in temporal order, to make predictions about future trends. Each approach employs distinct methodologies and assumptions on time series, as will be discussed in this chapter.

Before proceeding, it is useful to consider the notation used in time series analysis. Consider time series data  $(x_t, y_t) \in \mathbb{R}^k \times \mathbb{R}$ ,  $t = 1, \dots, n$ , where  $x_t$  is a vector of explanatory (or exogenous) features and  $y_t$  is the target value. Additionally,  $k$  is the number of explanatory variables, which can be represented by both stochastic and deterministic time series, as well as historical values of  $y_t$ , and  $n$  is the total number of observations in the data.

A forecasting method is a procedure that predicts future values of the target with the knowledge of historical values of both the target and other exogenous variables. It typically consists of designing a model  $\hat{\mu}$ , which is fitted on training data, that predicts future target values.

Furthermore, we can denote  $\mathcal{A}$  as a mapping algorithm that uses training data to fit the model. It formally maps all possible training sets collections to a space of regression functions  $\mathcal{R}$ , i.e.

$$\mathcal{A} : \bigcup_{m=1}^{n_0} (\mathbb{R}^k \times \mathbb{R})^m \mapsto \mathcal{R},$$

where  $n_0 \in \mathbb{N}$ ,  $n_0 < n$  represents the splitting index of the training data, comprising indices  $\{1, \dots, n_0\}$ . Therefore, it can be denoted

$$\hat{\mu} = \mathcal{A}((x_1, y_1), \dots, (x_{n_0}, y_{n_0})),$$

The situation described is called multivariate time series forecasting since other variables are available in addition to the target.

Time series forecasting can be classified based on the forecasting horizon. While being at time  $\tau$ , predictions can focus on different time steps in the future. The straightforward variant is a one-step-ahead prediction, which is usually the most accurate and aims to predict the target value at time  $\tau + 1$ . By generalizing to  $p \in \mathbb{N}$  steps ahead, a forecast at time  $\tau + p$  can also be made. The last type is multi-horizon forecasting, which focuses on predicting the entire sequence of values for time steps  $\tau + 1, \dots, \tau + p$ . Since the objective is to predict an entire sequence of values, this task is the most challenging one.

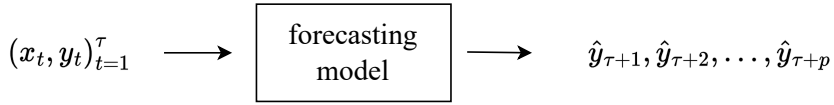


Figure 2.1: An example of multi-horizon forecasting. At given time  $\tau$ , past and present values of target  $y_t$  and explanatory variables  $x_t$  for all  $t = 1, \dots, \tau$  are used to forecast  $p$  target values  $\hat{y}_{\tau+1}, \hat{y}_{\tau+2}, \dots, \hat{y}_{\tau+p}$ .

There are several methods that can be used including statistical modeling, machine learning (ML) or deep learning (DL) models. Statistical models explicitly consider the temporal dependency and their pivotal role is identifying and exploring the internal structure of the data. These models are designed to capture trends, seasonal fluctuations and other dynamics within the time series. One of the drawbacks is the assumptions placed on the time series by specific models. Violations of these assumptions can lead to biased or inefficient estimates and forecasts. The most well-known statistical models are autoregressive integrated moving average (ARIMA), seasonal ARIMA (SARIMA), exponential smoothing (ES), or their variants using exogenous variables [33]. For more complex datasets, these models have been outperformed by ML and DL models [52, 61].

Although initially designed for supervised learning tasks, machine learning and deep learning models have been adapted for time series forecasting. This adaptation involves creating lags such that past observations are used as features to capture temporal dependencies. This method ensures that while the general dependency among observations is preserved through these lags, the individual observations can be treated as independent. After restructuring the time series as independent data, both ML and DL models can be applied effectively. The advantage of this approach is the ability to leverage the power of machine learning algorithms, especially in capturing complex nonlinear relationships and interactions in the data.

Several approaches can be employed in multi-horizon forecasting. One such method is multi-output regression, which involves modifying the predictive model to simultaneously forecast multiple future values. This approach directly targets the entire prediction horizon in a single model. Another strategy is the creation of an ensemble of models, where each model is trained to predict a specific time step within the entire forecast horizon. This method benefits from the diverse perspectives of multiple models. Alternatively, models can be applied in an iterative manner, where prediction for each step is fed into the next, gradually building the forecast sequence.

One specific type of DL model is Transformer [57], originally developed for natural language processing (NLP). Its ability to capture long-term dependencies and complex patterns is a good prerequisite for accurate time series prediction, especially when working with large data sets. Their application to time series is being investigated, and their application to time series data are one of the main aims of this thesis.

Before focusing on the Transformer model, we describe two machine learning models that have been used as benchmarks. We then focus on the Transformer, describing in detail the main concepts and components of the model. We conclude with a description of a state-of-the-art model called the Temporal Fusion Transformer [39], which is based on the original Transformer structure.

## 2.1 XGBoost

Gradient tree boosting (GTB) is a popular and powerful machine learning technique, applicable to a broad range of supervised learning problems due to its flexibility and scalability. Its advanced variant, eXtreme Gradient Boosting (XGBoost) [13], offers enhanced performance with faster execution owing to various optimizations and improvements. Similarly to decision trees, XGBoost iteratively adds trees that optimally reduce the objective function by leveraging the gradient and Hessian of the loss function.

### 2.1.1 Gradient boosting

The XGBoost model optimizes an objective function that consists of a training loss and a regularization term to control the complexity of the model. The CART method [7] is used to construct the decision tree ensemble. Each of the trees predicts a score and the total score of the ensemble is the sum of the scores of all the trees in the form

$$\hat{y}_i = \sum_{j=1}^K f_j(x_i), \quad f_j \in \mathcal{F}, \quad (2.1)$$

where  $K$  is the number of trees in the ensemble,  $f_j$  is a function representing a single tree with  $\mathcal{F}$  being the set of all possible trees, and  $x_i \in \mathbb{R}^d$  is an input vector.

The objective function is given in general form by

$$\text{obj} = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{j=1}^K \omega(f_j), \quad (2.2)$$

where  $l(y_i, \hat{y}_i)$  is a loss of  $i$ -th input vector and  $\omega(f_j)$  is the  $j$ -th tree complexity. Model training is iterative and employs an additive strategy, in which an already trained tree is locked, and another one is trained afterwards. This allows the new tree to be learned to correct the mistakes of the previous ones. For the scenario, where  $k$ -th tree is being optimized, predictive function (2.1) can be rewritten as

$$\hat{y}_i^{(k)} = \sum_{j=1}^k f_j(x_i) = \hat{y}_i^{(k-1)} + f_k(x_i), \quad (2.3)$$

where  $\hat{y}_i^{(t)}$  is the prediction of tree ensemble with indices  $\{1, \dots, k\}$ .

By substituting (2.3) into (2.2) modified for  $k$ -th tree, the objective function can be written as

$$\text{obj}^{(k)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(k-1)} + f_k(x_i)) + \omega(f_k) + C,$$

where  $C \in \mathbb{R}$  is a constant. We then develop the loss function into a second-order Taylor expansion and after arithmetic adjustments, the objective function simplifies to

$$\text{obj}^{(k)} = \sum_{i=1}^n [l(y_i, \hat{y}_i^{(k-1)}) + g_i f_k(x_i) + \frac{1}{2} h_i f_k^2(x_i)] + \omega(f_k) + C,$$

while the terms  $g_i$  and  $h_i$  denote the respective derivatives

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(k-1)}} l(y_i, \hat{y}_i^{(k-1)}), \\ h_i &= \partial_{\hat{y}_i^{(k-1)}}^2 l(y_i, \hat{y}_i^{(k-1)}). \end{aligned}$$

By removing the part of the objective function that does not depend on  $f_k$ , we get the final form of the objective function

$$\text{obj}^{(k)} = \sum_{i=1}^n [g_i f_k(x_i) + \frac{1}{2} h_i f_k^2(x_i)] + \omega(f_k).$$

The regularization term in XGBoost is formulated to include both L1 and L2 regularization, as well as penalization for tree complexity, expressed as follows

$$\omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 + \alpha \sum_{j=1}^T |w_j|.$$

In this equation,  $f$  is the tree,  $T$  is the number of leaves in the tree  $f$ ,  $w_j$  is the  $j$ -th leaf weight in  $f$ ,  $\gamma$  is the parameter that penalizes the number of leaves and  $\lambda$  and  $\alpha$  are the L2 and L1 regularization parameters on leaf weights.

Compared to the GTB algorithm, XGBoost has several improvements that make it such a popular method. Key features include advanced split-finding algorithms that efficiently find the best split in the feature space, or ability to handle sparse data. Important system enhancements include parallelization strategies that, together with cache-aware and out-of-core computation, give XGBoost improved computational performance.

For a comprehensive explanation of the utilization of the score in tree pruning within XGBoost, as well other details, please refer to [13].

## 2.2 MLP

A Multilayer Perceptron (MLP) is a variant of the feedforward artificial neural network (ANN), initially conceptualized by Frank Rosenblatt in 1958 [49]. An MLP is composed of an input and an output layer, as well as at least one hidden layer. Each layer consists of neurons that are fully connected to every neuron in both the subsequent and preceding layers (see Figure 2.2). Furthermore, each neuron-to-neuron connection is associated with a weight and every neuron is associated with a bias.

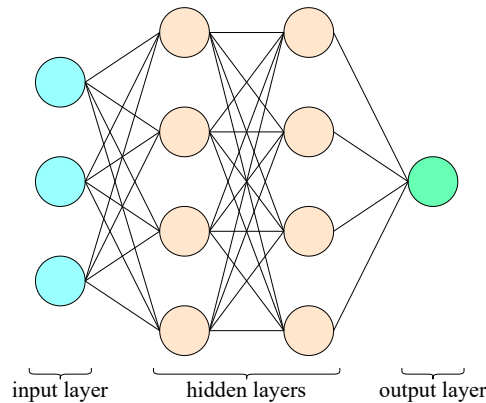


Figure 2.2: Architecture of a Multilayer Perceptron neural network with two hidden layers.

Associated weights indicate the importance of the specific connection between neurons, while bias is introduced to increase flexibility. Each neuron computes a weighted sum of its inputs,

achieved by multiplying each input by its respective weight and then summing these products, with the bias added to the sum. An activation function is then applied to this weighted sum, introducing nonlinearity to the neuron’s output. The output is subsequently passed to each neuron in the next layer. Each layer in a MLP, encompassing both the input and output layers, is characterized by its number of neurons, commonly referred to as the layer size.

The output of a neuron can be written as

$$o = f \left( \sum_j h_j w_j + b \right), \quad (2.4)$$

where  $f$  is an activation function,  $h_j$  is the  $j$ -th value from preceding layer with corresponding weights  $w_j$ , and  $b$  represents the neuron’s bias.

Let us now consider a MLP with  $M$  hidden layers with corresponding sizes  $l_1, \dots, l_M$  and let  $x \in \mathbb{R}^d$  be the input vector. Furthermore,  $w_{j,k}^{(s)}$  stands for the weight corresponding to the connection between the  $j$ -th neuron in the  $(s - 1)$ -th layer and the  $k$ -th neuron in the  $s$ -th layer and  $b_k^{(s)}$  stands for the bias of the  $k$ -th neuron in the  $s$ -th layer. Additionally,  $z^{(s)} \in \mathbb{R}^{l_s}$  is the output of  $s$ -th hidden layer and for simplicity, let us set  $z^{(0)} := x$ .

Adjusting the equation (2.4) for  $i$ -th neuron in  $s$ -th hidden layer, we get

$$z_i^{(s)} = f \left( \sum_j^{l_s} z_j^{(s-1)} w_{j,i}^{(s)} + b_i^{(s)} \right).$$

The input layer size depends on the input data and the number of neurons corresponds to the number of features. The output layer is shaped according to the task it is designed for. Usually, one neuron for regression and multiple neurons in case of classification.

Additionally, the activation function is a crucial part of MLP, as it introduces nonlinearity into the model. Without it, the whole MLP would be reduced to a composition of linear functions, hence also a linear function. In hidden layers of a multilayer perceptron, a variety of activation functions can be employed, commonly including sigmoid, tanh, or ReLU (see Figure 2.3). The type of activation function used in the output layer depends on task. In the regression problem considered in this thesis, the activation function in the output layer is identity activation.

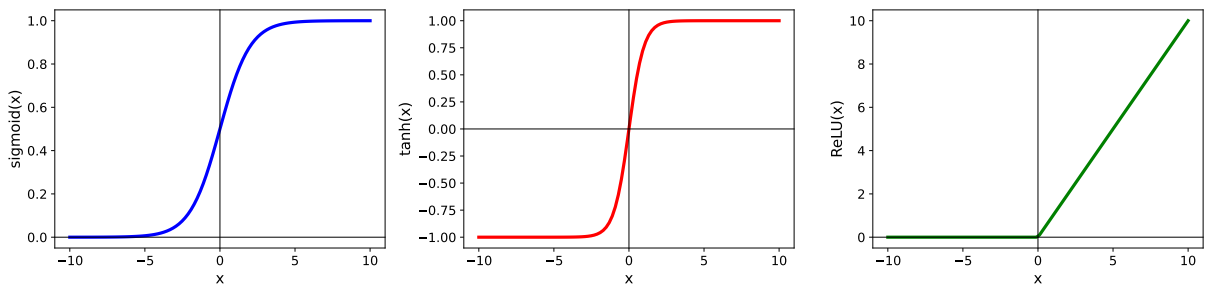


Figure 2.3: Examples of typically used activation functions.

The weights and biases in a MLP are updated using the gradient descent algorithm in the process called backpropagation [50], which is an algorithm used to compute the gradient of the loss function with respect to the network weights. It involves a forward pass to calculate the loss,

followed by a backward pass to compute gradients using the chain rule. The weights  $w_{j,k}^{(s)}$  and biases  $b_k^{(s)}$  are then updated in the direction that minimizes the loss function  $L$ . The updates can be expressed as

$$w_{j,k}^{(s)} = w_{j,k}^{(s)} - \alpha \frac{\partial L}{\partial w_{j,k}^{(s)}},$$

$$b_j^{(s)} = b_j^{(s)} - \alpha \frac{\partial L}{\partial b_j^{(s)}},$$

where  $\alpha$  is the learning rate,  $\frac{\partial L}{\partial w_{j,k}^{(s)}}$  and  $\frac{\partial L}{\partial b_j^{(s)}}$  are the partial derivatives of the loss function with respect to weights and bias. The mean squared error is typically used as the loss function. For detailed information on the multilayer perceptron and an in-depth explanation of backpropagation, reader is referred to [45].

## 2.3 Transformer

In solving problems of sequence transduction—also referred to as sequence-to-sequence modeling—in fields such as natural language processing (NLP), speech recognition, or time series forecasting, one frequently used method involves models based on recurrent or convolutional neural networks [54, 56, 63]. These complex models typically rely on the encoder-decoder architecture. The encoder processes the variable-length input and produces its abstract, fixed-length vector representation, while the decoder reverses this process to produce a variable-length output [14]. Improved results have been achieved using an attention mechanism that synergizes both components [4, 36]. However, these models face several complications.

Recurrent models process data sequentially, and at time  $\tau$ , the model’s output depends not only on the current input data but also on the hidden state from time  $\tau - 1$ . This interdependency makes parallelization impossible and causes the loss of long-term temporal dependencies in longer sequences due to the vanishing gradient problem [27]. The computational time increases with larger sequence lengths, and training such a model can reach extremely long computational time. Since these models do not benefit from parallelization, using hardware accelerators such as GPUs or TPUs does not reduce training time. Although gradient-based limitations have already been partially mitigated and performance has been improved, the primary challenge of their parallelization persists [51].

However, in 2017, Vaswani et al. [57] proposed a groundbreaking model called the **Transformer**. It relies solely on the attention mechanism and forgoes the use of recurrent and convolutional layers. This novelty enables parallel processing, resulting in significant training enhancements in terms of both resource utilization and computation time [55]. Furthermore, the Transformer achieves exceptional results that surpass the performance of recurrent models in certain tasks.

### 2.3.1 Model Architecture

Since the release of [57], numerous of articles have been published introducing new and improved Transformer architectures. In this chapter, however, our focus will be solely on the original paper and the Transformer architecture presented therein while introducing the model and illuminating its architecture. Notably, the Transformer model was originally developed to address the challenges of machine translation and we will follow this purpose throughout this chapter.

Modifications required to apply the Transformer to time series will be discussed later in this thesis, although the internal structure of the Transformer remains intact.

Initially, we will provide an overview of the Transformer as a whole, introducing its components and explaining their roles within the model. Special emphasis will be placed on the attention mechanism and the operations performed by each part of the model.

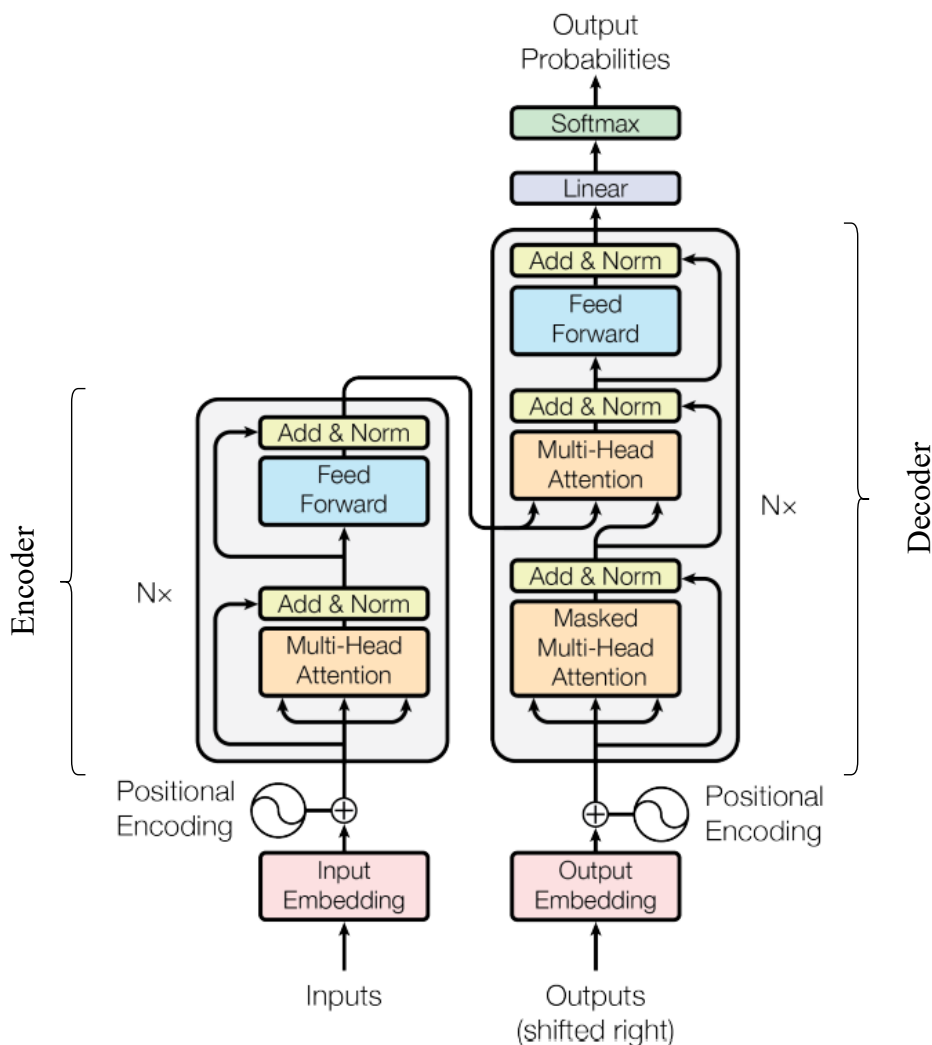


Figure 2.4: A Transformer architecture proposed in [57].

The original problem that Transformer was supposed to solve was machine translation, i.e. to receive a sequence of words in the source language as input and generate the corresponding translation in target language as output. As previously noted, the most proficient models consist of an encoder and a decoder, and this applies also to the Transformer architecture. Both encoder and decoder blocks are constructed using stacked identical layers, and the visual representation of this architecture is provided in 2.4.

### 2.3.1.1 Encoder block

The encoder block comprises multiple consecutive layers referred to as encoder layers. Each encoder layer is further composed of two sublayers. The initial sublayer is the multi-head self-attention layer, which will be extensively discussed in the Section 2.3.3. The subsequent sublayer is a fully-connected feedforward layer. Both layers are connected back-to-back with residual connections. The outputs of both sublayers undergo layer normalization [3].

At the beginning of the encoding process, a source sequence serves as an input to the encoder. This input sequence is passed through the input embedding, alongside with the positional encoding (described later in the Section 2.3.4.2). The encoder is responsible for understanding the input sequence and extracting relevant information and dependencies between tokens.

### 2.3.2 Decoder block

The decoder block is similarly constructed using identical layers, although their internal structure slightly varies. The decoder layer consists of three sublayers. The first comprises a masked multi-head self-attention layer, designed to prevent future token visibility in the sequence. The final sublayer features a fully-connected feedforward layer, similar to the one in encoder layer. The key distinction in the decoder layer is found in the intermediate sublayer: the multi-head cross-attention layer, also known as the encoder-decoder attention layer. This sublayer incorporates both the internal hidden states incoming from the decoder input and the outputs originating from the encoder output. This sublayer therefore provides interconnection of both encoder and decoder blocks. All sublayers are properly normalized and linked with residual connections.

The decoder input is a target sequence which is again enriched by input embedding and positional encoding. Once source sequence is processed by the encoder block, it is passed to the decoder where it enters the second sub-layer mentioned earlier, the multi-head cross-attention. The target sequence alias decoder input enters the first decoder sublayer, where masked multi-head self-attention is applied to it, and continues to the cross-attention layer, where the encoder output is also used for computation. The output is then passed to a fully-connected feedforward layer and, finally, softmax is applied to the output to receive probabilities, which are further used to predict the next word.

While the encoder provides language understanding, the decoder exercises its generative function and produces new text. In addition to machine translation, it is also suitable for other text generation tasks.

### 2.3.3 Attention mechanism

One of the key methods used in the Transformer is the attention mechanism. An attention assigns different levels of importance to different parts of the input. It allows the model to specifically focus on mutually-relevant parts and it is useful to capture contextual relationships and improve the processing of long-range dependencies in sequential data.

The attention function operates with three vectors called **query**, **keys** and **values**. It maps a query and a set of keys and values to an output vector. Multiple functions can be used to calculate attention, however, the authors of the original paper suggested scaled dot-product attention to be used.



### 2.3.3.1 Scaled Dot-Product Attention

The query, keys, and values vectors are incorporated into the calculation. Although the authors suggest that query and keys dimension  $d_k \in \mathbb{N}$  and values dimension  $d_v \in \mathbb{N}$  may be different, in practical use  $d_v = d_k$  is usually considered. Therefore, while explaining the theoretical background of the Transformer, we will consider the same dimension for all query, keys and values vectors.

Scaled dot-product attention is applied as follows. First, dot product of query with all keys is computed. Second, softmax is applied to the results and weights are obtained. Finally, the dot product of the weights with values is computed to obtain output vector. In this way, the attention is calculated for one query. However, we can apply attention function to all queries at once by stacking query vectors into matrix form.

Before we define the attention function, let us introduce the softmax function. It is an operation commonly used in machine learning and statistical modeling to transform a vector of real numbers into a probability distribution. Given a vector  $u = (u_1, u_2, \dots, u_d)^T \in \mathbb{R}^d$ , the softmax function is defined as

$$\text{softmax}(u_i) = \frac{e^{u_i}}{\sum_{j=1}^d e^{u_j}}, \quad \text{for } u = 1, 2, \dots, d.$$

The softmax function has an important property, which is that

$$\lim_{u_i \rightarrow -\infty} \text{softmax}(u_i) = 0. \quad (2.5)$$

Considering an arrangement of  $n$  vectors into a matrix  $\mathbf{U} \in \mathbb{R}^{n,d}$ , where  $\mathbf{U} = (u_{i,j})_{i,j=1}^{n,d}$ , the subsequent row-wise application of softmax can be denoted as follow

$$\text{softmax}(\mathbf{U})_{i,j} = \frac{e^{u_{i,j}}}{\sum_{l=1}^d e^{u_{i,l}}}, \quad \text{for } i = 1, 2, \dots, n, \text{ and } j = 1, 2, \dots, d.$$

With the newly introduced softmax, it is possible to define scaled dot-product attention.

**Definition 2.3.1** (Scaled Dot-Product Attention). *Let  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n,d_k}$  be matrices of queries, keys and values respectively with rows  $q_i^T, k_i^T, v_i^T \in \mathbb{R}^{d_k}, \forall i \in \{1, \dots, n\}$ . A **scaled dot-product attention** is given by*

$$z_i = \text{attention}(q_i, k_1, \dots, k_n, v_1, \dots, v_n)_i = \sum_{j=1}^n \text{softmax}\left(\frac{q_i^T k_j}{\sqrt{d_k}}\right) v_j, \quad (2.6)$$

which can be expressed in matrix form

$$\mathbf{Z} = \text{attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}. \quad (2.7)$$

The weighting of the values vector  $v_j$  depends solely on keys and query vectors  $k_j$  and  $q_i$  respectively. It also corresponds to the first element of cross-correlation between vectors  $k_j$  and  $q_i$ , which is for two real vectors  $a, b \in \mathbb{R}^d$  defined as

$$\text{corr}(a, b)_j = \sum_{i=1}^{n-j-1} a_i b_{i+j-1}, \quad \text{for } j = 1, 2, \dots, d.$$

The softmax operator serves as normalization and ensures that the sum of all weights is summed to unity. In a sense, the attention function calculates the correlation between query and keys vectors, which is then used as a weight for vector values. Vectors of values  $v_j$ , for which keys vector  $k_j$  is highly correlated with given query  $q_i$ , are given higher relevance.

As the dimension  $d_k$  increases, the dot product tends to yield larger values. The subsequent softmax is not very sensitive for these high values, so the scaling constant  $\frac{1}{\sqrt{d_k}}$  is used in the attention formula (2.6) and (2.7), respectively. It was proposed by the authors of [57] and is intended to prevent this phenomenon.

### 2.3.3.2 Multi-head Self-attention

In both the encoder and decoder blocks, the initial sublayer is the self-attention layer. It receives only the sequence itself as input, enriched with embedding and positional encoding. As the name implies, this process involves computing attention solely based on the input sequence.

The self-attention mechanism exclusively utilizes the sequence of  $n$  vectors  $x_1, \dots, x_n$  as input, where  $x_i \in \mathbb{R}^d$ . These vectors must be processed to create vectors of queries, keys, and values. To do so, three different linear transformations are applied to the input vectors. For this purpose, we define weight matrices such that

$$q_i = W_q^\top x_i, \quad k_i = W_k^\top x_i \quad \text{and} \quad v_i = W_v^\top x_i, \quad \forall i \in \{1, \dots, n\},$$

where  $W_q, W_k, W_v \in \mathbb{R}^{d, d_k}$  and  $d \in \mathbb{N}$  is known as model dimension. The weight matrices of each transformation are learnable, and their specific values are learned during the training phase.

To exploit deeper relationships in the data, it is beneficial to perform several parallel self-attention functions. As multiple heads of attention are performed, the information learning process could be approached differently and, even though we work with the same input data, the weight matrices may be learned differently for each head. In this way, new contexts can be mined from the data.

For further use, we introduce the concatenation operator, which facilitates the merging of matrices along the second dimension. Let  $\mathbf{A}_i \in \mathbb{R}^{n, d_i}, \forall i = 1, \dots, h$  be arbitrary matrices. Then the concatenation operator

$$\text{concat} : \mathbb{R}^{m, d_1} \times \dots \times \mathbb{R}^{m, d_h} \mapsto \mathbb{R}^{m, \sum d_i}$$

is defined as follows

$$\text{concat}(\mathbf{A}_1, \dots, \mathbf{A}_h) = \begin{pmatrix} \mathbf{A}_1 & \dots & \mathbf{A}_h \end{pmatrix}.$$

We set  $h$  as the number of heads used in the multi-head attention. If each head is to be employed equally,  $d$  must be divisible by the number of heads  $h$ . The following definition summarizes multi-head self-attention.

**Definition 2.3.2** (Multi-head self-attention). *Let  $\mathbf{X} \in \mathbb{R}^{n, d}$  be a matrix of input sequence data. Subsequently, let  $h \in \mathbb{N}$  be a number of attention heads,  $d \in \mathbb{N}$  the model dimension and  $d_k \in \mathbb{N}$  is the head dimension. The output of an  $i$ -th **attention head** can be denoted as*

$$\text{head}_i = \text{attention} \left( \mathbf{XW}_q^{(i)}, \mathbf{XW}_k^{(i)}, \mathbf{XW}_v^{(i)} \right),$$

where  $\mathbf{W}_q^{(i)}, \mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)} \in \mathbb{R}^{d, d_k}$  are learnable weight matrices of  $i$ -th head. a **multi-head self-attention** function is then given by

$$\text{MultiHead}(\mathbf{X}) = \text{concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}_o,$$

where  $\mathbf{W}_o \in \mathbb{R}^{d_k h, d}$  is a learnable weight matrix.

### 2.3.3.3 Cross-attention

In the decoder, the data flow deviates slightly from the encoder. The second sublayer of the decoder uses two inputs: the output of the final stacked encoder  $\mathbf{Z}_{(\text{enc})} \in \mathbb{R}^{n, d}$  and the input of the decoder  $\mathbf{Z}_{(\text{dec})} \in \mathbb{R}^{m, d}$ , which has already passed through the self-attention layer. Using these inputs, a multi-head cross-attention is performed, facilitating interaction between the encoder and decoder. The keys and values vectors are derived from the encoder data  $\mathbf{Z}_{(\text{enc})}$ , while the query vectors are generated by the decoder, specifically from the output  $\mathbf{Z}_{(\text{dec})}$  of its previous self-attention layer.

It is important to mention that the encoder output, coming from the encoder block to generate keys and values, is passed from the last stacked encoder. Thus, the same data enters each of the stacked decoders from the encoder block.

Therefore, let  $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d, d_k}$  be learnable weight matrices. Furthermore,

$$\mathbf{K}_{(\text{enc})} = \mathbf{Z}_{(\text{enc})} \mathbf{W}_k, \quad \mathbf{V}_{(\text{enc})} = \mathbf{Z}_{(\text{enc})} \mathbf{W}_v \quad \text{and} \quad \mathbf{Q}_{(\text{dec})} = \mathbf{Z}_{(\text{dec})} \mathbf{W}_q.$$

Then, the formula (2.7) is applied as follows

$$\mathbf{Z} = \text{attention}(\mathbf{Q}_{(\text{dec})}, \mathbf{K}_{(\text{enc})}, \mathbf{V}_{(\text{enc})}) = \text{softmax} \left( \frac{\mathbf{Q}_{(\text{dec})} \mathbf{K}_{(\text{enc})}^T}{\sqrt{d_k}} \right) \mathbf{V}_{(\text{enc})},$$

to get the cross-attention matrix  $\mathbf{Z} \in \mathbb{R}^{m, d}$ . The values of  $n$  and  $m$  correspond to the length of the embedded input sequence to the encoder and decoder, respectively. However, for the machine translation task, the sequences are usually padded to the same length, and therefore  $n = m$ .

Multi-head cross-attention function is defined similarly as in Section 2.3.2.

### 2.3.3.4 Masking

When using the model, it is crucial to ensure that it remains causal in its operations. It means that the prediction of the model must not depend on future values. When considering the encoder, there is no need to conceal any values within the model, as all information remains fully accessible and available.

In the case of a decoder, however, the situation is different and the model needs to mask future data, such as future words in a sentence for a machine translation model. To evade this situation, part of the decoder input data are masked, which prevents the model from learning from forthcoming data.

The masking takes place in the self-attention sublayer in the decoder. Let us consider the intermediate result of the attention calculation before applying softmax

$$\mathbf{Z}_{(\text{corr})} = \frac{\mathbf{Q} \mathbf{K}^T}{\sqrt{d_k}},$$

where  $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{m, d_k}$  and  $\mathbf{Z}_{(\text{corr})} \in \mathbb{R}^{m, m}$ .

The resulting correlation-style matrix  $\mathbf{Z}_{(\text{corr})}$  shows the relevance between each vector in the sequence. To achieve masking of future tokens, it is necessary to hide the values above the diagonal. In practical implementations of the model, masking of future values is achieved by substituting them with negative infinity. This takes place during the computation of the scaled dot-product attention before the softmax is applied. As mentioned in (2.5), the softmax function approaches 0 as its input tends towards negative infinity.

In practice, the softmax application therefore assigns zero weight to each masked element in the matrix, erasing all correlation information.

### 2.3.4 Input and output

We have already introduced the encoder and decoder blocks. Now, it remains to explain the format in which the data enters the whole encoder-decoder structure, and how the output is transformed to obtain predictions of following tokens.

When using Transformers for time series, the architecture differs fundamentally only in the input and output. In this section we will therefore limit ourselves to a brief description, supplemented by references that will provide the reader with a more detailed description.

#### 2.3.4.1 Linear projection and Softmax

We outlined that in the last stage the output of the encoder-decoder structure is passed to the linear layer with associated softmax.

The linear layer

$$\text{Linear} : \mathbb{R}^d \mapsto \mathbb{R}^{d_t}$$

guarantees a dimensionality transformation of the decoder output  $\mathbf{Z}_{(\text{out})} \in \mathbb{R}^{m,d}$  as follows

$$\text{Linear}(\mathbf{Z}_{(\text{out})}) = \mathbf{H},$$

where  $\mathbf{H} \in \mathbb{R}^{m,d_t}$ . Value  $d_t$  corresponds to the dimension of the target language dictionary.

Subsequently, a softmax operation is applied, introducing a non-linear transformation that scales values of the vector to fall within the range of 0 to 1. This results in a vector, where each element within signifies the probability of a specific word from the dictionary occurring as the next token in the sequence.

This output layer configuration is applied for machine translation. However, it is convenient to emphasize its adaptability. Depending on the task, the resulting layer can be modified.

#### 2.3.4.2 Embedding

The input data to the encoder and decoder are sequences of tokens, that need to be converted into numerical representations. This is done by embedding. Each token is first represented as a one-hot vector, and then transformed into continuous, dense embedding by an embedding matrix. This matrix learns contextual representations of the tokens, capturing semantic information.

If we consider the space of all possible words in the source language  $\mathcal{L}$ , then a token  $w$  from this space is first converted into a one-hot vector  $\tilde{w} \in \{0, 1\}^{d_s}$ , where  $d_s$  is the size of the source dictionary. Simplistically, such a vector contains a one at the position in the dictionary corresponding to the token  $w$  in  $\mathcal{L}$ , and zeros elsewhere. Words that do not appear in the dictionary are encoded as unknown token. In the last step, embedding is applied and  $\tilde{w}$  is converted to a numerical representation of  $x \in \mathbb{R}^d$ . For more detailed description, please refer to [37, 43, 48].

### 2.3.4.3 Padding

Padding is essential in Transformers due to its ability to process sequences of variable lengths in parallel. Sequences delivered during training are padded with a special padding token to ensure uniform length within a batch. This allows efficient batch processing and facilitating computation across multiple sequences simultaneously. Usually, all batches are padded to the same length, which is called maximum sequence length.

More information about padding can be found in [19].

### 2.3.4.4 Positional encoding

Among the remarkable advantages offered by the Transformer architecture, one of the most significant is the positional encoding (PE). In traditional recurrent models, data are processed sequentially, leading to a natural understanding of temporal order.

In Transformers, sequential information is not inherently captured and PE vectors are introduced to address it. They are added to the encoder and decoder input vectors post-embedding, infusing the data with information about its relative position in the sequence.

The use of positional encoding is proposed in [57], which offers an approach to encode information about relative temporal order information within the data. This addition ensures that despite parallel processing, the model retains an understanding of the temporal relationships, which is crucial for various sequence-based tasks.

Positional encoding involves the computation of vectors utilizing sinusoidal functions. An essential aspect of PE is that it requires no learnable parameters, making it both efficient and versatile.

**Definition 2.3.3.** *Let  $k \in \{1, \dots, n\}$  be a given position in an input sequence of length  $n$  and  $p_k \in \mathbb{R}^d$  be its corresponding encoding, where  $d \in \mathbb{N}$  is an even number representing the dimension of the model. Then a mapping function  $f : \{1, \dots, n\} \mapsto \mathbb{R}^d$  represents a generator of positional encoding and its  $i$ -th element is defined as follows*

$$(p_k)_i = f(k)_i := \begin{cases} \sin(\omega_l \cdot k), & \text{if } i = 2l \\ \cos(\omega_l \cdot k), & \text{if } i = 2l + 1, \end{cases} \quad (2.8)$$

where

$$\omega_l = \frac{1}{10000^{\frac{2l}{d}}}, \quad \forall l \in \{0, \dots, \frac{d}{2} - 1\}. \quad (2.9)$$

The frequencies in Definition 2.3.3 decrease as  $i$  approaches  $d$ . This results in a progression of wavelengths, ranging from  $2\pi$  to  $10000 \cdot 2\pi$ .

It is noteworthy that the choice of the number 10000 in (2.9) is not fixed [34]. Rather, it can vary depending on the length of the input sequence. This adaptability ensures that the PE can effectively capture the nuances of different sequence lengths, adding to its versatility and applicability.

The intuition behind positional encoding can be found in binary representation of numbers. If we consider a sequence of natural numbers and their corresponding binary representations, we observe that the least significant bit (0-1) changes with the highest frequency, i.e. the change is occurring with each transition between even and odd numbers. The second bit in the sequence (0-2) changes with a lower frequency, i.e. with each pair of consecutive numbers, and so on.

Consider the relative distance of two natural numbers. When the numbers are close to each other, then less significant bits in their representations change. On the other hand, if they are far apart, we observe a change in more significant bits. The idea of positional encoding can be seen in a similar way.

To further illustrate the motivation behind positional encoding, it is useful to establish its connection to the rotation matrix. As we mentioned, the PE does not encode the absolute position of the data in the sequence, but allows the model to learn relative positions. Thus, if we take a shift of any number of positions  $l$ , we can show that this shift is equivalent to applying a linear transformation, namely a rotation.

Let us state and prove a theorem that demonstrates this fact in two dimensions. This theorem was outlined in [1].

**Theorem 2.3.1.** *Let  $k \in \mathbb{N}$  be a given position in a sequence and let  $\phi \in \mathbb{N}$  be a fixed offset. For each sine-cosine pair corresponding to the frequency  $\omega$ , there exists a linear transformation  $\mathbf{M} \in \mathbb{R}^{2 \times 2}$  independent of  $k$  such that*

$$\mathbf{M} \cdot \begin{pmatrix} \sin(\omega \cdot k) \\ \cos(\omega \cdot k) \end{pmatrix} = \begin{pmatrix} \sin(\omega \cdot (k + \phi)) \\ \cos(\omega \cdot (k + \phi)) \end{pmatrix}.$$

*Proof.* Let  $\mathbf{M}$  be a matrix with elements  $m_{1,1}$ ,  $m_{1,2}$ ,  $m_{2,1}$  and  $m_{2,2}$ , so that

$$\begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix} \cdot \begin{pmatrix} \sin(\omega \cdot k) \\ \cos(\omega \cdot k) \end{pmatrix} = \begin{pmatrix} \sin(\omega \cdot (k + \phi)) \\ \cos(\omega \cdot (k + \phi)) \end{pmatrix}.$$

We can apply addition theorem to sine and cosine as follows

$$\begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix} \cdot \begin{pmatrix} \sin(\omega \cdot k) \\ \cos(\omega \cdot k) \end{pmatrix} = \begin{pmatrix} \sin(\omega \cdot k) \cos(\omega \cdot \phi) + \cos(\omega \cdot k) \sin(\omega \cdot \phi) \\ \cos(\omega \cdot k) \cos(\omega \cdot \phi) - \sin(\omega \cdot k) \sin(\omega \cdot \phi) \end{pmatrix},$$

which results in equations

$$\begin{aligned} m_{1,1} \sin(\omega \cdot k) + m_{1,2} \cos(\omega \cdot k) &= \cos(\omega \cdot \phi) \sin(\omega \cdot k) + \sin(\omega \cdot \phi) \cos(\omega \cdot k) \\ m_{2,1} \sin(\omega \cdot k) + m_{2,2} \cos(\omega \cdot k) &= -\sin(\omega \cdot \phi) \sin(\omega \cdot k) + \cos(\omega \cdot \phi) \cos(\omega \cdot k). \end{aligned}$$

By solving the equations, we get

$$\begin{aligned} m_{1,1} &= \cos(\omega \cdot \phi), \quad m_{1,2} = \sin(\omega \cdot \phi), \\ m_{2,1} &= -\sin(\omega \cdot \phi), \quad m_{2,2} = \cos(\omega \cdot \phi), \end{aligned}$$

and so the transformation matrix  $\mathbf{M}$  can be expressed as

$$\mathbf{M} = \begin{pmatrix} \cos(\omega \cdot \phi) & \sin(\omega \cdot \phi) \\ -\sin(\omega \cdot \phi) & \cos(\omega \cdot \phi) \end{pmatrix}.$$

□

Thus, the resulting rotation matrix does not depend on the absolute position  $k$ . The theorem can be generalized to higher dimensions, and its validity confirms that a PE vector  $p_{k+\omega}$  can be written as a linear transformation of a PE vector  $p_k$ .

### 2.3.5 Training and inference

To conclude this section, let us focus on the difference between training and testing a Transformer. As in the whole Section 2.3, it is presented on a machine translation task. However, the concept itself remains the same for time series.

#### 2.3.5.1 Training

During training, a major advantage over recurrent networks is that the entire process runs in parallel. The data are grouped into batches as part of the mini-batch optimization [15], but for simplicity we consider a batch size of 1.

The input to the encoder is a sequence of words in the source language, which is first padded to a given maximum sequence length. Next, the sequence is converted into numerical representations by embedding, and PE is added to preserve the temporal order. The data then enters the encoder, where it is processed by the encoder  $N_x$  times and then passed to the decoder.

The target language sequence at the decoder input is similarly padded, converted with input embedding and enriched with PE. In the decoder, masking is usually applied in the self-attention layer to prevent leakage of future information. On each pass through the decoder, the output of the last encoder is used in the encoder-decoder layer. Finally, the linear layer upscales the decoder output to the size of the target language dictionary and softmax outputs the probabilities of the words in the dictionary.

The loss function is computed, comparing the predicted probability vectors for the subsequent words against the ground truth targets. Based on this loss, the model parameters are optimized.

#### 2.3.5.2 Inference

During the inference phase, the encoder works identically as during training. However, the fundamental change is in the decoder, that performs predictions iteratively.

Ground truth values cannot be used while testing the model, as they would reveal information about true previous words, which is only applicable in the training phase. Therefore, only the representation of the token that marks the beginning of sentence is input to the decoder to get the first predicted word. This word is appended to the input of the decoder and the process is repeated iteratively. Word by word is predicted with new words being appended to the already predicted words at the input of the decoder at each step. The prediction is complete when the model predicts a token marking the end of sequence, or when the maximum length of the sequence is reached.

The disadvantage of this iterative prediction is that the predictions are not independent and depend on previous predictions. In addition to the standard error of the prediction, a multiplicative error must be considered that increases with the length of the predicted sequence.

## 2.4 Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) is a deep neural network based on the attention mechanism [39]. It is an architecture for multi-horizon forecasting with an enhanced form of interpretability, shown in Figure 2.5.

It incorporates components representing novel approaches to handle input data of different types, such as static and real, or of a different temporal positions, namely known, past, and future

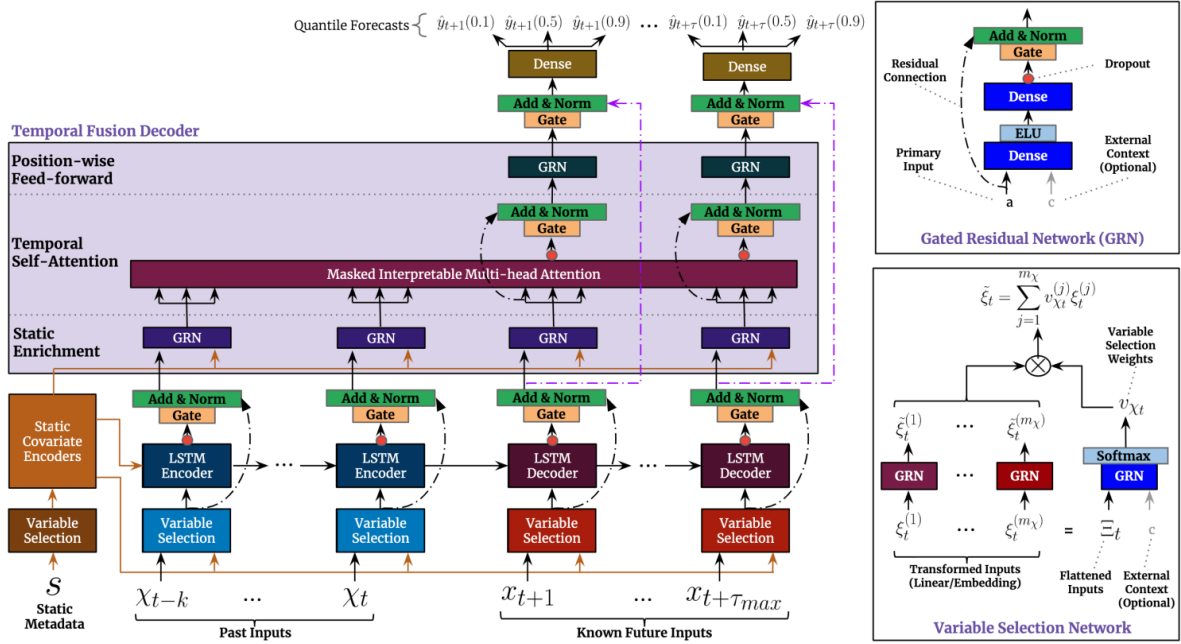


Figure 2.5: Temporal Fusion Transformer architecture proposed in [39].

inputs. In accordance with this division, the inputs are then processed separately in different parts of the model. We briefly review the different parts of TFT and discuss their contribution. For further description and details of the model, we refer the reader to the original paper.

The authors proposed a Gated Residual Network (GRN), based on the Gated Linear Unit (GLU) [16] mechanism, which gives the model the ability to skip non-linear processing when it is not needed. Thus, the model is able to vary its complexity depending on the type and size of the dataset.

Another component of the model are variable selection networks. In multi-variable datasets, it is often unknown which variables have the greatest predictive power. Variable selection can select the most significant variables at each point in time and suppress variables with potentially negative effects on predictive performance. The factor and continuous variables are first separately encoded in feature embeddings, the former by entity embedding [28], the latter with a linear layer. Within variable selection nets, the GRN is responsible for the selection itself.

The main part of the model is the Temporal Fusion Decoder, which is used to capture both short and long term dependencies. A modified version of the attention mechanism, called interpretable multi-head attention, is used together with masking to find relative dependencies in the data. Before the data are loaded into the Temporal Fusion Decoder, the sequence-to-sequence layer is applied, where LSTM blocks are used to encode the data into uniform temporal features. The information is encoded sequentially, preserving the temporal information. This step replaces the standard positional encoding.

Finally, the output of the Temporal Fusion Decoder is routed to a linear layer. As in [58], quantile regression is used to obtain prediction intervals. At each time step, a set of predictions is obtained for each quantile, which helps to determine the range of possible predicted values.



# Chapter 3

## Data

For the task of system imbalance prediction, the case of power exchange trading is simulated. Belgium was chosen for this purpose, as it participates in the European exchange EPEX Spot through short-term contracts on the intraday and day-ahead markets. This thesis focuses on intraday market forecasting, specifically observing 15-minute contracts.

The system imbalance is modeled using data with 15-minute granularity from the Belgian transmission system managed by the operator ELIA. The data are sourced from the ELIA web portal [21, 22, 23, 24] and are accessible free of charge via their website. Additionally, ELIA offers an API, facilitating the possibility for real-time data flow, which could be leveraged to deploy a real-time forecasting system.

The data were collected from January 2022 to August 2023. Although complex DL models (such as the Transformer) require a large amount of data, it was decided to use only this data for this work. Earlier data from 2022 and 2021 are affected by a number of factors, such as the unprecedented COVID-19 pandemic and the subsequent energy crisis. The data can deviate significantly from normal behavior and were therefore omitted to avoid biasing the model.

### 3.1 Temporal causality

When working with time series, it is necessary to maintain temporal causality, thus not breaking the natural sequential characteristics of the data. Consider the split times  $t_k$  that divide the timeline into 15-minute trading windows  $T_k = (t_k, t_{k+1})$ . During the training phase, the forecasting system must not use information that was not known at time  $t_k$  when predicting the system imbalance of window  $T_k$ . In other words, it must be ensured that there is no data leakage from the future.

There is no future data leakage in this thesis. However, it is important to consider an additional obstacle, which is the delay between the end of the contract and the publication of the corresponding data by the TSO. Consumption and production are monitored by a system of sensors and other devices. Variables such as load, wind and solar generation, or most importantly, system imbalance belonging to the quarter-hour window  $T_{k-1}$ , are collected over the entire time interval  $(t_{k-1}, t_k)$ . At the end of this period, the TSO has to acquire the data, summarize it, and then publish it. Thus, there is a lag of  $\Delta$  between  $t_k$  and the time of publication  $\tilde{t}_k$ , as shown in Figure 3.1.

From the perspective of the TSO, the delay is  $\Delta$ , or potentially even less, as the TSO is tasked with collecting the data and thus has access to it prior to its publication. Based on this data,

the TSO can evaluate the state of the grid and possibly use ancillary services to compensate any deviation in the system.

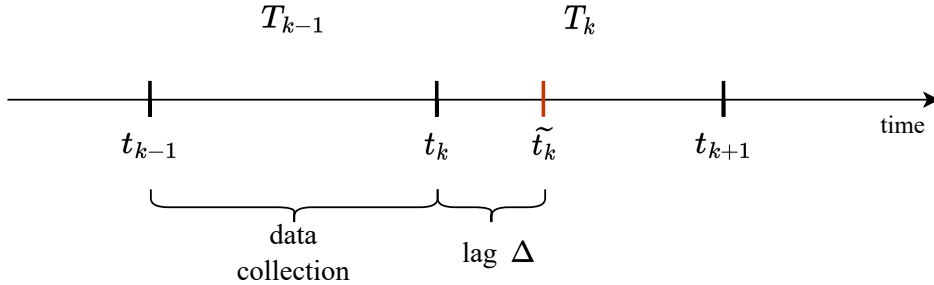


Figure 3.1: After data collection during interval  $(t_{k-1}, t_k)$ , there is a delay  $\Delta = \tilde{t}_k - t_k$  during which the TSO collects and publishes the data.

From the other side, i.e., the market participants, the situation is similar. The incentive to enter into a contract in the intraday market is to adjust their market position in view of the upcoming settlement. To decide whether to execute a trade, a market participant can use only the data available at the latest point in time when the contract can be traded. In the case of the EPEX Spot exchange, the closing time is 5 minutes prior to delivery. Therefore, even if the delay  $\Delta$  were completely neglected, the trade would still not be possible with the data collected at the end of window  $T_{k-1}$ .

As a result, it is effectively impossible at time  $t_k$  to have knowledge of all the variables used to predict the window  $T_k$ . This fact has to be openly admitted since this delay is omitted in the thesis and the first possible trading window would be the one starting at time  $t_{k+1}$ . This simplification was made to streamline the data collection process.

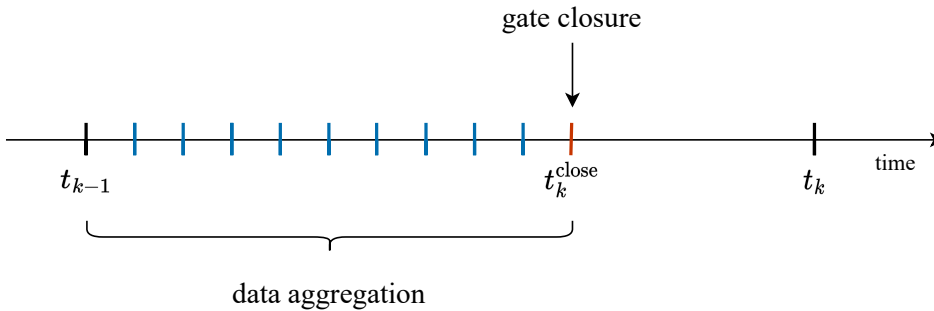


Figure 3.2: The diagram shows the use of data with 1-minute granularity to predict system imbalance for a 15-minute contract. Time  $t_k^{\text{close}}$  denotes the gate closure, e.i. the point at which trading is completed for block  $T_k$ . The operating time required to process the data, make the prediction and execute the trade is neglected.

On the other hand, there are several ways to replace the unavailable values at time  $t_k$  to model system imbalance of the window  $T_k$ . The problem arises for measured values of solar and wind production, total load, and also the target, system imbalance. To address this, the unavailable values of wind production, solar production, and total load can be effectively replaced with one of ELIA's published forecasts. ELIA provides forecasts a week and a day in advance, along with

a most recent forecast that is updated until just a few hours before delivery. These forecasts are of high quality, as demonstrated later by the correlations shown in Tabs 3.2 and 3.3. Similarly, it is possible to use lagged variables and thus settle for information from the  $T_{k-2}$  window. The correlation is comparable to the ELIA forecasts.

There are several methods to address the issue of unavailable system imbalance values. Firstly, forecasts can again be used, since ELIA publishes its own forecasts of system imbalance. This approach essentially forms an ensemble model, as it incorporates ELIA’s model predictions of the imbalance of  $T_{k-1}$  to predict the imbalance of  $T_k$ .

Secondly, ELIA publishes data at different granularities. While the most commonly used granularity is 15 minutes, aligning with the traded contracts, ELIA also provides data at 1-minute intervals, available nearly in real-time. This finer granularity data can be aggregated to determine the approximate imbalance for a corresponding 15-minute window, as shown in Figure 3.2.

Thus, although prediction is impossible in the first window, it is possible to substitute not available data based on these facts. How the subsequent prediction would be affected is not part of this thesis and is left unanswered.

## 3.2 Data preprocessing

An important part of the whole process of modeling system imbalance is to preprocess the data. The key steps are variable selection and following feature engineering, which includes lags generation, aggregation and feature encoding. These steps will be discussed in detail in the following sections.

As explanatory variables were selected those, that are likely correlated with the system imbalance. In this case, the dynamics of the whole transmission network is taken into account. Notably, special attention is directed towards RES, such as wind and solar production. The inherent fluctuations in these energy sources are recognized as significant contributors to grid imbalance, underscoring their importance in the analysis.

### 3.2.1 System imbalance

The key variable in forecasting is undoubtedly the target system imbalance itself. Historical values are instrumental in understanding the progression and trends of the time series. By analyzing past data, model can identify patterns and anomalies, enabling more accurate predictions about future imbalances. In addition, this historical insight allows the model to be refined, increasing its ability to adapt to changes in patterns over time, thereby improving the overall accuracy and reliability of the predictions.

In Figure 3.3, we observe the evolution of system imbalance during 7th May 2023. It is observed that the system imbalance often changes abruptly. However, it still fluctuates around the desired value of zero. This erratic behavior, marked by occasional extreme increases or decreases in the imbalance, poses significant challenges in its prediction.

Compared to other variables observed in the transmission system, system imbalance does not exhibit a clear pattern. As discussed, its behavior depends on unexpected changes in other variables within the grid.

Also related to system imbalance is the price set by the TSO in the aftermarket settlement. There is a clear dependence between these variables, that are negatively correlated. If the imbalance is

negative and there is a shortage in the grid, the price increases. Conversely, if there is a surplus, the price decreases and can even reach negative values.

The imbalance price is not used in this thesis. However, when the imbalance is precisely predicted, it becomes possible to anticipate market price movements, thereby creating an opportunity for speculation on the final price.

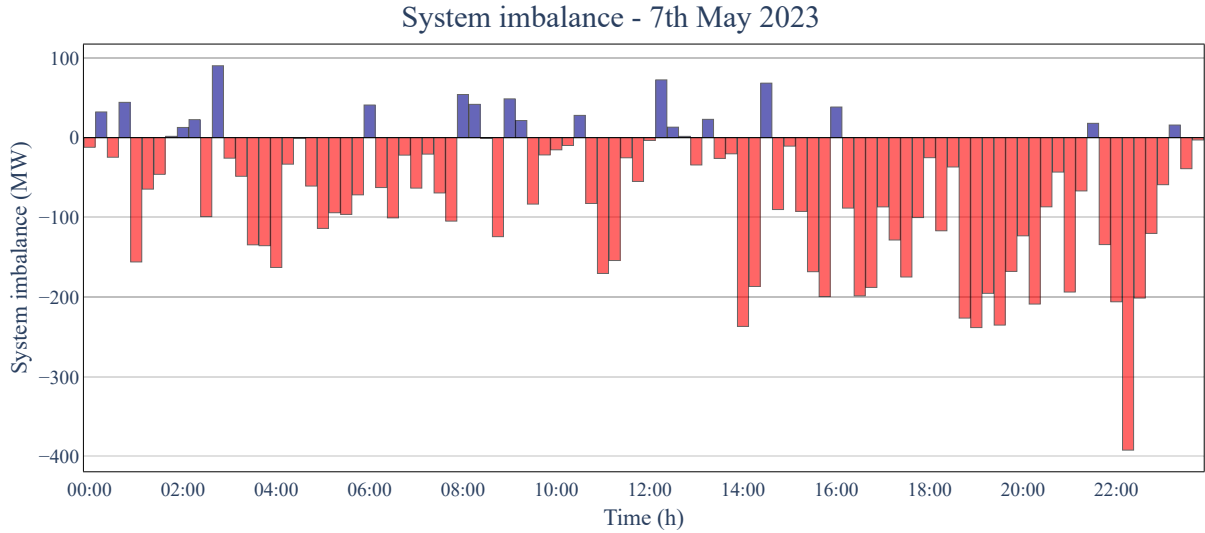


Figure 3.3: System imbalance values in Belgian grid during 7th May 2023. For clarity, positive imbalance is depicted in blue, whereas negative imbalance is depicted in red.

An analysis of the autocorrelation data in Table 3.1 reveals that the target sequence does show some degree of correlation. However, within the context of time series analysis, this correlation is not particularly strong. This finding is significant because it implies that traditional time series forecasting models, which primarily rely on historical data, might not be highly effective in predicting system imbalances. The relatively lower correlation observed in the sequence suggests the need for more sophisticated or alternative forecasting approaches.

lag	1	2	3	4	5	6	7	8
<b>autocorrelation</b>	0.652	0.459	0.397	0.433	0.292	0.198	0.162	0.174

Table 3.1: Autocorrelation of system imbalance at different lags.

How volatile system imbalance is can also be verified by examining the aggregated monthly data in Figure 3.4. Monthly standard deviations exceed 200 MW and extreme deviations reach in absolute value over 1000 MW. It indicates the complexity of the forecasting task.

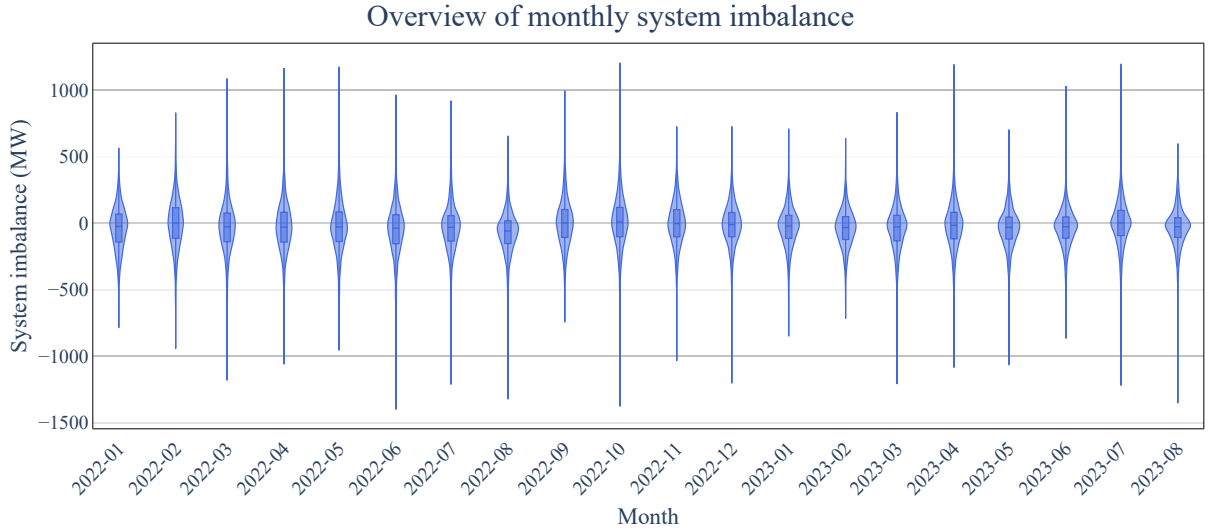


Figure 3.4: Preview of the distribution of monthly system imbalance values.

The use of lagged values of system imbalance was proposed in feature engineering. It involves extracting values from the past that are assumed to be similar, based on the periodicity of the market. Different lags can then be used according to the expected cyclicity.

In the model, lags serve as reference values that capture past behavior of the time series. For system imbalance, we expect a cyclical pattern that repeats daily and weekly.

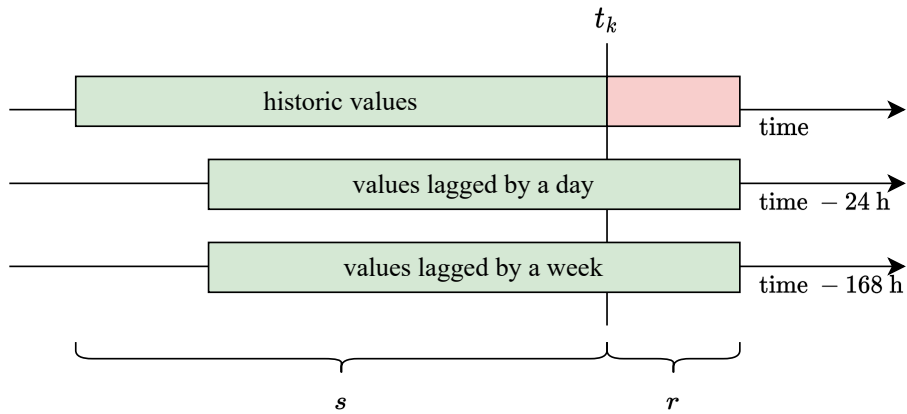


Figure 3.5: Scheme corresponding to the used lags for system imbalance.

Let us consider a data sequence of length  $s \in \mathbb{N}$  representing historic values, used as predictors, and a future sequence of length  $r \in \mathbb{N}$ , which is to be predicted. Lags are designed to cover window of length  $s$  one day back (i.e. 24 hours), and one week back (168 hours). Specifically, considering the situation shown in Figure 3.5, a future sequence of length  $r$  is predicted at time  $t_k$ . The historical sequence consists of data for windows  $(T_{k-s}, \dots, T_{k-1})$  and the predicted windows are  $(T_k, \dots, T_{k-1+r})$ .

Consider notation given by  $T_j^l = T_{j-l}$  for arbitrary  $j, l \in \mathbb{N}$ . Both sequences of lags, which are added to the input data, can be then written as  $(T_{k-s+r}^{96}, \dots, T_{k-1+r}^{96})$  and  $(T_{k-s+r}^{672}, \dots, T_{k-1+r}^{672})$ . The values 96 and 672 are numbers of delayed quarter hours for a day and a week, respectively. The length of both lags  $s$  corresponds to the length of the input sequence, ensuring compatibility with the input dimensions required by certain models.

### 3.2.2 Total load

Another important and highly monitored variable in the power grid is total load. It consists of all the electrical loads on the ELIA grid, including the connected distribution systems. Energy losses in the system are also taken into account. Before publication, the measured data are further processed by aggregation and extrapolation.

The total load is not affected by such frequent fluctuations and patterns are visible in the data. In particular, it is possible to identify so-called peak demands, i.e. periods when the grid experiences high load. The increase in energy consumption is attributed to operational demands of businesses and factories, mainly during weekdays. The highest workload occurs in daytime working hours and Figure 3.6 illustrates these weekly peak patterns. The EPEX Spot exchange is prepared for these predictable demands and therefore one of the trading windows is marked as peakload. It is defined as a 12-hour period from 8am to 8pm on weekdays.

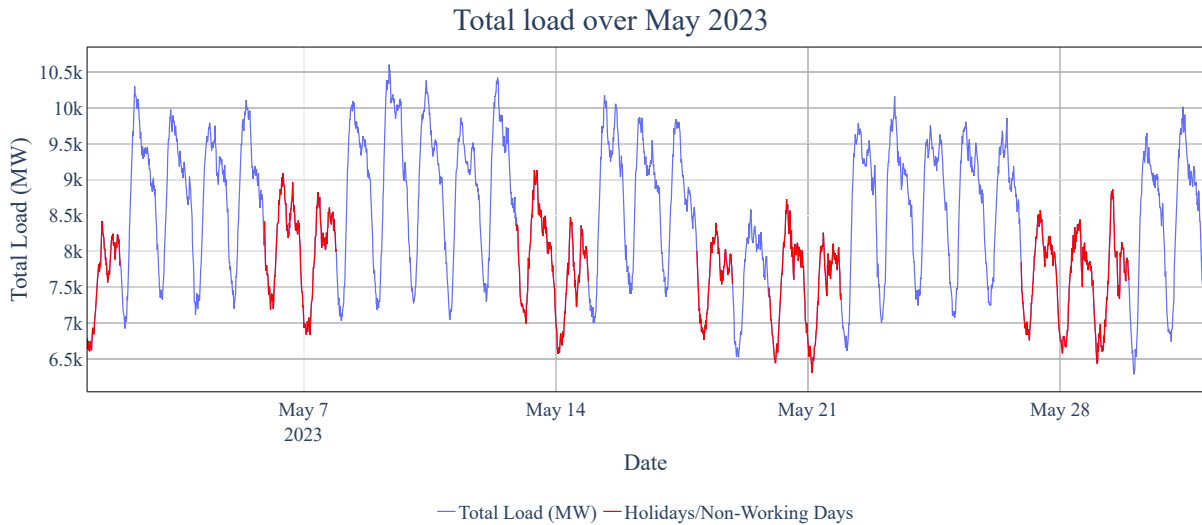


Figure 3.6: Total load during May 2023. It is evident that peaks during working days typically exceed those observed on holidays and non-working days.

In addition to the load itself, forecasts available from ELIA are used. The first is the most recent forecast, usually updated up to a few hours before the contract execution. The second is the day-ahead forecast, available at 6pm the previous day. In Table 3.2 we present the correlations that these forecasts have with the measured load.

Because of the high correlation, the use of forecasts in the model seems redundant. However, it is suspected that the direction of change in real consumption relative to forecasts provides additional information to the model.

forecast	correlation with load
most recent	0.987
day-ahead at 6PM	0.969

Table 3.2: Pearson correlation between measured total load and its forecasts provided by ELIA.

### 3.2.3 Wind and solar

Wind and solar energy production is inherently variable and highly dependent on weather conditions. This has a direct impact on the imbalance in the grid and is therefore an important variable in forecasting. As discussed in Section 1.1, renewable resources, led by wind and solar, are prone to unpredictable changes in generation. These deviations have to be taken into account in the model.

In the data used, production for each time window is available separately for each region in Belgium. Furthermore, the wind data are split according to whether the wind farm is onshore or offshore. Despite the loss of information, the production data were summed. Thus, only the total wind and solar production can be used. Since the system imbalance is aggregated as well, it is considered that these aggregated variables are sufficient for predicting the imbalance in the whole grid.

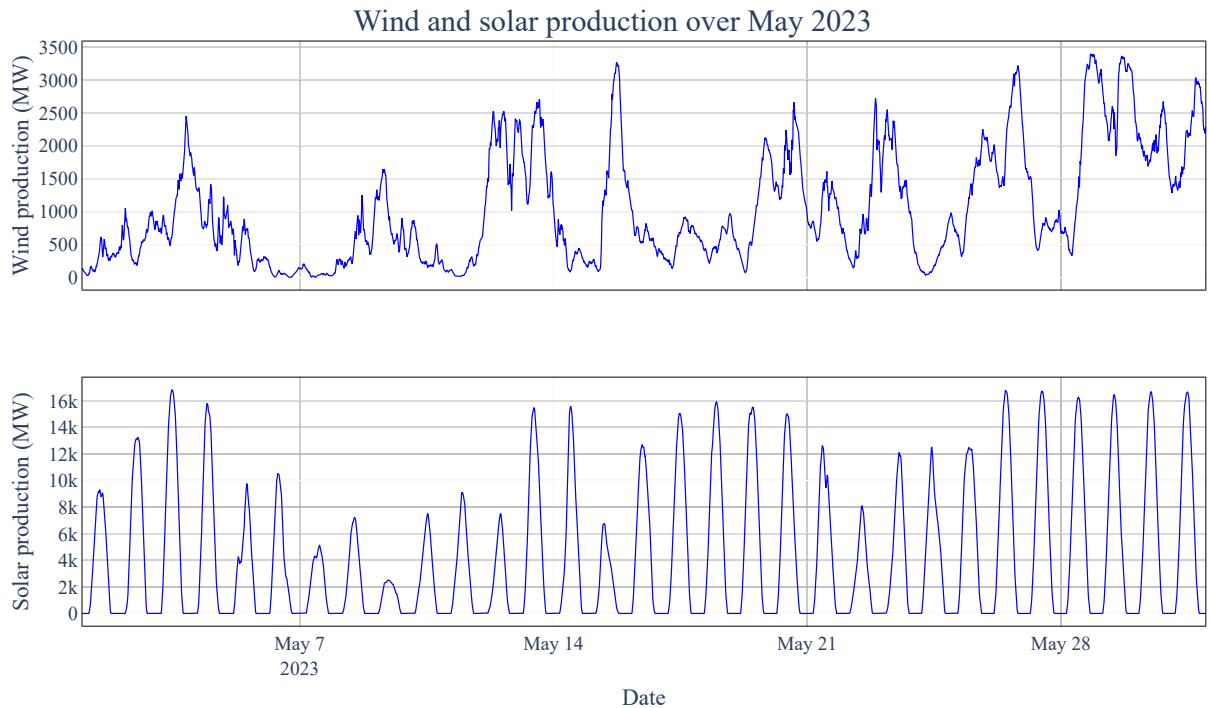


Figure 3.7: Wind and solar production during May, 2023.

As with total load, there are forecasts available as well and they are the most recent and the day-ahead forecasts. Both have been summed as well as real production. Table 3.3 presents Pearson correlation between these forecasts with real production for both wind and solar.

The progression of solar and wind production can be seen in Figure 3.7. In the case of wind, a drop in production can be observed during windless periods. For solar, the pattern is much more obvious, with peaks occurring during the day and production ceasing at night.

forecast	correlation with wind	correlation with solar
most recent	0.973	0.993
day-ahead at 6PM	0.956	0.982

Table 3.3: Pearson correlation between measured variables (wind and solar) and their forecasts provided by ELIA.

### 3.2.4 Calendar data

Using the calendar data, it is possible to extract a lot of useful information. This is because each time window can be uniquely identified by its timestamp. In the case of Transformer, this data can be used to preserve temporal order information. Unlike NLP tasks, where the only temporal component is given by the order of the words in a sentence, time series forecasting can benefit from this explicit temporal data. Alternatively, it can be used instead of positional encoding by incorporating additional variables that specify temporal order.

Several pieces of information can be extracted from datetime and are used to train the model. In particular, these variables include:

- Information indicating the year, month, and day of the year.
- A variable specifying the day of the week. In 3.2.2, we have already discussed how this information is crucial in the case of load, so it can also carry essential information for modeling system imbalance.
- The hour and quarter-hour indicate the window for which the contract is traded. This might be important information to identify daily peaks.
- Variables indicating whether a day is a holiday or a day off. It was collected for Belgium [44] and serves as additional information. During these days, the market behaves more like a weekend day, so it is useful to inject this knowledge into the data.

Factor time variables must be encoded into numerical representations to be used in modeling. Essentially, there are two ways in which encoding can be performed.

The first one is **one-hot encoding**, which is a process of converting categorical variable into binary vector for each category of the variable. Let  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$  be a set of  $k \in \mathbb{N}$  unique categories. For all  $i = 1, 2, \dots, k$ , a given category  $C_i$  is encoded as a one-hot vector  $e_i \in \{0, 1\}^k$  as

$$e_i = (\delta_{ij})_{j=1}^k,$$

where  $\delta_{ij}$  is the Kronecker delta, defined as

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

The second option is to use **ordinal encoding**. This is a technique in which a numerical value is assigned to each category of a factor variable based on the given order. Let  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$



be an ordered set of  $k$  unique categories. The ordinal encoding of a category  $C_i$  is represented by a mapping function  $f : \mathcal{C} \mapsto \{1, \dots, k\}$ , where:

$$f(C_i) = i, \quad \forall i = 1, 2, \dots, k.$$

Categories must have meaningful ordering, often specified explicitly. An example might be the order of days of week from Monday to Sunday, or months from January to December.

Both types of encoding have their disadvantages. One-hot encoding increases the dimensionality of the data and for  $k$  categories it introduces  $k$  new variables to the data. For high values of  $k$ , this increase may be excessive. In addition, this added data are sparse, which reduces its informative value. Similarly, ordinal encoding introduces a notion of quantitative relationship into the data that may not be true. For example, the encoding of weekdays gives the false impression that Monday (0) and Sunday (6) are very far apart, whereas they are adjacent days. There are observable patterns in the market that repeat periodically on a daily, weekly and so on basis. In order to preserve cyclicity, temporal consistency, and avoid a rocketing increase in dimensionality, the trigonometric functions sine and cosine were chosen to transform the data.

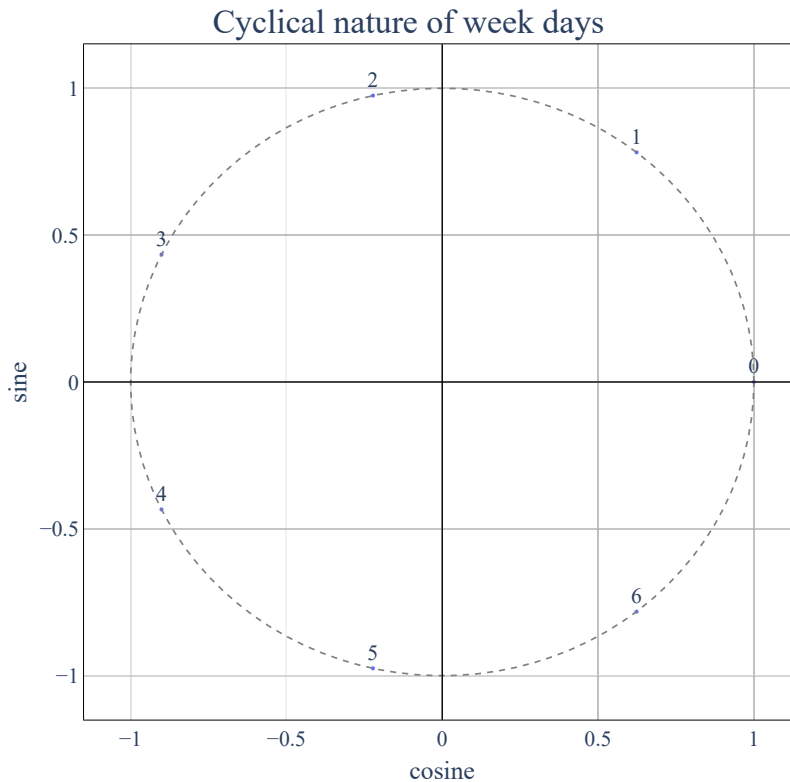


Figure 3.8: Both the sine and cosine components of the cyclically encoded feature map it to a two-dimensional space on a circle. In the example of weekdays, the transition from Sunday (6) to Monday (0) is given as any other transition between two adjacent days.

Consider an ordered set of  $k$  unique categories with categories  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$  and a cyclical categorical variable  $u \in \mathcal{C}^n$ , that repeats every  $T$  steps. Next, consider that the variable  $u$  was encoded element-wise using ordinal encoding map  $f$ . Therefore, the numerical representations

are encoded as  $\tilde{\mathcal{C}} = \{1, 2, \dots, k\}$  and the encoded vector  $u$  is denoted as  $v$ . Subsequently, the **cyclical feature encoding** with sine and cosine transformations of  $v$  is defined as

$$\begin{aligned} x^s &= \sin\left(\frac{2\pi \cdot v}{T}\right), \\ x^c &= \cos\left(\frac{2\pi \cdot v}{T}\right). \end{aligned}$$

The variable  $v$  is thus effectively mapped onto a circle using the  $x^s$  and  $x^c$ . The periodic nature of the data are preserved along with the information about the continuity of the data at the end of the cycle, as shown in Figure 3.8. This approach is particularly useful in time series forecasting, where cyclical features play a key role in predicting future values.

### 3.3 Data processing

The preprocessed data were initially split into training, validation and test sets, standardized, and then formatted appropriately for input depending on the model used.

Since we work with time series, it is necessary to split the data properly. The commonly used procedure of shuffling the data and then performing a random split cannot be used, as this would break the causality in the data and introduce information leakage from the future. Therefore, a time split should be performed, and the data should be split into whole blocks, with training performed on the oldest available data and validation and testing on the more recent data. The data were split according to Table 3.4.

set	start date	end date	% of dataset
training	2022-01-01	2023-02-28	70
validation	2023-03-01	2023-05-31	15
test	2023-06-01	2023-08-31	15

Table 3.4: Defined partitioning of data into training, validation and test sets.

There are missing data in the following features: wind production (0.04 %), load (0.27 %), and day-ahead load forecast (0.16 %). Although this represents a small portion of the data, it is important to note that the majority was found in the test set. The cause of this missing data are likely due to a measurement error in the transmission system or an incorrect publication by ELIA.

Instead of using common techniques to fill in missing data, such as mean or median imputer, it was suggested to use forecasts to replace them. As ELIA offers multiple forecasts for these variables at various times, the most recent forecast can substitute for the actual value, and any missing forecast can be replaced by an earlier one.

All features were standardized except for binary variables and time variables encoded by cyclical feature encoding. The target variable system imbalance was also scaled in the same way.

Let us consider a variable  $u \in \mathbb{R}^n$ . The standardization is given by formula

$$v_i = \frac{u_i - \bar{u}}{s_n}, \quad i = 1, \dots, n,$$

where  $\bar{u}$  is the estimated mean and  $s_n$  is the estimate of standard deviation.

Subsequently, these estimates are defined as follows

$$\bar{u} = \frac{1}{n} \sum_{j=1}^n u_j, \quad s_n = \sqrt{\frac{1}{n-1} \sum_{j=1}^n (u_j - \bar{u})^2}.$$

Standardization is performed for training, validation, and test data using mean estimation and standard deviation calculated on the training set to avoid data leakage.

Once all three datasets are available, they need to be converted into a format suitable for model input, which essentially depends on the type of model. For the sake of simplicity, we will also consider a batch size of one throughout the rest of this section. In addition, in this and subsequent chapters, we will occasionally consider matrices where one of the dimensions is 1. This notation is intentional and is intended to correspond to the data during implementation of the models.

As mentioned in Chapter 2, some of the models used were not originally designed to work with time series and therefore need to be modified. Observations representing individual time steps must be grouped into sequences of length  $s$ , representing history, and future sequences of length  $r$ . Let  $x_t \in \mathbb{R}^p$  be a vector of explanatory features and  $y_t \in \mathbb{R}$  be the target value, where  $t \in \{1, \dots, n\}$  and  $p \in \mathbb{N}$  is the number of explanatory variables. Then  $\mathbf{E}_t \in \mathbb{R}^{s,p+1}$  is the input data matrix, given by

$$\mathbf{E}_t = \begin{pmatrix} y_{t-s+1} & y_{t-s+2} & \dots & y_{t-1} & y_t \\ x_{t-s+2} & x_{t-s+2} & \dots & x_{t-1} & x_t \end{pmatrix}^T$$

and  $\mathbf{G}_t \in \mathbb{R}^{r,1}$  is the target matrix given as

$$\mathbf{G}_t = (y_{t+1} \quad y_{t+2} \quad \dots \quad y_{t+r-1} \quad y_{t+r})^T.$$

In order to construct the matrices  $\mathbf{E}_t$  and  $\mathbf{G}_t$  correctly, the index domain must be reduced to  $t \in \{s, \dots, n-r\}$ . This initial setting is valid for all models implemented in this thesis.

Standard machine learning models, such as MLP and XGBoost, are directly fed by the input vector batch. The input matrix  $\mathbf{E}_t$  has to be flattened to the input vector  $e_t$ , where it does not matter in which dimension this operation is performed. If we define the flattening operation for any  $d_1, d_2 \in \mathbb{N}$  as

$$\text{vec} : \mathbb{R}^{d_1, d_2} \mapsto \mathbb{R}^{d_1 \cdot d_2},$$

then  $\forall t \in \{s, \dots, n-r\}$  holds

$$\text{vec}(\mathbf{E}_t) = e_t,$$

where  $e_t \in \mathbb{R}^{(k+1) \cdot s}$ .

The specific setup for transformer-based models is described for each model in the next chapter.



## Chapter 4

# Transformer for Time Series

In Section 2.3, we thoroughly discussed the Transformer and its initially defined architecture in [57], that was originally released for machine translation. Therefore, there are necessary changes that need to be made in order to rebuild the model to the use case of time series forecasting. Major modifications must be made in the processing of input data, while leaving the internal structure of the encoder and decoder intact. Further modifications are also required in the output layer to adapt the model to solve the regression problem.

Since the original paper, numerous studies have been published dealing with the application of transformers to time series. Some retain the internal architecture and attention mechanism, such as [11, 61]. Others, including [38, 40, 47, 60, 65] introduce significant changes that alter the architecture, modify the attention mechanism or optimize the overall functionality of the transformer.

In this section, we first discuss general changes in the transformer structure for time series. We then present the proposed models and describe the mechanisms used.

### 4.1 Input Data

Let us first discuss the differences that we need to bridge. The most important are the data itself in the form of a time series with explanatory variables. Instead of a sequence of words for the NLP task, there is a sequence of vectors that provide numerical values of the explanatory variables and the target at each observed time  $t$ . In the straightforward case, we only consider numerical explanatory variables, since we assume that all variables have been already encoded and transformed into numeric representations, as described in Chapter 3.

Therefore, consider a matrix of input values representing the sequence of  $s$  historical observations  $\mathbf{E}_t \in \mathbb{R}^{s \times p+1}$ , as defined in Section 3.3. Unlike machine learning models, transformer-based models use the input matrix  $\mathbf{E}_t$  directly.

The first general change is that since all input data are aligned to the length of the sequence  $s$ , there is no need to use padding at all. For embedding, there are two approaches that can be used to mimic the transformations of the original transformer:

- Instead of embedding, a linear layer

$$\text{Linear} : \mathbb{R}^{p+1} \mapsto \mathbb{R}^d$$

can be used, which converts the data into numerical representations, i.e. transforms the data from dimension  $p + 1$  to model dimension  $d$ . Numerical features can be converted

to representations with higher dimensionality, but this step removes the direct contextuality of the data.

- Since the data are already in numerical form, the use of embedding can be completely omitted. However, it must be set that the model dimension is  $d = p + 1$ .

To preserve contextuality, the latter approach is used in this thesis and only normalized data, where  $d = p + 1$ , are input into the encoder.

The last stage of the data before entering the encoder is the addition of PE, but the procedure is the same as for the language transformer. Alternatively, the transformed temporal variables can be used, which have already been discussed in Section 3.2.4. Since it was assumed that the use of temporal variables would not only preserve temporal order but also aid in prediction, it was decided to use them instead of PE.

Let us now introduce the implemented models.

## 4.2 Vanilla Transformer

The first model used for forecasting is Vanilla transformer, which except for the necessary changes for the time series forecasting use case, follows the original structure from [57]. As mentioned above, we have already made modifications to the encoder input data.

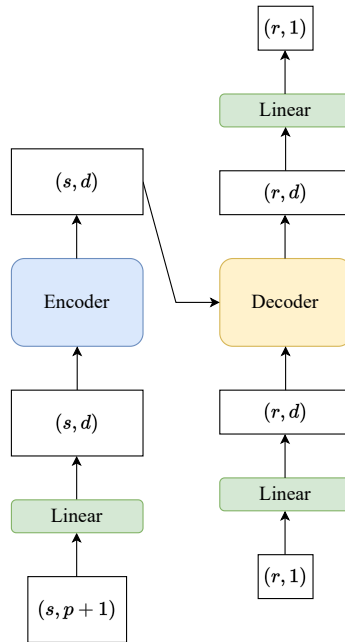


Figure 4.1: Architecture and dimension changes of the data in Vanilla transformer. In our case,  $d = p + 1$ , so the linear layer at the encoder input is suppressed.

Let us focus now on the target sequence that is passed into the decoder. Instead of a sequence of words in the target language, there is a sequence of  $r$  future target values. A matrix  $\mathbf{D}_t \in \mathbb{R}^{r,1}$  of decoder input must be constructed, which is used during training and is given by

$$\mathbf{D}_t = (y_t \quad y_{t+1} \quad \dots \quad y_{t+r-2} \quad y_{t+r-1})^T. \quad (4.1)$$

Note that matrices  $\mathbf{D}_t$  and  $\mathbf{G}_t$  (defined in Section 3.3) are almost identical, except for the index shift. The target value  $y_t$  is also shared with the encoder input  $\mathbf{E}_t$ . This value is equivalent to the token of beginning of sentence and does not bring any new information value, since it was already exposed to the encoder.

The challenge is that the target sequence matrix does not have the same dimensions as the encoder input sequence. For this purpose, a linear layer

$$\text{Linear} : \mathbb{R} \mapsto \mathbb{R}^d$$

is used as an embedding to upscale dimensionality to  $d$ .

Output layer is adapted to the regression problem and consists of a linear layer

$$\text{Linear} : \mathbb{R}^d \mapsto \mathbb{R}^r$$

to transform the output of the decoder into a sequence of  $r$  predicted values. The exact dimensionality change of the input and output data are shown in Figure 4.1.

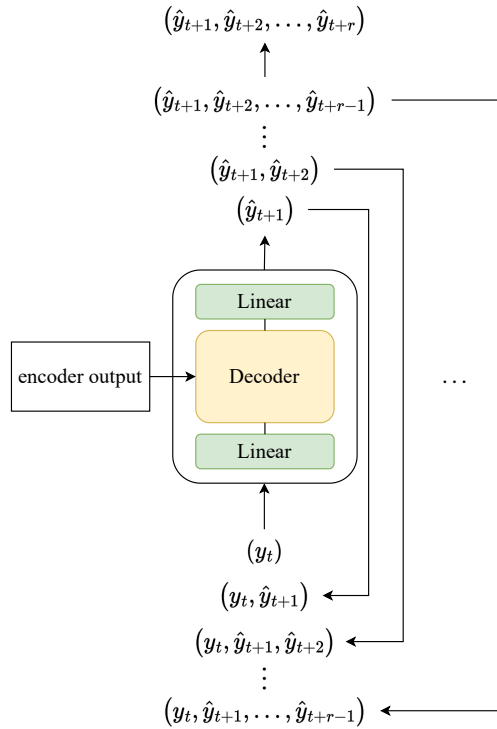


Figure 4.2: During iterative inference, the Vanilla transformer predicts successively values  $\hat{y}_{t+1}, \dots, \hat{y}_{t+r}$ , which are added to the initialization value  $y_t$  at each step. In the last step, the entire sequence  $(\hat{y}_{t+1}, \dots, \hat{y}_{t+r})$  is predicted.

The architectural changes discussed are sufficient to apply the model to time series data. A masking layer is applied during training to mask future targets. In the testing phase, new values are iteratively predicted at each step, using the previous predictions as shown in Figure 4.2. At each

subsequent prediction step, the model adds a single new piece of information, which is the previous predicted value. However, this is a prediction that is inherently subject to error. Thus, the first prediction gives the trend of the entire predicted sequence and can significantly bias the whole sequence, in case of a significant error.

### 4.3 Encoder model

Another implemented model is a transformer using only the encoder part. This type of model is inspired by the architecture popularized by the BERT language model [17]. Encoder models use the encoder to find relevant information in the input sequence. Although these models are used for NLP tasks such as text classification and question answering, they can be similarly adapted for time series. Because of the missing decoder, the only input of the model is the one into encoder.

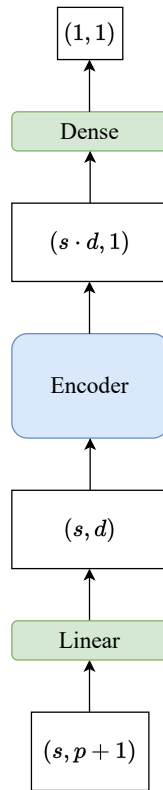


Figure 4.3: Encoder model architecture with marked dimensions of the data.

Compared to transformers with an encoder-decoder structure, it cannot perform sequential inference. To design a model for multi-horizon prediction, a similar approach to that used in machine learning is required. This involves either modifying the linear output layer to predict multiple values simultaneously or training multiple distinct models, that share the same input and each is dedicated to predict  $k$  steps ahead. In this thesis, the latter approach is chosen for  $k \in \{1, \dots, r\}$ . While the input data are processed as in Section 4.2, the encoder output layer has to perform a thorough downscaling. The output matrix is flattened to a vector and then, since it is a significant dimension reduction, a dense layer is used, which is defined as follows



$$\text{Dense} : \mathbb{R}^{s \cdot d} \mapsto \mathbb{R}.$$

In the terminology of neural networks, dense layer refers to a MLP and in this scenario it is convenient due to its hidden layer to facilitate downscaling. The complete architecture of Encoder model is shown in Figure 4.3.

The model applies attention without masking to all observations in the sequence entering the encoder. It is suspected that the encoder could be used to find contexts in the data that are subsequently processed by the dense layer. Thus, the efficiency of the benchmark MLP model could be improved by this model.

#### 4.4 Transformer with future lags

Although both models proposed so far make effective use of the attention mechanism, the fundamental flaw lies in the input data itself. Neither model uses any additional information belonging to the predicted values. Both models base their predictions solely on information that occurred in the past and are part of the input sequence of historical values.

The most comprehensive transformer architecture in this thesis addresses this shortcoming by using so-called **future lags**. The idea is that information known in the future can be leveraged to make predictions. Some deterministic features, such as temporal information and holidays (refer to Section 3.2.4), are available for any prediction window in the future, since this information is known. This principle has already been used in [39], where these variables enter the model as known features.

However, not all features are deterministic and therefore known. Variables corresponding to measured values, such as load and RES generation, are examples of this. It has already been mentioned in Section 3.1, that among the available data from ELIA are several forecasts that are highly correlated with the realizations. These can be used to replace the measured values and to construct a new feature vector for each forecast window, which we call the future lag.

Let us denote a future lag vector as  $\xi_t \in \mathbb{R}^p$ . Given our specific data, a future lag vector can be constructed such that it includes all the explanatory variables, or their corresponding forecasts. For the decoder input, we use the matrix  $\mathbf{D}_t$  defined as (4.1) in Section 4.2, extended with future lags. The enriched matrix  $\mathbf{D}_t^\xi \in \mathbb{R}^{r \cdot p + 1}$  is then given by

$$\mathbf{D}_t^\xi = \begin{pmatrix} y_t & y_{t+1} & \dots & y_{t+r-2} & y_{t+r-1} \\ \xi_{t+1} & \xi_{t+2} & \dots & \xi_{t+r-1} & \xi_{t+r} \end{pmatrix}^\text{T} = (\mathbf{D}_t \quad \boldsymbol{\Xi}_t),$$

where we denote

$$\boldsymbol{\Xi}_t = (\xi_{t+1} \quad \xi_{t+2} \quad \dots \quad \xi_{t+r-1} \quad \xi_{t+r})^\text{T}.$$

Matrix  $\mathbf{D}_t^\xi$  is used to train the model. During inference, it must take into account that the matrix  $\mathbf{D}_t^\xi$  is formed sequentially. Similarly to Figure 4.2, the ground truth values are successively replaced by predictions in the matrix  $\hat{\mathbf{D}}_t^\xi$ . The whole process is initialized by the input

$$\hat{\mathbf{D}}_t^1 = \begin{pmatrix} y_t \\ \xi_{t+1} \end{pmatrix}^\text{T}.$$

The predicted value  $\hat{y}_{t+1}$  then forms a new input together with the matrix  $\hat{\mathbf{D}}_t^0$ , which is denoted as

$$\hat{\mathbf{D}}_t^2 = \begin{pmatrix} y_t & \hat{y}_{t+1} \\ \xi_{t+1} & \xi_{t+2} \end{pmatrix}^T.$$

Finally, for prediction steps  $k = 2, \dots, r$ , the input to the decoder consists of

$$\hat{\mathbf{D}}_t^k = \begin{pmatrix} y_t & \hat{y}_{t+1} & \dots & \hat{y}_{t+k-1} \\ \xi_{t+1} & \xi_{t+2} & \dots & \xi_{t+k} \end{pmatrix}^T.$$

In the last step, matrix  $\hat{\mathbf{D}}_t^r$  enters the decoder, which outputs the last predicted value  $\hat{y}_{t+r}$ . This completes the final prediction sequence.

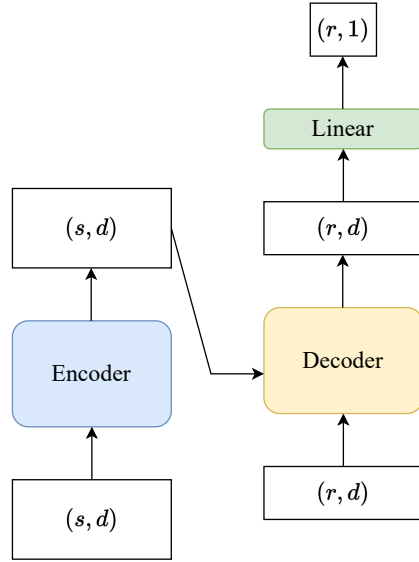


Figure 4.4: Transformer with future lags.

Compared to the vanilla transformer, each prediction step adds additional information in the form of lags to the previous prediction, which can significantly improve the next prediction. The architecture can be viewed in Figure 4.4, which includes the labeling of each data dimension in the model.

# Chapter 5

## Results

The previous chapters laid out the theoretical underpinnings of each model, including their modifications for time series. We also discussed the transformations performed on the data, the feature engineering and the split into training, validation and test sets.

In this chapter, the implemented transformer-based models are compared with benchmark machine learning models. The comparison is based on three commonly used metrics for model evaluation. Let us denote target values  $y_k$  and a predicted values  $\hat{y}_k$  for  $k = 1, \dots, n$ , where  $n$  is a number of observations and  $\bar{y} = \frac{1}{n} \sum_{i=k}^n y_k$ . Then the metrics are defined as follows:

- **Root Mean Square Error (RMSE)**

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

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

- **Mean Absolute Error (MAE)**

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

It represents the average magnitude of the errors in a set of predictions, without considering their direction.

- **R-squared ( $R^2$ )**

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

R-squared quantifies the proportion of the variance in the target variable that is explained by the explanatory variables.

All implemented models were subjected to hyperparameter optimization prior to actual training. The actual tuning was done on the combined training and validation data set. It is not possible to use the standard k-fold method for time series, as it randomly shuffles the data, which would cause data leakage from the future.

Therefore, it is necessary to split the data according to the time sequence and use a time split method. Such a data split is shown in Figure 5.1. The model is trained on a set of variable size, which always represents history. Validation is performed on the fixed length set, which follows the set to be trained. At each fold, the validation set is added to the training set and the model is trained and evaluated again.

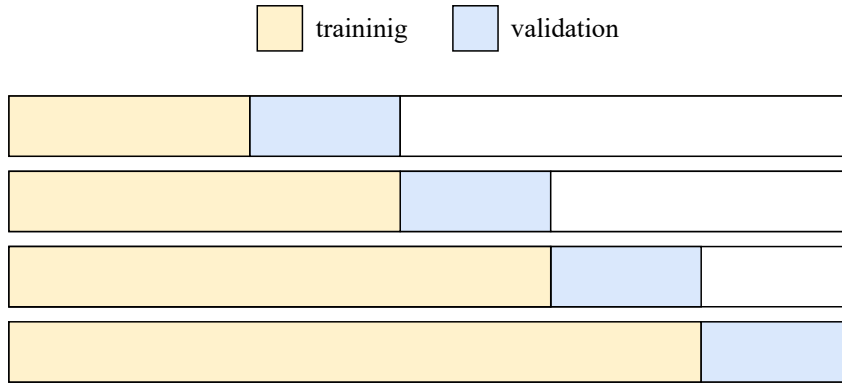


Figure 5.1: An indication of how the time split is performed in k-fold cross validation for hyperparameter tuning. In this thesis, 4 splits were used.

The selection of hyperparameters was conducted through a randomized search, where the parameter grid was explored uniformly without the use of any advanced heuristics. The chosen parameter grid, encompassing all possible combinations of hyperparameters, was based on the tested behavior of the models or aligned with parameters specified in the corresponding original paper. Due to computational complexity, the number of searched parameter combinations was limited for certain models.

In this chapter, we will further concentrate on comparison of the models employed in this thesis. We will present the transformer-based models relevant to this work, encompassing details on hyperparameter tuning and training processes. The benchmark models, used primarily for comparative purposes, will be presented without this additional information. Comprehensive details, including all the source code, are accessible in the GitHub repository [46] corresponding to this thesis.

It is worth mentioning a fundamental difference between the models compared. As mentioned in Chapter 2, ML models in particular, which are not designed for the sequence-to-sequence task, can be applied in two ways. The model can be modified to be multi-output, i.e. it can be constructed to predict the whole sequence. However, this is at the expense of the predictive power of the model. The second way is to construct  $r$  models so that each performs a prediction on a learned time window. The assembly of models thus has a higher prediction accuracy, but it is compensated by a higher computational complexity. This is because each model is trained separately. In this thesis, the latter approach was used, including hyperparameter tuning for each of  $r$  separate models.

Both traditional approaches have a limitation: they cannot predict sequences longer than a pre-defined length  $r$  without training a new model or multiple models. In contrast, transformer-based models are inherently designed to predict entire sequences of any length. They can extend the predicted sequence to  $\tilde{r} > r$  without needing multiple models or retraining. This makes transformers more flexible, as they use just one stand-alone model for varying sequence lengths.

This thesis does not delve into the computational complexity of training of the employed models. However, it is clear that machine learning models generally have lower complexity, making their training computationally less intensive. A future research direction could explore under what conditions the process of selecting hyperparameters and training multiple machine learning models becomes more computationally demanding than tuning and training a single, but more complex, Transformer model. This would help in understanding the trade-offs between model simplicity and computational efficiency.

## 5.1 Conformal prediction intervals

In general case of prediction, whether using classical statistical modeling methods or machine learning and neural networks, only point predictions are obtained. Although the model is optimized on training data to minimize the point prediction error, when validating the model on unseen data, we have no information on the quality of the new prediction.

This problem is addressed by a novel Model Agnostic Prediction Interval Estimator (MAPIE) package [41], which is based on [5, 35, 62]. It includes a variety of methods that provide prediction intervals in addition to the point prediction.

Given  $n_0 < n$ , then index set  $\{1, \dots, n_0\}$  corresponds to training data. The goal is to construct an interval around the point prediction  $\hat{y}_k$  that is likely to contain a ground truth target value  $y_k$ , where  $k \in \{n_0 + 1, \dots, n\}$ . With a significance level  $\alpha$ , we construct prediction interval  $\hat{C}_{n_0, \alpha}(x_k)$  as

$$\mathbb{P} \left\{ y_k \in \hat{C}_{n_0, \alpha}(x_k) \right\} \geq 1 - \alpha,$$

where

$$\hat{C}_{n_0, \alpha} : \mathbb{R}^p \mapsto \mathbb{R}^2.$$

The actual probability, that the predicted value  $\hat{y}_k$  ends up inside the conformal prediction interval, is with respect on both the training data and  $(x_k, y_k)$ .

Various resampling methods used for regression problems, such as those in [5], are valid only under the assumption of the exchangeability hypothesis. This hypothesis allows arbitrary rearrangement of the data, which is not possible in the context of time series, hence invalidating the hypothesis.

Therefore, a method based on the jackknife+-after-bootstrap method [35], known as the Ensemble batch prediction intervals (EnbPI) [62], is proposed for time series. The procedure for constructing prediction intervals involves the use of the bootstrap method and is used as follows:

- The training data are resampled  $K$  times using the block bootstrap method, which involves randomly selecting blocks of the same size and joining them together. We refer the reader to [42] for more details. Thus, we obtain  $K$  bootstraps  $B_1, \dots, B_K$  and their complementary sets  $B_1^C, \dots, B_K^C$ .
- Next,  $K$  models are trained. For each model index  $j$ , where  $j = 1, \dots, K$ , the model is represented as  $\hat{\mu}^j$ .
- For every  $i = 1, \dots, n_0$ , we compute

$$\hat{\mu}_{-i}(x_i) = \phi \left( \left\{ \hat{\mu}^j(x_i) \mid i \in B_j^C \right\} \right),$$

where  $\phi$  represents mean aggregating function. The error of  $i$ -th observation

$$\epsilon_i = y_i - \hat{\mu}_{-i}(x_i)$$

is then calculated.

- Finally, we construct  $\beta$  and  $(1 - \alpha + \beta)$  quantiles as

$$\hat{q}^- = \hat{q}_{n_0, \beta} \{\epsilon_i \mid i \in \{1, \dots, n_0\}\}, \text{ and } \hat{q}^+ = \hat{q}_{n_0, (1-\alpha+\beta)} \{\epsilon_i \mid i \in \{1, \dots, n_0\}\},$$

where  $\beta$  is a parameter, which is being minimized during the training process.

The resulting prediction interval for  $x_k \in \{n_0 + 1, \dots, n\}$  is then defined as

$$\hat{C}_{n_0, \alpha}^{\text{EnbPI}}(x_k) = (\hat{\mu}_\phi(x_k) + \hat{q}^-, \hat{\mu}_\phi(x_k) + \hat{q}^+),$$

where

$$\hat{\mu}_\phi(x_k) = \phi(\{\hat{\mu}^j(x_k) \mid i \in \{1, \dots, n_0\}\}).$$

However, the resulting coverage of the prediction interval for this method is not absolute but asymptotic. Further details of the EnbPI method, can be found in [41, 62].

## 5.2 Model evaluation and comparison

We divide the comparison into two distinct cases. The first case involves the use of models without future lags, where only historical information is used to predict the entire sequence ahead. In this scenario, we compare the Vanilla Transformer and the encoder model against the benchmark models, namely MLP and XGBoost.

The second method involves the use of future lags, introduced in Section 4.4. This approach is implemented for the Transformer with future lags but is not applicable to other transformer-based models. Benchmark models, such as MLP and XGBoost, can be easily modified to take advantage of future lags.

To formally indicate the input data, let us recall the input vector  $e_t \in \mathbb{R}^{(p+1) \cdot s}$  for MLP and XGBoost. If future lags are used for modeling, they are simply added to the input vector. Denoting future lags vector as  $\xi_t \in \mathbb{R}^p$ , the enriched input vector is written as

$$\hat{e}_t^k = \text{concat}(e_t^T, \xi_{t+1}^T, \dots, \xi_{t+k}^T)^T,$$

where the vector dimension of  $\hat{e}_t^k$  is  $(p+1) \cdot s + kp$  and  $k \in \{1, \dots, r\}$  corresponds to the time window on which the model is predicting. Thus, differently large input data are used for each of the models.

For the experiments, the input sequence length  $s$  is set to 32, and the output sequence length  $r$  to 8, corresponding to 8 hours of historical data (with a 15-minute granularity) and predictions for the next 2 hours. Although all time windows are treated equivalently in our analysis, predictions for the first hour, comprising the initial four time windows, are of main interest.

### 5.2.1 Results without future lags

In scenarios where future lags are not utilized, the metrics achieved for each prediction window by models are detailed in Table 5.1. These results are provided for the test set, and all models were optimized using the mean squared error (MSE) loss function.

Step	Model	Metrics		
		RMSE	MAE	R2
1	MLP	109.07	81.07	0.56
	XGBoost	110.36	81.22	0.55
	Transformer	134.11	93.38	0.34
	Encoder	112.31	82.84	0.53
2	MLP	134.10	97.57	0.34
	XGBoost	134.01	97.79	0.34
	Transformer	146.22	103.89	0.21
	Encoder	134.88	97.63	0.33
3	MLP	142.35	103.11	0.25
	XGBoost	143.92	104.02	0.24
	Transformer	152.58	108.56	0.14
	Encoder	144.87	103.59	0.23
4	MLP	148.52	106.92	0.19
	XGBoost	148.85	106.48	0.18
	Transformer	154.37	109.29	0.12
	Encoder	149.28	106.85	0.18
5	MLP	154.58	109.80	0.12
	XGBoost	155.44	110.79	0.11
	Transformer	159.08	112.94	0.07
	Encoder	157.35	113.14	0.09
6	MLP	157.56	111.66	0.08
	XGBoost	159.59	113.23	0.06
	Transformer	160.81	114.26	0.05
	Encoder	158.58	112.74	0.07
7	MLP	158.46	112.12	0.07
	XGBoost	160.82	113.67	0.05
	Transformer	161.72	115.14	0.03
	Encoder	161.53	113.58	0.04
8	MLP	159.73	112.14	0.06
	XGBoost	161.07	113.57	0.04
	Transformer	161.44	114.85	0.04
	Encoder	161.69	115.03	0.03

Table 5.1: Comparison of MLP, XGBoost, Transformer and the Encoder model across different metrics. The models are compared for each time window separately. None of the models use future lags.

Surprisingly, the MLP model emerges as a highly effective option, dominating almost uniformly across all time windows. Both XGBoost and the Encoder yield competitive capabilities, man-

aging to nearly match or even slightly surpass MLP in certain steps. The Vanilla Transformer model, employing a distinct prediction principle discussed earlier, lags behind in performance. Contrary to expectations, the Encoder model, despite leveraging an advanced attention mechanism, did not surpass the performance of the MLP. This outcome challenges the initial assumption that an attention-based model would improve on the MLP on this specific task.

An assessment of performance using specific metrics reveals that the MLP shows absolute dominance in R-squared values. Additionally, MLP model exhibits strong performance in both RMSE and MAE metrics across most steps. However, in step 4, the rest of the models marginally outperform the MLP in terms of MAE. Furthermore, in step 2, XGBoost slightly outperforms MLP in terms of RMSE.

Comparing the XGBoost and Encoder model, the latter dominates in a 7 out of 8 time steps in RMSE. On the other hand, the Encoder model showcases superior performance over XGBoost in steps 1, 3, 4, 6, and 7 concerning MAE. Finally, the Vanilla Transformer level increases only in later time steps, where it outperforms the Encoder model in terms of MAE.

However, a notable trend across all models is a tendency for their predictive accuracy to decline as the forecasting window extends into the future. This decline is evident in the gradual reduction of the R-squared values, suggesting that all models face increasing challenges in maintaining accuracy over longer prediction horizons.

Finally, we provide details of the final parameter grid that was used to find suitable parameters. Table 5.2 shows these values for the Encoder model, where  $d_f$  is the size of the hidden layer in the output dense layer and  $d_d$  is the size of the hidden layer in the encoder and decoder blocks. Other hyperparameters include the number of heads  $h$ , the number of sequential applications of encoder and decoder  $N_x$ , dropout, and learning rate. Subsequently, the resulting hyperparameters are also listed in Table 5.3.

Hyperparam	Possible values
$d_f$	64, 128, 256
$d_d$	256, 512
$N_x$	1, 2, 4
$h$	2
dropout	0.2, 0.4
lr	0.0005, 0.00025

Table 5.2: Grid of hyperparameters for Encoder model.

Hyperparam	1	2	3	4	5	6	7	8
$d_f$	256	128	256	128	256	128	256	128
$d_d$	512	256	512	256	512	256	512	256
$N_x$	1	2	1	2	1	2	1	2
$h$	2	2	2	2	2	2	2	2
dropout	0.2	0.4	0.2	0.4	0.2	0.4	0.2	0.4
lr	0.0005	0.00025	0.0005	0.00025	0.0005	0.00025	0.0005	0.00025

Table 5.3: Selected hyperparameters for the Encoder model for each step ahead.

Similarly to the Encoder model, we present a hyperparameter grid for the Vanilla Transformer. Table 5.4 lists both the examined values and the values selected as the best.



Hyperparam	Possible values	Selected value
$d_f$	128, 256	256
$N_x$	4, 6	6
$h$	2	2
dropout	0.1, 0.25, 0.5	0.25
lr	0.00075, 0.00025	0.00075

Table 5.4: Hyperparameter grid with selected values for Vanilla Transformer.

To illustrate the prediction behavior, we visualize the prediction comparison in Figure 5.2 for each model on the selected date of June 28, 2023, focusing particularly on the differences in their predictions. The encoder model appears to attempt to capture peaks, which is a potential result of slightly higher RMSE. Subsequently, the predictions of MLP and XGBoost are quite similar, which aligns with the calculated metrics.

Most notably, the behavior of the Vanilla Transformer model stands out. For reasons to be determined, its predictions are very cautious, limiting the range of predicted values to the  $(-131.71, 107.67)$  on the whole test set. Therefore, in areas where the real system imbalance curve exhibits extremely high or low values, the Transformer prediction curve shows a constant-like behavior. This pattern suggests a limitation in the ability of this model to accurately predict peak values and shows a conservative approach that keeps the Transformer prediction relatively close to zero compared to other models. Due to this deficiency, it results in higher RMSE errors.

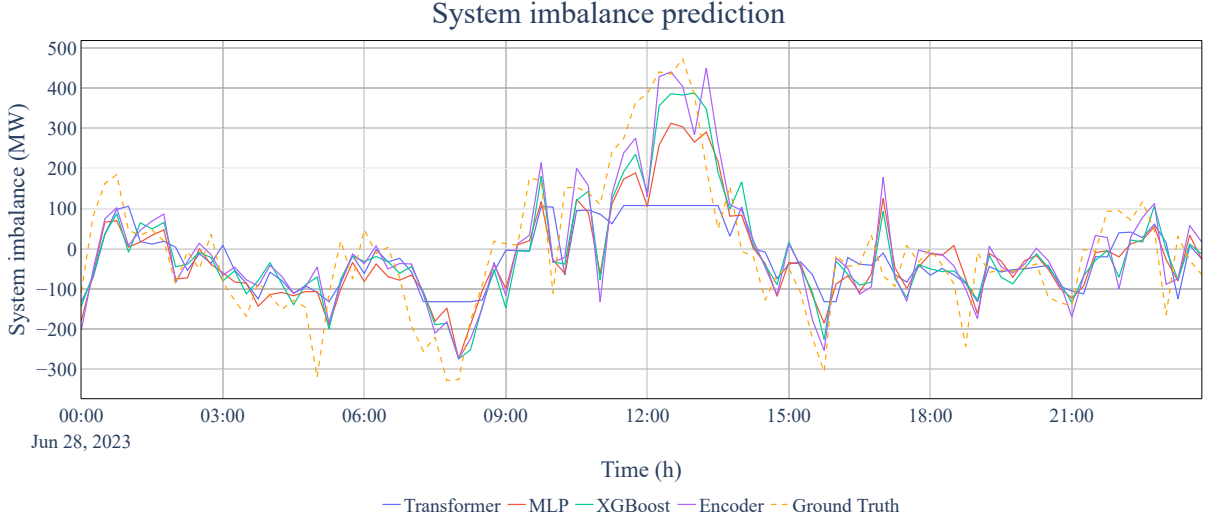


Figure 5.2: Prediction comparison for models without the use of future lags.

Another phenomenon that can be seen on the figure is the delayed peak identification, likely attributable to the high volatility of the data. This delay in peak recognition suggests a lagged response in all the models. Such behavior underscores the complexity of system imbalance forecasting and highlights the need for method that can adapt to this problem and capture these volatile patterns.

### 5.2.2 Results with future lags

Let us now focus on models that incorporate future lags and are able to leverage forecasts and additional information to enhance prediction accuracy. As with the two ML models, the performance of the Transformer is expected to improve. In addition to these models, we also present results for the Temporal Fusion Transformer specifically trained for system imbalance prediction. While the TFT model does not directly implement future lags in its architecture, it uniquely incorporates known future inputs in modeling the target variable. This characteristic justifies its inclusion in our comparative analysis despite its architectural differences.

Step	Model	Metrics		
		RMSE	MAE	R2
1	Transformer	119.11	87.25	0.48
	MLP	109.61	80.98	0.56
	XGBoost	110.36	81.22	0.55
	TFT	116.14	86.12	0.50
2	Transformer	138.79	100.60	0.29
	MLP	132.33	97.05	0.35
	XGBoost	134.03	97.99	0.34
	TFT	136.91	100.18	0.31
3	Transformer	146.70	105.18	0.21
	MLP	141.35	101.87	0.26
	XGBoost	143.57	103.35	0.24
	TFT	146.53	106.54	0.21
4	Transformer	150.38	106.62	0.17
	MLP	149.28	106.55	0.18
	XGBoost	148.63	106.09	0.18
	TFT	152.56	110.27	0.14
5	Transformer	155.96	110.60	0.10
	MLP	153.08	110.06	0.13
	XGBoost	154.47	109.68	0.12
	TFT	157.53	113.64	0.08
6	Transformer	158.54	112.16	0.07
	MLP	155.15	110.28	0.11
	XGBoost	157.96	112.31	0.08
	TFT	160.31	115.31	0.05
7	Transformer	160.07	113.16	0.05
	MLP	155.52	110.78	0.11
	XGBoost	159.69	112.59	0.06
	TFT	162.00	116.20	0.03
8	Transformer	160.69	113.63	0.05
	MLP	155.36	111.32	0.11
	XGBoost	159.53	112.40	0.06
	TFT	162.62	116.42	0.02

Table 5.5: Comparison of MLP, XGBoost, Transformer and TFT model across different metrics. The models are compared for each time window separately. Models incorporate future lags or, in case of TFT, another method to leverage known future variables to refine the prediction.

The metrics for each time window, detailed in Table 5.5, reveal a pattern similar to our previous comparison. Implemented models were optimized using the MSE loss function, whereas quantile loss was utilized to optimize TFT. The MLP model dominates in terms of R-squared across all time steps. As for RMSE, it outperforms other models in all but one time step, which is the fourth time step, where it is marginally surpassed by XGBoost. The latter model achieves better MAE results than MLP in time steps 4 and 5.

Both Transformer and TFT lag behind MLP and XGBoost. However, as the time step increases, they start catching up with the benchmark models. Although TFT performs better than the Transformer in the first three time windows for RMSE and the first two for MAE and R-squared, the Transformer surpasses TFT in all subsequent windows.

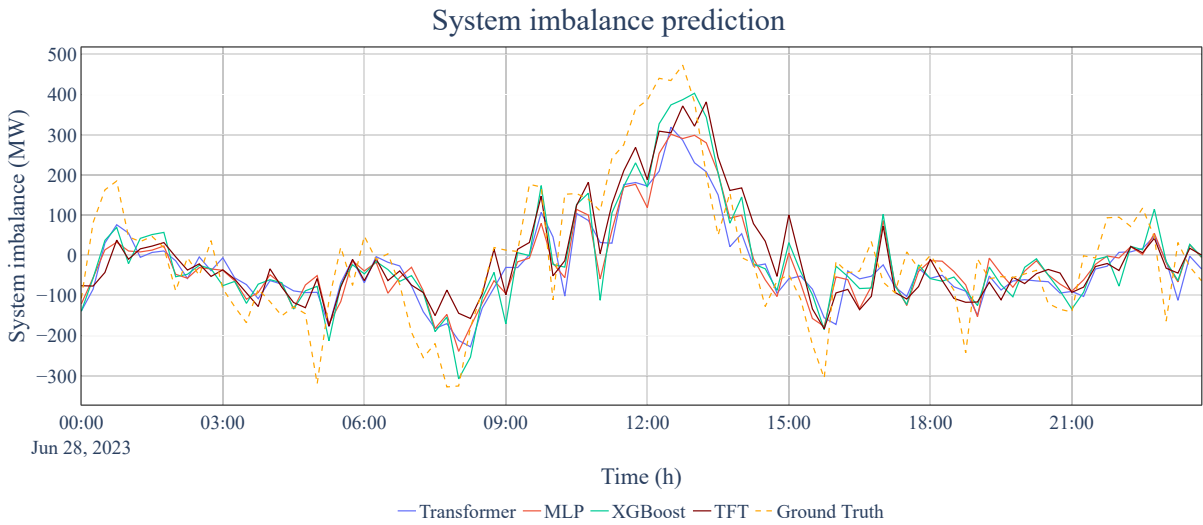


Figure 5.3: Prediction comparison for models with the use of future lags.

The quality of the predictions can be seen in Figure 5.3, where, as in the previous case, we demonstrate the situation for the June 28, 2023. Both MLP and XGBoost exhibit similar behavior as without the use of future lags. The predictions of the TFT model are characterized by changes in the direction of large deviations in an attempt to capture peaks. As for the Transformer, we again obtain more conservative predictions, where the model captures the main trend of the system imbalance, but not the extreme fluctuations.

Contrary to expectations, transformer-based models did not surpass the machine learning benchmarks in our thesis. However, it should be taken into account the reliance of benchmarks on multiple models for prediction, as opposed to the single-model approach of sequence-to-sequence models. Additionally, transformer-based models tend to accumulate errors in prediction. The high volatility and lower autocorrelation of Belgian data present a significant challenge, which is probably one of the aspects why the transformers were outperformed.

Despite falling short of the benchmarks, the Transformer model with future lags did outperform the state-of-the-art TFT model in terms of RMSE and MAE in time windows 4 to 8. Additionally, its conservative forecasts can be particularly relevant in applications where only the direction of the imbalance, i.e., whether the imbalance is positive or negative, is required. In such scenarios, a Transformer with conservative predictions may be a suitable choice.

Across different tasks, XGBoost usually appears to be a superior model to MLP, but this is not

the case in our task. The reason is most likely due to the size of the input data, which is not handled by XGBoost as easily as by MLP. To this end, feature selection could be performed to select only significant variables, but this optimization was not performed for this benchmark model.

To maintain consistency, we provide a grid search performed for the Transformer model with future lags, which is summarized in Table 5.6. Both possible hyperparameter values and those selected during tuning are listed.

Hyperparam	Possible values	Selected value
$d_f$	256, 512, 1024	512
$N_x$	4, 6	6
$h$	2	2
dropout	0.1, 0.2	0.1
lr	0.000075, 0.00025, 0.0005	0.0005

Table 5.6: Hyperparameter grid with selected values for Transformer with future lags.

One of the major contributions of this thesis is certainly the introduction and implementation of future lags. Their effect can be directly compared in the MLP and XGBoost models, which add future lags to the input data vector. Similarly, the effect on the Transformer with future lags and its prediction accuracy compared to the Vanilla Transformer can be observed.

Contrary to expectations, the ML models show no improvement. The error rates remain unchanged, any improvement is negligible and some time windows even show a slight decline. This outcome implies that each model within the ensemble, though optimized independently, may not effectively incorporate the additional variables.

We observe significant improvements for the Transformer, and we report the percentage decrease in terms of RMSE and MAE in Table 5.7. Compared to the ML models, this is most likely due to the use of the attention mechanism, which utilizes associations found between the future lags vectors, subsequently employed in the prediction.

Step	RMSE decrease (%)	MAE decrease (%)
1	11.18	6.56
2	5.08	3.17
3	3.85	3.11
4	2.58	2.44
5	1.96	2.07
6	1.41	1.84
7	1.02	1.72
8	0.46	1.06

Table 5.7: The table shows the percentage decrease in the RMSE and MAE metrics of Transformer with future lags compared to the Vanilla Transformer.

The Transformer model achieves its greatest improvement in the first window, where it exceeds 11 %. However, with subsequent time steps, the rate of improvement diminishes significantly. Since transformer-based models, specifically Transformer with future lags and TFT, are not equivalent to MLP and XGBoost in terms of training. Sequential models optimize the prediction

on  $r$  windows simultaneously, which includes optimization over the entire sequence, in contrast to ML models that optimize for each window separately.

Therefore, both models were re-trained focusing solely on one-step-ahead prediction. The results achieved are shown in Table 5.8 and are compared with the MLP and XGBoost models for the first time window.

Step	Model	Metrics		
		RMSE	MAE	R2
1	Transformer	112.55	83.22	0.53
	MLP	109.61	80.98	0.56
	XGBoost	110.36	81.22	0.55
	TFT	109.61	81.11	0.56

Table 5.8: Comparison of TFT and Transformer trained on one step ahead prediction with benchmarks.

It turns out that when optimized for one-step-ahead prediction, the Transformer significantly approaches the prediction error of both MLP and XGBoost. Under the same conditions, TFT even matched MLP in terms of RMSE. Admittedly, these results are indicative of the capabilities of transformer-based models, although only when predicting one step ahead.

### 5.3 Ensemble

To improve the results even further, an ensemble model approach was proposed with models that use future lags. An ensemble refers to a system that uses the outputs of other models to improve the accuracy of the prediction. The aim is to combine the strengths of various models to improve predictive power. In general, any type of model can be build upon ensemble, simpler models are often preferred.

To maintain objectivity and prevent bias, the ensemble model was trained only on data from the original validation set. Then, it was tested on the original test data to evaluate its performance. For detailed information on ensemble models and their applications, please refer to [10].

Step	RMSE	MAE	R2
1	107.55	79.40	0.57
2	130.07	95.33	0.38
3	140.31	101.49	0.27
4	146.14	104.20	0.21
5	151.17	107.56	0.16
6	154.33	109.40	0.12
7	155.58	110.06	0.11
8	154.70	109.71	0.12

Table 5.9: Performance Metrics for Different Models

Several ensemble models have been proposed in this thesis. In addition to a simple averaging method, linear regression and decision trees [6] have been implemented. In addition to the predictions from the 4 models with future lags (MLP, XGBoost, TFT, and Transformer), selected

features of the input data of original models were added to the ensembles. In the case of decision trees, a grid search was also implemented to find the best hyperparameters.

Eventually, the most effective ensemble model turned out to be a linear model that utilized only the predictions from the original models as predictors. A more complex random forest struggled with overfitting despite hyperparameter optimization. A significant challenge in this context involved the size of the training data for the ensemble models, which matched the size of the test data. This may have caused a poor generalization of the model.

## 5.4 Prediction intervals

As mentioned in the 5.1, in addition to point estimates, prediction intervals can also be obtained. A combined training and test dataset was used to use the EnbPI method. For model training, four block bootstraps were performed, and MLP, XGBoost, and Transformer models were trained, all including future lags. The confidence level was set to  $\alpha = 0.05$ .

Of the MLP and XGBoost models, only the submodels designed for the first time step were used, and the Transformer was specifically trained to predict one step ahead.

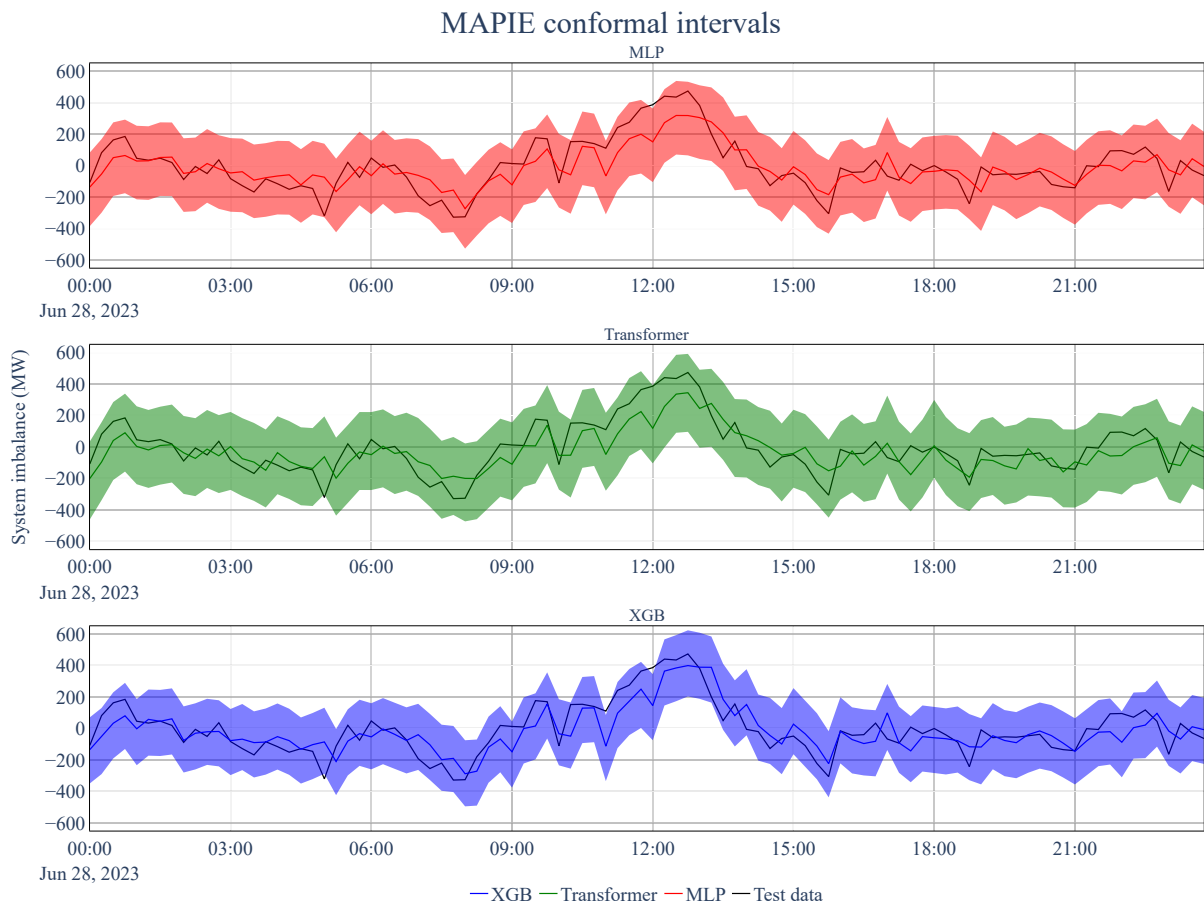


Figure 5.4: Conformal interval predictions for 28th June, 2023.

The results for June 28, 2023, can be seen in Figure 5.4. Next, we need to specify the important global values for each model, namely the width of prediction intervals and the coverage, which

indicates the actual coverage of the predictions made by the intervals. We also report the RMSE achieved on the test set. However, it should be noted that the models were trained on different parts of the joint training and test set, and the EnbPI model used performs predictions using an ensemble of models. Thus, the outputs are not directly comparable to those in the previous sections.

Model	Coverage	Width	RMSE
MLP	0.9577	467.50	108.67
Transformer	0.9633	495.07	112.69
XGBoost	0.9402	419.91	109.32

Table 5.10: Performance metrics for conformal interval predictions.

In Table 5.10, we show the results obtained using the EnbPI method. We can see that although MLP achieves the lowest prediction RMSE, XGBoost shows the narrowest interval width, indicating the most reliable prediction. As for coverage, XGBoost slightly underperforms compared to the set coverage, as it is guaranteed asymptotically.





# Conclusion

Recent transformer-based models, such as the Temporal Fusion Transformer [39] and the Informer [65], illustrate that the attention mechanism can be effectively applied to time series forecasting, achieving state-of-the-art results in certain datasets. In contrast, the findings of several studies, including [18, 64], present a dissenting view, arguing against the efficacy of Transformers and attention mechanism.

In this thesis, our focus is on Transformers and their comparative analysis with benchmark machine learning models, namely MLP and XGBoost. The initial chapter of this thesis presents an overview of the electric transmission system, introduces the concept of system imbalance, and discusses the motivation behind its forecasting. In the second chapter, the theoretical foundations of the Transformer model and its internal structure are detailed. Additionally, the fundamental concepts of benchmark models are presented. The third chapter analyzes the Belgian data utilized in this thesis and describes the preprocessing of this data for its application in the models. The necessary modifications for adapting transformer-based models to time series forecasting are detailed in the subsequent chapter, including the introduction of the specific models proposed in this thesis. The concluding chapter presents results and comparative analysis.

In our research, transformer-based models did not surpass the performance of popular machine learning models, such as MLP and XGBoost, in predicting system imbalance at 8 time steps ahead. However, their prediction metrics were closely comparable to those of XGBoost and MLP. When trained for one-step-ahead prediction, the prediction accuracy of the Transformer approached that of the benchmark models, while the TFT even matched the benchmark prediction error.

An important contribution of this thesis is the incorporation of future lags, which utilize available forecasts and known future data. This approach significantly enhanced the Transformer model, demonstrating its potential effectiveness. Conformal prediction intervals were also calculated to increase the understanding of prediction quality, while provide additional insights into the reliability and accuracy of the point estimates.

There are several possibilities to improve the results obtained. The first of these is certainly the enhancement of the architecture and improving its complexity. Additional possible improvements include enhancing hyperparameter optimization with heuristic algorithms, expanding the data to include more variables or increase the volume of training data.

Time series forecasting still remains an open task with great potential for improvement, especially in the context of rapidly evolving models and methodologies. With the emergence of novel models, such as TimeGPT [29], which utilize transfer learning and large datasets, the question emerges as to whether these models will dominate in time series analysis.



# Bibliography

- [1] S. Ahmed, I. E. Nielsen, A. Tripathi, S. Siddiqui, R. P. Ramachandran, and G. Rasool. Transformers in time-series analysis: A tutorial. *Circuits, Systems, and Signal Processing*, pages 1–34, 2023.
- [2] R. Aïd. *Electricity Derivatives*. Springer International Publishing, 2015.
- [3] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.
- [4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016.
- [5] R. F. Barber, E. J. Candes, A. Ramdas, and R. J. Tibshirani. Predictive inference with the jackknife+, 2020.
- [6] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.
- [7] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall, 1984.
- [8] L. Bretschger and L. Zhang. Nuclear phase-out under stringent climate policies: A dynamic macroeconomic analysis. *The Energy Journal*, 38(1):167–194, 2017.
- [9] J. Browell and C. Gilbert. Predicting electricity imbalance prices and volumes: Capabilities and opportunities. *Energies*, 15, 2022.
- [10] G. Brown. *Ensemble Learning*, pages 312–320. Springer US, Boston, MA, 2010.
- [11] L. Cai, K. Janowicz, G. Mai, B. Yan, and R. Zhu. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Transactions in GIS*, 24(3):736–755, 2020.
- [12] H. Chen, T. N. Cong, W. Yang, C. Tan, Y. Li, and Y. Ding. Progress in electrical energy storage system: A critical review. *Progress in Natural Science*, 19(3):291–312, 2009.
- [13] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.
- [14] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.
- [15] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better mini-batch algorithms via accelerated gradient methods, 2011.

- [16] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks, 2017.
- [17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [18] Y. Dong, J.-B. Cordonnier, and A. Loukas. Attention is not all you need: pure attention loses rank doubly exponentially with depth. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2793–2803. PMLR, 18–24 Jul 2021.
- [19] M. Dwarampudi and N. V. S. Reddy. Effects of padding on lstms and cnns, 2019.
- [20] EEX Group. *EEX Group Annual Report 2022*. European Energy Exchange, 2022.
- [21] Elia. Imbalance Prices per Quarter-Hour (Historical Data). <https://opendata.elia.be/explore/dataset/ods047/information/>, 2023. Accessed: 2023-09-26.
- [22] Elia. Measured and Forecasted Total Load on the Belgian Grid (Historical Data). <https://opendata.elia.be/explore/dataset/ods001/information/>, 2023. Accessed: 2023-09-26.
- [23] Elia. Photovoltaic Power Production Estimation and Forecast on Belgian Grid (Historical). <https://opendata.elia.be/explore/dataset/ods032/information/>, 2023. Accessed: 2023-09-26.
- [24] Elia. Wind Power Production Estimation and Forecast on Belgian Grid (Historical). <https://opendata.elia.be/explore/dataset/ods031/information/>, 2023. Accessed: 2023-09-26.
- [25] Enerdata. Wind & solar share in electricity production data. <https://yearbook.enerdata.net/renewables/wind-solar-share-electricity-production.html>, 2023. Accessed: 2023-11-18.
- [26] European Network of Transmission System Operators for Electricity (ENTSO-E). Member companies, 2023. Accessed: 2023-11-18.
- [27] H. Fei and F. Tan. Bidirectional grid long short-term memory (bigridlstm): A method to address context-sensitivity and vanishing gradient. *Algorithms*, 11(11):172, 2018.
- [28] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*, 29, 2016.
- [29] A. Garza and M. Mergenthaler-Canseco. Timegpt-1, 2023.
- [30] Global Wind Energy Council. Global wind report 2023. <https://gwec.net/globalwindreport2023/>, 2023. Accessed: 2023-11-18.
- [31] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning: Adaptive Computation and Machine Learning series*. The MIT Press, 2016.
- [32] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O’Reilly Media, Sebastopol, CA, 2019.

- [33] J. D. Hamilton. *Time Series Analysis*. Princeton University Press, 2020.
- [34] A. Kazemnejad, I. Padhi, K. N. Ramamurthy, P. Das, and S. Reddy. The impact of positional encoding on length generalization in transformers, 2023.
- [35] B. Kim, C. Xu, and R. F. Barber. Predictive inference is free with the jackknife+-after-bootstrap, 2020.
- [36] Y. Kim, C. Denton, L. Hoang, and A. M. Rush. Structured attention networks, 2017.
- [37] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966. PMLR, 2015.
- [38] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.
- [39] B. Lim, S. O. Arik, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv preprint*, 2019.
- [40] S. Luetto, F. Garuti, E. Sangineto, L. Forni, and R. Cucchiara. One transformer for all time series: Representing and training with time-dependent heterogeneous tabular data, 2023.
- [41] MAPIE contributors. MAPIE: Model Agnostic Prediction Interval Estimator. <https://mapie.readthedocs.io/en/latest/index.html>, 2023. Accessed: 2023-12-15.
- [42] S. Mignani and R. Rosa. The moving block bootstrap to assess the accuracy of statistical estimates in ising model simulations. *Computer Physics Communications*, 92(2):203–213, 1995.
- [43] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.
- [44] M. Montel and A. Yakovets. holidays. <https://pypi.org/project/holidays/>, 2023. Accessed: 2023-12-15.
- [45] M. A. Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.
- [46] V. Obhlidal. System imbalance forecasting. <https://github.com/obhlivoj/System-Imbalance-Forecasting>, 2024.
- [47] I. Padhi, Y. Schiff, I. Melnyk, M. Rigotti, Y. Mroueh, P. Dognin, J. Ross, R. Nair, and E. Altman. Tabular transformers for modeling multivariate time series. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3565–3569. IEEE, 2021.
- [48] O. Press and L. Wolf. Using the output embedding to improve language models, 2017.
- [49] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

- [50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation, 1985.
- [51] A. Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, March 2020.
- [52] S. Siami-Namini, N. Tavakoli, and A. S. Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401. IEEE, 2018.
- [53] SolarPower Europe. Global market outlook for solar power 2023 - 2027. SolarPower Europe, 2023. Accessed: 2023-11-18.
- [54] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.
- [55] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey, 2022.
- [56] U. Ugurlu, I. Oksuz, and O. Tas. Electricity price forecasting using recurrent neural networks. *Energies*, 11(5):1255, 2018.
- [57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [58] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka. A multi-horizon quantile recurrent forecaster, 2018.
- [59] World Nuclear Association. World nuclear performance report 2023. World Nuclear Association, 2023. Accessed: 2023-11-18.
- [60] H. Wu, J. Xu, J. Wang, and M. Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.
- [61] N. Wu, B. Green, X. Ben, and S. O’Banion. Deep transformer models for time series forecasting: The influenza prevalence case, 2020.
- [62] C. Xu and Y. Xie. Conformal prediction for time series, 2023.
- [63] W. Yin, K. Kann, M. Yu, and H. Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.
- [64] A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting?, 2022.
- [65] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.