MASTER THESIS

Comparison of Frameworks for Time Series Forecasting

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science (M.Sc.)

Supervisor: Prof. Dr. Martin Spindler

Submitted by: Aziz Abdoul Nabi

[redacted]

[redacted]

Master of Science in Business Administration

[redacted]

Aziz.abdoulnabi@gmail.com

Issuing Date: 15.04.2024

Submission Date: 15.10.2024

Table of Contents

List of Figures III

List of Tables IV

List of Abbreviations V

1 Introduction 1

2 Literature Review 3

2.1 Local and global models 4

2.2 Hierarchical global models 6

3 Forecasting models and evaluation metrices 12

3.1 Traditional statistical models 13

3.1.1 ARIMA 13

3.1.2 Exponential smoothing 15

3.2 Machine Learning models 17

3.3 Deep learning models 20

3.3.1 Recurrent-based models 24

3.3.2 CNN-based models 27

3.3.3 MLP-based models 30

3.3.4 Transformer-based models 33

4 Empirical application 39

4.1 Local and global models (M4 dataset) 39

4.1.1 Constructed dataset 40

4.1.2 Data preprocessing and engineering 41

4.2 Hierarchical forecasting (M5 dataset) 44

4.2.1 Constructed dataset 45

4.2.2 Data preprocessing and engineering 47
4.3 Model selection 49
4.4 Experimental setup and tuning 52
4.5 Evaluation metrics 53
5 Results and analysis 55
5.1 Local and global models (M4 dataset) 55
5.2 Hierarchical forecasting (M5 dataset) 58
5.3 Model comparison and discussion 61
6 Conclusion and future work 64
Appendix 65
Reference List 71

List of Figures

Figure 1 Simple Hierarchical Structure. 8
Figure 2 Decision Tree Structure. 18
Figure 3 XGBoost and LightGBM Tree Growth. 19
Figure 4 A simple MLP with one hidden layer. 21
Figure 5 Neural Network Optimization Problem. 22
Figure 6 Bias-Variance Trade-off in Neural Networks. 23
Figure 7 RNN Architecture 25
Figure 8 LSTM Architecture 26
Figure 9 TimesNet Function Overview. 29
Figure 12 NHITS Frequency and Signal Capture 32
Figure 13 NHITS Architecture 33
Figure 14 Transformer Architecture with Multi-head Attention. 36
Figure 15 PatchTST Data Processing. 39
Figure 16 First 5 Time Series in the M4-Dataset. 40
Figure 17 Data Ranges Before Normalization 41
Figure 18 Autocorrelation and Partial Autocorrelation of M4 Dataset 43
Figure 19 Components for Randomly Selected Time Series 44
Figure 20 Hierarchical Structure of Constructed M5 Dataset. 46
Figure 19 Plot of sales on the country and state levels. 47
Figure 20 Autocorrelation and Partial Autocorrelation of M5 Data 48
Figure 21 Holiday and Event-related sales spikes. 49
Figure 22 Best-performing Hybrid Model 58

List of Tables

Table 1 Overview of M5 Dataset.....45
Table 2 Summary of the data used in my case studies. 51
Table 3 Results of local and global forecasting models. 57
Table 4 Out-of-Sample MAPE for Hierarchical Forecasting Approaches 60

List of Abbreviations

Abbreviation/Variable	Definition
ANN	Artificial Neural Network
ARIMA	Auto Regressive Integrated Moving Average
CNN	Convolutional Neural Network
DL	Deep Learning
FFN	Feedforward Neural Network
KAN	Kolmogorov-Arnold Networks
LSTM	Long Short-Term Memory
LightGBM	Light Gradient Boosting
MAE	Mean Absolut Error
MASE	Mean Absolute Scaled Error
MAPE	Mean Absolute Percentage Error
MSE	Mean Squared Error
ML	Machine Learning
MLP	Multilayer Perceptron
NBEATS	Neural basis expansion analysis for interpretable time series
NHITS	Neural Hierarchical Interpolation for Time Series
OLS	Ordinary Least Squares
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SOTA	State of the art
TSF	Time Series Forecasting
XGBoost	Extreme Gradient Boosting

1 Introduction

The idea of foreseeing the future is captivating; however, forecasting is not about predicting with absolute certainty but rather about providing insights into potential outcomes. This glimpse into what might unfold enables more informed planning and decision-making. Time Series Forecasting (TSF) involves predicting future values based on historical data recorded at regular intervals, such as hourly, daily, or monthly (Box George and Jenkins Gwilym (1976). By analyzing past patterns and trends, TSF has become a fundamental tool across fields where temporal dynamics shape key decisions, including finance, economics, meteorology, and engineering. This process leverages historical data to anticipate future trends, making it invaluable for strategic planning and operational efficiency.

The purpose of this study is to offer an overview of various state-of-the-art methods and models in the domain of time series forecasting. This is accomplished by using two real-world datasets from the M forecasting competition series as benchmarks: daily finance data from the M4 competition and daily sales data from the M5 competition[1]. The first one is daily finance data from the M4 competition and the second is daily sales data from the M5 competition. For each approach I compare multiple algorithms ranging from simple statistical models to sophisticated deep learning neural networks. I compare multiple algorithms within each approach, spanning from simple statistical models to advanced deep learning neural networks. Additionally, I examine the differences between two distinct forecasting approaches: the recursive approach, commonly used in autoregressive models, and the direct approach, frequently utilized in machine learning and deep learning frameworks. In the recursive approach, each predicted value is used as an input for the subsequent prediction. In contrast, the direct multi-step approach aims to predict

[1] The M-Competitions are organized by Professor Makridakis. For more information, see https://www.unic.ac.cy/iff/research/forecasting/m-competitions/

values at specified future time points, with each prediction generated by a separate model trained by a machine learning or deep learning algorithm.

The first attempt at a time series forecasting model is introduced by Yule (1927) through his work on autoregressive (AR) models for studying natural phenomena, such as sunspot numbers. In the 1970s, Box and Jenkins (1976)expand on the AR model by incorporating additional components and developing the Autoregressive Integrated Moving Average (ARIMA) model. These models assume that future values of a time series can be expressed as a linear combination of past values and stochastic error terms, making them suitable for handling non-stationary data through differencing techniques. In the following decade, further advancements were made with the introduction of Vector Autoregressive (VAR) models, which allow for the prediction of multivariate time series, which involves data with multiple interdependent variables, such as inflation and interest rates (Sims, 1980).

However, this study primarily focuses on univariate methods for TSF. Machine Learning (ML) models are not inherently designed for TSF but can be effective when trained on relevant variables that capture temporal structure, achieved through feature engineering (Bontempi et al., 2013). In contrast, Deep Learning (DL) models are based on neural network architectures, often specifically designed for handling a certain type of data, like sequential data. This structure enables them to automatically identify relevant information from raw data, reducing the need for extensive feature engineering (Lim and Zohren, 2021). This work demonstrates the strengths and limitations of each of these techniques, illustrating how the best model choice is influenced by the structure of the data and the specific forecasting problem.

This paper is structured as follows: First, I explain the difference between local and global forecasting models, detailing how local models are typically trained on individual time series, while global models leverage multiple time series simultaneously to capture shared patterns and dependencies.

In the literature review section, I discuss the advantages and limitations of each approach, providing a theoretical understanding of the function of global TSF in data with and without hierarchical structure. Next, I provide a comprehensive overview of the various forecasting architectures considered in this work, beginning with traditional statistical models such as ARIMA and exponential smoothing. I then proceed to more advanced non-linear ML models and state-of-the-art (SOTA) DL models, with a significant portion of Section 3 dedicated to exploring various DL architectures suitable for TSF.

Following this theoretical foundation, I describe the experimental setup and datasets used for benchmarking. In Section 4, I provide an in-depth exploration of the datasets, focusing on relevant components such as seasonality, essential for feature engineering and model selection. I then discuss the rationale for selecting specific models for each dataset and the matrices used to evaluate model performance. Additionally, I outline my hyperparameter tuning strategy for model optimization. In Section 5, I present the results of both case studies, discussing how they align with existing literature on time series forecasting and the outcomes of the M4 and M5 competitions. While statistical models, as demonstrated in the M4 competition, can still offer robust performance, ensemble models can surpass them (Makridakis et al., 2018). While SOTA DL models demonstrate superior performance on the M5 dataset, this comes at a computational cost, making it a trade-off in most cases. With proper feature engineering, LightGBM can achieve competitive performance, while maintaining lower training time than most DL models, except for NBETAS, which is overall the best model in my M5 benchmarking. My findings are consistent with the M5 competition's results (Makridakis et al., 2022). Finally, I summarize my insights and outline potential directions for future research.

2 Literature Review

In this section, I explain the distinctions between local and global forecasting approaches, illustrating their advantages and limitations. Additionally, I highlight

the role of hierarchical structures in time series forecasting, where a global model effectively captures shared patterns across time series at different levels of the hierarchy, ensuring coherent predictions through reconciliation techniques.

2.1 Local and global models

Given the exponential growth of collected data in the last decade, the need for more scalable and efficient tools to analyze this data has increased. Recently, a new approach for TSF has gained significant attention, especially in TSF competitions. This approach suggests that, given the similarity of time series in a single domain, for instance, the time series of multiple stores within the same supermarket chain, one global model can be trained on all these similar time series, rather than training a separate model for each series. By leveraging a single model for all related series, the model captures characteristics common to these series, which can enhance performance and improve generalization, while also reducing computational costs (Montero-Manso et al., 2020).

For businesses and financial institutions, the ability to forecast future trends using historical data is crucial for decision-making. Traditionally, this has been done using local models, where a separate forecasting model is built for each individual time series. While effective in capturing specific patterns of each series, this approach becomes increasingly resource-intensive and impractical as the number of time series grows into the thousands or even millions (Akiba et al., 2019).

Scalability challenges in local modeling have led to increased interest in global models. Unlike local models, which are tailored to each individual time series, global models train on collections of related series, allowing them to leverage shared patterns. For example, in finance, a global model might be trained across multiple stocks within the same industry, while in retail, it could span various products in a category. This approach enables global models to generalize more effectively, often achieving robust performance on individual series, especially when data is sparse or noisy (Montero-Manso and Hyndman, 2021).

Research shows that global models frequently outperform local models, especially when the time series share similar seasonal or trend components. Studies by Montero-Manso et al. (2020); Montero-Manso and Hyndman (2021) demonstrate that global models achieve superior accuracy in large-scale forecasting tasks. By pooling information across multiple time series, global models can capture underlying structures that may not be evident when each series is modeled independently, resulting in improved predictive performance.

Moreover, global models offer significant computational advantages. By consolidating the forecasting process into a single model, they reduce the overhead associated with maintaining and updating thousands of separate models, as would be required in a local modeling approach. This simplification not only streamlines the modeling process but also allows for more efficient use of computational resources. Recent advancements in machine learning, particularly in deep learning, further enhance the capabilities of global models. Methods such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are particularly well-suited for capturing temporal dependencies in time series data across multiple series simultaneously (Bandara et al.,2020).

The literature also highlights the benefits of global models in terms of generalization and transfer learning. For instance, Salinas et al., 2020) demonstrate that their model, DeepAR, improves performance by training on multiple related time series, allowing knowledge transfer from one series to another, particularly valuable in cases with limited data. This transfer learning capability makes global models robust in scenarios where individual series lack sufficient data for training reliable local models. Additionally, global models like N-BEATS, proposed by Oreshkin et al. (2019) show that DL architecture can achieve high performance across diverse datasets by leveraging shared features and patterns across multiple time series.

However, global models are not without their challenges. One of the primary concerns is the potential loss of specificity; while global models are powerful in capturing general patterns, they might overlook unique characteristics of individual series that a local model would capture. This trade-off between generalization and specificity is a key consideration when choosing between local and global approaches. Another challenge is the complexity of designing and training global models, which often require more sophisticated techniques and a deeper understanding of the relationships among the time series (Montero-Manso and Hyndman, 2021).

Recent studies have also explored hybrid approaches that combine the strengths of both local and global models. These hybrid models aim to leverage the generalization ability of global models while retaining the specificity of local models. For example, in his work on the M4 forecasting competition, Smyl (2020), demonstrates that combining local and global approaches can lead to significant improvements in forecast accuracy, particularly in challenging datasets where both unique and common patterns exist.

In conclusion, the choice between local and global models largely depends on the specific context and objectives of the forecasting task. While local models offer precision tailored to individual series, global models provide scalability and robustness, especially in environments characterized by large numbers of similar time series. Furthermore, the next section explores hierarchical forecasting, which builds on the principles of global TSF by structuring forecasts across different levels in a hierarchy. This approach leverages global models to ensure coherent predictions across aggregated and disaggregated series, enhancing both accuracy and consistency in multi-level forecasting.

2.2 Hierarchical global models

Companies typically organize their data in a hierarchical structure with different levels, such as global, regional, and local levels or divisional categories like store

and department. For example, in retail, overall sales forecasts can help predict individual product category sales, as demonstrated by Athanasopoulos et al. (2009). However, this approach might miss unique patterns at lower levels, potentially leading to inaccuracies. When forecasting time series within a hierarchical structure, it is crucial to maintain coherence across levels. Specifically, the forecasts for lower-level time series should collectively align with the aggregate forecast at higher levels, a requirement known as "aggregate consistency" (Wickramasuriya et al., 2015). The process of adjusting forecasts to achieve this alignment is referred to as forecast reconciliation (Wickramasuriya et al., 2019). Hierarchical reconciliation methods offer several advantages, ensuring alignment between forecasts at lower and higher levels, which is particularly valuable in organizational settings where various departments or product lines must adhere to strategic goals. Additionally, these methods can integrate multiple information sources, improving accuracy across aggregation levels (Hyndman et al., 2011);Athanasopoulos et al., 2009). To illustrate, I use an example from Hyndman and Athanasopoulos (2018) depicting a three-level hierarchy. The top level contains the most aggregate sum of the data. The top level contains the total aggregate, followed by two series (A and B), each aggregating sublevels (AA, AB, AC for A; BA, BB for B). The total number of series is n= 8, with has m=*5* time series at the bottom level.

In the following, I present the most widely used reconciliation methods, as explained in Hyndman and Athanasopoulos (2018).

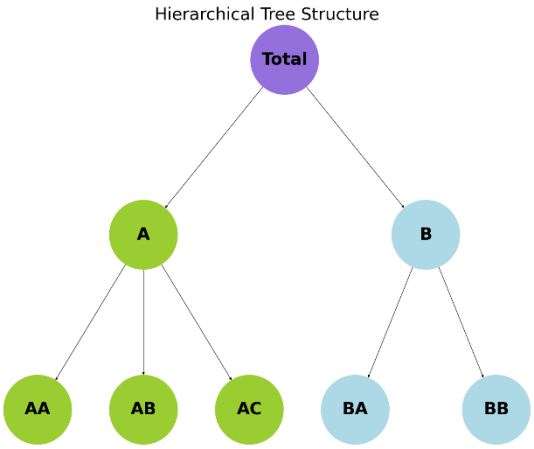


Figure 1 Simple Hierarchical Structure.

Source: Own representation based on Hyndman and Athanasopoulos (2018). Chapter: 11.1 Hierarchical and grouped time series

The bottom-up approach

This approach involves aggregating forecasts from individual series to produce predictions at higher levels. It captures specific patterns in each series and has been successfully applied in fields like telecommunications demand forecasting (Hyndman et al., 2011). While the bottom-up approach can achieve high accuracy at lower levels, it may encounter challenges such as data sparsity and high variance, particularly with volatile series.

The top-down approach

In this approach, forecasting begins with the total aggregated data, which is then broken down into more detailed levels. The total forecast is disaggregated by proportions through the hierarchy. Three primary methods exist for calculating these proportions:

- Average historical proportions: To determine the proportion p_j of each lower-level series, j=1, …., *m*, the average historical share of each bottom-level time series $y_{j,t}$ relative to the total aggregate y_t over the period $t = 1, \ldots, T$ is calculated as follows:

$$p_j = \frac{1}{T}\sum_{t=1}^{T}\frac{y_{j,t}}{y_t}\ , \tag{1}$$

- Proportions of the historical averages:

  A slightly different approach is to calculate the proportion by dividing the average of a bottom level series by the average of the total aggregate.

$$p_j = \sum_{t=1}^{T}\frac{y_{i,t}}{T} \Big/ \sum_{t=1}^{T}\frac{y_t}{T}\ , \tag{2}$$

  The simplicity of these approaches comes at cost, namely the loss of information due to aggregation (Hyndman and Athanasopoulos, 2018).

- Forecast proportions.

  Since time series change over time, their proportions may also vary. However, the historical methods do not account for this. To address this issue Athanasopoulos et al. (2009) suggest using forecasts to calculate proportions.

Due to the hierarchical structure of the data, reconciliation can be performed on an intermediate level. The middle-out approach is a hybrid technique that adjusts forecasts both upwards and downwards, balancing detail and overall information. This makes it suitable for situations with both stable and volatile segments. Forecasts are first calculated at a middle level, then aggregated upwards and disaggregated downwards.

However, these methods also have drawbacks. The top-down approach can overlook local nuances, leading to less accurate forecasts at lower levels, while the bottom-up approach may suffer from data sparsity and high variability when there are many series or when the series are volatile (Moreover, the computational complexity of hierarchical models can pose a barrier, requiring significant resources to be implemented effectively, especially with large and complex datasets.

This hierarchical structure can be represented in matrix notation as follows:

$$\begin{bmatrix} y_t \\ y_{A,t} \\ y_{B,t} \\ y_{AA,t} \\ y_{AB,t} \\ y_{AC,t} \\ y_{BA,t} \\ y_{BB,t} \\ y_{BY,t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{AA,t} \\ y_{AB,t} \\ y_{AC,t} \\ y_{BA,t} \\ y_{BB,t} \end{bmatrix} \Leftrightarrow y_t = Sb_t, \quad (3)$$

Here, the vector y_t includes all observations across each level in the hierarchy at time *t*, the summing matrix is represented by S with $n \times m$ dimensions and the observations in the bottom level of the hierarchy are in the m-dimensional vector *b.*

For a forecast horizon *h,* the base forecast without any regard to aggregation or grouping can be denoted by $\hat{y}_h$. The following expression shows how coherence forecasts can be achieved using coherence reconciliation approaches:

$$\tilde{y}_h = SG\hat{y}_h \quad (4)$$

Where $m \times n$ G is a matrix defined according to the approach used, for instance when applying a bottom-up approach the time series in the bottom level are set to 1 in the matrix, as follows:

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (5)$$

Classical models are limited to the use of information from a single level, since they are using 1 and 0 for a specific reconciliation approach (bottom-up or top down). Wickramasuriya et al. (2019) addressed this issue by introducing the Minimum Trace (MinT) optimal approach. By leveraging information from multiple levels a matrix G can be found to minimize the total forecast variance of the reconciled forecasts, thus achieving coherence while increasing the accuracy. Hyndman and Athanasopoulos (2018) suggest multiple methods that can be used to find the matrix G:

1. Ordinary Least Squares (OLS): Assuming equal variance across levels, an OLS model can determine the matrix G to minimize the variance between base forecasts and coherent forecasts.
2. Weighted Least Squares (WLS-OLS): When assuming different variances across the levels, these variances can be scaled by the base forecasts, then these weighted differences can be minimized using OLS.
3. Weighted Structure: If residuals are unavailable and variances cannot be calculated, the matrix G can instead be derived using the aggregation structure as a proxy for actual data.

In comparison, hierarchical forecasting methods and ML approaches each have unique strengths and limitations in time series prediction. Hierarchical methods are particularly effective when preserving structural relationships within data is essential, such as ensuring aggregate consistency across organizational levels. They are often more interpretable, making them suitable for settings where understanding the underlying process is as important as the forecast itself.

Machine and deep learning models, can capture complex nonlinear relationships and interactions within data, potentially resulting in higher predictive accuracy, particularly when dealing with large datasets (Bandara et al., 2020). However, these models often require substantial training data to perform well and may struggle with interpretability and maintaining hierarchical consistency unless additional

adjustments are applied (Ben Taieb et al., 2017). The choice between hierarchical and ML approaches often depends on the specific forecasting context, such as data availability, required accuracy, and the need for interpretability.

To effectively implement hierarchical forecasting in real-world applications, several best practices should be considered. First, it is crucial to understand the hierarchical structure of the data thoroughly, as this understanding will guide the selection of the most appropriate forecasting method, whether it be top-down, bottom-up, or middle-out (Hyndman et al., 2011). Organizations should leverage software and tools that support hierarchical forecasting to handle large data volumes and complex calculations efficiently.

Athanasopoulos et al. (2009) recommend combining methods to optimize performance, such as using a hybrid approach that integrates top-down and bottom-up forecasts. Continuous validation of the models with new data and regular revision is necessary to adapt to changing conditions or patterns (Ben Taieb et al., 2017). This involves back testing and comparing the performance of different models to ensure they provide accurate and actionable forecasts. Additionally, involving stakeholders in the process can improve model acceptance and usefulness by ensuring forecasts align with organizational needs and constraints.

Overall, hierarchical forecasting methods offer valuable frameworks for multi-level prediction, but their application requires careful consideration of the data structure and operational needs. As forecasting becomes increasingly critical due to complex global supply chains and multi-channel marketing strategies, understanding and implementing these methods will be essential for accurate and actionable predictions.

3 Forecasting models and evaluation metrices

The forecasting models used in this study are divided into two primary categories: linear statistical models and non-linear machine learning (ML) models.

In the following subsections I provide an overview of the most popular statistical and ML algorithms used for forecasting problems. The overview begins with well-established statistical autoregressive models, followed by a discussion on how advances in computing power have enabled the use of more sophisticated models.

3.1 Traditional statistical models

In the following I present the well-established autoregressive statistical models.

3.1.1 ARIMA

One of the most well-known statistical models for time series forecasting is the Autoregressive Integrated Moving Average (ARIMA), introduced in the 1970s by Box and Jenkins (1976). It consists of three parts: An autoregressive and moving average parts.

The autoregressive (AR) component captures dependencies between lagged and current values, while differencing (I) is used to eliminate trends or seasonality, achieving stationarity[2]. Lastly, the moving average is the average for a window of time and is used to smooth out short-term fluctuations, hence explaining sudden shocks.

These three components incorporate three parameters:

1. AR(p): The number of lag observations used in the model is represented by p.
2. I(d): The number of times row observations are differentiated, is represented by d.
3. MA(q): The size of the moving average window is represented by q.

Thus, an ARIMA(5,1,10) model uses 5 lagged data values, applies first-order differencing to achieve stationarity, and incorporates the last 10 lagged forecast

[2] The process remains in statistical equilibrium with probabilistic properties that do not change over time.

errors in the moving average component. The model incorporates the previous 10 forecast errors to adjust the prediction.

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right)(1 - L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right)\varepsilon_t, \quad (6)$$

Where:

X_t: The time series at time *t.*

ϕ_i: Coefficients of the autoregressive (AR) terms

p: Number of autoregressive terms.

L: Lag operator, where $LX_t = X_{t-1}$.

d: Number of differences for stationarity.

θ_i: Coefficients of the moving average (MA) terms.

q: The number of moving averages.

ε_t: Error term (white noise).

ARIMA requires parameter specification for each time series, which can be time-consuming with large datasets; therefore, I use an automated version, Auto-ARIMA.

Auto-ARIMA automates the selection of the optimal parameters (p, d, q) for ARIMA and their seasonal counterparts (P, D, Q) if necessary. This is achieved by iterating through different combinations of these parameters, fitting the model and evaluating the quality of each set using a predefined criterion. Akaike Information Criterion (AIC) is a commonly[3] used criterion in Auto-ARIMA. AIC aims at balancing the complexity and the fit of the model. The model with the lowest model is selected.

$$AIC = 2k - 2\ln(L), \quad (7)$$

[3] See chapter 8.6 of Hyndman, R. J., and G. Athanasopoulos, 2018. *Forecasting: Principles and practice* (OTexts).for more details.

Where:

k is the number of parameters which is a proxy for the model's complexity

L the likelihood function of the model.

The AIC balances goodness of fit, represented by L, with a penalty for excessive parameters to prevent overfitting.

In the following section, I explain a commonly used statistical method, which I use as a baseline model additional to ARIMA. Establishing a solid baseline is essential for determining whether a more complex approach is beneficial.

3.1.2 Exponential smoothing

Exponential Smoothing is a time series forecasting method that applies weighted averages to past observations, with weights that decrease exponentially over time. Brown (2004) introduces a simple exponential smoothing (SES) for discrete data which solely smooths the level making it a suitable approach for data without a clear trend or seasonality. This method is later extended by (Holt, 2004)[4] who incorporates a trend component. Winters (1960) extends this method to incorporate a seasonal component.

This approach, now known as the Holt-Winters method, includes three components: level, trend, and seasonality. The additive version of the Holt-Winters method is expressed as follows:

Forecast equation:

$$\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t+h-m}, \tag{8}$$

Level smoothing:

$$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}), \tag{9}$$

[4] The paper was originally published in 1957.

Trend smoothing:

$$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1 - \beta^*) + b_{t-1}\,, \tag{10}$$

Seasonality smoothing:

$$s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1 - \gamma)s_{t-m}, \tag{11}$$

Where ℓ_t denotes an estimate of the level of the data at time *t*. For this estimation a smoothing parameter α is used, $0 \leq \alpha \leq 1$. The slop b_t in the forecasting equation denotes an estimate of the trend of the data at time *t*. Again, a smoothing parameter β^* is used to estimate the trend, $0 \leq \beta^* \leq 1$. For the smoothing factor s_t a parameter is m is used the frequency of seasonality in year, corresponding to 12 for monthly data with a monthly seasonality or 4 for quartal seasonality.

The seasonality component can be represented as follows:

$$s_t = \gamma * (y_t - \ell_t) + (1 - \gamma *)s_{t-m}, \tag{12}$$

With $\gamma *$ the smoothing seasonality parameter, $0 \leq \gamma * \leq 1$. (Hyndman and Athanasopoulos, 2018).

In the multiplicative version of the Holt-Winters method, the seasonality component is multiplied rather than added. This is used when seasonal variations are proportional to the level of the series.

Forecast equation:

$$\hat{y}_{t+h|t} = (\ell_t + hb_t) \times s_{t+h-m}, \tag{13}$$

Level smoothing:

$$\ell_t = \alpha \frac{y_t}{s_{t-m}} + (1 - \alpha)(\ell_{t-1} + b_{t-1}), \tag{14}$$

Trend smoothing:

$$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1 - \beta^*) + b_{t-1}\,, \tag{15}$$

Seasonality smoothing:

$$s_t = \gamma \frac{y_t}{(\ell_{t-1} - b_{t-1})} + (1-\gamma)s_{t-m}, \quad (16)$$

In this multiplicative approach, the seasonal indices are relative factors that scale the forecast up or down, depending on the time of year. This method is particularly effective when the amplitude of seasonal fluctuations changes with the level of the time series.

3.2 Machine Learning models

In the context of machine learning (ML), forecasting is framed as a type of regression problem. There exists a wide range of models that can be considered for this problem, from a simple linear regression to more complex tree-based models. Decision trees split data into subsets based on input features, forming a tree-like structure that can be applied to both classification and regression problems (Breiman et al., 1984). This approach aims to capture complex patterns in the data by splitting the data into subcategories before making the prediction based on the splits. After a certain point further splitting of the data results in learning noise and random fluctuations, thus reducing its ability to generalize to unseen data and leading to a high variance.

One of the earliest methods to address overfitting (see Section 3.3 for more details) is bootstrap aggregation, or bagging, introduced by Breiman (1996). Bagging works by growing multiple deep trees independently and training them on a randomly selected subsample chosen from the data with replacement. Depending on the problem type a voting or averaging of the predictions is done, hence yielding in improved accuracy and stability by reducing the variance. Breiman (2001) extends this technique to build an ensemble of uncorrelated decision trees, each grown from a randomly selected subset of features.

Boosting addresses the issue of underfitting by training a sequence of weak learners, each iteration focusing on correcting the errors of the previous model (Schapire, 1990).

Freund and Schapire (1996) extends this technique to allow for adaptive boosting (Adaboost) by assigning greater weights to misclassified instances. Adaboost follows a unique process: initially, it constructs a weak learner, slightly better than random guessing, by assigning equal weights to each data point. In subsequent iterations, new learners are incorporated with adjusted weights, increasing the emphasis on misclassified instances while reducing it on correctly classified ones. This iterative refinement aims to enhance accuracy, with the final prediction derived from a weighted combination of weak learners' outputs. One weakness of this approach is its susceptibility to outliers and noise.

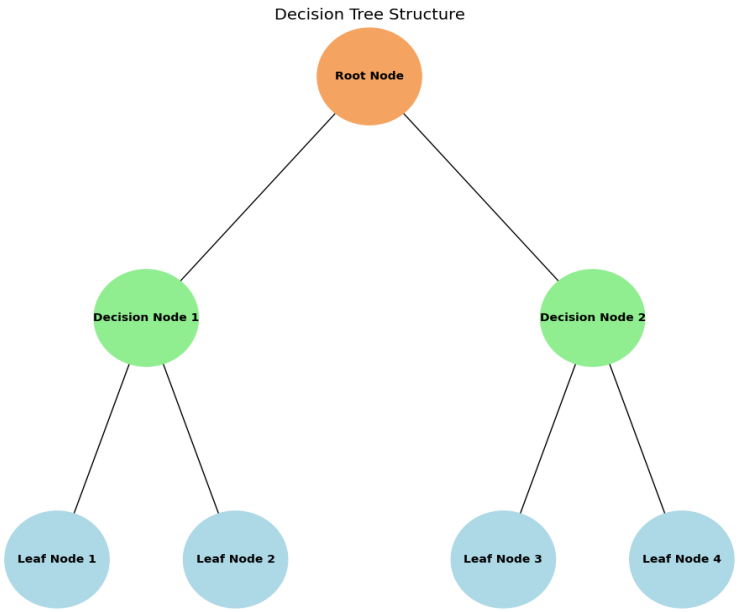


Figure 2 Decision Tree Structure.

The lines between the nodes are called branches. Source: Own representation based ob the description of trees byBreiman et al. (1984).

Friedman (2001) proposes Gradient Boosting, a more resilient alternative to Adaboost. Rather than adjusting weights for individual data points, Gradient Boosting optimizes performance by updating based on the gradient of the loss function, emphasizing areas with higher errors. Thus, enhancing the model's

stability by making it more robust against noisy data. Among gradient boosting algorithms, XGBoost has gained prominence due to its extensive optimization features, such as a customizable learning rate (eta) and regularization options to prevent overfitting, such as shrinkage techniques. Additionally, the support for parallel computation across multiple CPU cores enables it to efficiently handle large datasets.

Additionally, XGBoost facilitates out-of-core learning, contributing to its scalability, as highlighted by Chen and Guestrin (2016). Building on this architecture, Ke et al. (2017) introduce LightGBM, which further enhances computational efficiency and memory usage compared to XGBoost. LightGBM uses leaf-wise (vertical) tree growth instead of level-wise (horizontal), allowing for greater loss reduction, improved accuracy, and faster processing. Unlike XGBoost, which uses a pre-sorted algorithm to build trees by enumerating over all possible points based on pre-sorted feature values, LightGBM uses a histogram-based algorithm instead. This alternative approach buckets continues variables into discrete bins, from which the best split point is chosen, allowing for the construction of feature histogram. As a result, it reduces memory usage and accelerates the split-process, making LightGBM particularly useful for large datasets.

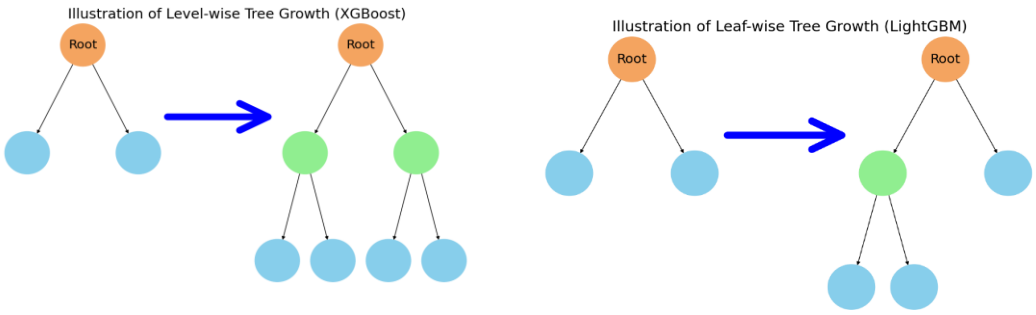


Figure 3 XGBoost and LightGBM Tree Growth.
The green highlighted circle is the leaf and the first circle in the tree is referred to as node.
Source: Own representation based on the description of the algorithms.

Due to their versatility and robustness, boosting models are a popular choice for forecasting competitions like the M5 competition, where tree-based models outnumbered deep learning models in entries (Januschowski et al., 2022).

In the following Section I explain the extension of ML architecture to build more complex models using artificial neural networks with many layers allowing to capture the more complex structure of high-dimensional data better than traditional ML models (LeCun et al., 2015).

3.3 Deep learning models

Over the past two decades, a new type of ML approach called Deep Learning (DL) has proven to be state-of-the-art for solving complex classification and regression problems. Deep learning algorithms are particularly effective for tasks such as image and speech recognition (LeCun et al., 2015). In the this section, I explain which deep learning algorithms are used for my time series forecasting. I consider four different architectures: Recurrent Neural Networks, Convolutional Neural Networks, Multi-Layer-Perceptron (MLP)[5] and Transformers. First, I provide a brief introduction to the function of neural networks, covering key concepts and methods in deep learning based on Zhang et al. (2021).

How do neural networks work?

Inspired by the human brain, Artificial Neural Networks (ANNs) aim to process information in a similar way. ANNs typically consist of layers of interconnected neurons that process input data through weighted connections. The architecture consists of two main layers: an input layer and an output layer. Additional layers between these two essential layers are called hidden layers. A simple ANN consists of at least one hidden layer, hence three layers in total. Each neuron calculates a weighted sum of its inputs and then applies an activation function to this sum to produce the output. For example, with a rectified linear unit (ReLU) activation function, as proposed by Nair and Hinton (2010), the output is the max of x and 0, ensuring a non-negative result.

[5] A Multilayer Perceptron (MLP) is a type of neural network consisting of multiple fully connected layers of neurons.

$$ReLU(x) = max(x, 0), \tag{17}$$

In a feedforward neural network, the input data is passed through the network from the input layer to the output layer. This also involves calculating weights, biases and the activation function outputs. The goal of an NN is to learn the true function that describes the data the best. So, the empirical loss should be minimized by finding the best weights and biases for this.

$$\hat{\theta} := arg\ arg\ E_n\ \left[l\left(Y, f(X, \theta)\right)\right], \tag{18}$$

$$\nabla_\theta E_n\left[l\left(Y, f(X, \theta)\right)\right], \tag{19}$$

where the gradient of f is defined as $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$

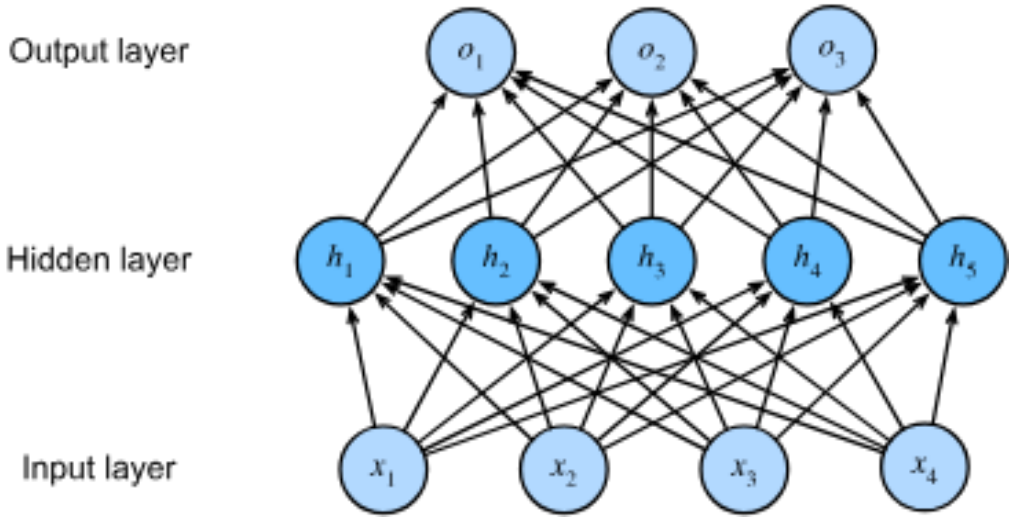


Figure 4 A simple MLP with one hidden layer.
A Fully Connected Feedforward Neural Network, also known as Multilayer Perceptron (MLP).

Source : https://d2l.ai/chapter_multilayer-perceptrons/mlp.html Zhang et al. (2021)

To achieve this goal efficiently, Stochastic Gradient Descent (SGD) is used to update weights and biases during backpropagation. The calculated errors are used to determine the gradient, which is propagated back through the network to adjust the weights and minimize errors. SGD allows for different data processing options: using a subset of the data is called a batch, a smaller subset is called a minibatch, and using the entire dataset is called a full batch.

For each iteration, a batch is randomly sampled from the dataset (without replacement), and the resulting errors are used to update the model's weights. A complete pass over the entire dataset is referred to as an epoch, during which the model processes all available data. For example, a dataset containing 1,024 data points with a batch size of 64 results in 16 batches per epoch. Since the model processes data in batches, it completes one full epoch after seeing all 16 batches. Each update is referred to as an optimization step, since the model updates its weight after each batch. If trained for 100 epochs, the model undergoes 1,600 optimization steps. There are two ways to control training frequency: in Darts[6], training frequency is set by specifying the number of epochs, while in Nixtla[7], it is managed by defining the number of optimization steps.

This procedure addresses the risk of the model becoming trapped in a local minimum, a common issue when using gradient descent without batch processing, as noted by Bottou (1991).

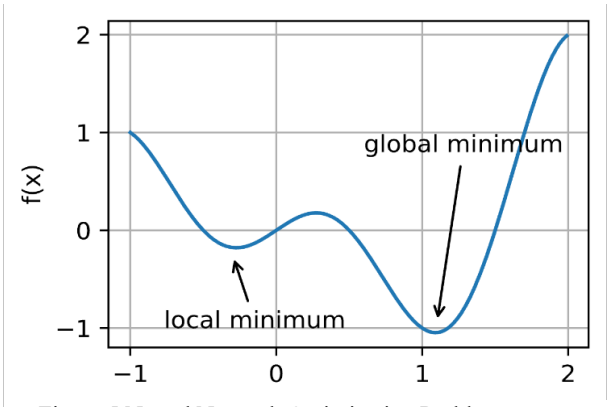


Figure 5 Neural Network Optimization Problem.
Source : https://d2l.ai/chapter_optimization/optimization-intro.html

To achieve good generalization, the model to perform well on new, out-of-sample (OOS) data, the model should avoid learning the training data's structure too precisely. Otherwise, it may not generalize effectively. This phenomenon is known

[6] The used library for the M4 study, for more see https://unit8co.github.io/darts/

[7] The used library for the hierarchical forecasting, for more see https://Nixtlaverse.Nixtla.io/

as overfitting (see Figure 6), and various regularization techniques are used to mitigate it.

To test the generalization ability of a model, new data is needed, therefore the used dataset can be split into training and test data. The train data is used to train the model, then the performance is tested the test data. Ideally the minimum error for both datasets should be achieved. Cross-validation is another common method to evaluate generalization. In cross-validation, the training data is divided into k equal folds (e.g., 5 folds). One-fold is used as a validation set, while the remaining folds are used for training. This process is repeated k times, with each fold serving as the validation set once. The errors from each iteration are then averaged to provide an overall performance estimate.

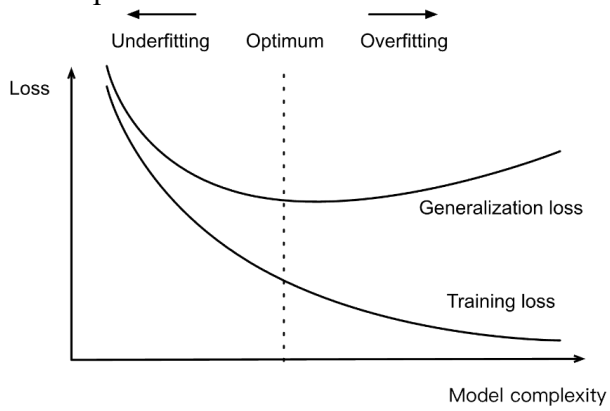


Figure 6 Bias-Variance Trade-off in Neural Networks.
Source : https://d2l.ai/chapter_linear-regression/generalization.html. Zhang et al. (2021)

Backpropagation through time

Applying backpropagation to sequential data is known as Backpropagation Through Time (BPTT) (Werbos, 1990). BPTT involves unrolling the RNN's computational graph over each time step, effectively transforming it into a feedforward network with recurring parameters at each timestep. This setup allows the application of the chain rule to propagate gradients backward through the network. The gradients of each parameter are aggregated across all instances in the unrolled network, a concept similar to weight tying in convolutional neural networks

However, challenges arise when dealing with lengthy sequences, often comprising exceeding thousands of tokens, where computational demands and memory requirements increase significantly. The input traverses more than a thousand matrix products before reaching the final output and requires an equal number of products to compute gradients. The paper further discusses potential pitfalls and practical solutions to these issues.

3.3.1 Recurrent-based models

Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are specifically designed to capture temporal dynamics, which makes them well-suited for time series forecasting. RNNs achieve this by updating the hidden state at each time step using a recurrent function:

$$H_t = \phi(W_{hx}X_t + W_{xh}H_{t-1} + b_H), \tag{20}$$

Where H_t represents the hidden state at time *t*, and ϕ is some activation function. The weights matrices are $W_{hx} \in R^{h\times h}$, and $W_x \in R^{x\times h}$, and the bias vector $b \in R^h$. control how the hidden state H_t is updated using the current input X, the old hidden state H_{t-1}, and a bias b. This recurrent update process is responsible for the "recurrent" aspect of the RNN architecture (see Figure 7).

At each time step t the model generates an output O_t using the following equation:

$$O_t = H_t W_{ho} + b_O, \tag{21}$$

Where $\mathrm{W_{ho}} \in \mathrm{R^{h\times o}}$ and $b_\mathrm{o} \in \mathrm{R^h}$. are the output layer's weight matrix and bias vector, respectively. RNNs are deep learning models that capture dynamic dependencies in sequential data.

With BPTT the hidden state H_t is updated at each time step as a sequence is processed. However, this recursive structure introduces challenges when dealing with long sequences. As the network unfolds over time, it essentially forms a very deep neural network, where each layer shares the same weights (LeCun et al. (2015).

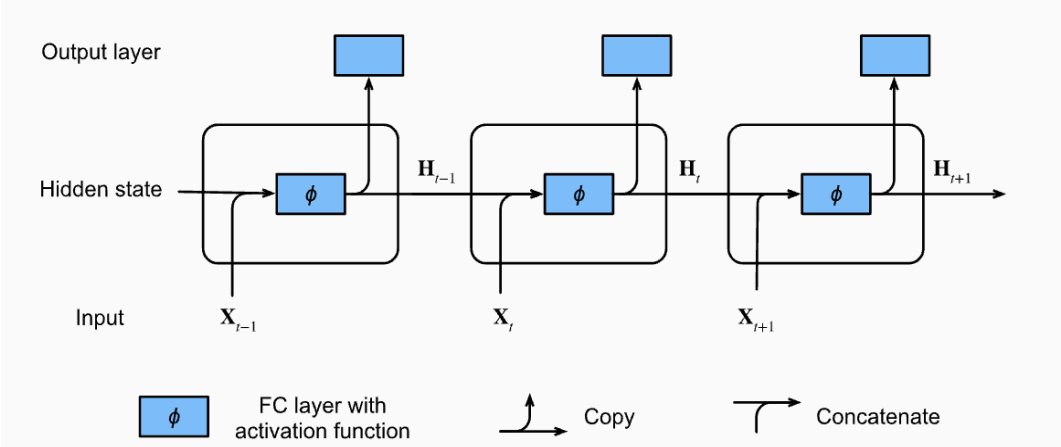


Figure 7 RNN Architecture
Source : https://d2l.ai/chapter_recurrent-neural-networks/rnn.html (Zhang et al. (2021)

For very long sequences, gradient computation becomes computationally expensive, and the gradients tend to diminish with each time step, a phenomenon known as the vanishing gradient problem (Hochreiter, 1998)[8]. Consequently, RNNs often struggle to capture long-term dependencies effectively. Hochreiter and Schmidhuber (1997) introduce the Long-Short-Term-Memory (LSTM) model to address this limitation. LSTM is specifically designed to address the vanishing gradient problem and capture long-term dependencies. The following section provides an overview of the LSTM architecture and its role in time series forecasting.

[8] Hochreiter shows this problem in his diploma thesis in 1991 and later publish it in 1998.

LSTM

The LSTM architecture addresses RNN limitations by introducing a memory cell, which functions similarly to a hidden state but enables better handling of long-term dependencies. The LSTM memory unit includes three gates: the input gate $I_t \in R^h$, forget gate $F_t \in R^h$, and an output gate $O_t \in R^h$.

$$I_t = \sigma(W_{xi}X_t + W_{hi}H_{t-1} + b_i), \tag{22}$$

$$F_t = \sigma\left(W_{xf}X_t + W_{hf}H_{t-1} + b_f\right), \tag{23}$$

$$O_t = \sigma(W_{xo}X_t + W_{ho}H_{t-1} + b_o), \tag{24}$$

Here, the weight matrices are $W_{xi}, W_{xf}, W_{xo} \in R^{d\times h}$ and $W_{hi}, W_{hf}, W_{ho} \in R^{h\times h}$, and the biases vectors are $b_i, b_f, b_o \in R^{1\times h}$. The values for each gate are calculated using the sigmoid activation function σ, hence the values are between 0 and 1. Similar to the hidden state in RNNs, the memory cell C_t is updated at each time step t using a candidate memory cell $\widetilde{C_t}$, which uses a hyperbolic tangent (*tanh)* activation function with a value range (-1, 1).

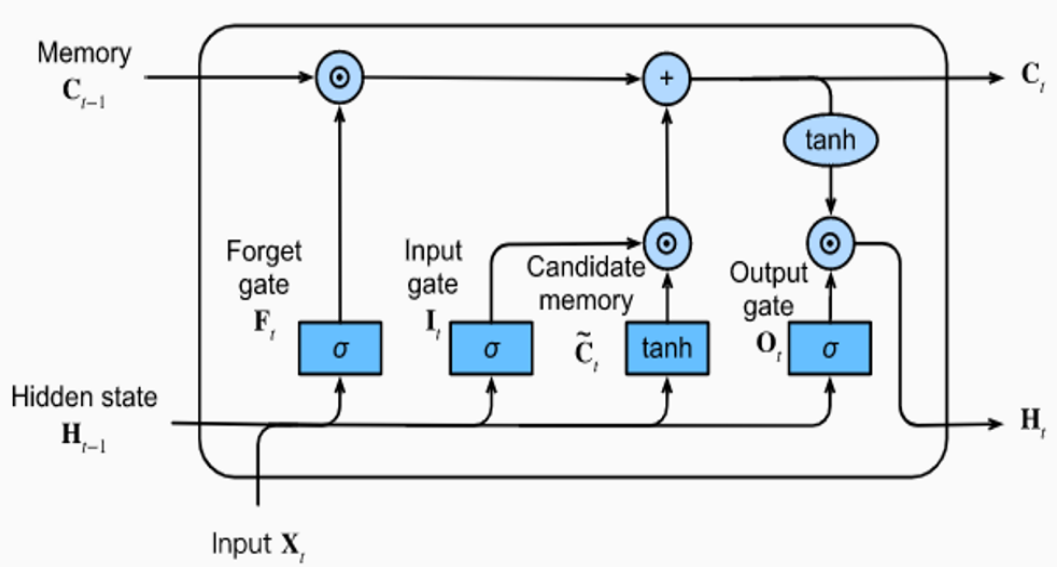


Figure 8 LSTM Architecture
Source: https://d2l.ai/chapter_recurrent-modern/lstm.html (Zhang et al. (2021)

$$\widetilde{C_t} = tanh(W_{xc}X_t + W_{hc}H_{t-1} + b_c), \quad (25)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \widetilde{C_t}, \quad (26)$$

Here, the weight matrices are $W_{xc} \in R^{d\times h}$ and $W_{hc} \in R^{h\times h}$, and the bias vector is $b_c \in R^{1\times h}$. In the new memory cell at time *t,* the forget gate determines how much of the previous memory cell C_{t-1} be retained through elementwise multiplication denoted by $\odot$. The value of $F_t = 0$ means that the previous memory cell is completely reset, while $F_t = 1$ fully retains it. The candidate memory cell $\widetilde{C_t}$ allows the model to reduce or increase the impact of an input I or even reduce its value, since $\widetilde{C_t}$ can negative, at time *t*. Finally, the hidden state H_t is updated by elementwise multiplying the output gate O_t with the tanh activation of the memory cell C_t, as follows:

$$H_t = O_t \odot tanh(C_t), \quad (27)$$

3.3.2 CNN-based models

Originally developed for image recognition, Convolutional Neural Networks (CNNs) use convolutional filters, to extract information from image data, enabling pattern detection and accurate classification (LeCun et al., 1998). CNNs achieve this by applying small matrices, known as filters or kernels, to extract spatial information from images by sliding over the entire image in a process known as convolution. Similarly, time series contain hidden patterns within their temporal sequences, which can be extracted using convolutional methods.

One of the earliest attempts to demonstrate how CNNs can be useful for TSF is a a generic Temporal Convolutional Networks (TCNs) architecture (Bai et al., 2018), which introduces two new convolutions to CNN, causal and dilated. Causal convolutions ensure that predictions depend only on past values. However, there are two limiting factors to how far a simple convolution can look back, namely the size of the kernel and the depth of the network. The filter size *k* dictates how many

previous time steps are considered to predict future values. For instance, a kernel size of 7 implies that the future value depends on the current time step and the 6 preceding time steps. As the neural network becomes deeper, each layer can consider all time steps from the previous layer, but these factors still limit the model's receptive field. To overcome this limitation, dilated convolutions introduce a dilation *d,* which increases progressively with each network layer to expand the receptive field. The receptive field of a single layer is (k-1) d, hence exponentially increasing d allows the network to capture longer-range dependencies more efficiently.

Both TCNs and RNNs aim to capture short- and long-term dependencies in sequential data. Unlike RNNs, TCNs apply the same filter across all time steps, allowing for parallel computation and removing the sequential processing constraint. Additionally, this architecture does not suffer from gradient-related issues like RNNs, such as vanishing or exploding gradients, because gradients are calculated after each predicted value and then propagated back through the layers to update the corresponding kernel weights.

This TCN architecture demonstrates that CNNs can be a suitable solution for TSF problems. Recent models, such as the Multi-Input Convolutional Network (MICN) introduced by Wang et al. (2023), extend the TCN architecture for multivariate TSF by applying separate convolutions to each input stream. This operation allows to capture local patterns as well as interactions across multiple time series. Furthermore, a more recent model called TimesNet (Wu et al., 2023)introduces two key strategies to address limitations of models like TCN. First, it implements a flexible multi-resolution strategy, allowing the model to capture temporal patterns at varying time scales by adjusting kernel sizes dynamically. Second, it utilizes a hierarchical learning approach that processes data at different granular levels, enabling the model to capture both coarse and fine fluctuations in the data. These techniques enhance TimesNet’s ability to model complex temporal dynamics across different time horizons.

TimesNet transforms one-dimensional sequential data into a two-dimensional structure, based on identified periods and frequencies within the data (see Figure 9). Each column represents the values of a specific frequency, and each row represents different periods in the time series. This transformation allows the model to capture both intra-period (within-period) and inter-period (across-period) variations using 2D kernels. For instance, when dealing with daily data, the model can identify variations between the days within a period (such as a week or month) and the variations of specific days across different periods. This method introduces a novel approach for handling sequential data by treating it as an image. After transforming it into 2D format, convolutional kernels capture spatial variations both within and across periods.

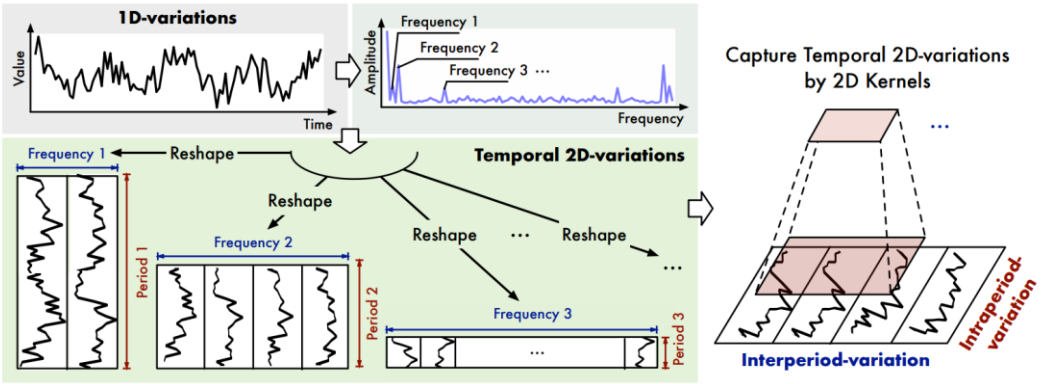


Figure 9 TimesNet Function Overview.

Transform of Time Series into 2D data. Source:Wu et al. (2023).

Both RNNs and CNNs often employ fully connected layers (dense layers) at their output to produce the predictions. There are indeed models that solely rely on these layers for TSF. In the following Section I present some of the most popular Multi-Layer Perceptron (MLP) based models.

3.3.3 MLP-based models

NBEATS

Neural basis expansion analysis for interpretable time series or NBEATS is a new type of neural networks developed by Oreshkin et al. (2019) specifically for time series. The goal of NBEATS is to provide a straightforward model for time series forecasting that does not require feature engineering, unlike statistical models like ARIMA or smoothing methods. Unlike RNNs and LSTMs, NBEATS processes sequences without recurrence, relying instead on a feedforward architecture. Instead, the model uses a feedforward architecture where the entire input sequence is processed at once without looping back through the sequence. This architecture allows for a more efficient and easier train, since it does not have to deal with backpropagation through time (BPTT). Additionally, these models benefit from the possibility of parallel processing which reduces the training time substantially. The basic component of NBEATS is a fork-shaped block composed of four fully connected layers with ReLU activations, designed to capture complex non-linear relationships. This block generates a backcast and a forecast. The first component is a backward-looking operation that tries to capture the reconstruct the input time series to help the model understand the underlaying temporal patterns. The second component aims to generate forecasts based on the learned patterns from the backcast. NBEATS consists of a K blocks stacked sequentially using a technique known as doubly residual stacking. This technique builds upon the work of He et al. (2016), who introduce a framework specifically developed to optimize the training of deep neural networks, particularly those that suffer from gradient-related issues such as the vanishing gradient problem. He et al. (2016) argue that by using residual connections, allowing the gradient to skip one or multiple layers, neural networks can be trained more effectively.

Residual connections pass the input of each layer not only to the next layer but also directly to the output, allowing the model to preserve important information across layers.

This residual connection is represented mathematically as:

$$f(x) + x, \tag{28}$$

where x is the input from the previous layer and $f(x)$ is the transformation of the input x, produced by the layer. When the weights in the transformation function f are set to zero the output y is equal to x. Thus, the residual block effectively performs identity mapping, allowing the input to pass through unchanged as the output. This property is one of the reasons why residual networks can maintain performance even as they grow deeper, as they can easily learn to preserve the input when necessary. NBEATS extends this concept by introducing a second level of residual connections that operate not just within individual layers or blocks but also across multiple blocks within the same network stack, hence the name doubly residual stacking. The NBEATS model consists of M stacks, with each stack learning distinct aspects of the time series. For instance, one stack might be tuned to capture seasonality, while another might be used to focus on the trend. Olivares et al. (2023) extend NBEATS to allow to incorporate exogeneous factors, thus allowing to enhance the performance when including relevant external variables. NBEATS paved the way for new state-of-the-art models that build on its innovative architecture such as NHITS.

NHITS

The Neural Hierarchical Interpolation for Time Series (NHITS) is a model developed by Challu et al. (2023) to address challenges in long-horizon forecasting, such as prediction volatility and computational complexity. NHITS introduces two additional techniques to NBEATS to enhance the forecasting performance, especially for long horizons. The primary innovation, hierarchical interpolation, enables the model to decompose time series data across multiple hierarchical levels,

allowing NHITS to focus on various temporal scales effectively, each corresponding to a different frequency, allowing the model to capture distinct patterns that occur at various frequencies such as short or low frequencies (see Figure 10). Unlike NBEATS, which focuses on seasonality and trend decomposition, NHITS decomposes time series data into frequency bands. This approach allows the model to capture patterns that occur at different frequencies. NHITS also introduces multi-rate signal sampling, which enables the model to flexibly focus on various frequency scales within each block, providing adaptability for long-horizon forecasting. Each block includes an additional MaxPooling layer with a kernel size k, allowing the model to adjust its focus on different frequency scales and levels of detail within the data. For long-horizon forecasts a larger kernel size ensures that the model pay more attention to large scale/ low frequency. For instance, on a high-level resolution (e.g., weekly or monthly) the model captures high frequency components. The block architecture ensures that each sequential block refines the prediction of the previous block.

The model's final output is an aggregate of each stack's output, thus ensuring that the model captures the full spectrum of patterns across different resolutions and frequencies. With the use of a MaxPool layer with a kernel size k at each block, the model offers the flexibility to focus on analyzing components at a specific scale, for instance use a larger Multi-Rate Signal Sampling.

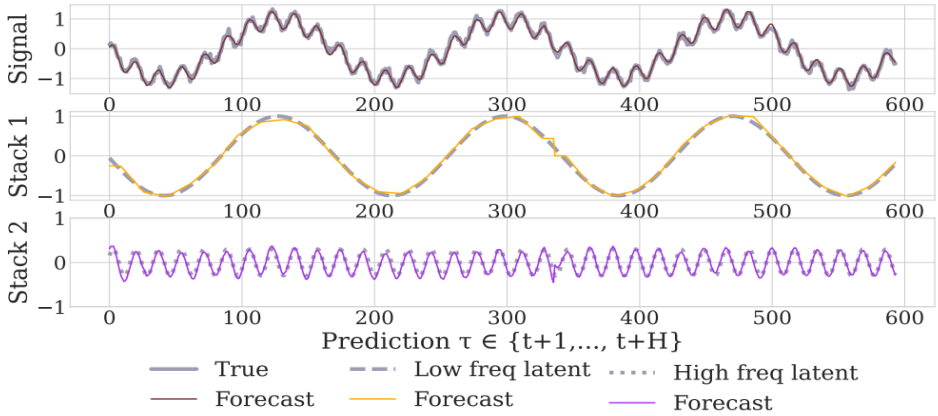


Figure 10 NHITS Frequency and Signal Capture
Source Challu et al. (2023).

More recent approaches suggest using mixing architectures to capture complex dependencies within temporal data. TSMixer (Chen et al., 2023)employs two MLPs, a feature- and a time-mixing MLP, while TimeMixer (Wang et al., 2024), employ a decomposable multi-scale mixing architecture.

3.3.4 Transformer-based models

The introduction of the new neural network architecture by Vaswani et al. (2017) called Transformer is considered as a pivotal point in the field of deep learning. Using the attention mechanism transformers allows us to solve more complex tasks such as understanding natural language by capturing the dependencies over long sequential data such as text data. This innovation extends beyond Natural Language Processing (NLP) to domains like TSF, where capturing both short- and long-term dependencies within sequences is crucial for accurate predictions. This is achieved solely through a method called self-attention which extends the attention mechanism introduced by Bahdanau (2014) enhance models in sequence-to-sequence tasks such as machine translation. Attention allows the model to identify relevant parts of a sequence to focus on when predicting a word, rather than

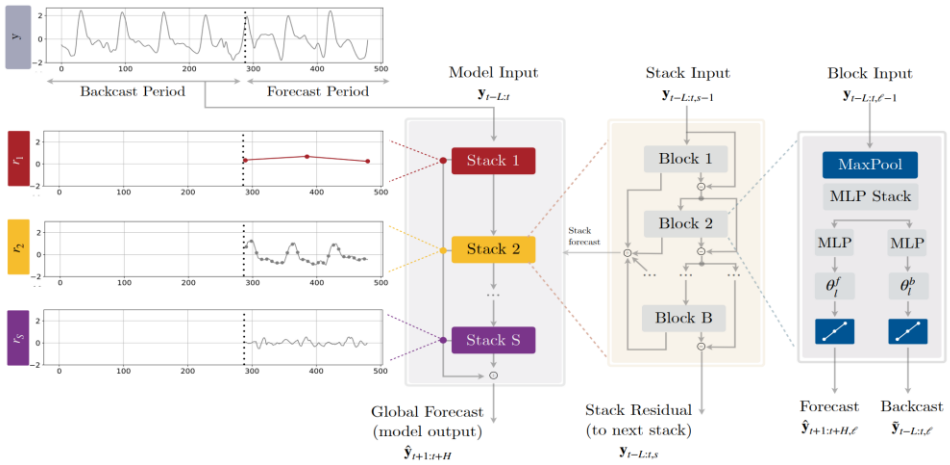


Figure 11 NHITS Architecture
Source: Challu et al. (2023).

considering the inputs of a sequence.

A recurrent model encodes a source sentence into a fixed-length vector which can lead to information loss (Cho (2014). The attention mechanism resolves this by

dynamically computing a weighted sum of all input elements, assigning higher weights to more relevant parts of the sequence. The self-attention mechanism works by deriving three vectors: Query (Q), Key (K), and Value (V), from the input sequence. The query vector Q represents the current element, whether it's a word in NLP or a data point in TSF, for which the model seeks relevant information from the entire sequence. The query vector Q is compared to all other elements in the sequence using their corresponding key vectors K, which are derived from a linear transformation of the input sequence. By computing dot product between Q and K vectors the model can identify the relative importance of one input to another. The dot product between Q and K is scaled by the square root of the key's dimensionality to stabilize the values. This result is then passed through a SoftMax function, transforming the score into a probability distribution over the sequence elements to ensure that the sum of all probabilities is equal to 1. Finally, the actual data represented by V are weighted by the calculated attention scores. attention probabilities as expressed mathematically in the following equation:

$$Attention\ (Q, K, V) = softmax\ \left(\frac{QK}{\sqrt{d_t}}\right) V, \tag{29}$$

In self-attention, the prefix "self" indicates that all three components: Q, K, and V are derived from the same input sequence. While in the original attention mechanism the query is generated from the decoder, i.e., the target/output sequence. Self-attention allows the model to learn internal relationships and dependencies within the sequence itself. This is parallel to TSF where the target is to predict an output based on historical observations by capturing short- and long-dependencies. In TSF, Transformers address the limitations of models like LSTM, which often struggle with retaining information over long sequences, by using self-attention to effectively capture dependencies at varying time horizons.

The transformer proposed in Vaswani et al. (2017) uses an encoder and decoder architecture, each is composed of $N = 6$ identical layers. The encoder consists of two sublayers, a multi-head-attention[9] (see right side of Figure 12) and fully connected position wise feed-forward neural network (FFN). Positional encoding is added to the input embeddings to provide the model with information about the position of each token or time step in the sequence.

The FFN consists of two non-linear transformations with ReLU activation function in between, as expressed in the following equation:

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2, \tag{30}$$

In this equation:

- Input x represents the output from the self-attention mechanism.
- Weight Matrices W_1 and W_2 are learned parameters
- Biases b_1 and b_2 adjust the output values after each linear transformation, adding flexibility to the model.

The decoder adds a third sublayer to apply multi-head-attention to the output of the encoder stack. Additionally, to target the gradient-related problems that RNNs could suffer from, transformer architecture employs residual connections (He et al., 2016) that add the input of each sublayer to its output before applying normalization. This similarity between language preprocessing and TSF allows the development of algorithms specifically designed to learn the patterns such as seasonality and trend in a time series. Indeed, since the introduction of transformers multiple transformer-based algorithms were developed for TSF.

[9] Multiple self-attention mechanisms run in parallel to allow the model to capture various types of dependencies.

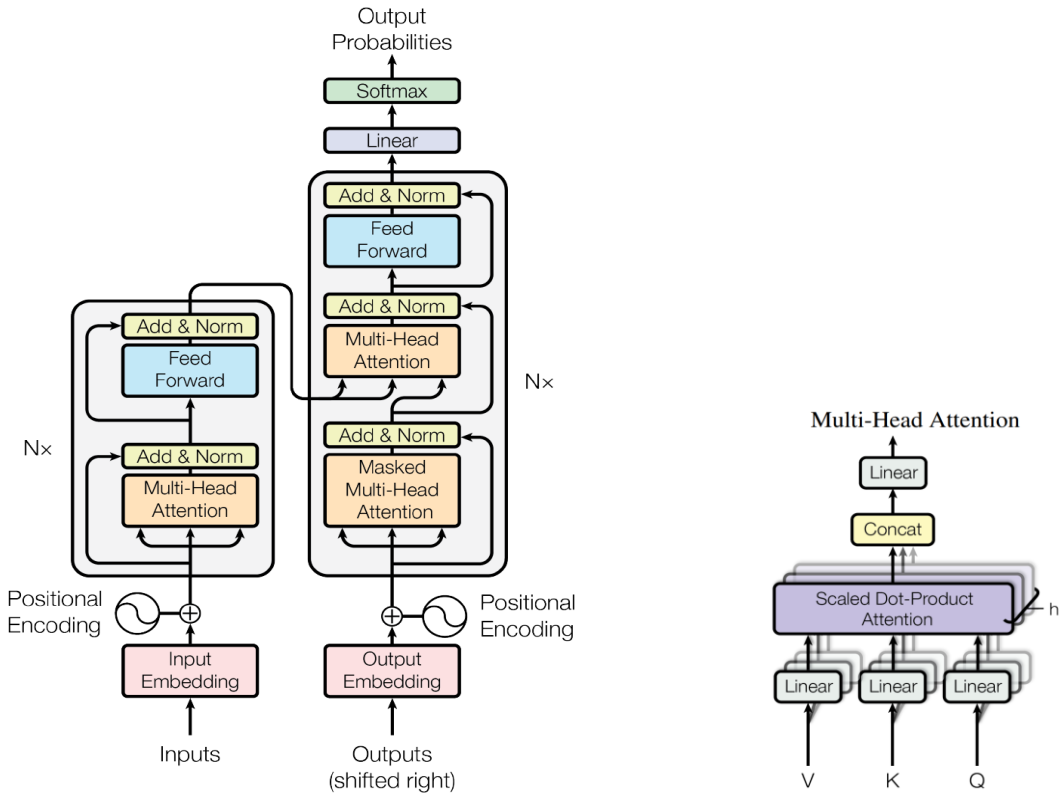


Figure 12 Transformer Architecture with Multi-head Attention.

Source: He et al. (2016)

The Informer model by Zhou et al. (2021) addresses the challenge of capturing long-term dependencies in large datasets by introducing ProbSparse, a probabilistic self-attention mechanism. ProbSparse focuses only on the most relevant queries, significantly improving scalability for datasets with lengthy sequences. Autoformer, introduced by Wu et al. (2021), replaces traditional self-attention with an auto-correlation mechanism specifically designed for time series forecasting. This mechanism captures periodic patterns by identifying similarities within sub-series based on data periodicity, rather than computing the relevance between all element pairs as standard self-attention does.

Autoformer's mechanism relies on decomposing time series into trend and seasonal components, similar to ARIMA, allowing it to analyze each component independently. The auto-correlation mechanism aggregates similar sub-series

across different periods, enabling the model to effectively learn and predict cyclic behavior in the time series, such as daily, weekly, or yearly patterns, without being computationally intensive like standard self-attention. Autoformer's decomposition-based approach, enhanced by an autocorrelation mechanism, improves both efficiency and accuracy for long-term forecasting. This approach is particularly effective in datasets with strong seasonal or recurrent patterns. By extending Autoformer to better capture frequencies, FEDformer, developed by Zhou et al. (2022), extends Autoformer by incorporating frequency analysis using Fourier transforms. This enhancement allows FEDformer to capture frequency-specific patterns efficiently, making it especially suitable for cyclical datasets, such as those related to energy consumption.

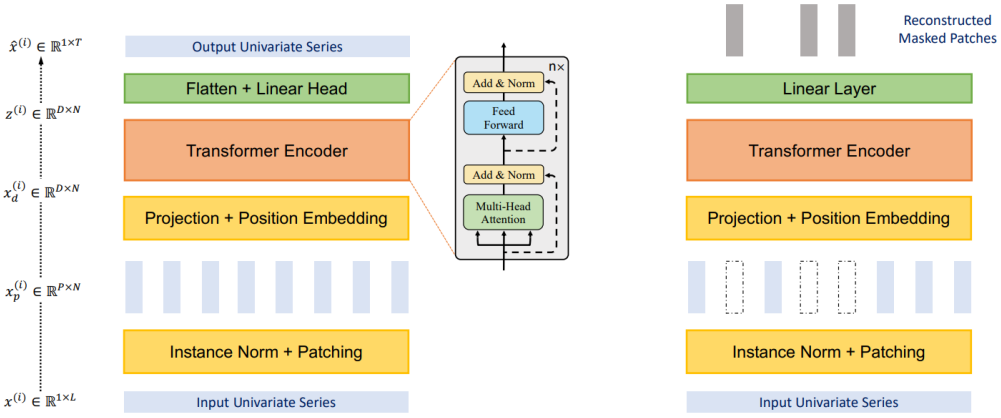
The Temporal Fusion Transformer (TFT), introduced by Lim et al. (2021) addresses the complexities of multi-horizon time series forecasting by combining self-attention with RNNs. This combination helps overcome the limitations of traditional recurrent models, which often struggle to capture long-term dependencies effectively. In TFT, a sequence-to-sequence encoder captures short-term dependencies, while an attention mechanism in the decoder focuses on long-term dependencies, enabling balanced forecasting across various time horizons. Unlike traditional attention mechanisms that capture correlations between time stamps, TFT applies attention to each time series independently, with no information shared across series. This design, while effective, increases computational complexity compared to simpler time series models. Due to the use of attention and RNNs, TFT is computationally more expensive than simpler time series models. iTransformer (Liu et al., 2023) is another model that uses series-wise processing to capture inter-series dependencies in multivariate data.

Patch Time Series Transformer (PatchTST), developed by Nie et al. (2022) is a Transformer-based model designed for multivariate forecasting. This model introduces two new methods, Patching and Channel-independence. Using an

independent channel ensures a multivariate time series is divided into univariate time series independently processed.

Each univariate time series is processed patch-wise, with a typical patch size of 64. By feeding the model patches instead of tokens (point-wise inputs), PatchTST becomes more efficient by reducing computational costs and model complexity. For example, a sequence of 1000 steps would generate 1000 tokens in a point-wise model but only 16 patches when processed patch-wise, significantly reducing the input size and improving efficiency. By processing input data in patches, PatchTST reduces the quadratic complexity of the original Transformer model from $O(N^2)$ to $O((L/S)^2)$, where NNN represents the number of tokens, LLL is the sequence length, and SSS is the stride. This optimization enables PatchTST to handle longer sequences with significantly reduced computational demands. PatchTST applies instance normalization to each patch, adjusting it to have a zero mean and unit variance. This normalization reduces the influence of different time series scales, while positional embeddings are added to ensure the model retains temporal information within each patch. To ensure that the model retains temporal information, positional embeddings are added to the patches. Once normalized and embedded with positional information, the patches are processed by the supervised Transformer encoder (shown on the left side of Figure 13). In the encoder, the full multi-head self-attention mechanism, as in the vanilla Transformer (Vaswani et al. (2017), is applied to enable the model to capture both short- and long-term dependencies across the time series data. The encoder output is then flattened and passed through a linear layer, which maps the patches back into future predictions. This supervised Transformer backbone aims to minimize the loss between the predictions and actual values, typically using a loss function such as Mean Squared Error (MSE). Additionally, PatchTST employs a self-supervised learning technique as a pre-training step, helping the model learn generalized representations of the data by masking randomly selected patches and training the model to reconstruct them (see right side of Figure 13).

The parameters learned during this self-supervised pre-training phase are then transferred to the main supervised training, enhancing the model's ability to capture complex time series patterns. PatchTST's architecture allows it to achieve competitive performance in time series forecasting, efficiently handling both short- and long-term dependencies. Benchmarking studies by Qiu et al. (2024); Shao et al. (2023) demonstrate that PatchTST consistently achieves the lowest error rates across various transformer-based models, proving its effectiveness in diverse forecasting scenario.



(b) Transformer Backbone (Supervised) (c) Transformer Backbone (Self-supervised)

Figure 13 PatchTST Data Processing.

4 Empirical application

For my case studies I use data from the M-forecasting competition. I focus on daily data, however for the first comparison I use daily finance data, which is noisy and could be challenging to forecast. For the second case study I use sales data with a clearer (weekly) seasonality.

4.1 Local and global models (M4 dataset)

The M4 competition dataset includes 100,000 time series from various domains, such as macroeconomics, microeconomics, and finance, across different frequencies (Makridakis et al., 2020). For this study, I use a small subset of the original dataset.

For modeling, I use the Darts[10] library, which provides a comprehensive suite of statistical, machine learning, and deep learning algorithms tailored for both local and global forecasting.

4.1.1 Constructed dataset

The constructed dataset comprises 100 time series of daily financial data. Due to the typically noisy nature of this data, I utilize nearly three years of observations, totaling 1,081 data points per series. Overall, this dataset contains $1{,}081 \times 100 = 109{,}100$ observations. Descriptive statistics reveal that values in the dataset range from a minimum of 446.68 to a maximum of 19,466.50. The average value is 5,866.74, with a slightly lower median, suggesting a right-skewed distribution. Figure 16 depicts a plot of the first five time series is provided below to illustrate the dataset's appearance. This data displays a high degree of randomness, posing a challenge for forecasting models, which offers a good contrast to the next study with the sales M5 sales data.

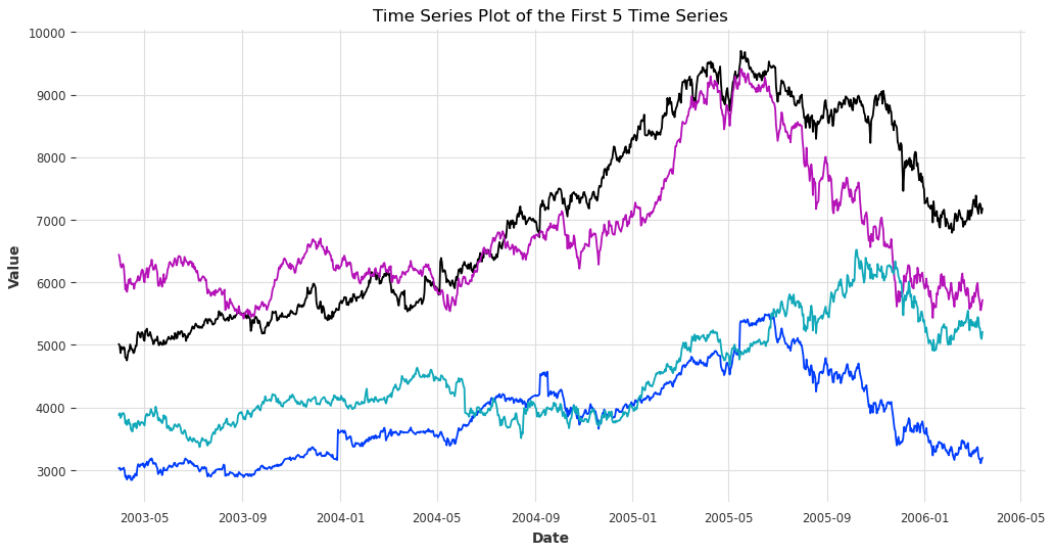


Figure 14 First 5 Time Series in the M4-Dataset.

Source: Own representation.

[10] For more, see https://unit8co.github.io/darts/

4.1.2 Data preprocessing and engineering

Due to the varied value ranges in the data, I apply scaling before training the ML and DL algorithms. Using the Darts library's built-in scaler, which wraps the sklearn MinMax scaler, I normalize the values to a range between 0 and 1. This scaling step is critical, especially for DL models, to prevent issues in gradient-based optimization. Large scale differences among input features can lead to slow convergence or unstable training, causing gradients to either vanish (vanishing gradient problem) or explode (Goodfellow et al., 2016). When features in the input data have very different scales, it can lead to problems such as slow convergence or unstable training, as the gradients can become too small (vanishing gradient problem) or too large (exploding gradient problem).

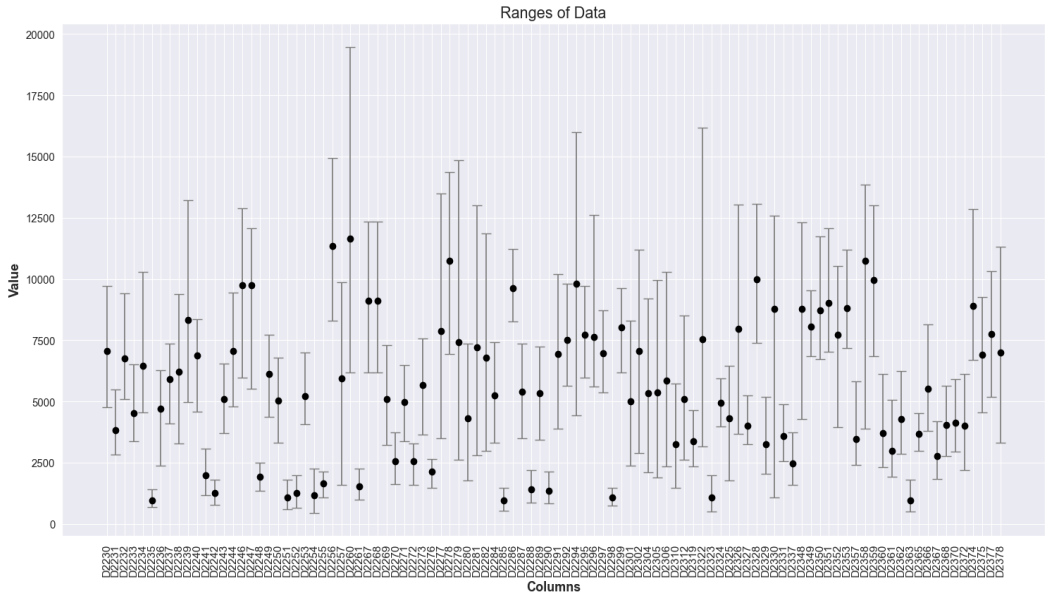


Figure 15 Data Ranges Before Normalization
Source: Own representation.

Without scaling, features with larger magnitudes may dominate the learning process, skewing model performance. Scaling mitigates this, allowing the model to focus on relative differences rather than absolute magnitudes (Goodfellow et al., 2016). DL models can capture complex, non-linear relationships directly from raw data, reducing the need for extensive feature engineering. Their multi-layered

architectures facilitate automatic learning of relevant features, a key advantage for handling complex datasets (LeCun et al. (2015).

Through multiple layers of representation learning, DL models can automatically learn relevant features from the data, which is one of their key strengths. In contrast, ML algorithms like boosting models (e.g., XGBoost, LightGBM) rely heavily on feature engineering for optimal performance, as they lack the inherent ability to learn additional features beyond the input data (Zheng (2018).

Lag features capture autocorrelation within the data sequence by shifting observations by a specified number of steps. For example, a lag of 1 implies that the value at t_{-1} is used as a feature at time t. To determine the number of lags to use, a visual illustration of the correlations between the current and lagged values can be help. The autocorrelation plot (Left plot in Figure 16) visualizes correlations between the time series and its lagged versions across various lags, helping to identify patterns like seasonality. The partial autocorrelation plot (Right plot in Figure 16), illustrates the conditional correlation between each observation and its lagged values while controlling for the effects of any intermediate lags. In the figure above, I present the autocorrelation and partial autocorrelation plots for a randomly selected, yet representative, time series from my dataset. The autocorrelation plot on the left indicates a decaying relationship between observations and lag values, which remains consistently strong over 30 lags, suggesting a persistent trend in the data. The partial autocorrelation plot, however, shows a high correlation with the first lag, with subsequent lags dropping close to zero by lag 14. This pattern implies that the data is non-stationary, exhibiting a clear trend without any seasonal patterns. There are additional techniques for identifying seasonality in time series data, often decomposing the series into trend, seasonality, and residual components. This breakdown aids in isolating the underlying trend. Traditional methods may struggle to capture changes in seasonality effectively over time (Hyndman and Athanasopoulos, 2018).

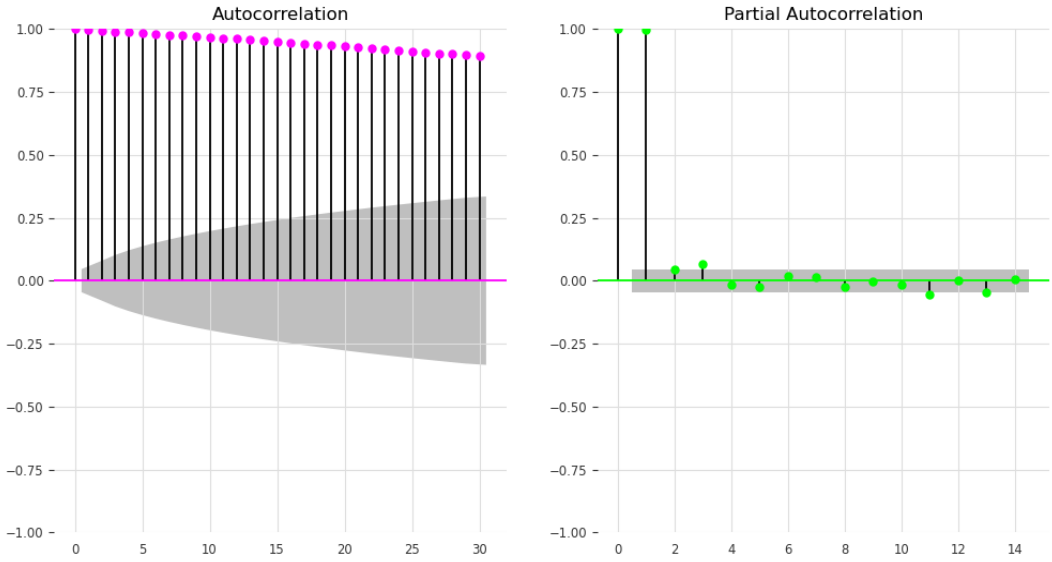


Figure 16 Autocorrelation and Partial Autocorrelation of M4 Dataset

Source: Own representation.

A more flexible alternative is the Seasonal-Trend decomposition using LOESS (STL), developed by Cleveland et al. (1990). STL allows for seasonality to vary over time, providing a robust means to capture evolving patterns. By leveraging LOESS (locally estimated scatterplot smoothing), STL decomposes the time series into three primary components:

1. Trend: The long-term movement in the data, either increasing or decreasing over time.
2. Seasonal: The recurring patterns observed at regular intervals, such as weekly, monthly, or yearly cycles.
3. Residual: The remaining part of the series after accounting for trend and seasonality, which reflects random noise or unexplained variations.

Figure 17 illustrates these components for a randomly selected time series, demonstrating how STL captures variations without assuming constant seasonality. Unlike classical additive or multiplicative decomposition, STL is non-parametric and data-driven, making it especially useful for real-world datasets where seasonal amplitude or frequency can fluctuate.

Given the varying seasonality and the decaying correlation with lagged values, I include the first 4 lags, along with seasonal lags (7, 14, and 28) as features in the LightGBM model. Additionally, I extract calendar-based features, such as month and week, to enhance forecasting performance. Incorporating these features allows the model to capture both short-term dependencies and seasonal patterns, which is particularly beneficial for improving accuracy in time series with irregular seasonal effects. In the next subsection, I explore the M5 dataset, comparing its features to this dataset, despite both being composed of daily data.

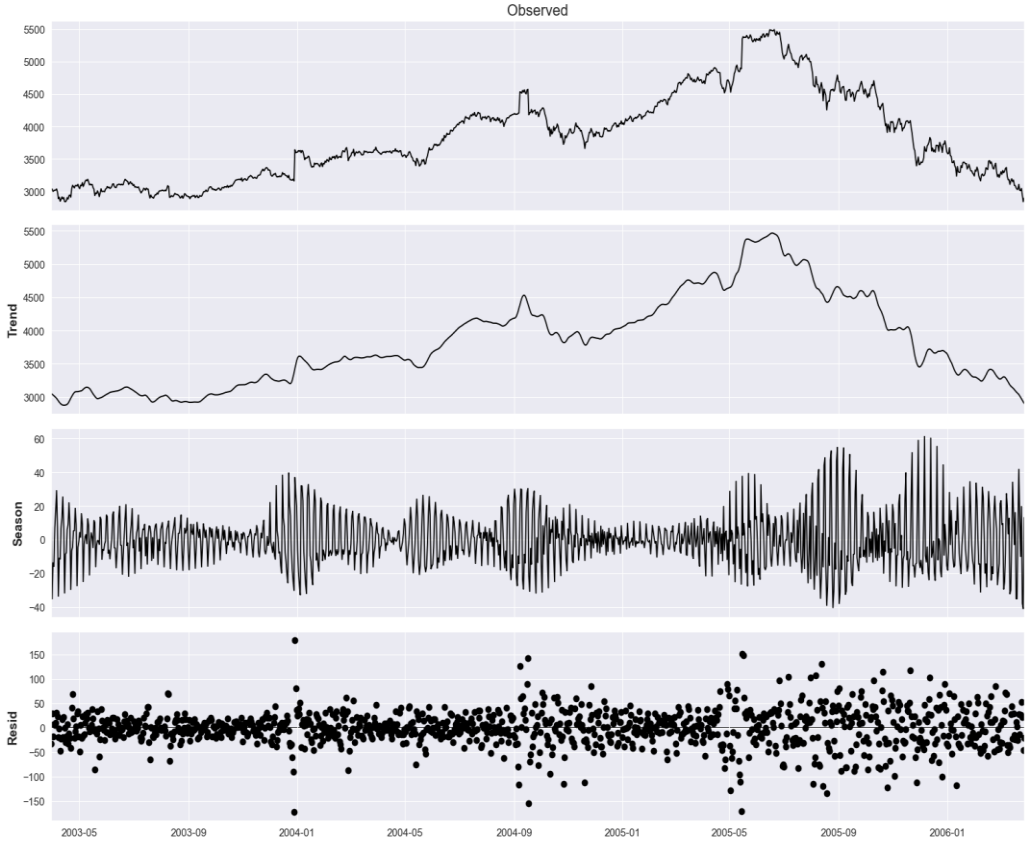


Figure 17 Components for Randomly Selected Time Series
Source: Own representation.

4.2 Hierarchical forecasting (M5 dataset)

For my second case study, I use the M5 dataset, which contains daily sales data from Walmart. The original M5 competition dataset includes more than 40,000 time

series; however, a significant portion of these series represents products that may have numerous zero-sales periods (Makridakis et al., 2022). In this section, I present the customized subset of the M5 dataset used in my study, along with the rationale for my selection. Additionally, I describe my feature engineering process, focusing on constructing a dataset that addresses the challenges associated with intermittent demand, i.e. predicting future values, which can be zero.

4.2.1 Constructed dataset

The focus of this study is to demonstrate how hierarchical models operate and to identify the model with the best forecasting performance. To achieve this, I construct a smaller subset of the dataset, emphasizing higher hierarchical levels to avoid issues related to intermittent demand at the product level.

Table 1 Overview of M5 Dataset

Level	Number of series	Total series per level
USA	1	1
State	3	3
Store	3-4	10
Category	3	30
Department	2-3	70
Total		114

After aggregating the data at the bottom level and including a series representing the total sum for each day, I reduce the dataset to 114 time series arranged in a 5-level hierarchy. The highest level represents the country (USA), where the time series is calculated as the sum of all sales from the underlying state level. For example, the total sales in the USA are obtained by summing sales from the three states represented: California, Texas, and Wisconsin. At each state level, data from 3 or 4 different stores are considered. For instance, in California, state-level sales are derived by aggregating sales from four stores. This pattern continues down the

hierarchy: each level's sales represent the sum of all sales at the level directly below it. At the bottom level of this dataset, departmental sales represent the sum of individual product sales within each department. By structuring the dataset this way, I maintain the hierarchical integrity necessary for hierarchical forecasting while managing data sparsity by avoiding the product level directly. This approach enables the models to focus on capturing aggregated patterns across each hierarchical level. The hierarchical structure is depicted in Figure 18.

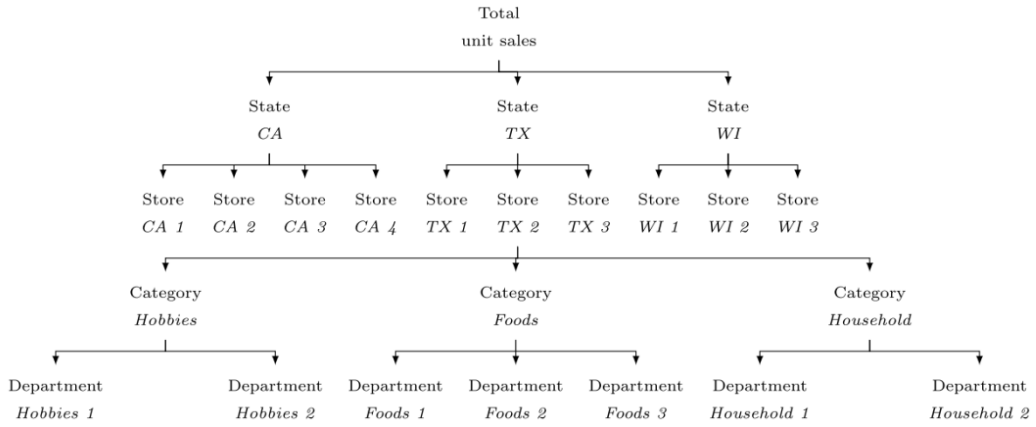


Figure 18 Hierarchical Structure of Constructed M5 Dataset.
Source: Makridakis et al. (2022)

In total I have 5 levels, which are calculated as follows:

1. Total: Sum of state forecasts.

$$\hat{y}_{(T,h)} = \hat{y}_{(CA,h)} + \hat{y}_{(WI,h)} + \hat{y}_{(TX,h)}, \tag{31}$$

2. State: Sum of store forecasts for each state.

$$\hat{y}_{(State,h)} = \sum_i^C \hat{y}_{(store_i,h)}, \tag{32}$$

3. Store: Sum of categories forecasts for each store.

$$\hat{y}_{(Store,h)} = \sum_i^C \hat{y}_{(cat_i,h)}, \tag{33}$$

4. Department: Sum of departments for each category.

$$\hat{y}_{(cat,h)} = \sum_i^C \hat{y}_{(deprt_i,h)}, \tag{34}$$

5. Department: Sum of products sold in each department.

$$\hat{y}_{(deprt,h)} = \sum_i^C \hat{y}_{(prod_i,h)}, \tag{35}$$

The daily sales data ranges from the 29th of January 2011 to the 4th of April 2016, spanning a total of 1,913 days. Figure 19 shows the time series at both the country and state levels. The plot also indicates some level of seasonality, which is typical in sales data.

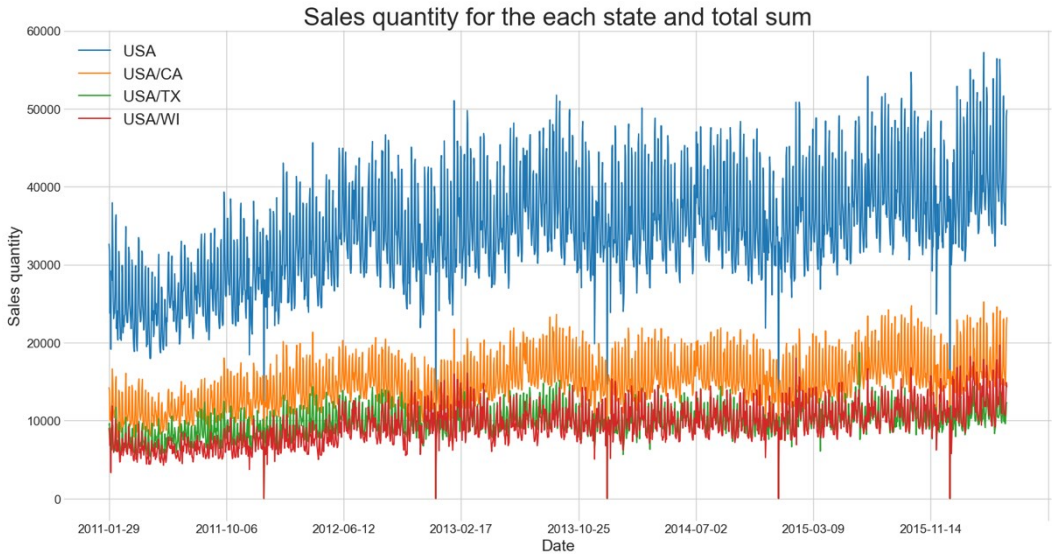


Figure 19 Plot of sales on the country and state levels.
Source: Own representation.

4.2.2 Data preprocessing and engineering

By applying decomposition analysis, the time series can be separated into three components: trend, seasonality, and residuals. A decomposition of the time series at the country level as depicted in Figure 20, shows that the data does exhibit clear constant seasonality, along with a clear upward trend over the observed time period.

The autocorrelation and partial autocorrelation plots, as seen in Figure 21, unveils the presence of distinct spikes every 7 days confirms a weekly recurring pattern in the data. This observation suggests that cycles of 7-day intervals provide valuable insights for forecasting. Consequently, lags that are multiples of 7 can capture this

weekly seasonality and should be included as relevant features in the model indicate a cyclical weekly pattern.

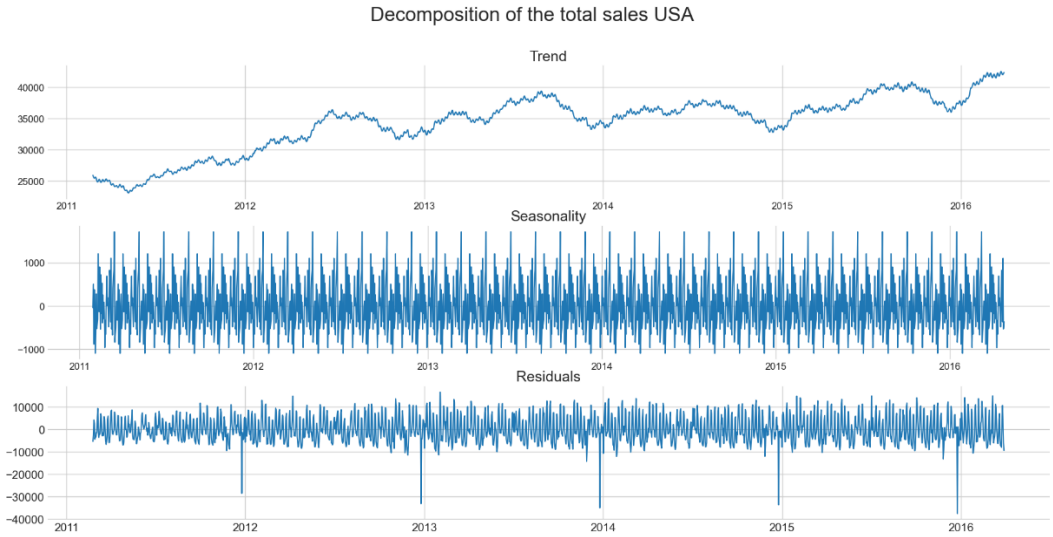


Figure 21 Sales Decomposition at Country Level.
Source: Own representation.

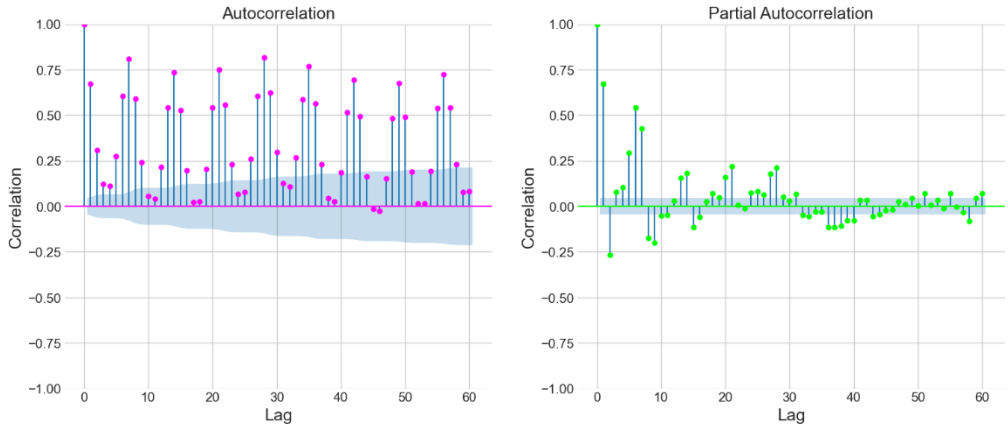


Figure 20 Autocorrelation and Partial Autocorrelation of M5 Data
Source: Own representation

Incorporating additional calendar features can significantly improve the performance of the forecasting model by accounting for specific sales patterns tied to holidays and major events. For instance, events like the Super Bowl or holidays such as Christmas often lead to predictable spikes in sales, as illustrated in Figure 22. Using such event-related data as covariates allows the model to anticipate these variations, enhancing the forecast accuracy. Sales often show notable increases both before and after these events, a pattern that is captured in the chart by spikes above

the weekly moving average. I construct a dummy variable to capture all days with sales spikes, allowing the model to account for the impact of sudden increases in sales associated with specific events or patterns. By including this variable, the model can distinguish between regular daily fluctuations and significant spikes, further improving forecast accuracy during high-demand periods.

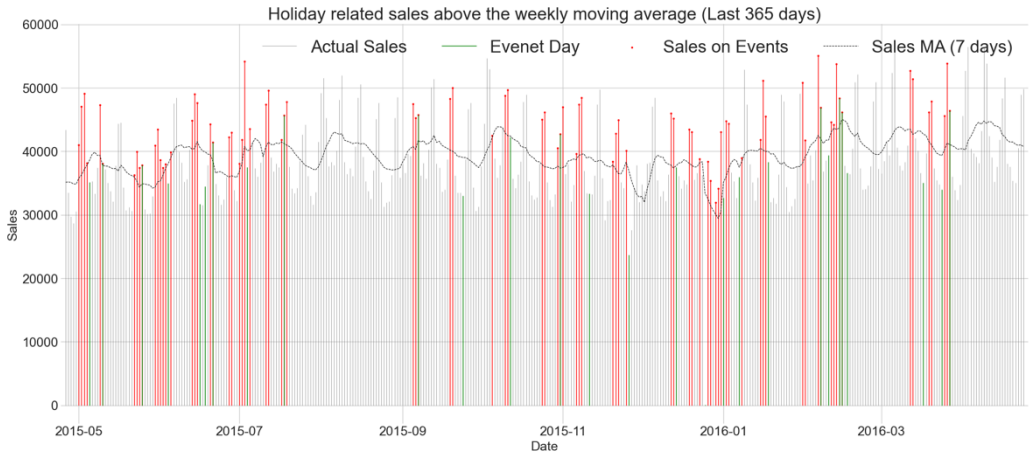


Figure 22 Holiday and Event-related sales spikes.
Source: Own representation

4.3 Model selection

For the M4 dataset, I start with a drift model, a variation of simple naïve methods such as the seasonal naïve model. The drift method allows the predicted values to vary over time by adding a drift term, which is equal to the average change observed in the data (Hyndman and Athanasopoulos, 2018).

As for statistical models, I use the Holt-Winters approach which is a triple exponential smoothing. A simple yet powerful model with low computational costs that can offer a comparable or even better performance than complex models, depending on the data, as reflected by the M4-competition results (Makridakis et al., 2018). Additionally, I include the well-known ARIMA model and rely on Auto-ARIMA to automatically select the best parameters due to the large number of time series. The first machine learning model I employ is LightGBM (Ke et al., 2017), which is an improvement on the already efficient extreme gradient boosting model XGBoost (Chen and Guestrin, 2016).

LightGBM's efficiency and scalability make it preferable to bagging methods such as Random Forests (Sheridan et al., 2016) particularly in high-dimensional data with engineered features.For deep learning (DL) models, I include one representative model from each architecture category. Since the M4 dataset does not exhibit constant seasonality, I choose BlockLSTM from Darts, which employs an encoder-MLP decoder structure rather than a traditional encoder-decoder LSTM. BlockLSTM is based on the architecture proposed by Wen et al. (2017), who show that this architecture is particularly suitable when dealing with shifting seasonality.

For MLP-based models, I select NHITS instead of NBEATS due to its hierarchical interpolation capabilities and multi-rate sampling mechanism, which allows it to handle diverse frequency components effectively. For CNN-based models, I select the vanilla TCN due to its causal convolutions and residual connections. Bai et al. (2018) highlight that the use of dilated causal convolutions and residual connections allow TCN to capture long-term dependencies effectively while avoiding overfitting. The limited receptive field is actually advantageous in this case, since short-term fluctuations are far more influential on future predictions than long-term dependencies for daily finance data. Models with broader receptive fields might learn less meaningful older patterns, hence introducing noise. By focusing on a shorter horizon, the model prioritizes relevant, short-term trends, ensuring more accurate forecasts. For the transformer-based model, I employ the vanilla Transformer architecture (Vaswani et al., 2017) which offers a simpler alternative to more complex variants. The model's ability to handle both short- and long-term dependencies without excessive computational costs makes it suitable for datasets without constant seasonality, as is the case here. It should be noted that, Darts offer less implementations of DL models than Nixtla, however, for consistency, I choose the best possible model from each architecture family.

For the M5 dataset, I include Holt-Winters and Auto-ARIMA as statistical baselines. LightGBM is also applied here due to its effectiveness with engineered features. Among DL models, I select LSTM for its ability to capture recurrent

patterns. NBEATS is chosen for its architecture, which consists of two stacks to capture trend and seasonality, making it a more suitable choice than NHITS. As for the CNN-based model, I use TimesNet, which extends the TCN architecture and solves its shortcomings by increasing the receptive field. Additionally, its ability to capture patterns within and between time series by transforming 1D data into a 2D format, make it suitable for hierarchical data, where seasonality and trend may vary across different levels of granularity.

As discussed in Section 3.3.4 transformer-based models vary in how they process and structure input data, with different approaches offering distinct advantages depending on the dataset's characteristics. Given the scale of the M5 dataset, I consider a model with patch-wise technique, which divides the time series into patches before feeding it into the model, to be the most suitable. This approach allows the model to capture local dependencies, such as weekly seasonality, within each patch, while reducing the computational costs of the attention mechanism compared to the standard point-wise method. This enhances the comparability of the models, since I am factoring in training time as a critical metric. PatchTST represents a strong candidate for my use-case, due its channel-independence setup processes each time series independently using patches, which allows to learn local as well as global patterns using the self-attention mechanism to capture dependencies between patches. A summary of the model used is shown in Table 2.

Table 2 Summary of the data used in my case studies.

Model Category	M4 daily finance data	M5 daily sales data
Statistical models	Exponential Smoothing (Holt-Winters) ARIMA	
ML	LightGBM	
DL (RNN)	Block LSTM	LSTM
DL (CNN)	TCN	TimesNet
DL (MLP)	NHITS	NBEATS(x)
DL (Transformer)	Base Transformer	PatchTST

4.4 Experimental setup and tuning

In Darts, the TimeSeries class is used to construct one object from multiple univariate or multivariate time series, whereas with hierarchical data, Nixtla transforms the data into a long format. I use a DataFrame[11] to construct a TimeSeries object with the first column being temporal information (datetime), which is an essential requirement. The TimeSeries object has three dimensions, time, component, and sample. The time index represents temporal information about the data, and it works with sequential data with or without known time stamps. The component dimension refers to the time series, which are the columns of the DataFrame in my case. In Nixtla, however, data is required in a "long" format, where each time series observations are represented as a single row with a unique identifier, timestamp and the observed value. Thus, a DataFrame in Nixtla has three main columns: Unique_id, ds, y. In this global approach, unique_id is used to identify the observations of a specific time series. After training a model on the data, reconciliation methods can be applied to achieve coherence predictions.

The forecasting horizon H is the suggested horizon for this frequency in the M4 competition, which is 14 days. Given the short forecasting horizon and for better comparability with autoregressive models, I choose a recursive approach for all models. All DL models are trained for a maximum of 100 epochs. Early stopping with a patience of 10 epochs is implemented. To ensure reproducibility, a random state of 42 is used for all the models. For some DL models the hyperparameters search method, Optuna (Akiba et al., 2019), can find the best combinations, while for others I find manual tuning to lead to better performance. After testing various values for the input size, the best input size seems to be ranging between 2-4 for the M4 dataset study and 3 or 6 for the M5 dataset, hence between 28-64 and 84 or 168

[11] A class provided by the Pandas package to store tabular data. For more see: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

respectively. As a batch size, I find the best values to be between 32 or 64 for all models except for BlockLSTM which is trained with a batch size of 128.

Recursive vs. direct multistep forecasting

ML models allow flexibility to perform forecasting using two different strategies. The first one is an iterative approach by predicting one step at the time, the model uses the predicted value is an input for the next. This is comparable to an autoregressive model. However, as the forecasting horizon increases, the accuracy of the model decreases due to the accumulation of errors from using predicted values as inputs to predict the next step. For longer horizons, an alternative approach can be used to overcome the shortcomings of the recursive forecasting. This approach directly forecasts all of the steps at once by building h-step-ahead models to predict the steps from y_t to y_{t+H} for a time horizon H (Ben Taieb and Hyndman, 2012).

4.5 Evaluation metrics

There are multiple commonly used metrices to evaluate the performance of forecasting models with each having its advantages and limitations. The presented metrics are thoroughly discussed in Hyndman and Athanasopoulos (2018). One of the widely used metrices is calculated is the Root Mean Square Error (RMSE), which is calculated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}, \tag{36}$$

Using quadratic terms in these metrices making them sensitive to outliers, which can be advantageous for optimizing models to minimize the deviations (Hyndman and Koehler (2006). However, Chai and Draxler (2014) highlight that RMSE can unveil the performance difference in the data, due to its sensitivity to the magnitude of the data. A scale-invariant alternative is the Mean Absolute Error (MAE), which offers straightforward interpretability and robustness to outliers. MAE calculates the average of the absolute difference between the actual and predicted values.

$$MAE = \frac{1}{n}|y_t - \widehat{y_t}|, \tag{37}$$

One downside of this metric is that it is scale-dependent, hence when dealing data with different magnitudes, the interpretation can be difficult. A solution to this issue is to extend the metric by calculating the percentage error, which helps standardize the results and improve interpretability across different scales. The Mean Absolute Percentage Error (MAPE) is calculated as follows:

$$MAPE = \frac{1}{n}\left|\frac{y_t - \widehat{y_t}}{\widehat{y_t}}\right| \times 100, \tag{38}$$

A third possibility, besides scale-dependent and percentage errors, is to compute scaled errors. Hyndman and Koehler (2006) propose the Mean Absolute Scaled Error (MASE), which scales the data based on the MAE of in-sample seasonal naïve forecasts. MASE is calculated as follows:

$$MASE = \frac{1}{h}\frac{\frac{1}{h}\sum_{t=1}^{h}|y_t - \widehat{y_t}|}{\frac{1}{n-m}\sum_{t=m+1}^{n}|y_t - y_{t-m}|}, \tag{39}$$

Where:

y_t is the actual values of the time series at point t.

$\widehat{y_t}$ is the forecasted values.

h is the forecasting horizon.

m is the frequency of the data (i.e. 12 for monthly series).

n is the number of observations.

To provide a comprehensive understanding of the models' performance, I employ three different error measures: the scale-dependent RMSE, the percentage-based MAPE, and a scaled error measure. Additionally, considering that the global model approach is designed for scalability, I monitor the training times for both local and global models.

5 Results and analysis

In this section, I present the results of both case studies and highlight the strengths and limitations of the models used. The primary findings reveal that global models can achieve performance on par with local statistical models while considerably reducing training time. Additionally, outperforming the statistical baseline is only achieved by combining different forecasting methods or models.

5.1 Local and global models (M4 dataset)

A simple naïve model, naïve drift, provides good predictions by achieving a MAPE of 4.50%. The ARIMA model slightly outperforms Holt-Winters, with MAPE scores of 4.15% and 4.53%, respectively. More sophisticated machine learning and DL models perform fall behind statistical models, with the worst being TCN, which has a MAPE of around 5% and an RMSE just under 300. Although ARIMA achieves the lowest MAPE, NHITS delivers lower errors in terms of RMSE and MASE. These results suggest that NHITS is more effective at capturing overall patterns across the dataset, while ARIMA is more accurate on time series with lower magnitudes or less volatility.

To explore the potential of global models, I train the same ML and DL locally by training a model for each time series. The global versions of these models achieve better results, with the only exception of LSTM, while significantly reducing the training time. For instance, the global version of NHITS reduces the training time by 10 folds, while reducing the MAPE by 13.5% (see Table A *1*). In all cases, the training time of global models is significantly less than local models with LightGBM being the only exception. This can be explained by the fact that the global model is trained with 100 times more variables than the local model, resulting in a huge increase in the number of variables. While DL models can handle larger datasets if tuned correctly, overly complex models risk overfitting, which limits generalization to out-of-sample data.

Table 4 summarizes the results. It includes 8 single models, whereas naïve drift, Holt-Winters, and ARIMA serve as baseline models. Additionally, I test if a an ensemble of different models using the 8 models with the recursive approach can outperform the best single model. A total of 28 ways exist to build an ensemble of two global models. Following Wolpert (1992), stacked generalization approach, I use a linear regression model as the meta-learner to combine the outputs of the base models. Additionally, I consider building an ensemble using a simple averaging of predictions. This doubles the number of possible combinations by 2, hence 56 combinations are possible. To explore further possibilities, I re-train each global model using a direct forecasting approach instead of recursive, then combine the outputs of the recursive and direct methods to build 2 different types of ensembles, like building an ensemble of different models. These models are referred to as "combi" in Table 4. When extended to ensembles of three models, the number of possible combinations grows to 56 unique pairs (i.e., $56 \times 2 = 112$), and including the two ensemble combinations yields a total of 168 ensembles.

A simple averaging of the naïve drift predictions and those from the hybrid LSTM model achieves the lowest error, with a MAPE of 3.36%, reducing the initial LSTM model's MAPE of 4.84% by over 40%. The best-performing two-model ensemble combines a simple statistical model, naïve drift, with the combined version of LSTM, hence improving over the performance of the same ensemble, but with the recursive version of LSTM. Adding a third model to this combination using a meta-learner does not enhance the performance, however this ensemble model, with naïve drift, LSTM and NHITS is the best 3-model ensemble.

Figure 23 illustrates how the combination of the two forecasting models, recursive and direct, for LSTM enhances the performance. The recursive forecasting model is good at capturing the trend, however due to the autoregressive style, the model misses to capture short-term fluctuations leading to smooth predictions as illustrated in the figure. A direct multi-step approach predicts each step directly.

Figure *23* Best-performing Hybrid Model

Table 3 Results of local and global forecasting models.

This table shows the results of local and global forecasting models abbreviated as LM (Local Model) and GM (Global Model). Local models are trained individually for each time series using a loop, while global models are trained on the whole dataset, treating all 100-time series as one TimeSeries object. DL models are tuned with Optuna for 50 trials to identify the optimal hyperparameters. The table is structured into four sections: individual models (3 local and 5 global), 5 Combi models, and the best ensemble model. The plus sign indicates an ensemble of these two models. The bold values indicate the best overall results, whereas the underlined values are the best in the section.

Model	RMSE	MAPE	MASE	Training Time (s)
Global naïve drift (GND)	282.12	4.50	1.58	0.28
Holt-Winters	273.43	4.53	1.65	29.12
ARIMA	263.98	<u>4.15</u>	1.68	506.63
LightGBM	297.4	4.82	1.58	560.58
LSTM	274.59	4.84	1.57	41.69
TCN	298.18	5.01	1.5	89.92
NHITS	<u>256.69</u>	<u>4.26</u>	<u>1.39</u>	216.7
Transformers	282.56	4.61	1.57	174.97
Transformer_Combi	267.88	4.26	1.51	266.38
LSTM_Combi	<u>225.93</u>	<u>3.46</u>	<u>1.22</u>	88.86
TCN_Combi	298.63	4.59	1.64	177.77
NHITS_Combi	258.0	4.01	1.42	220.5
Transformer_Combi	267.88	4.26	1.51	266.38
LSTM+GND	235.45	3.73	1.29	41.97
LSTM_Combi+GND	**222.14**	**3.36**	**1.19**	89.11
GND+LSTM+NHITS (Reg)	235.4	3.57	1.27	0.49

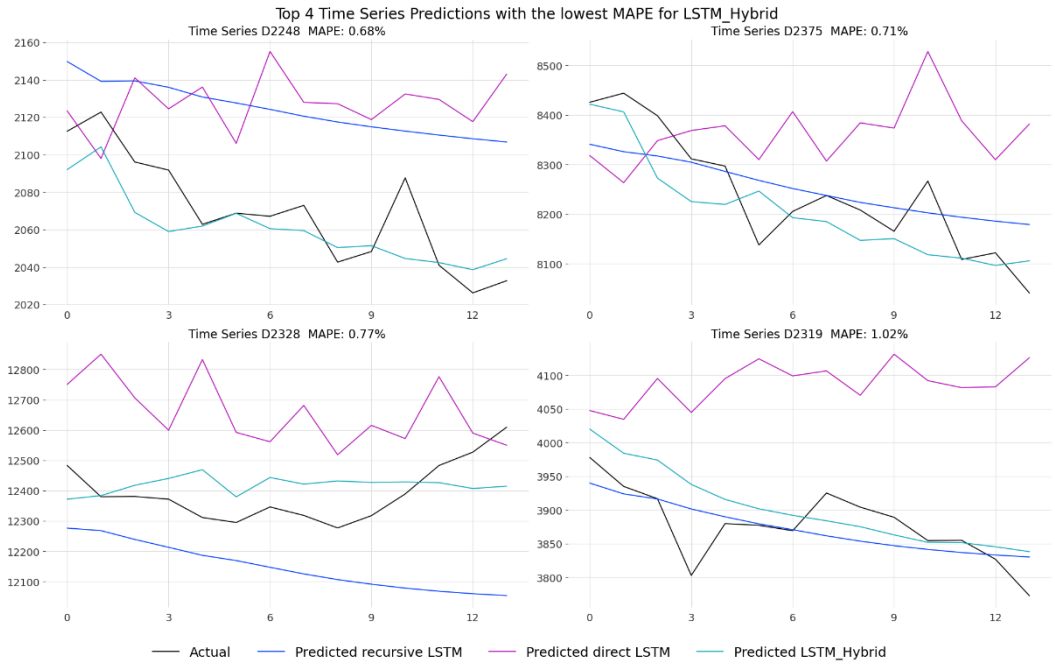


Figure 23 Best-performing Hybrid Model
Source: Own representation.

The goal of this operation is to show how just by combining two forecasting methods, recursive and direct, of the same model without changing the returning the model, the accuracy can be increased. By combining the two methods using a simple regression model as a meta learner, I can leverage the strengths of each method and mitigate the smoothness of recursive forecasting, thus achieving a superior performance with lower prediction errors.

5.2 Hierarchical forecasting (M5 dataset)

In this Section I present my results for the hierarchical forecasting study. First, I start by illustrating the problem of incoherence mentioned in Section 2. For the state of California, I calculate the sum of forecasts predicted by ARIMA at each level. As seen in the figure, the lower the level the higher the deviation from the top level with the forecast at the bottom level deviating by 2% on average. Therefore, reconciliation is essential to achieve coherence forecasts.

In the following, I present the results of four different reconciliation approaches, namely the two classical ones: bottom-up, top-down and two alternatives, middle-

out and optimal. I focus on comparing the MAPE of the models, presented in Table 4, while RMSE measures are reported in appendix (see Table A ***4***). Middle-out proves to be the best across the models and levels by consistently providing the lowest error across the models. Two models have almost identical errors, with LightGBM having the third best overall MAPE of 11.28%, while TimesNet providing a minimal improvement with 11.26% using the middle out reconciliation method. However, when applying the optimal reconciliation method, LightGBM can outperform TimesNet with a MAPE of 11.23%. The best performing model overall with a MAPE of 11.14% is NBEATS, which provides consistent solid performance across all levels.

When examining the errors across the hierarchical levels, LightGBM outperforms NBEATS at the top 3 levels. At the country level, LightGBM achieves a 13% improvement compared to NBEATS and a 50% improvement over the best-performing statistical model, Holt-Winters, with a MAPE of just under 3%. The errors increase across the borders, as the granularity increases. On the bottom level, department level, LightGBM achieves almost identical performance compared with the best model NBEATS, with 13.36% and 13.35% respectively. However, LightGBM struggles at the category level LightGBM is coming at 4th place after TimesNet, NBEATS and PatchTST using the middle out performance, while the optimal method yields a slightly lower MAPE for LightGBM. Overall LightGBM is offering competitive performance, especially considering the training time. When comparing the overall MASE, however, LightGBM is only outperforming LSTM from the DL models (see Figure A3). NBEATS, on the other hand, is still dominating by achieving the lowest overall MASE and lowest training time. LSTM is overall the slowest model with the highest errors, which can be attributed to its recursive forecasting approach. When compared to other transformer models, the reduction of training time is noticeable in the PatchTST architecture. PatchTST achieves a MASE of 0.50 compared to 0.49 and 0.487 for TFT and Informer,

respectively (see Figure A3). However, the training time is 80% (90%) less compared to TFT (Informer) which can be explained using patch-wise processing.

Table 4 Out-of-Sample MAPE for Hierarchical Forecasting Approaches

level	Holt-Winters	ARIMA	Light-GBM	LSTM	Times-Net	NBEATS	Patch-TST
Level 1: Country							
Top-Down	5.78	6.68	4.90	4.91	3.57	3.35	4.02
Bottom-Up	5.72	6.44	3.73	5.24	3.44	3.41	4.02
Middle-Out	5.90	6.23	**2.93**	4.98	3.39	<u>3.10</u>	3.84
Optimal	5.77	6.50	3.35	5.06	3.42	3.19	3.85
Level 2: State							
Top-Down	7.14	7.13	5.65	5.98	4.73	4.93	5.19
Bottom-Up	7.38	6.94	5.46	6.33	4.54	4.99	5.04
Middle-Out	7.08	6.84	**4.32**	5.87	<u>4.54</u>	4.77	4.99
Optimal	7.16	7.00	4.53	6.03	4.55	4.84	5.04
Level 3: Store							
Top-Down	7.87	7.94	6.53	7.07	5.70	6.00	5.96
Bottom-Up	8.12	7.76	6.54	7.33	5.61	6.08	5.90
Middle-Out	7.92	7.57	**5.70**	7.04	5.57	<u>5.91</u>	5.81
Optimal	7.88	7.77	5.58	7.10	5.61	5.95	5.85
Level 4: Category							
Top-Down	10.24	10.92	9.65	10.31	8.52	8.75	9.10
Bottom-Up	10.37	10.89	9.79	10.49	8.48	8.75	9.15
Middle-Out	10.29	10.82	9.25	10.25	**8.45**	<u>8.62</u>	9.01
Optimal	10.30	10.51	8.99	10.44	8.45	8.72	9.13
Level 5: Department							
Top-Down	18.80	16.16	13.56	14.81	13.67	13.40	14.06
Bottom-Up	19.02	15.84	13.66	15.01	13.69	13.35	14.00
Middle-Out	18.74	15.89	<u>13.36</u>	14.80	13.66	**13.35**	14.03
Optimal	18.96	16.57	15.03	15.30	13.86	13.55	14.33
Overall							
Top-Down	15.17	13.74	11.63	12.63	11.29	11.22	11.72
Bottom-Up	15.36	13.51	11.71	12.83	11.28	11.20	11.69
Middle-Out	15.15	13.50	11.28	12.60	<u>11.25</u>	**11.14**	11.66
Optimal	15.28	13.86	**12.23**	12.96	11.38	11.29	11.88

5.3 Model comparison and discussion

The performance differences observed between simpler statistical models and complex DL models are largely attributable to the characteristics of the training data. Financial daily data, for instance, often reflects short-term fluctuations influenced by market news, which may not correspond to intrinsic patterns, such as seasonality. This lack of inherent structure means that simpler models, like naïve methods, often perform well, indicating that less complex architecture is more suitable for noisy data. However, when the data exhibits clear, recurrent patterns, complex models tend to outperform, as shown in Section 5.2.

The limitations of individual models can be mitigated by combining multiple models, as demonstrated by the winner of the M4 competition, who leveraged exponential smoothing and RNNs to create a hybrid model that utilizes both local and global forecasting approaches (Smyl, 2020). My findings align with the results of the M4 and M5 competitions. Statistical models are difficult to outperform and ensembles, as shown in Section 5.2, are superior to single models (Makridakis et al., 2020). Additionally, I show that combining recursive- and direct-multistep predictions to achieve superior performance compared to using one of these techniques, which is consistent with the findings of Ben Taieb and Hyndman (2012).

In Section 5.2, LightGBM demonstrates a strong balance between performance and efficiency, offering a competitive MAPE and minimal error at the highest aggregation level (country level). These results echo findings from the M5 competition, where boosting models have shown superior performance (Makridakis et al., 2022). A key factor in the popularity of boosting models is their rapid training time, which enables quicker model improvements, as Januschowski et al. (2022) argue in their discussion about forecasting with trees.

In contrast to classification tasks, where historical data is used to categorize new inputs, time series forecasting presents unique challenges due to the inherent

unpredictability of temporal changes. Forecasting in the absence of clear constant seasonality, for example, is much more complex than predicting data with regular, recurrent patterns. In both scenarios, a global approach that leverages multiple time series for pattern learning can be advantageous. With increasingly large datasets available, DL models are particularly well-suited for such tasks, achieving strong results when appropriately tuned.

While powerful, single models are limited by their inherent weaknesses. To mitigate these limitations, combining multiple models, such as statistical and DL approaches, or integrating forecasting strategies like recursive and multi-step forecasting, can enhance overall model performance as demonstrated in the M5 competition (Makridakis et al., 2022).

Predicting sequential data becomes more manageable when it's embedded within a text, as model accuracy can significantly improve with access to extensive text data. However, TSF lacks this simplicity; it is often influenced by external factors like holidays, events, or economic indicators such as inflation or interest rates. In this study, I demonstrate that more advanced machine learning and deep learning models are not always superior; in certain cases, even a straightforward naïve forecast outperforms complex methods. This underscores the importance of a deep understanding of both the dataset and the forecasting context to select the most suitable model for each problem. Combined forecasting methods or models is a valuable solution for overcoming model and methods limitations and boosting predictive accuracy. Moreover, the trade-off between model accuracy and computational cost is a crucial consideration, as slight performance improvements can often come at the expense of significantly increased processing time, which is the case with many transformer-based models due to the associated quadratic complexity $O(N^2)$ as Oliveira and Ramos (2024) discuss. Even when using patching, transformers fail to achieve the efficient performance of MLP models like NBEATS (see Figure A1). These findings are consistent with Oliveira and Ramos (2024) who use the M5 dataset as a benchmark to test different transformer models

and show that TFT and Informer are achieving better performance than PatchTST, while this comes at a higher computational cost.

The recent development of Kolmogorov-Arnold Networks (KANs) (Liu et al., 2024), which utilizes learnable activation function, introduces an alternative to MLPs by eliminating the need for linear weight. Additionally, KANs introduce interpretable architecture. Building on this novel approach, Xu et al. (2024) show how KANs' strengths can be leveraged to be used for TSF providing good accuracy, while addressing the interpretability challenges traditional MLPs struggle with.

Recent introduction of the state space model Mamba (Gu and Dao, 2023) a potential alternative to transformer models with a significant performance boost in processing sequential data. This is achieved without the attention mechanism; therefore, the complexity is substantially reduced with a near-linear complexity. An attempt to modify this architecture for TSF is simple Mamba (S-Mamba) by Wang et al. (2024) who demonstrate a superior performance of their model compared to other SOTA transformers-based models such as iTransformer and PatchTST, while noticeably reducing the training and inference time, thus providing a better scalability then transformer-based models with the attention-mechanism.

6 Conclusion and future work

In this work, I demonstrate that more advanced ML and DL models are not always superior; in some cases, a simple naïve forecast may be the most effective option. However, it all depends on the nature of the data, therefore it is crucial to have a good understanding of the data and forecasting context to find the best model for the specific problem. Once an initial model is chosen, exploring combinations of different forecasting approaches or models can help mitigate the model's limitations and improve prediction accuracy.

It is important to consider the trade-off between accuracy and computational cost when selecting a model, as some configurations offer only slight performance gains at a significantly higher computational expense. This challenges the purpose of a global approach, which aims to deliver better scalability at a lower cost compared to training separate models for each time series. Given recent advancements in time series forecasting (TSF) literature, it would be valuable to compare current SOTA models with newer architectures like Mamba or KANs. Additionally, balancing performance and computational cost is especially relevant when efficiency is a critical criterion, such as with large datasets. As the demand for accurate, scalable forecasting solutions grows, a combination of model interpretability, robustness to data variability, and computational efficiency will be key criteria guiding the choice of models in both academic and applied settings.

Appendix

Table A 1 Comparison of local and global forecasting approaches

Local Models							
	Baseline and statistical models		Machine and deep learning models				
metric	Holt-Winters	ARIMA	Light-GBM	LSTM	TCN	Nhits	Trans-former
RMSE	273.43	285.52	390.32	270.76	269.82	305.86	435.56
MAPE	4.35	4.21	6.81	4.40	4.42	5.14	6.77
MASE	1.51	3.9	5.53	5.97	1.55	3.97	2.52
Time	29.12	684.39	42.72	1764.58	1613.97	1393.84	4273.45
Global Models							
metric	Global Naïve Seasonal	Global Naïve Drift	Light-GBM	LSTM	TCN	Nhits	Trans-former
RMSE_R	321.32	265.66	297.67	283.3	298.18	274.83	282.56
RMSE_D			342.79	458.20	424.09	379.11	437.79
MAPE_R	5.35	4.08	4.65	4.84*	5.01	4.53	4.61
MAPE_D			6.0	8.84	7.55	6.79	8.23
MASE_R	1.89	1.44	1.61	1.57	1.50	1.5	1.57
MASE_D			2.01	2.78	2.32	2.27	2.80
Time_R	0.24	0.25	142.18	43.99	89.92	149.79	139.84
Time_D			142.85	44.87	87.85	93.03	137.05

Note that the stated training time is the total time required to train the model while also checkpointing to save the tensorboard data and retrieve the best to predict using the best performing model. This aims to reduce overfitting, thus enhancing the performance on OOS. See Append table xy to see the performance of the models without early stopping and checkpoints. Statistical and ML models are trained on the CPU, therefore a direct comparison of training time with DL models is not possible, since DL models are trained on a GPU. The used hardware's are Ryzen 2600 processor with 6 cores and a NVIDIA RTX 2070.

*Without early stopping and checkpoints, the MAPE is almost double with a value of 10.83 which is an indication of overfitting.

Table A2 Data Model Hyperparameters (M4)

Model Name	Input Parameters	Training Parameters
Light GBM	Input: [-1, -2, -3, -4, -7, -14, -21, -28] Cyclic encoders: [month, week, day-of-week] Relative position of past values	Estimators: 500 Learning rate: 0.1 Min split gain : 0.0001 Max depth: 6
Block LSTM	Input: 28 RNN layers: 4, Hidden dimension: 128	Batch size: 128 Learning rate: 0.002
TCN	Input: 28 Kernel size: 14 Number of filters: 96	Batch size: 32 Learning rate: 0.001
NHITS	Input: 42 Stacks: 2 Blocks: 1 Layers: 2 Layer width: 128	Batch size: 64 Learning rate: 0.001
Transformer	Input: 56 Model dimension: 88 Heads: 8 Encoder layers: 1 Decoder layers: 3 Activation function: ReLU	Batch size: 32 Learning rate: 0.001

Table A3 Data Model Hyperparameters (M5)

Model Name	Input Parameters	Training Parameters
LightGBM	Lags: 28 Date features: day-of-week, month, quarter, year Seasonal Rolling Mean (28-day window, 7-day season length) and rolling standard deviation with lagged values of: 1, 7, 14, 28 Rolling means: 7 days, lag of 7 14 days, lag of 14 28 days, lag of 28 90 days, lag of 1	No. Estimators: 2500 Learning rate: 0.1 Max depth: 12, No. leaves: 500 Min child samples: 75 Colsample bytree: 0.71
LSTM	Input size: 168 Encoder hidden size: 300 Encoder layers: 2 Decoder hidden size: 512 Context size: 5	Learning rate: 0.00055 Max steps: 1000 Batch size: 32
TimesNet	Input size: 168 Hidden size: 16, Convolution hidden size: 7	Learning rate: 0.001, Batch size: 64, Max steps: 400, Step size: 1
NBEATS	Input size: 168 Stack types: ['identity', 'trend', 'seasonality']	Batch size: 64 Max steps: 600
PatchTST	Input size: 84 Patch length: 16 Hidden size: 256, Heads: 16	Learning rate: 0.0006, Revin: True, Max steps: 1000 Batch size: 32 Windows batch size: 1024 Random seed: 17

Note: For M5 data, statistical models and LSTM use the recursive forecasting approach, while other models use the direct approach.

Table A 4 Out-of-Sample RMSE for Different Hierarchical Forecasting Approaches

level	Holt-Winters	ARIMA	Light-GBM	LSTM	Times-Net	NBEATS	Patch-TST
Level 1: Country							
Top-Down	3042.53	3697.51	2565.30	2771.67	1967.73	1953.02	2143.85
Bottom-Up	3023.24	3520.74	2148.09	2831.71	1908.52	1906.99	2249.44
Middle-Out	3120.26	3430.46	**1732.37**	2744.01	1880.85	1851.70	2074.08
Optimal	3037.85	3583.99	1859.51	2780.26	1907.24	1874.95	2130.79
Level 2: State							
Top-Down	1293.26	1367.26	1037.59	1126.34	839.97	859.58	914.44
Bottom-Up	1311.66	1299.78	986.78	1172.68	822.28	870.39	961.53
Middle-Out	1296.42	1280.62	837.37	1108.71	**814.01**	829.86	887.83
Optimal	1296.11	1323.99	843.55	1132.14	822.13	841.29	909.49
Level 3: Store							
Top-Down	451.07	455.91	368.87	402.96	310.59	318.18	324.30
Bottom-Up	462.15	440.57	358.32	418.02	305.51	317.89	337.40
Middle-Out	455.31	433.72	322.25	397.17	**303.50**	311.87	318.26
Optimal	452.59	445.46	315.93	404.39	305.75	313.65	322.85
Level 4: Category							
Top-Down	184.27	181.35	153.42	166.47	133.56	135.69	138.04
Bottom-Up	187.40	180.84	151.03	171.06	132.47	134.55	140.72
Middle-Out	185.17	179.13	141.25	165.15	**131.80**	132.84	136.57
Optimal	184.85	179.68	140.37	166.95	132.35	134.03	137.80
Level 5: Department							
Top-Down	102.95	92.15	77.23	84.03	70.21	70.68	72.52
Bottom-Up	104.52	92.67	76.55	85.54	69.71	70.38	73.92
Middle-Out	103.66	92.03	73.18	83.59	**69.51**	69.67	71.95
Optimal	103.26	92.12	73.96	84.30	69.93	70.18	72.60
Overall							
Top-Down	212.00	212.71	169.96	184.70	144.87	146.77	152.17
Bottom-Up	215.07	208.23	162.99	189.91	142.84	146.14	157.05
Middle-Out	213.81	205.49	147.61	182.87	**141.91**	**143.18**	149.60
Optimal	212.50	209.20	148.57	185.35	142.95	144.46	151.79

Figure A1

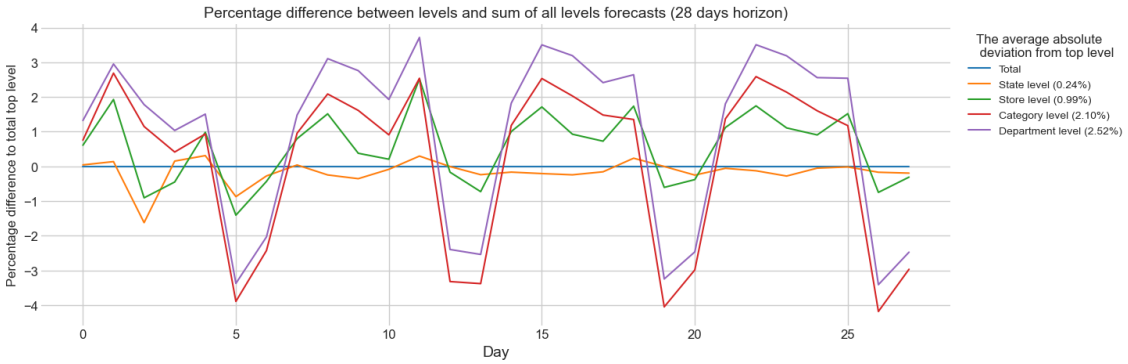


Figure A1 Forecast Incoherency Across Hierarchical Levels Without Reconciliation

Figure A2

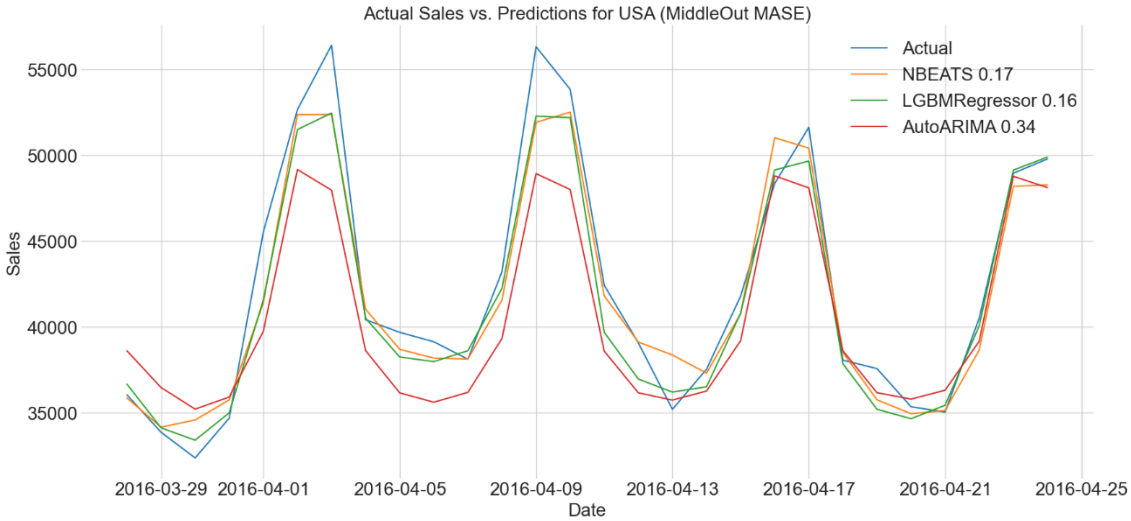


Figure A2 Comparison of Actual Sales vs. Predictions for USA

TimesNet as Overall Best and LightGBM as Country-Level Top Performer

Source: Own representation.

Figure A3

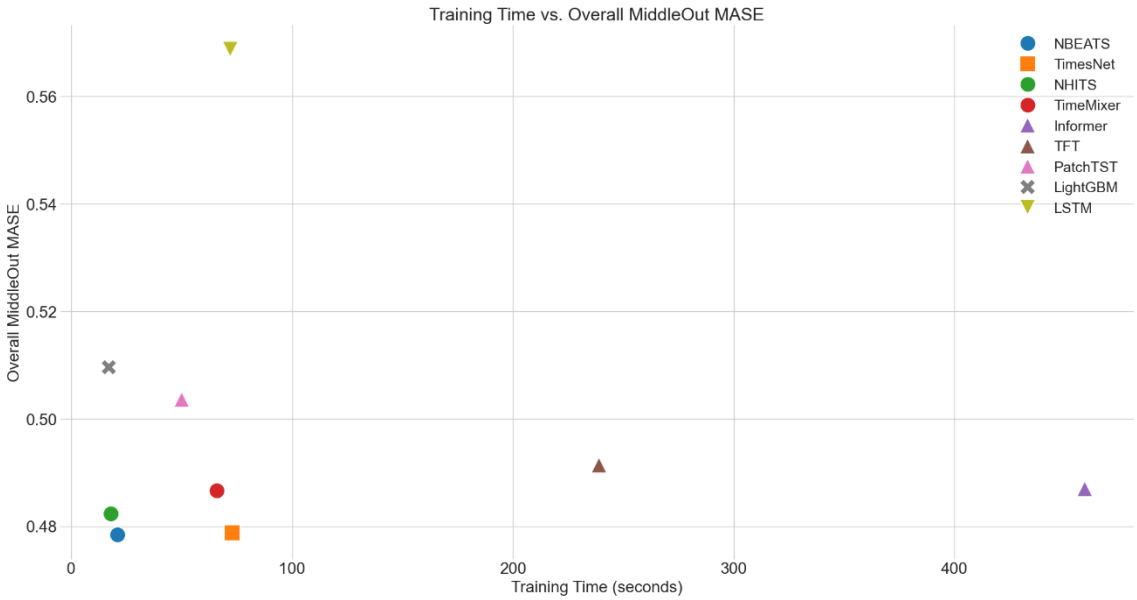


Figure A3 Plot of the training time and achieved MASE score.

Models of the same architecture family are represented with the same shape. All the models are trained using NIVIDIA RTX 2070 GPU. The only exception is LightGBM which is trained on 2600 Ryzen CPU with 6 cores. Therefore, a direct comparison of training time cannot be drawn.

Source: Own representation.

Reference List

Akiba, T., S. Sano, T. Yanase, T. Ohta, and M. Koyama, 2019, Optuna: A next-generation hyperparameter optimization framework, Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining.

Athanasopoulos, G., R. A. Ahmed, and R. J. Hyndman, 2009, Hierarchical forecasts for australian domestic tourism, *International Journal of Forecasting,* 25(1), 146-166.

Bahdanau, D., 2014, Neural machine translation by jointly learning to align and translate, *arXiv preprint arXiv:1409.0473*.

Bai, S., J. Z. Kolter, and V. Koltun, 2018, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, *arXiv preprint arXiv:1803.01271*.

Bandara, K., C. Bergmeir, and S. Smyl, 2020, Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach, *Expert systems with applications,* 140112896.

Ben Taieb, S., and R. J. Hyndman, 2012. *Recursive and direct multi-step forecasting: The best of both worlds* (Department of Econometrics and Business Statistics, Monash Univ.).

Ben Taieb, S., J. W. Taylor, and R. J. Hyndman, 2017, Coherent probabilistic forecasts for hierarchical time series, International Conference on Machine Learning (PMLR).

Bontempi, G., S. Ben Taieb, and Y.-A. Le Borgne, 2013, Machine learning strategies for time series forecasting, *Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures 2,* 62-77.

Bottou, L., 1991, Stochastic gradient learning in neural networks, *Proceedings of Neuro-Nımes,* 91(8), 12.

Box George, E., and M. Jenkins Gwilym, 1976, Time series analysis: Forecasting and control, *San Francisco: Holden Bay*.

Breiman, L., 1996, Bagging predictors, *Machine learning,* 24123-140.

Breiman, L., 2001, Random forests, *Machine learning,* 455-32.

Breiman, L., J. Friedman, R. Olshen, and C. Stone, 1984, Classification and regression trees.

Brown, R. G., 2004. *Smoothing, forecasting and prediction of discrete time series* (Courier Corporation).

Chai, T., and R. R. Draxler, 2014, Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature, *Geoscientific model development,* 7(3), 1247-1250.

Challu, C., K. G. Olivares, B. N. Oreshkin, F. G. Ramirez, M. M. Canseco, and A. Dubrawski, 2023, Nhits: Neural hierarchical interpolation for time series forecasting, Proceedings of the AAAI conference on artificial intelligence.

Chen, S.-A., C.-L. Li, N. Yoder, S. O. Arik, and T. Pfister, 2023, Tsmixer: An all-mlp architecture for time series forecasting, *arXiv preprint arXiv:2303.06053*.

Chen, T., and C. Guestrin, 2016, Xgboost: A scalable tree boosting system, Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.

Cho, K., 2014, Learning phrase representations using rnn encoder-decoder for statistical machine translation, *arXiv preprint arXiv:1406.1078*.

Cleveland, R. B., W. S. Cleveland, J. E. McRae, and I. Terpenning, 1990, Stl: A seasonal-trend decomposition, *J. off. Stat,* 6(1), 3-73.

Freund, Y., and R. E. Schapire, 1996, Experiments with a new boosting algorithm, icml (Citeseer).

Friedman, J. H., 2001, Greedy function approximation: A gradient boosting machine, *Annals of statistics,* 1189-1232.

Goodfellow, I., Y. Bengio, and A. Courville, 2016. *Deep learning* (MIT press).

Gu, A., and T. Dao, 2023, Mamba: Linear-time sequence modeling with selective state spaces, *arXiv preprint arXiv:2312.00752*.

He, K., X. Zhang, S. Ren, and J. Sun, 2016, Deep residual learning for image recognition, Proceedings of the IEEE conference on computer vision and pattern recognition.

Hochreiter, S., 1998, The vanishing gradient problem during learning recurrent neural nets and problem solutions, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* 6(02), 107-116.

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

Holt, C. C., 2004, Forecasting seasonals and trends by exponentially weighted moving averages, *International journal of forecasting,* 20(1), 5-10.

Hyndman, R. J., R. A. Ahmed, G. Athanasopoulos, and H. L. Shang, 2011, Optimal combination forecasts for hierarchical time series, *Computational statistics & data analysis,* 55(9), 2579-2589.

Hyndman, R. J., and G. Athanasopoulos, 2018. *Forecasting: Principles and practice* (OTexts).

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

Januschowski, T., Y. Wang, K. Torkkola, T. Erkkilä, H. Hasson, and J. Gasthaus, 2022, Forecasting with trees, *International Journal of Forecasting,* 38(4), 1473-1481.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, 2017, Lightgbm: A highly efficient gradient boosting decision tree, *Advances in neural information processing systems,* 30.

LeCun, Y., Y. Bengio, and G. Hinton, 2015, Deep learning, *nature,* 521(7553), 436-444.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner, 1998, Gradient-based learning applied to document recognition, *Proceedings of the IEEE,* 86(11), 2278-2324.

Lim, B., S. Ö. Arık, N. Loeff, and T. Pfister, 2021, Temporal fusion transformers for interpretable multi-horizon time series forecasting, *International Journal of Forecasting,* 37(4), 1748-1764.

Lim, B., and S. Zohren, 2021, Time-series forecasting with deep learning: A survey, *Philosophical Transactions of the Royal Society A,* 379(2194), 20200209.

Liu, Y., T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, 2023, Itransformer: Inverted transformers are effective for time series forecasting, *arXiv preprint arXiv:2310.06625*.

Liu, Z., Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, 2024, Kan: Kolmogorov-arnold networks, *arXiv preprint arXiv:2404.19756*.

Makridakis, S., E. Spiliotis, and V. Assimakopoulos, 2018, The m4 competition: Results, findings, conclusion and way forward, *International Journal of forecasting,* 34(4), 802-808.

Makridakis, S., E. Spiliotis, and V. Assimakopoulos, 2020, The m4 competition: 100,000 time series and 61 forecasting methods, *International Journal of Forecasting,* 36(1), 54-74.

Makridakis, S., E. Spiliotis, and V. Assimakopoulos, 2022, M5 accuracy competition: Results, findings, and conclusions, *International Journal of Forecasting,* 38(4), 1346-1364.

Makridakis, S., E. Spiliotis, and V. Assimakopoulos, 2022, The m5 competition: Background, organization, and implementation, *International Journal of Forecasting,* 38(4), 1325-1336.

Montero-Manso, P., G. Athanasopoulos, R. J. Hyndman, and T. S. Talagala, 2020, Fforma: Feature-based forecast model averaging, *International Journal of Forecasting,* 36(1), 86-92.

Montero-Manso, P., and R. J. Hyndman, 2021, Principles and algorithms for forecasting groups of time series: Locality and globality, *International Journal of Forecasting,* 37(4), 1632-1653.

Nair, V., and G. E. Hinton, 2010, Rectified linear units improve restricted boltzmann machines, Proceedings of the 27th international conference on machine learning (ICML-10).

Nie, Y., N. H. Nguyen, P. Sinthong, and J. Kalagnanam, 2022, A time series is worth 64 words: Long-term forecasting with transformers, *arXiv preprint arXiv:2211.14730*.

Olivares, K. G., C. Challu, G. Marcjasz, R. Weron, and A. Dubrawski, 2023, Neural basis expansion analysis with exogenous variables: Forecasting electricity prices with nbeatsx, *International Journal of Forecasting,* 39(2), 884-900.

Oliveira, J. M., and P. Ramos, 2024, Evaluating the effectiveness of time series transformers for demand forecasting in retail, *Mathematics,* 12(17), 2728.

Oreshkin, B. N., D. Carpov, N. Chapados, and Y. Bengio, 2019, N-beats: Neural basis expansion analysis for interpretable time series forecasting, *arXiv preprint arXiv:1905.10437*.

Qiu, X., J. Hu, L. Zhou, X. Wu, J. Du, B. Zhang, C. Guo, A. Zhou, C. S. Jensen, and Z. Sheng, 2024, Tfb: Towards comprehensive and fair benchmarking of time series forecasting methods, *arXiv preprint arXiv:2403.20150*.

Salinas, D., V. Flunkert, J. Gasthaus, and T. Januschowski, 2020, Deepar: Probabilistic forecasting with autoregressive recurrent networks, *International journal of forecasting,* 36(3), 1181-1191.

Schapire, R. E., 1990, The strength of weak learnability, *Machine learning,* 5197-227.

Shao, Z., F. Wang, Y. Xu, W. Wei, C. Yu, Z. Zhang, D. Yao, G. Jin, X. Cao, and G. Cong, 2023, Exploring progress in multivariate time series forecasting: Comprehensive benchmarking and heterogeneity analysis, *arXiv preprint arXiv:2310.06119*.

Sheridan, R. P., W. M. Wang, A. Liaw, J. Ma, and E. M. Gifford, 2016, Extreme gradient boosting as a method for quantitative structure–activity relationships, *Journal of chemical information and modeling,* 56(12), 2353-2360.

Sims, C. A., 1980, Macroeconomics and reality, *Econometrica: journal of the Econometric Society,* 1-48.

Smyl, S., 2020, A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting, *International journal of forecasting,* 36(1), 75-85.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, 2017, Attention is all you need, *Advances in neural information processing systems,* 30.

Wang, H., J. Peng, F. Huang, J. Wang, J. Chen, and Y. Xiao, 2023, Micn: Multi-scale local and global context modeling for long-term series forecasting, The eleventh international conference on learning representations.

Wang, S., H. Wu, X. Shi, T. Hu, H. Luo, L. Ma, J. Y. Zhang, and J. Zhou, 2024, Timemixer: Decomposable multiscale mixing for time series forecasting, *arXiv preprint arXiv:2405.14616*.

Wang, Z., F. Kong, S. Feng, M. Wang, H. Zhao, D. Wang, and Y. Zhang, 2024, Is mamba effective for time series forecasting?, *arXiv preprint arXiv:2403.11144*.

Wen, R., K. Torkkola, B. Narayanaswamy, and D. Madeka, 2017, A multi-horizon quantile recurrent forecaster, *arXiv preprint arXiv:1711.11053*.

Werbos, P. J., 1990, Backpropagation through time: What it does and how to do it, *Proceedings of the IEEE,* 78(10), 1550-1560.

Wickramasuriya, S. L., G. Athanasopoulos, and R. J. Hyndman, 2015, Forecasting hierarchical and grouped time series through trace minimization.

Wickramasuriya, S. L., G. Athanasopoulos, and R. J. Hyndman, 2019, Optimal forecast reconciliation for hierarchical and grouped time series through trace

minimization, *Journal of the American Statistical Association,* 114(526), 804-819.
Winters, P. R., 1960, Forecasting sales by exponentially weighted moving averages, *Management science,* 6(3), 324-342.
Wolpert, D. H., 1992, Stacked generalization, *Neural networks,* 5(2), 241-259.
Wu, H., T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, 2023, Timesnet: Temporal 2d-variation modeling for general time series analysis, *arXiv preprint arXiv:2210.02186*.
Wu, H., J. Xu, J. Wang, and M. Long, 2021, Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, *Advances in neural information processing systems,* 3422419-22430.
Xu, K., L. Chen, and S. Wang, 2024, Kolmogorov-arnold networks for time series: Bridging predictive power and interpretability, *arXiv preprint arXiv:2406.02496*.
Yule, G. U., 1927, Vii. On a method of investigating periodicities disturbed series, with special reference to wolfer's sunspot numbers, *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character,* 226(636-646), 267-298.
Zhang, A., Z. C. Lipton, M. Li, and A. J. Smola, 2021, Dive into deep learning, *arXiv preprint arXiv:2106.11342*.
Zheng, A., 2018. *Feature engineering for machine learning: Principles and techniques for data scientists* (O'Reilly Media, Inc).
Zhou, H., S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, 2021, Informer: Beyond efficient transformer for long sequence time-series forecasting, Proceedings of the AAAI conference on artificial intelligence.
Zhou, T., Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, 2022, Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting, International conference on machine learning (PMLR).

Eidesstattliche Erklärung

„Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Vergöttlichungen übernommen wurden, sind als solche kenntlich gemacht. Alle Internetquellen sind der Arbeit beigefügt. Des Weiteren versichere ich, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und dass die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht."

Ort, Datum Unterschrift

Hamburg, 15.10.2024