CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Time-Series Forecasting for Industrial Demand: A Comparative Study

Comparative Evaluation of Traditional and State-of-the-Art Time-Series Models for Industrial Demand Forecasting

Master's thesis in Computer science and engineering

ERIK ANDERSSON
ANDREAS PERSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Time-Series Forecasting for Industrial Demand: A Comparative Study

Comparative Evaluation of Traditional and State-of-the-Art Time-Series Models for Industrial Demand Forecasting

ERIK ANDERSSON
ANDREAS PERSSON

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Time-Series Forecasting for Industrial Demand: A Comparative Study
Comparative Evaluation of Traditional and State-of-the-Art Time-Series Models for Industrial Demand Forecasting

ERIK ANDERSSON
ANDREAS PERSSON

Examiner: Richard Johansson, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2025

Time-Series Forecasting for Industrial Demand: A Comparative Study
Comparative Evaluation of Traditional and State-of-the-Art Time-Series Models for Industrial Demand Forecasting
ERIK ANDERSSON
ANDREAS PERSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Accurate demand forecasting is critical for industrial manufacturers to optimize production planning and resource allocation. This study evaluates state-of-the-art time series forecasting models on a real-world dataset of monthly order intake for commercial vehicles from Company X, organized hierarchically by market, product group, and power supply.

In this study, classical statistical methods (AutoETS, AutoARIMA) and gradient-boosting trees (XGBoost, LightGBM) are implemented and compared with recently developed state-of-the-art deep learning architectures (Temporal Fusion Transformer, TimeXer, TiDE, N-HiTS, SOFTS). Expanding-window cross-validation is employed to generate multi-horizon forecasts up to 12 months and accuracy is evaluated using RMSE and sMAPE. Models are compared with and without the utilization of exogenous variables to highlight their robustness in processing external signals to aid forecasting accuracy.

The results indicate that tree-based and statistical models, particularly LightGBM and AutoARIMA when augmented with exogenous variables, achieve the lowest average errors, suggesting that state-of-the-art models are not competitive in this setting. However, while these models are less accurate on aggregate metrics, they tend to produce forecasts with richer temporal dynamics, capturing trends and seasonality more effectively.

Overall, the findings suggest that no single approach consistently outperforms others; model effectiveness varies depending on the forecast horizon, aggregation level, and the availability of external covariates. These results highlight the importance of selecting models based on forecasting context, particularly in data-scarce industrial environments. An interesting direction for future research could be to explore hybrid or ensemble approaches to combine accuracy with actionable temporal structure.

Keywords: Industrial demand forecasting, time-series, ETS, ARIMA, gradient boosting, LightGBM, XGBoost, Transformers, MLP, TFT, TimeXer, TiDE, N-HITS, SOFTS, exogenous variables, multi-horizon evaluation.

Acknowledgements

We would like to express our sincere gratitude to our supervisor and examiner, Professor Richard Johansson, for his valuable guidance, encouragement, and expertise throughout the course of this thesis.

We also wish to extend our heartfelt thanks to our company supervisors and domain experts, who have supported us throughout the project. Their insights, feedback, and continuous engagement have been instrumental in shaping our work and helping us navigate both technical and practical challenges.

Erik Andersson and Andreas Persson, Gothenburg, 2025-06-16

Contents

List of Figures

List of Tables

1

Introduction

Accurate demand forecasting is essential for industrial manufacturing companies to effectively respond to market needs, optimize resource allocation, and strengthen their competitive position. In particular, within the commercial vehicle manufacturing sector, precise demand forecasts significantly enhance production planning, facilitating informed decision-making and improved operational efficiency. These forecasts are typically based on historical sales data, which naturally form a time series.

In many industrial settings, demand forecasting is still largely driven by human intuition and experience. Multiple data-driven methods have been developed to support and improve the accuracy of the forecasting process, ranging from statistical models to machine learning (ML) based approaches. More recently, research within time-series forecasting has moved toward more advanced deep learning (DL), utilizing methodologies such as multilayer perceptrons (MLP) and transformers, showing good performance on commonly used benchmarking datasets [1]–[5]. However, despite the availability of advanced tools and techniques, practical challenges remain. Among these is the difficulty of understanding how well such models generalize to real-world industrial applications.

Recognizing these challenges, the primary objective of this thesis is to investigate how data-driven approaches, specifically state-of-the-art (SoTA) time-series forecasting techniques utilizing DL, perform when applied to a real-world demand forecasting problem. To this end, the study conducts a comparative analysis of methods ranging from well-established statistical models to SoTA deep learning methods, which have undergone only limited testing in practical settings.

1.1 Background

Demand forecasting is fundamentally a time-series problem, where the goal is to predict future values based on historically observed data, under the assumption that the patterns, trends, and dependencies observed in the past will to some extent persist into the future. Statistical methods have served as the foundation of time-series forecasting for a long time. Techniques such as exponential smoothing [6]–[8] and the Autoregressive Integrated Moving Average (ARIMA) family of models [9] provided mathematically interpretable and computationally efficient solutions, especially for stable or low-variability time series.

With advances in computational power and increased data availability, attention shifted toward more flexible ML methods capable of capturing complex, nonlinear relationships. In particular, tree-based ensemble models such as Gradient Boosted Decision Trees (GBDTs) gained popularity for their strong predictive performance, with popular implementations including XGBoost [10] and LightGBM [11]. These models proved effective in a wide range of forecasting competitions and industry applications [12].

More recently, DL models have shown promising results for time-series forecasting. Recurrent architectures such as Long Short-Term Memory (LSTM) networks [13] were among the first to be adopted, followed by attention-based transformer models. Since the introduction of the original Transformer architecture in 2017 [14], considerable effort has been devoted to adapting self-attention mechanisms for temporal data. Models such as the Temporal Fusion Transformer (TFT) [1], Informer [15], Autoformer [16], and FEDFormer [17] demonstrated SoTA performance on various benchmark datasets, leading to the perception that transformers were inherently superior for time-series forecasting.

This perception was challenged by Zeng et al. [18], who argued that the strong performance of transformer-based models may stem from the use of direct multi-step forecasting (DMS) strategies rather than the attention mechanism itself. To support this claim, they proposed an "embarrassingly simple" linear model using DMS, that outperformed several transformer architectures on a suite of standard datasets.

In response, a wave of research has explored the use of MLP-based models using DMS for time-series forecasting. Recent architectures such as TiDE [3], N-HiTS [4], and SOFTS [2] have demonstrated that MLPs can match or exceed the performance of more complex models in many settings.

While many SoTA models have been evaluated extensively on public datasets, their generalizability to industry-specific settings, such as commercial vehicle manufacturing, remains largely unexplored. Furthermore, recent studies have challenged the assumption that transformer-based models outperform simpler alternatives. In light of this, it is of interest to evaluate whether SoTA models offer advantages over traditional statistical and ML methods.

1.2 Problem Formulation

At the company studied in this thesis, a commercial vehicle manufacturer referred to as Company X, demand forecasts are produced on a monthly basis for horizons extending 12 to 24 months into the future. These forecasts are generated at a disaggregated level, specifically for each combination of market, product category, and power supply.

Currently, the forecasting process at Company X relies heavily on human expertise and manual adjustments informed by market knowledge, historical sales data, and contextual judgment. While this approach leverages valuable domain experience, it is inherently subjective and may struggle to consistently capture complex temporal dynamics, which can lead to suboptimal planning outcomes.

The problem addressed in this thesis is to develop and evaluate modern, data-driven forecasting techniques tailored to this setting. The aim is to complement the existing manual process with models capable of learning from historical sales patterns and modeling temporal structures across varying forecast horizons. Specifically, the models will be trained to produce forecasts of order intake with a 12-month horizon. Given the inherently hierarchical structure of the data, forecasts will be generated at multiple levels of aggregation, ranging from the most disaggregated level to the fully aggregated level.

1.3 Aim

The aim of this study is to evaluate SoTA time-series forecasting models on a real-world industrial dataset that has not previously been examined in the academic literature. Specifically, the following research questions (RQ) guide this work:

> **RQ1:** *How well do SoTA time-series forecasting models perform on commercial demand forecasting for vehicles compared to traditional statistical and machine learning models?*
>
> **RQ2:** *What modifications to the data, model architecture, or training procedure can improve forecasting performance for this specific use case?*

2

Theory

This chapter introduces the theoretical foundations of time-series forecasting, covering classical statistical, modern machine learning, and deep learning models. It begins with core concepts such as modeling strategies, forecasting horizons, and the role of trend, seasonality, and exogenous variables. The chapter then reviews the key models used in this study, highlighting their structure, assumptions, and relevance to practical forecasting tasks.

2.1 Time-Series Forecasting

Time-series forecasting is a well-established field of research that has evolved over several decades. It involves predicting future values based on previously observed data points, typically collected at consistent intervals over time [19]. The primary objective is to identify patterns, trends, and seasonal variations within the data to make informed predictions about future occurrences.

Time series can be classified into two main types: *univariate time series* and *multivariate time series* [19]. Formally, a univariate time series is defined as an ordered sequence of real-valued observations, denoted as $\{y_t\}_{t=1}^{T}$, where each observation $y_t \in \mathbb{R}$. In this setting, the primary objective is to forecast future values y_{T+h}, for one or more steps ahead, $h > 0$.

Conversely, a multivariate time series consists of multiple interrelated variables and can be represented as an ordered sequence of $\mathbf{m}$-dimensional real-valued vectors, denoted as $\{\mathbf{y}_t\}_{t=1}^{T}$, where $\mathbf{y}_t = [y_{t,1}, y_{t,2}, \ldots, y_{t,m}]^\top \in \mathbb{R}^m$ [19]. The forecasting task for a multivariate series involves predicting the future vectors $\mathbf{y}_{T+h}$, encompassing each of the m variables simultaneously.

2.1.1 Time-Series Modeling Approaches

Related to the two notions of univariate and multivariate time-series, there is the concept of *local* and *global* training of models on time-series data [20]. Local training involves fitting one model for each individual time-series, whereas global training involves training a single model for multiple time-series, learning shared patterns across the time-series. A local training approach can be applied to both univariate and multivariate data, while typically global training is applied only to multivariate data. Consequently, local or global training refers to whether or not data are pooled

across series, whereas univariate and multivariate pertains to the dimensionality of the observed variables.

Furthermore, multi-horizon forecasting, that is, forecasting more than one time-step ahead, introduces two different methods: *Iterative Multistep Forecasting* (IMF) and *Direct Multistep Forecasting* (DMF) [21]. IMF is an iterative (recursive) one step ahead method where if the forecasting horizon is $h > 1$, then the model forecasts one step ahead, adds that forecast as a synthetic past value to use for the subsequent forecasting steps. This iterates until the specified horizon is reached. This approach has empirically been shown to be efficient, however, it relies on correctly specified models [21]. Indeed, the IMF method does forecast based on previously forecasted values, ultimately increasing the risk of error propagation for longer horizons.

DMF is a modeling approach in which each forecasting horizon $h > 1$ is addressed by its own predictive model. This approach relies exclusively on observed historical data and does not incorporate any forecasted target values as inputs. Under an H-step forecasting scheme, one estimates and optimizes H separate models, each tailored to predict y_{t+1}, y_{t+2}, ..., y_{t+H}. While this strategy entails the specification and calibration of a distinct estimator for every horizon, for example twelve individual models for a twelve month forecasting exercise, it eliminates the risk of sequential error accumulation that arises when a one step ahead forecast model is applied recursively.

Although theory suggest that IMF can outperform DMF under a perfectly specified one-step model, DMF is often favored for its robustness to misspecification and its avoidance of error propagation [21]. The choice between IMF and DMF depends on the data's characteristics and remains an empirical question.

2.1.2 Trend and Seasonality

Trend and seasonal components represent two important aspects in time-series forecasting [22]. Seasonality describes periodic, recurring patterns within data that repeat consistently over defined intervals. Such patterns can arise from diverse factors, and a single dataset may include multiple seasonalities at different frequencies. Examples of sources of seasonality include weather variations, year-end promotional activities, or recurring economic events.

Trend describes the long-term, non-periodic movement in a time-series, reflecting sustained rises or falls over an extended horizon [22]. Such movements can be linear or nonlinear, driven by underlying shifts like technological innovation, demographic change, or economic expansion. Examples of trend sources include population growth driving rising demand, cumulative capital investment boosting GDP, or technological progress increasing productivity.

Time series exhibiting trend, seasonality, or both, are considered non-stationary [22]. A stationary time-series holds the properties that it does not depend on the observed time of the series [19]. The concept of stationarity is significant as many forecasting models assume stationarity to effectively capture underlying patterns within the data. A commonly employed technique for inducing stationarity in an

otherwise non-stationary series is *differencing* [19]. This transformation involves replacing the original observations with the differences between successive values, thereby stabilizing the variance of the series before model fitting.

2.1.3 Exogenous Variables

Incorporating exogenous variables into time-series forecasting models can enhance the predictive performance, particularly for mid- to long-term forecasts where uncertainty is more pronounced [23]. Exogenous variables, external covariates that influence the target variable but are not influenced by it, can include macroeconomic indicators, weather data, or internal metrics such as product, inventory levels, or marketing promotions. These variables provide additional data that can improve the ability of a model to capture underlying patterns and signals within the data.

Exogenous variables can be categorized into three types: static, historical, and future exogenous variables [1].

- Static exogenous variables are time-invariant attributes that serve as identifiers for data points. Examples of static exogenous variables are product group and geographical area.
- Historical exogenous regressors are temporal variables which only include past observed values. These are commonly incorporated into models when it is believed that a historic value of the regressor affects the future outcome of the target variable. As an example, this could be if testing shows that the value of exogenous variable E_x at time t affects the outcome of the target variable at time $t + m$, where $m > 0$.
- Future exogenous regressors are used when accurate future values of the regressor is believed to exist. The model is then trained on the historical observed values of exogenous variables, and future forecasts of these exogenous variables are passed to the model during inference. Examples of future exogenous variables include the number of business days in a month or forecasts of economic indicators such as GDP.

Incorporating exogenous variables in a model can be achieved through various methods, depending on the nature of the variables and the specific requirements of the forecasting task. Certain architectures can accommodate the full spectrum of exogenous variable types [1], [3], [4], whereas others are restricted to particular classes (e.g., future exogenous variables) [5], and a subset of models entirely excludes the integration of exogenous inputs [2], [24].

2.2 Statistical Time-Series Forecasting Models

This section introduces two traditional statistical approaches for time-series forecasting: exponential smoothing and ARIMA models. The section concludes by describing how automated procedures, specifically AutoETS and AutoARIMA, can be used to make these classical models practical for large-scale forecasting applications.

2.2.1 Exponential Smoothing

Exponential smoothing is a class of time-series forecasting methods that rely on the idea of recursively updating forecasts as new observations become available. The central principle of these methods is that forecasts are generated as weighted averages of past observations, where the weights decrease exponentially with time. This design gives greater importance to more recent data points, allowing the models to be responsive to structural changes such as level shifts or trend reversals.

The earliest form of exponential smoothing was introduced by Brown [6] for inventory control applications, and later generalized by Holt [7] and Winters [8] to include trend and seasonal components. These developments have collectively led to the family of methods now referred to as the Holt-Winters or exponential smoothing models. A comprehensive statistical foundation for these methods was later provided by Hyndman et al. [24], who developed the *innovations state space framework* and formalized the class of models now known as ETS (Error-Trend-Seasonal) models.

Simple Exponential Smoothing (SES): This method is suitable for time-series data without trend or seasonality [6]. Let x_t be the observation at time t, and s_t the smoothed level estimate. The recursive formula is:

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1}, \quad \text{for } 0 < \alpha < 1,$$

where α is the smoothing parameter. The forecast for all future time points is simply the last smoothed value, i.e., $\hat{x}_{t+h|t} = s_t$ for $h \geq 1$. This model assumes a constant level over time and is optimal for series that fluctuate around a stable mean.

Holt's Linear Method: To account for linear trends, Holt [7] extended SES by introducing a trend component b_t, resulting in the following update equations:

$$\begin{aligned} s_t &= \alpha x_t + (1 - \alpha)(s_{t-1} + b_{t-1}), \\ b_t &= \beta(s_t - s_{t-1}) + (1 - \beta) b_{t-1}, \end{aligned}$$

where β is the smoothing parameter for the trend. Forecasts are given by:

$$\hat{x}_{t+h|t} = s_t + h b_t.$$

This model adapts to persistent upward or downward trends and assumes that the trend continues indefinitely.

Holt-Winters' Exponential Smoothing: For time series with both trend and seasonality, the Holt-Winters method [8] introduces a seasonal component c_t and supports two forms: additive (for constant-amplitude seasonality) and multiplicative (for seasonality proportional to the level). In the additive case, the equations are:

$$\begin{aligned} s_t &= \alpha(x_t - c_{t-m}) + (1 - \alpha)(s_{t-1} + b_{t-1}), \\ b_t &= \beta(s_t - s_{t-1}) + (1 - \beta) b_{t-1}, \\ c_t &= \gamma(x_t - s_{t-1} - b_{t-1}) + (1 - \gamma) c_{t-m}, \end{aligned}$$

where m is the seasonal period and γ is the seasonal smoothing parameter. The h-step ahead forecast is:

$$\hat{x}_{t+h|t} = s_t + h b_t + c_{t+h-m\lfloor (h-1)/m \rfloor}.$$

The multiplicative form replaces additive seasonal terms with ratios and is preferred when seasonal fluctuations scale with the level of the series.

The ETS Model Class: The classical exponential smoothing methods (SES, Holt, and Holt-Winters) can all be viewed as special cases within a unified framework known as ETS (Error-Trend-Seasonal) models. Introduced by Hyndman et al. [24], this state space formulation models the error, trend, and seasonality components as either none (N), additive (A), multiplicative (M), or damped (Ad, Md).

Table 2.1: The fifteen ETS model configurations

Trend / Seasonality	**N (None)**	**A (Additive)**	**M (Multiplicative)**
N (None)	ETS(N,N,N)	ETS(N,N,A)	ETS(N,N,M)
A (Additive)	ETS(A,N,N)	ETS(A,A,A)	ETS(A,A,M)
Ad (Additive damped)	ETS(Ad,N,N)	ETS(Ad,A,A)	ETS(Ad,A,M)
M (Multiplicative)	ETS(M,N,N)	ETS(M,A,A)	ETS(M,A,M)
Md (Multiplicative damped)	ETS(Md,N,N)	ETS(Md,A,A)	ETS(Md,A,M)

Several of the ETS configurations correspond directly to the classical methods introduced earlier. Specifically, the basic smoothing model described in the Simple Exponential Smoothing section is represented by ETS(N,N,N), while the trend-adjusted model in Holt's method corresponds to ETS(A,N,N) or ETS(Ad,N,N) if a damped trend is applied. Likewise, the additive and multiplicative versions of the Holt-Winters method align with ETS(A,A,N) and ETS(A,M,N), respectively. The remaining configurations extend these foundations to support alternative combinations of error, trend, and seasonality that, while less commonly used, follow the same underlying principles.

2.2.2 The ARIMA Models

The ARIMA family, which stands for Auto-Regressive Integrated Moving Average, encompasses a class of stochastic models used for time-series forecasting. These models assume that the underlying time series is stationary or can be transformed into stationarity through differencing. The foundation of ARIMA modeling was laid by Box and Jenkins in the 1970s, who introduced a systematic procedure for model identification, estimation, and validation, commonly referred to as the Box-Jenkins methodology [9].

An ARIMA model is specified by three parameters (p, d, q), where p is the order of the autoregressive (AR) part, d is the degree of differencing required to achieve stationarity, and q is the order of the moving average (MA) component. When applied to a non-stationary series, differencing of order d is first performed, after which an ARMA model is fitted to the transformed series.

Autoregressive (AR) Models: In AR models, the future values of the series are predicted based on its past values. A first-order AR model is defined as:

$$Y_t = c + \phi_1 Y_{t-1} + \epsilon_t,$$

where ϕ_1 is the autoregressive parameter and ϵ_t is a white noise error term. In general, an AR(p) process is expressed as:

$$Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \epsilon_t.$$

Moving Average (MA) Models: MA models predict the current value based on past forecast errors. A first-order MA model is given by:

$$Y_t = c + \epsilon_t - \theta_1 \epsilon_{t-1},$$

and in general, an MA(q) model is written as:

$$Y_t = c + \epsilon_t - \sum_{j=1}^{q} \theta_j \epsilon_{t-j}.$$

Autoregressive Moving Average (ARMA): ARMA models combine both AR and MA components and are used for stationary time series. An ARMA(p, q) model has the form:

$$Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \epsilon_t - \sum_{j=1}^{q} \theta_j \epsilon_{t-j}.$$

Autoregressive Integrated Moving Average (ARIMA): ARIMA extends ARMA to non-stationary time series by introducing differencing. The model includes an additional parameter d, representing the number of times the data must be differenced to achieve stationarity:

$$\Delta^d Y_t = c + \sum_{i=1}^{p} \phi_i \Delta^d Y_{t-i} + \epsilon_t - \sum_{j=1}^{q} \theta_j \epsilon_{t-j}.$$

This structure allows ARIMA models to effectively model a wide range of time-series patterns including trends.

Seasonal ARIMA (SARIMA): An extension of ARIMA introduces seasonal autoregressive, differencing, and moving average terms to capture repeating seasonal patterns. The model is denoted as ARIMA$(p, d, q)(P, D, Q)_m$, where m is the seasonal period. The additional parameters (P, D, Q) represent seasonal AR, differencing, and MA terms respectively [9].

ARIMAX / SARIMAX: A further extension is to incorporate one or more exogenous variables to account for external influences on the time series. The inclusion of covariates could improve model accuracy when relevant external signals are available. This extension is commonly applied in practical settings, such as incorporating macroeconomic indicators in capacity planning [25].

All mentioned models above collectively belong to the broader class of *linear time-series models*, which assume that the current observation is a linear function of past values and past errors [26].

Non-exhaustive Summary of Models within the ARIMA Family:

- **ARIMA(p,0,0):** Equivalent to an autoregressive model AR(p).
- **ARIMA(0,0,q):** Equivalent to a moving average model MA(q).
- **ARIMA(p,0,q):** Equivalent to ARMA(p, q), for stationary series.
- **ARIMA(p,d,q):** The general ARIMA model.
- **ARIMA(p,d,q)(P,D,Q)$_m$:** Also known as SARIMA.
- **ARIMAX / SARIMAX:** Extensions of ARIMA and SARIMA that include one or more exogenous variables.

While a comprehensive treatment of the theory and variants within the ARIMA family is beyond the scope of this thesis, readers are referred to Box et al. [9] for more details. For a mathematically rigorous introduction to linear time-series models, including ARIMA and its subcomponents, see Brockwell and Davis [26].

2.2.3 AutoETS & AutoARIMA

In large-scale forecasting environments, such as demand forecasting, it is often necessary to generate forecasts for thousands of time series simultaneously. These time series can differ significantly in their characteristics, some may exhibit clear seasonality, others may trend upward or downward, while many may show only noise. In such contexts, manually specifying and fitting appropriate models for each series becomes infeasible. Furthermore, the level of statistical expertise required to reliably choose between exponential smoothing and ARIMA models can be limited in practice. These challenges have driven the development of automatic forecasting methods that can select and estimate suitable models with minimal human intervention.

A major contribution in this area was made by Hyndman and Khandakar [27], who proposed a unified framework for automatic model selection and estimation within both the ETS and ARIMA model families. Their methodology combines statistical tests, information criteria, and efficient search algorithms to identify the best-performing model from a predefined space of candidates.

In their work, Hyndman and Khandakar explains that the automatic procedure for exponential smoothing involves evaluating all admissible combinations of error, trend, and seasonal components within the ETS model class. Each configuration is estimated using maximum likelihood, and the model that minimizes an information criterion, typically the corrected Akaike Information Criterion (AICc), is selected as the best. This approach ensures that the model's complexity is penalized appropriately, reducing the risk of overfitting. AutoETS handles both additive and multiplicative components, as well as damping for trends, providing a flexible and data-driven solution for level, trend, and seasonal time series.

Furthermore, the authors also explain that automatic ARIMA modeling addresses a more complex model space that includes differencing, autoregressive terms, moving average terms, and their seasonal counterparts. The algorithm first determines the required level of non-seasonal and seasonal differencing using unit root tests such

as the KPSS and Canova-Hansen tests. After differencing, the algorithm performs a stepwise search through candidate ARIMA models, evaluating them based on an information criterion, commonly AICc. To avoid exhaustive enumeration, the stepwise approach adds or removes parameters incrementally, selecting the direction that improves model fit.

Hydman and Khandakar argues that these automated procedures make statistical time-series models scalable and accessible for use in real-world applications. By systematically selecting models based on the data and penalizing for complexity, AutoETS and AutoARIMA offer robust forecasting performance across a wide variety of time-series structures, without requiring manual tuning or expert intervention.

2.3 Machine Learning Time-Series Forecasting Models

Many real-world time-series problems often exhibit nonlinear interactions among covariates and more complex temporal structures than can not be captured by linear filters alone. Machine learning methods offer a natural generalization of the statistical paradigm where rather than specifying a rigid linear form, they learn an arbitrary mapping from a set of inputs to the target.

Although most classical machine learning algorithms were not originally designed for sequential data, many can be effectively repurposed for forecasting tasks when temporal dependencies are encoded appropriately. This is typically achieved by constructing feature vectors from lagged values, rolling statistics, calendar effects, and exogenous variables. Under this formulation, standard regression-based ML models, including support vector machines, artificial neural networks, and especially tree-based ensembles, can learn intricate interaction patterns and adapt to non-stationary or heterogeneous time series.

Among the most prominent methodologies in this domain are gradient boosting decision trees (GBDTs), which have demonstrated state-of-the-art performance in a wide array of forecasting benchmarks. Due to their robustness, scalability, and ability to model nonlinearities and high-order interactions, GBDTs are frequently adopted as the primary forecasting model in applied machine learning pipelines.

2.3.1 Gradient Boosting Trees

Gradient Boosting is an ensemble learning paradigm applicable to both classification and regression tasks [28]. It is a technique that builds predictive models in a sequential manner by combining a series of weak learners, typically decision trees. Each new model in the sequence is trained to approximate the negative gradient of the loss function with respect to the ensemble's current predictions, which are referred to as the pseudo residuals [29]. By incorporating the tree that targets these pseudo residuals, the model moves in the direction of steepest descent of the loss function and thus achieves a progressive reduction in the overall prediction error

through iterative refinement. The ensemble of these models is commonly referred to as Gradient Boosting Decision Trees (GBDTs).

Let $\{(x_i, y_i)\}_{i=1}^n$ be a dataset with n observations and m features. A GBDT model predicts the response $\hat{y}_i$ by aggregating the outputs of K regression trees $f_k \in \mathcal{F}$, where $\mathcal{F}$ denotes the space of regression trees:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}.$$

Each function f_k represents a decision tree characterized by a structure q mapping an input vector to a leaf index, and associated leaf weights w. Formally, the space of tree functions is defined as:

$$\mathcal{F} = \left\{ f(x) = w_{q(x)} \,\middle|\, q : \mathbb{R}^m \to \{1, \ldots, T\},\ w \in \mathbb{R}^T \right\},$$

where T is the number of leaves in the tree.

Despite their origins in general-purpose prediction tasks, GBDT models have been shown to be highly effective in time-series forecasting [12]. This is achieved when relevant temporal features are included in the models, such as lagged values acting as a fixed window of past observed targets (look-back), rolling statistics, and calendar-based indicators.

Prominent implementations include Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machine (LightGBM), both of which have demonstrated robust empirical performance across a variety of forecasting contexts [10], [11].

2.3.1.1 XGBoost

XGBoost was introduced by Chen and Guestrin [10] as a scalable implementation of GBDT, optimized for efficient handling of large-scale data and complex modeling scenarios. Its effectiveness has been demonstrated extensively, and is one of the more popular implementations of GBDT.

Chen and Guestrin implements several concepts that enhance its performance compared to existing GBDT. Key among these is regularization, which helps in controlling model complexity and reducing the risk of overfitting.

Formally, given training pairs $\{(x_i, y_i)\}_{i=1}^n$ and an additive ensemble

$$\hat{y}_i \;=\; \phi(x_i) \;=\; \sum_{k=1}^{K} f_k(x_i),$$

where each regression tree f_k has leaf weight vector w and T leaves, XGBoost minimizes

$$\mathcal{L}(\phi) \;=\; \sum_{i=1}^{n} l\big(\hat{y}_i, y_i\big) \;+\; \sum_{k=1}^{K} \Big[\gamma\, T + \tfrac{1}{2}\,\lambda\, \|w\|^2\Big],$$

in which l is a differentiable convex loss and γ, λ are hyperparameters controlling the penalties on tree complexity and on leaf weight magnitudes, respectively [10]. The term $\gamma\, T$ penalizes the number of leaves in the tree, analogous to an L_0 penalty on model complexity, while the term $\frac{1}{2}\,\lambda\,\|w\|^2$ is an L_2 penalty on the leaf weight vector. XGBoost also supports an optional L_1 penalty on the weights via the `reg_alpha` parameter.

Additionally, Chen and Guestrin explains that XGBoost implements various methods for efficient data handling, such as parallelization and sparse data optimization. Sparse data optimization is important for datasets containing numerous zero or missing values, which are common in real-world datasets. XGBoost effectively handles sparse data by implementing sparsity-aware algorithms, which ignore missing or zero-values features during tree construction and splitting.

2.3.1.2 LightGBM

Light Gradient Boosting Machine (LightGBM) was introduced by Ke et al. [11] as an efficient implementation of the GBDT framework, specifically designed to overcome the computational bottlenecks of existing solvers while retaining, or even improving, state-of-the-art predictive performance. As with XGBoost, LightGBM optimizes a regularized objective of the form

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l\Big(y_i, \hat{y}_i\Big) \;+\; \sum_{k=1}^{K} \Omega\Big(f_k\Big), \quad \Omega(f) = \gamma\, T + \tfrac{1}{2}\,\lambda\,\|w\|^2,$$

where each tree f_k has T leaves with weights w, and γ, λ control complexity and weight-shrinkage penalties [11].

Unlike the level-wise tree induction used in many GBDT implementations, LightGBM employs a *leaf-wise* (best-first) growth strategy: at each boosting iteration, it identifies the single leaf whose split yields the greatest reduction in objective, and grows that leaf. Denoting by G_j and H_j the sums of first and second derivatives of the loss over the samples in leaf j, and similarly G_l, H_l and G_r, H_r for the candidate child leaves, the gain from splitting leaf j is

$$\text{Gain}_j = \tfrac{1}{2}\Big[\tfrac{G_l^2}{H_l+\lambda} + \tfrac{G_r^2}{H_r+\lambda} - \tfrac{G_j^2}{H_j+\lambda}\Big] - \gamma.$$

By greedily choosing the split with maximal Gain_j, LightGBM can build deeper, more expressive trees with fewer splits, often yielding faster convergence on large datasets [11].

To further accelerate training, LightGBM integrates two novel techniques:

- **Gradient-based One-Side Sampling (GOSS)**: GOSS reduces the effective data size for split finding by retaining a fraction α of instances with the largest absolute gradients $|g_i|$ and randomly sampling β of the remaining instances. To correct for the sampling bias, small-gradient instances are re-weighted by $\frac{1-\alpha}{\beta(1-\alpha)}$, yielding an unbiased estimate of the true gradient distribution. This preserves the informative, high-error examples while discarding

many low-impact ones, substantially reducing computation without degrading accuracy [11].

- **Exclusive Feature Bundling (EFB)**: In high-dimensional sparse settings (e.g., one-hot encoded categoricals), many features rarely take nonzero values simultaneously. EFB exploits this by grouping such non-overlapping features into bundles. Formally, it partitions the feature set $\{x_j\}$ into bundles $\{B_k\}$ such that for any $x_i, x_j \in B_k$, $x_i \cdot x_j = 0$ for the vast majority of samples. Each bundle is then treated as a single feature during histogram construction, drastically reducing memory footprint and split-finding time [11].

Empirical benchmarks demonstrate that LightGBM can achieve speedups on the order of 5-10x compared to other GBDT packages, often without sacrificing accuracy, and has consistently ranked among the top performers in large-scale machine-learning competitions, including a first-place finish in the 2020 Kaggle M5 Forecasting challenge [12].

2.4 Artificial Neural Networks

This section focuses on three core neural network architectures relevant for time-series forecasting: Multilayer Perceptrons (MLP), Recurrent Neural Networks (RNN), and Transformer models. Each of these architectures serves as a foundation for the more specialized models discussed in subsequent sections.

Artificial Neural Networks (ANNs) are a class of machine learning models that are designed to approximate complex, non-linear functions by learning patterns from data. In the context of supervised learning, an ANN attempts to learn a mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ from input to output by minimizing a loss function that quantifies the difference between the predicted and true values.

The fundamental building block of an ANN is the artificial neuron, also known as a perceptron. Each neuron receives one or more input signals, applies a weighted linear combination, and passes the result through a non-linear activation function. Formally, the output of a neuron can be written as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right),$$

where x_i are the input features, w_i are the corresponding weights, b is a bias term, and $f(\cdot)$ is the activation function.

When multiple neurons are arranged in numerous layers and stacked together, they form a Deep Neural Network (DNN). A DNN typically consists of an input layer, several hidden layers, and an output layer. The depth of the network, defined by the number of hidden layers, enables it to learn hierarchical representations of the input data. DNNs are trained via backpropagation [30] and gradient-based optimization, where the model iteratively adjusts its weights to minimize the loss function using training data.

2.4.1 Multilayer Perceptrons

Multilayer Perceptrons (MLPs) represent one of the most fundamental architectures within artificial neural networks [31]. It is a fully connected, feedforward architecture in which each neuron in a given layer is connected to every neuron in the next layer. MLPs are composed of an input layer, one or more hidden layers, and an output layer, with non-linear activation functions applied between layers. As a feedforward model, information in an MLP flows strictly from inputs to outputs, without any feedback or internal state retention.

Mathematically, an MLP with K hidden layers can be described as a composition of affine transformations followed by non-linear activations. Given an input vector $x \in \mathbb{R}^n$, the output of the network $y \in \mathbb{R}^m$ is computed as:

$$y = f_K(W_K f_{K-1}(\cdots f_1(W_1 x + b_1) \cdots) + b_K),$$

where W_k and b_k denote the weight matrix and bias vector for the k-th layer, and $f_k(\cdot)$ is the activation function applied element-wise. For a visual representation of the MLP see Figure 2.1.

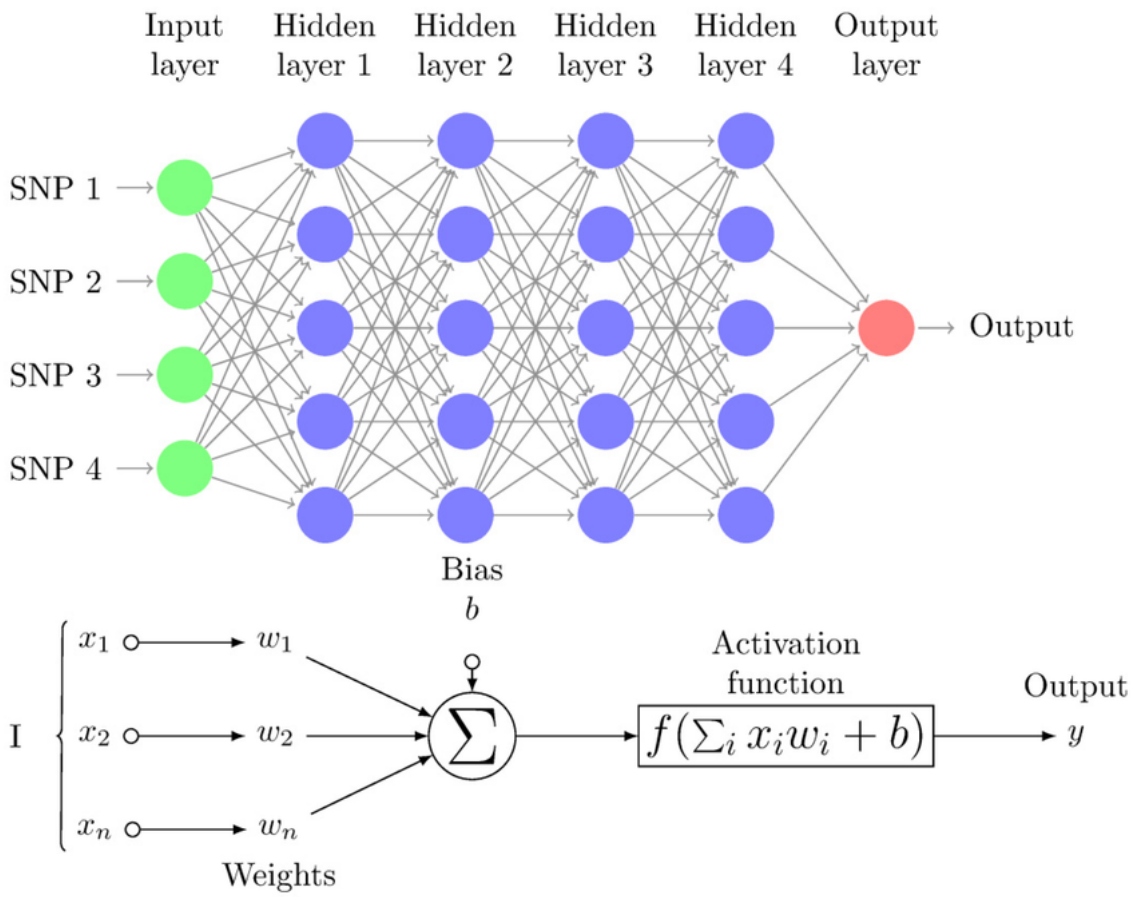

Figure 2.1: Illustration of a Multilayer Perceptron (MLP) with four input features and four hidden layers, each consisting of multiple fully connected neurons. The lower part of the figure shows the computation inside a single neuron: a weighted sum of inputs plus a bias, passed through an activation function. Reproduced from [32], used under the Creative Commons Attribution License [33].

In the context of time-series forecasting, MLPs are typically used in a window-based framework. A fixed-length window of past observations is used as input, and the model is trained to predict one or more future values. For a univariate time series $\{x_t\}$, the model learns a mapping of the form:

$$\hat{x}_{t+h} = f(x_t, x_{t-1}, \ldots, x_{t-L+1}),$$

where L is the window length and h is the forecast horizon.

2.4.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural network architectures specifically designed to model sequential data [34]. Unlike feedforward networks such as MLPs, RNNs maintain an internal state that is updated over time, allowing them to capture temporal dependencies across sequences. This makes them well-suited for time-series forecasting tasks, where past values inform future predictions.

At each time step t, a vanilla RNN takes as input the current observation x_t and the hidden state from the previous time step h_{t-1}, and updates its hidden state h_t through a recurrent transformation:

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b),$$

where W_x and W_h are weight matrices, b is a bias vector, and $\sigma(\cdot)$ is a non-linear activation function.

Although the simple vanilla RNN can, in principle, capture long-term dependencies, they often suffer from vanishing or exploding gradients during training, which limits their effectiveness for longer sequences. To address this limitation, Hochreiter and Schmidhuber [13] introduced the Long Short-Term Memory (LSTM) network.

LSTMs enhance the RNN architecture by introducing a memory cell c_t and a set of gating mechanisms that control the flow of information through the network. These gates (input, forget, and output) allow the network to learn when to retain or discard information, enabling better modeling of long-range dependencies. The LSTM cell is updated as follows:

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}$$

where $\odot$ denotes element-wise multiplication, and each W, U, and b are trainable parameters for the respective gates. For a visual represntation of the LSTM cell see Figure 2.2.

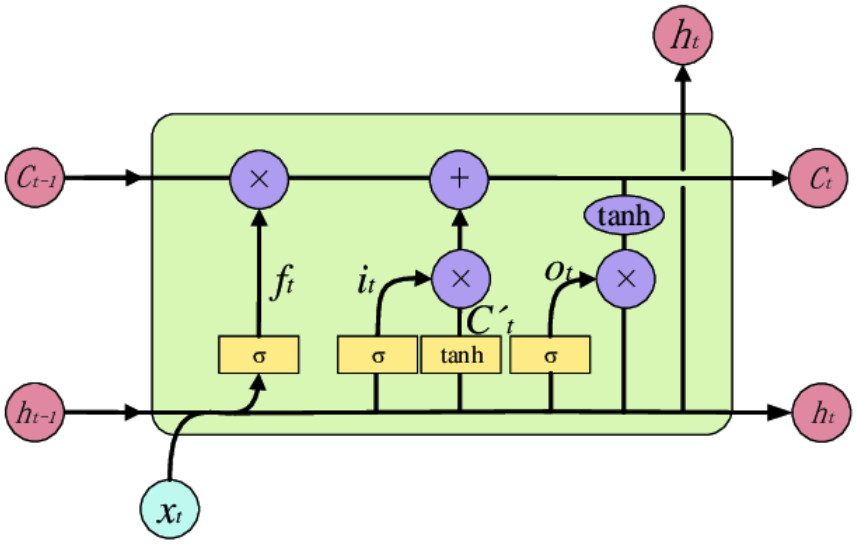

Figure 2.2: Illustration of the LSTM cell, including its key components: forget, input, and output gates, as well as the cell state and hidden state transitions. Reproduced from [35], used under the Creative Commons Attribution License [33].

In time-series forecasting, LSTMs are typically applied by feeding in sequences of past observations and using the final hidden state, or the entire sequence of hidden states, to produce forecasts.

2.4.3 Transformers

The Transformer architecture represents a paradigm shift in sequence modeling by relying entirely on self-attention mechanisms, rather than recurrence, to capture dependencies in sequential data [14]. Originally developed for natural language processing tasks, the Transformer has since been widely adopted and adapted across domains, including time-series forecasting through models such as the Temporal Fusion Transformer [1].

Unlike RNNs, which process input sequentially and propagate information through hidden states, the Transformer enables direct access to all positions in the input sequence via the self-attention mechanism. This allows the model to capture both short- and long-range dependencies with greater parallelization and improved gradient flow, alleviating challenges such as vanishing gradients that often arise in recurrent architectures.

At the core of the Transformer is the *scaled dot-product attention* mechanism. Given an input matrix $X \in \mathbb{R}^{T \times d_{\text{model}}}$, the model computes three learned projections, queries Q, keys K, and values V, through linear transformations:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are trainable parameter matrices and d_k is the dimensionality of the attention space.

The attention weights are then calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where the softmax function normalizes the attention scores across the sequence.

To allow the model to jointly attend to information from different representation subspaces, the Transformer uses *multi-head attention*, where the attention mechanism is applied in parallel h times with distinct linear projections:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O,$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),$$

where $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ is a learned output projection.

Because the self-attention mechanism is invariant to input order, positional encodings are added to the input embeddings to retain information about sequence position. These encodings, typically added once before the first layer in the original Transformer, or at multiple layers in modern implementations, enable the model to distinguish between positions in the sequence. Figure 2.3 illustrates the original Transformer architecture, including input/output embeddings with positional encodings, multi-head attention, feed-forward layers, and the encoder-decoder structure used in sequence-to-sequence tasks. While this figure reflects the general architecture, many time-series models, including those used in this thesis, employ encoder-only variants or modified attention mechanisms.

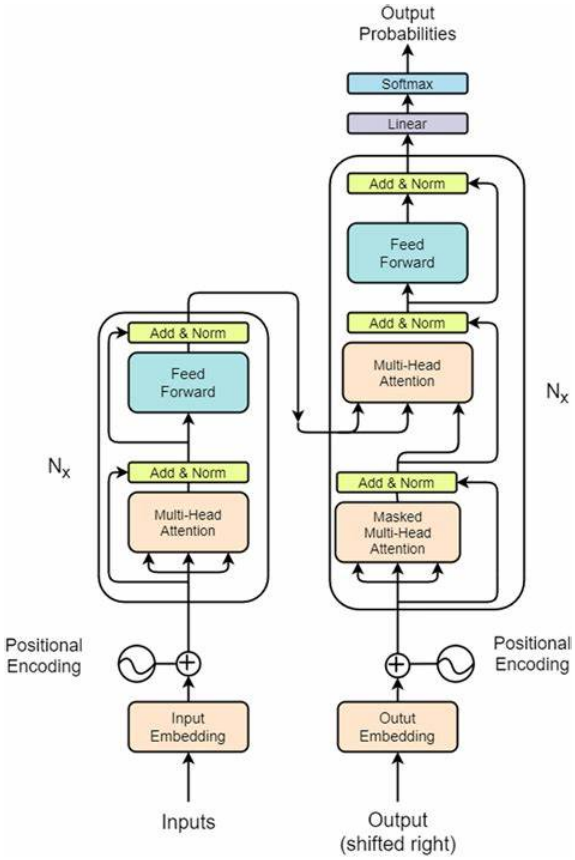

Figure 2.3: Illustration of the Transformer architecture, including its key components: input and output embeddings with positional encodings, multi-head attention, feed-forward layers, and the encoder-decoder structure. Reproduced from original paper [14] with permission for scholarly use.

2.5 Transformer-Based Models

This section presents two modern Transformer-based architectures for time-series forecasting. It introduces the Temporal Fusion Transformer, designed for interpretable multi-horizon forecasting with covariate selection, and TimeXer, a recently developed model that improves the handling of exogenous variables through a dual-stream attention mechanism.

2.5.1 TFT

The Temporal Fusion Transformer (TFT) is a deep learning architecture developed for interpretable multi-horizon time-series forecasting [1]. It combines multi-head self-attention with sequence modeling and input selection mechanisms, allowing it to effectively handle heterogeneous inputs and produce both accurate and interpretable forecasts.

A key component of TFT is the *Gated Residual Network* (GRN), which serves as a flexible building block across the architecture, including in the variable selection

modules (see Figure 2.4). The GRN enables the model to control the level of non-linear processing applied to its inputs, and to preserve or skip information paths depending on the data characteristics. Given a primary input vector $\mathbf{a} \in \mathbb{R}^d$ and an optional context vector $\mathbf{c} \in \mathbb{R}^d$, a GRN computes:

$$\eta_2 = \mathrm{ELU}(W_2\mathbf{a} + W_3\mathbf{c} + \mathbf{b}_2),$$
$$\eta_1 = W_1\eta_2 + \mathbf{b}_1,$$
$$\mathrm{GRN}(\mathbf{a}, \mathbf{c}) = \mathrm{LayerNorm}\left(\mathbf{a} + \mathrm{GLU}(\eta_1)\right),$$

where ELU is the Exponential Linear Unit activation, LayerNorm is standard layer normalization, and W_i, $\mathbf{b}_i$ are learned weights and biases. The GLU gating mechanism is defined as:

$$\mathrm{GLU}(\gamma) = \sigma(W_4\gamma + \mathbf{b}_4) \odot (W_5\gamma + \mathbf{b}_5),$$

where $\sigma(\cdot)$ is the sigmoid function and $\odot$ denotes the element-wise product. This structure allows the model to suppress unnecessary transformations when the data suggests a simpler mapping suffices.

Another central component of TFT is the *Variable Selection Network* (VSN), which provides instance-wise feature selection for both static and time-dependent covariates (see Figure 2.4). Before selection, all input variables are first transformed into a common feature space of dimension d_{model}. Specifically, categorical variables are encoded using learned entity embeddings, while time dependent variables are projected through learned linear layers. This results in transformed feature representations $\boldsymbol{\xi}_t^{(j)} \in \mathbb{R}^{d_{\text{model}}}$ for each of the m input variables at time t.

These transformed vectors are concatenated into a single flattened input $\boldsymbol{\Xi}_t \in \mathbb{R}^{m \cdot d_{\text{model}}}$, which is then passed, along with an optional context vector $\mathbf{c}_s$, through a Gated Residual Network to generate instance-specific attention weights:

$$\mathbf{v}_t = \mathrm{Softmax}(\mathrm{GRN}(\boldsymbol{\Xi}_t, \mathbf{c}_s)).$$

The weights $\mathbf{v}_t \in \mathbb{R}^m$ are then used to compute a weighted sum over the transformed inputs, thereby producing an aggregated feature representation that emphasizes the most relevant variables for prediction at time t. This mechanism allows the model to suppress noisy or redundant inputs dynamically and concentrate its modeling capacity on the most informative signals. Separate variable selection networks are applied for static features, historical covariates, and known future inputs, each using the same structure but with distinct parameters, denoted by different colors in Figure 2.4.

Temporal dependencies in TFT are modeled using a combination of LSTM encoders, which capture short- and medium-range dynamics, and multi-head self-attention layers, which enable the model to focus on relevant time steps across longer horizons. Forecasts are produced via *quantile regression*, which enables the model to generate

probabilistic outputs in the form of prediction intervals. Specifically, for a given quantile level $\tau \in (0, 1)$, the model predicts $\hat{y}_t^\tau$ by minimizing the loss:

$$\mathcal{L}_\tau(y_t, \hat{y}_t^\tau) = \max\left(\tau(y_t - \hat{y}_t^\tau), (\tau - 1)(y_t - \hat{y}_t^\tau)\right).$$

This loss penalizes under- and over-estimation asymmetrically, and allows the model to estimate a full distribution rather than just the mean. A visual overview of the full architecture, including the GRN and VSN modules, is shown in Figure 2.4.

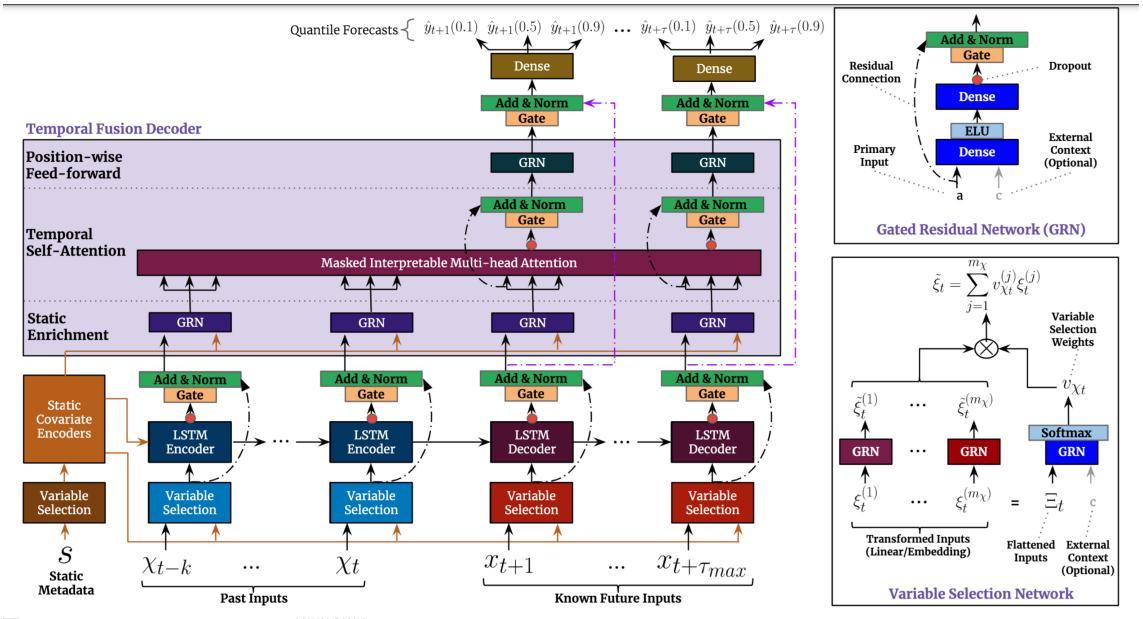

Figure 2.4: Illustration of the TFT architecture, including its key components: static covariate encoders, variable selection networks (VSNs), LSTM encoders, gated residual networks (GRNs), and multi-head attention. Reproduced from the original paper [1], used under the Creative Commons Attribution License [33].

2.5.2 TimeXer

TimeXer is a Transformer-based architecture tailored for multivariate time-series forecasting with exogenous variables. It addresses key limitations in prior attention-based models by introducing a novel two-branch representation and attention design [5].

The model separates the modeling of endogenous and exogenous variables into two distinct representation streams. Given an input time series $\mathbf{x}_{1:T} \in \mathbb{R}^{T \times d}$, TimeXer treats each endogenous variable separately and applies a patch-wise embedding. Specifically, the series is split into $N = \lfloor T/P \rfloor$ non-overlapping patches $\{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$, each of length P. These are projected into temporal tokens via a learned projection:

$$\mathbf{P_{en}} = \text{PatchEmbed}(\mathbf{s}_1, \ldots, \mathbf{s}_N),$$

where each $\mathbf{s}_i \in \mathbb{R}^P$ is flattened and linearly mapped to a D-dimensional embedding. To facilitate interaction with exogenous features, a separate learnable global token

is initialized per target series:

$$\mathbf{G_{en}} = \text{Learnable}(\mathbf{x}),$$

which serves as a global representation of the target variable across time.

In parallel, each exogenous series is embedded as a single-variate token using a dedicated variate-level encoder:

$$\mathbf{V_{ex,i}} = \text{VariateEmbed}(\mathbf{x}_i), \quad i = 1, \ldots, d_x,$$

where d_x is the number of exogenous features, and $\mathbf{V_{ex,i}} \in \mathbb{R}^D$ summarizes the entire history of the i-th exogenous variable.

TimeXer employs a two-stage attention mechanism: in the first stage, self-attention is applied jointly over the patch embeddings $\mathbf{P_{en}}$ and the global token $\mathbf{G_{en}}$, enabling three key interactions: Patch-to-Patch attention to capture local temporal dependencies, Patch-to-Global attention, where the global token aggregates information across patches, and Global-to-Patch attention, where the global context is broadcast back to the individual patches.

Then, cross-attention is used between the global token $\mathbf{G_{en}}$ and the exogenous tokens $\mathbf{V_{ex,i}}$ to inject external context into the prediction pathway. This separation allows exogenous signals to adjust the forecasting process without overwhelming the temporal representation.

As shown in Figure 2.5, TimeXer reuses the standard Transformer architecture without architectural modifications, while applying distinct embedding strategies to endogenous and exogenous inputs. Temporal-wise dependencies within the target series are modeled via self-attention over patch and global tokens, whereas variate-wise dependencies from exogenous inputs are captured through cross-attention, mediated by a learnable global token. By omitting positional encodings for exogenous variables and isolating their influence to the global token, TimeXer achieves robustness to missing values, frequency misalignments, and noise, while remaining fully parallel and compatible with existing Transformer acceleration techniques.

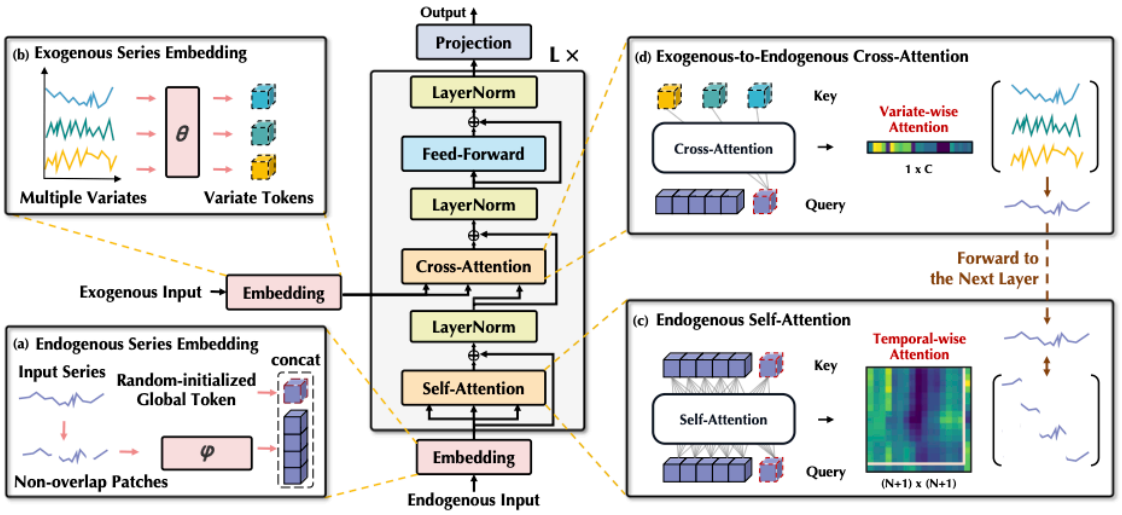

Figure 2.5: Illustration of the TimeXer architecture, which separates temporal and variate streams and integrates exogenous variables via a learnable global token that interacts through cross-attention. Reproduced from the original paper [5], used under the Creative Commons Attribution License [33].

2.6 Multilayer Percepton Models

This section presents three time-series forecasting models based on MLPs. It introduces TiDE, a fully MLP-based model for long-horizon forecasting with covariates; N-HiTS, a hierarchical model that captures temporal structure through interpolation and residual learning; and SOFTS, which leverages cross-channel fusion via an aggregation-redistribution mechanism.

2.6.1 TiDE

The Time-series Dense Encoder (TiDE) is a deep learning architecture for long-term time-series forecasting that explicitly avoids self-attention, recurrent, and convolutional modules in favor of simple MLPs [3]. By relying exclusively on MLP-based residual blocks, TiDE achieves linear computational complexity in both the context length L and forecast horizon H, while retaining the capacity to model non-linear dependencies and incorporate exogenous covariate information.

Formally, given a univariate series $\{y_t^{(i)}\}_{t=1}^{L}$, dynamic covariates $\{x_t^{(i)}\}_{t=1}^{L+H}$, and static attributes $a^{(i)}$, the model produces forecasts

$$\hat{y}_{L+1:L+H}^{(i)} \;=\; \mathrm{TiDE}\Big(y_{1:L}^{(i)},\, x_{1:L+H}^{(i)},\, a^{(i)}\Big).$$

The architecture comprises four main stages: feature projection, dense encoding, dense decoding, and temporal decoding (see Figure 2.6).

TiDE applies the same structure of residual blocks to all components in the architecture. Each residual block is a two-layer MLP with a ReLU activation in the

hidden layer, followed by a linear output layer with dropout. A parallel linear skip connection is added to the block's input, and layer normalization is applied to the sum of main and skip paths.

To avoid prohibitively large inputs, each dynamic covariate vector $x_t^{(i)} \in \mathbb{R}^r$ is mapped via a residual block to a lower-dimensional vector $\tilde{x}_t^{(i)} \in \mathbb{R}^{\tilde{r}}$:

$$\tilde{x}_t^{(i)} \;=\; \mathrm{ResBlock}\Big(x_t^{(i)}\Big), \quad \tilde{r} \ll r.$$

In the dense encoder, the model then flattens and concatenates the past targets $y_{1:L}^{(i)}$, all projected covariates $\tilde{x}_{1:L+H}^{(i)}$, and static attributes $a^{(i)}$, feeding the resulting vector into n_e stacked residual blocks. The encoder output is a fixed-size embedding

$$e^{(i)} = \mathrm{Encoder}\left(\mathbf{y}_{1:L}^{(i)}; \tilde{\mathbf{x}}_{1:L+H}^{(i)}; \mathbf{a}^{i}\right)$$

In the first decoding phase, a second stack of n_d residual blocks maps $e^{(i)}$ to a vector $g^{(i)} \in \mathbb{R}^{pH}$, which is then reshaped into

$$D^{(i)} \;=\; \mathrm{Reshape}\Big(g^{(i)}\Big) \;\in \mathbb{R}^{p\times H},$$

so that column $d_t^{(i)}$ represents the decoded feature for time $L+t$ $(t = 1, \ldots, H)$.

For each horizon step t, a temporal decoder residual block consumes the concatenation of $d_t^{(i)}$ and $\tilde{x}_{L+t}^{(i)}$ to yield the final forecast:

$$\hat{y}_{L+t}^{(i)} \;=\; \mathrm{TemporalDecoder}\Big[d_t^{(i)};\, \tilde{x}_{L+t}^{(i)}\Big], \quad t = 1, \ldots, H.$$

In addition, a global linear residual connection maps the input series $y_{1:L}^{(i)}$ directly to a length-H vector, which is added to $\hat{y}_{L+1:L+H}^{(i)}$. This ensures that a purely linear variant of TiDE is a special case of the full model.

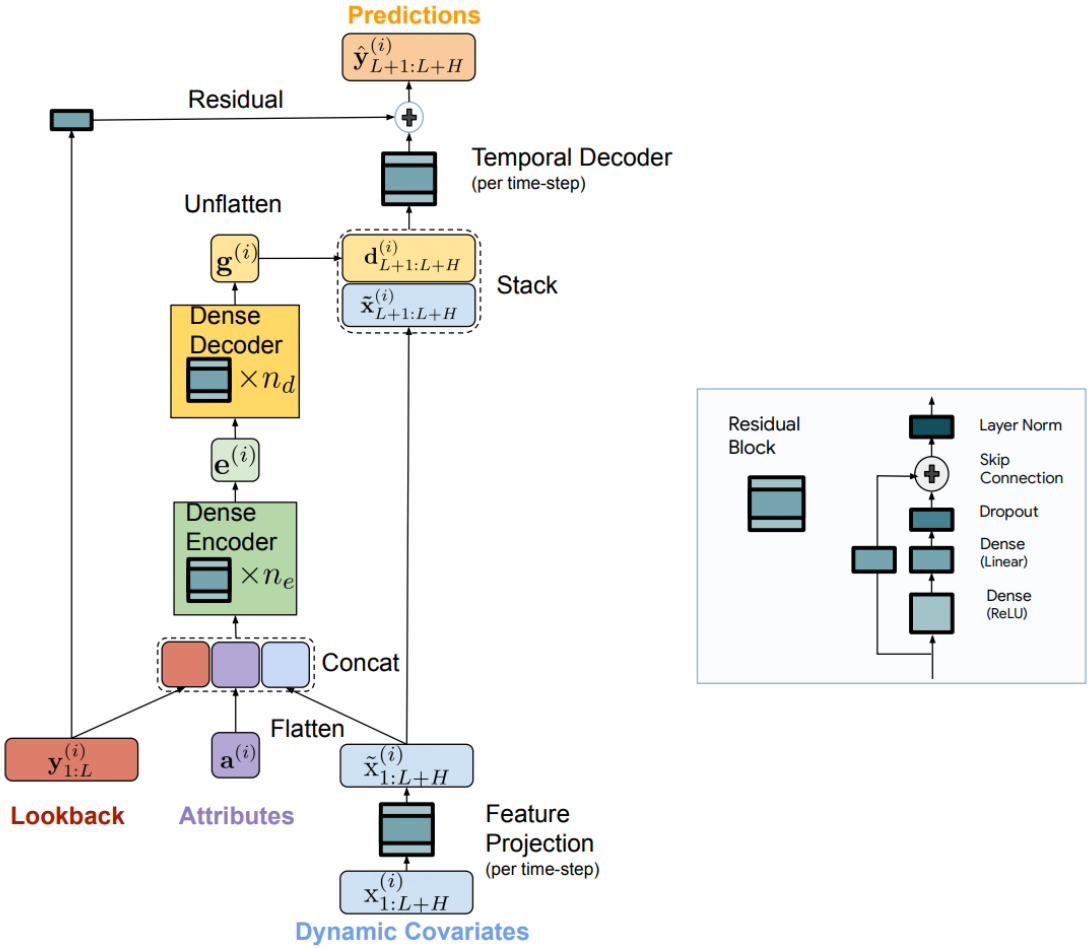

Figure 2.6: Illustration of the TiDE architecture, including the dense encoder and decoder, and temporal decoder combined with a global residual. Reproduced from the original paper [3], used under the Creative Commons Attribution License [33].

2.6.2 N-HiTS

The Neural Hierarchical Interpolation for Time Series Forecasting (N-HiTS) model is a deep learning architecture proposed for accurate and efficient long-term forecasting [4]. Building upon the residual block structure of N-BEATS [36], N-HiTS introduces two key innovations: hierarchical interpolation and multi-rate input sampling. These mechanisms allow the model to capture patterns at multiple temporal resolutions, enabling it to efficiently handle long input sequences and extended forecast horizons.

N-HiTS adopts a hierarchical stack-block architecture, where the model is composed of multiple *stacks*, and each stack consists of several *blocks*. Each block applies a learned downsampling (via max pooling) to its input, processes the result through a multilayer perceptron (MLP), and generates two outputs: a *forecast* and a *backcast*. The forecast represents the block's prediction of future values, while the backcast approximates the input component the block can explain. This backcast is subtracted from the input and passed as a residual to the next block, enabling sequential refinement.

Formally, given an input time series $y \in \mathbb{R}^L$ and a forecast horizon H, each block b processes a downsampled version of the residual signal:

$$\tilde{y}_b = D_b(y_{\text{res}}),$$

where D_b is a max-pooling operator with block-specific kernel size k_b. The downsampled input is processed via an MLP to produce a coarse forecast $f_b \in \mathbb{R}^{H_b}$, which is upsampled using a temporal interpolation operator:

$$\hat{y}_b = \text{Interp}_b(f_b), \quad \hat{y}_b \in \mathbb{R}^H.$$

The model accumulates forecasts additively across blocks:

$$\hat{y} \leftarrow \hat{y} + \hat{y}_b,$$

while updating the residual input as:

$$y_{\text{res}} \leftarrow y_{\text{res}} - \text{Backcast}_b(\tilde{y}_b).$$

Crucially, each stack in N-HiTS is designed to operate at a distinct temporal resolution. Early stacks use larger pooling kernels and coarser interpolation (low expressiveness ratio), allowing them to model long-term trends and low-frequency components of the signal. Later stacks use smaller pooling kernels and higher interpolation resolution, refining the forecast by capturing medium- and high-frequency components. Residuals are passed between stacks in the same way as between blocks.

By composing the final forecast as the sum of interpolated outputs from blocks across all stacks, N-HiTS effectively performs a hierarchical decomposition of the signal. This design allows it to isolate and model patterns at various frequencies, from slow-moving trends to rapid fluctuations. For a visual representation of the N-HiTS architecture, see Figure 2.7.

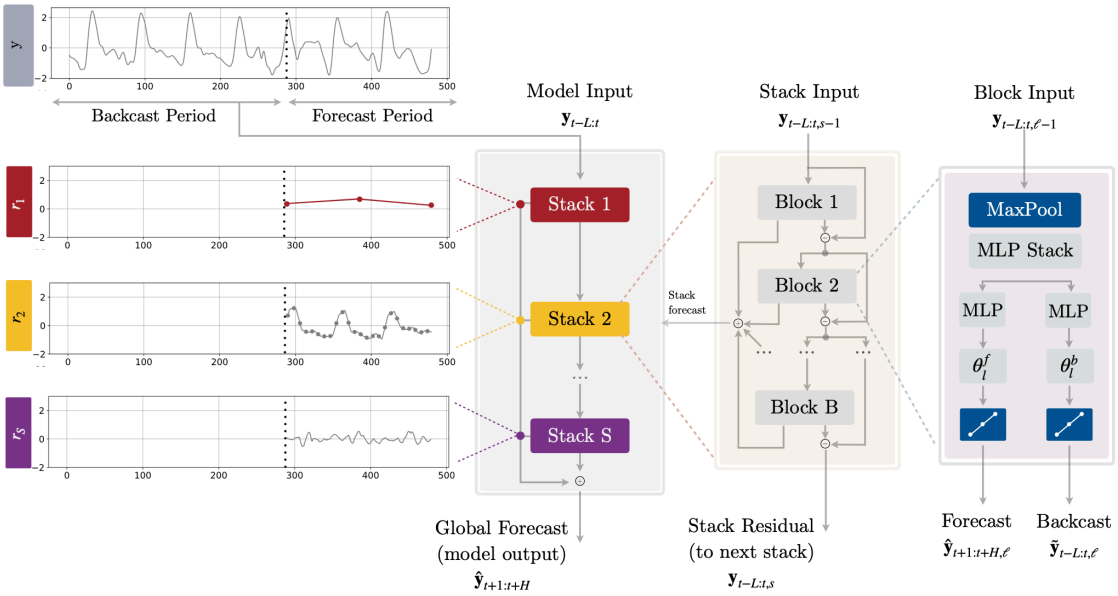

Figure 2.7: Illustration of the N-HiTS architecture. Each stack consists of multiple residual blocks operating at a specific temporal resolution. Blocks downsample their input, perform non-linear projections, and output both a forecast and a backcast. Forecasts are interpolated to the target horizon and summed to form the final prediction. Reproduced from the original paper [4], used under the Creative Commons Attribution License [33].

2.6.3 SOFTS

The Series-cOre Fused Time Series forecaster (SOFTS) is an MLP-based implementation designed for efficient forecasting of multivariate time series data [2]. SOFTS introduces a unique module called STar Aggregate-Redistribute (STAR), which efficiently capture correlation among channels with linear computation complexity, addressing common limitations of attention mechanisms. It is noted that, within this context, "channels" refer to the distinct time series to be forecasted, and that SOFTS does not accommodate exogenous variables.

Formally, given historical observations $y \in R^{C \times L}$, where C denotes the number of channels and L the length of the historical window, SOFTS generates forecasted values $\hat{y} \in \mathbb{R}^{C \times H}$, where horizon $H > 0$.

Prior to model input, SOFTS applies reversible instance normalization to each channel of the input series $X \in \mathbb{R}^{C \times L}$, centering and scaling to unit variance, and subsequently inverts this normalization on the forecasted outputs to restore the original data scale [2].

The normalized series is projected via a single linear layer to obtain initial channel embeddings

$$S_0 = \text{Embedding}(y), \quad S_0 \in R^{C \times d},$$

where d is the hidden dimensionality of the embeddings.

Subsequently, these channel embeddings are iteratively refined via multiple layers of the STAR module. Each STAR module comprises two sequential operations: aggregation and redistribution.

The aggregation step condenses inter-channel information. A global core representation $o_i \in \mathbb{R}^{d'}$ is computed by applying a two-layer MLP with GELU activation $(\mathrm{MLP}_1 : \mathbb{R}^d \to \mathbb{R}^{d'})$ to each channel embedding, followed by stochastic pooling across channels

$$o_i = \mathrm{Stoch\_Pool}\Big(\mathrm{MLP}_1(S_{i-1})\Big).$$

Stochastic pooling, as employed here, introduces regularization by selecting output values according to a probability distribution derived from the pooling map, in contrast to deterministic methods such as max pooling [2], [37].

During the redistribution step, the core vector o_i is concatenated with each preceding channel embedding, yielding

$$F_i = \mathrm{Repeat\_Concat}\Big(S_{i-1},\, o_i\Big) \;\in\; \mathbb{R}^{C\times(d+d')},$$

which is then mapped back to $\mathbb{R}^d$ via another MLP $(\mathrm{MLP}_2 : \mathbb{R}^{d+d'} \to \mathbb{R}^d)$ and combined with a residual connection

$$S_i = \mathrm{MLP}_2(F_i) + S_{i-1}.$$

Stacking N STAR modules yields progressively refined embeddings S_N.

The final embeddings are linearly projected to the forecasting horizon H for each channel to generate the final prediction

$$\hat{y} = \mathrm{Linear}(S_N), \quad \hat{y} \in \mathbb{R}^{C\times H}$$

See Figure 2.8 for a visual overview of the SOFTS architecture.

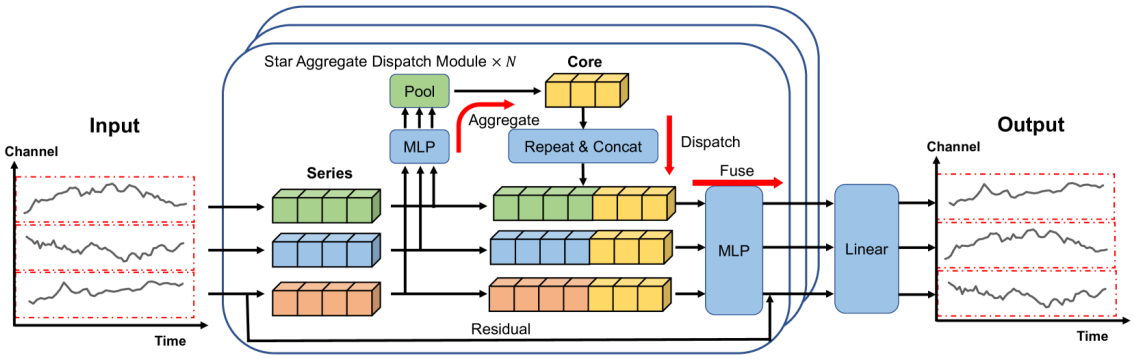

Figure 2.8: Architecture of the SOFTS model, illustrating how the STar Aggregate-Dispatch (STAD) module centralizes global information via a core representation and then fuses it back into each channel. Reproduced from the official project repository [38], used under the MIT License [39].

3

Methods

Having laid out the theoretical foundations in Chapter 2, Chapter 3 presents the practical framework in which those models are implemented. It describes data sources utilized and the preprocessing steps taken. Model implementation via the utilized libraries is then detailed. Several key transformations and modeling decisions are also described to improve the forecasting performance of the specific dataset used. Finally, the evaluation protocol based on rolling origin forecasts with sMAPE and RMSE metrics across multiple forecast horizons is outlined.

3.1 Data

The datasets utilized in this study span the years 2014 through 2024. The primary data source is the order book, which records historical orders placed, and is supplemented by several exogenous variables. Additionally, historical forecasts produced using Company X's current procedures are available to provide insight into the existing forecasting process.

3.1.1 Order book

The order book constitutes a dataset in which each row corresponds to a single vehicle order. It comprises 75 features, which can be classified into the following categories: (1) geographical features, (2) product features, (3) temporal features, and (4) miscellaneous features. These categories are summarized in Table 3.1.

Table 3.1: Summary of the order book dataset.

Category	**Description**	**Number of features**
Geographical features	Hierarchy from larger regions to a specific country.	4
Product features	Hierarchy from product usage description to specific model variants such as power supply, gearbox, and more.	33
Dates	Ranging from initial quotation date until delivery to customer	15
Miscellaneous	Specific order details that are of less interest for this study.	23

The geographical features indicate the location where an order was placed. These features adhere to an organizational structure that aligns with existing processes for demand forecasting. The highest hierarchical level corresponds to a broad sales region comprising multiple countries, whereas the lowest level represents individual countries.

The dataset includes 13 date-related features, ranging from the initial quotation date to the delivery date of the vehicle to the customer. Among these, the order date serves as the ground truth and constitutes the target variable for forecasting. This date reflects order intake, that is, the point at which the order is formally recognized. Since each record in the order book corresponds to a unique order with a specific order date, the dataset can be aggregated at various levels of temporal granularity.

Furthermore, the dataset contains 33 product-related features. The product features encompass both high-level classifications, such as the intended vehicle application, and more detailed specifications, including power supply and engine power.

Finally, the dataset includes a number of miscellaneous fields that do not fall within the previously mentioned categories. Among these, only the order status variable is considered in this study. This field indicates whether an order has been placed, fulfilled, or canceled.

Figure 3.1 presents a selection of the time series analyzed in this study. At the highly aggregated Europe 30 level (reflecting the complete dataset), distinct seasonal patterns are readily apparent. By contrast, at the country level (for example, Spain), the series display greater stochasticity. Certain disaggregated series, such as Spain Product Group 1 and Power Supply 2 (PG1, PS2), exhibit no observations prior to specific dates, reflecting the introduction of new products. Additionally, the post-COVID-19 period is marked by heightened variance and an increased frequency of outliers. Together, these characteristics, random fluctuations, episodic volatility, and structural breaks due to product launches, introduce considerable complexity, highlighting the imperative for suitable data transformations before model training.

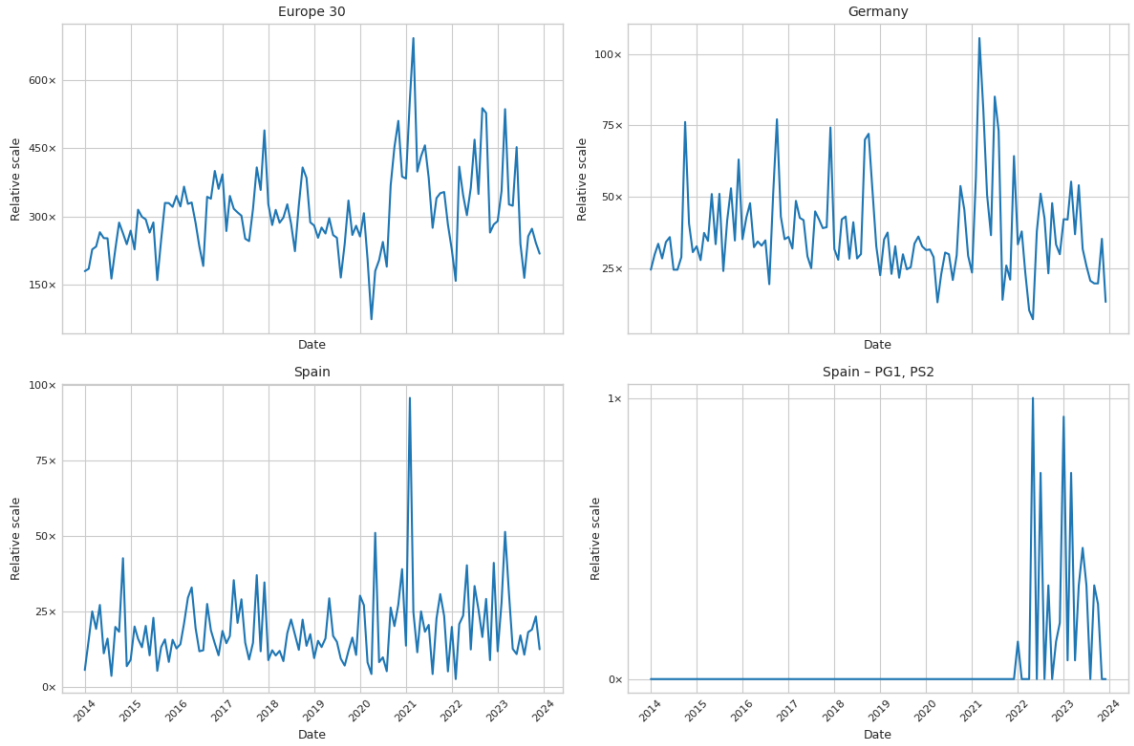

Figure 3.1: Illustration of four representative time series used in this study. The aggregated Europe 30 series (top-left) exhibits pronounced seasonal patterns and trend components. The Germany market order intake (top-right) is characterized by numerous outliers and elevated variability during the COVID-19 period. The Spain market series (bottom-left) shows stochastic fluctuations. The series for a specific product group and power supply in Spain (bottom-right) highlights structural breaks associated with new product introductions. The y-axis have been normalized and scaled due to confidentiality.

3.1.2 Exogenous Regressors

In addition to the order book, the Total Market Forecast (TMF) constitutes another internal data source available for model training and inference. This dataset captures the projected number of vehicle registrations, irrespective of manufacturer. It is aggregated on a yearly basis and is generated quarterly, providing high-quality forecasts for both the current and subsequent year. The geographical resolution is at the country level. The TMF dataset contains two numerical fields: the forecasted value and the corresponding actual value.

Alongside the TMF, gross domestic product (GDP) was incorporated as an additional exogenous regressor. GDP, a recognized indicator of a country's economic situation, was chosen on the basis of domain expertise at Company X. However, the absolute GDP series displays a pronounced upward trend, meaning that future monthly values would likely exceed any previously observed levels and could violate the stationarity assumptions of some forecasting methods. To mitigate this issue and better align with model requirements, GDP growth was used instead, defined

as the percentage change in GDP from one month to the next.

3.1.3 Data Selection

The datasets employed in this study originate from a large number of countries. To ensure the feasibility of the analysis, a selected subset of countries was used in combination with one higher-level aggregation: Germany, Spain, and Europe 30, the latter representing an aggregation of 30 European countries.

At the product level, the data have also been aggregated. In alignment with current practices at Company X, the dataset has been organized into three product categories and three power supply categories.

Given the hierarchical nature of the problem, forecasts will be generated at multiple levels. Table 3.2 illustrates an example of this structure. The combination of hierarchical features results in 35 time series utilized in this study.

Table 3.2: Example of the hierarchy used in the study.

Level	**Time series**
Level 1	Europe 30
Level 2	Europe 30 / Product Group 1
Level 2	Europe 30 / Germany
Level 3	Europe 30 / Germany / Product Group 1
Level 4	Europe 30 / Germany / Product Group 1 / Power Supply 1

3.2 Preprocessing

In preparation for model training, data preprocessing was undertaken. The raw dataset was subjected to a series of cleaning and reformatting operations. Insights obtained from data exploration guided the identification of required data transformations.

3.2.1 Data Cleaning and Hierarchical Aggregation

The order book dataset employed in this study required modification to support model development. First, the hierarchical structure was established in accordance with Company X's operational procedures. As detailed in Section 3.1.1, each record in the order book corresponds to a single order at the most detailed level. To construct the desired hierarchy, product categories and geographical groupings were merged and, where necessary, further disaggregated.

Once the groupings were defined, the data were aggregated to monthly intervals by summing the number of orders for each specific hierarchical level. Where a particular grouping recorded no orders in a given month, a value of zero was imputed. All

monthly timestamps were standardized to the first day of each month. Thus, the final modeling dataset comprised every month for each time series and consists of the following columns:

- The hierarchical series identifier.
- A timestamp column, indicating the date to which each observed value pertains.
- The target variable.

To integrate the exogenous covariates with the order book time series, each country- or product-level metric was first aligned to its corresponding hierarchy node. Specifically, the relevant indicators were extracted from both internal and external data sources at every hierarchical level, and then concatenated into a single panel indexed by the order book series identifier and date.

Because the covariates comprise both realized observations and forecasts, a "cut-off" mechanism was implemented to prevent look-ahead bias in temporal cross-validation and backtesting. For each series and forecast origin, the latest timestamp at which forecasts would have been available in real time was recorded. This cut-off date was derived by reconstructing the historical forecast releases. Any covariate values with timestamps later than the cut-off were excluded from model fitting at that origin.

The resulting exogenous dataset therefore consists of:

- The hierarchical series identifier (matching the order book).
- A cut-off date, representing the information horizon for each forecast origin.
- A timestamp column, indicating the date to which each observed or forecasted value pertains.
- One column per exogenous feature, each reflecting either the observed value or the forecast available as of the cut-off.

3.2.2 Data Splitting

Partitioning data to evaluate models in time-series contexts requires careful consideration to prevent data leakage. Prior to any data exploration and hyperparameter tuning, a test set comprising data from the year 2024 was set aside. This subset remained unused until the final evaluation of the developed models. Consequently, the training and validation set spans ten years of data, resulting in an approximate split of 90% for training and validation and 10% for testing.

3.2.3 Data transformations

Some of the resulting time series contained values at or near zero which can induce instability in forecasting models and result in negative point predictions. To address this issue, all series were transformed using the natural logarithm. This transformation enforces non-negative forecasts and stabilizes variance across series, thereby

promoting homoscedastic behavior [19]. Specifically, a log plus one transformation was used to handle zero-values data points. In addition, the logarithmic transformation aids in normalizing the distribution of values and mitigates the influence of outliers. The normalization will primarily aid models trained globally, while all models benefit from reduced influence of outliers.

3.3 Implementation

Once data transformation procedures were complete, the implementation of forecasting models was initiated. This section details the application of established Python libraries for model implementation and describes the systematic hyperparameter-tuning framework used to search for optimal configurations.

3.3.1 Model implementation

All models were implemented using the NIXTLA library to ensure methodological consistency and reproducibility across model classes. Statistical forecasting methods were accessed through the *statsforecast* library [40], which provides efficient implementations of classical time-series models. GBDT algorithms, including XGBoost and LightGBM, were integrated using their native Python libraries via NIXTLA's *mlforecast* interface [41], which facilitates feature engineering and model training in a time-series context. Deep learning architectures were implemented using the *neuralforecast* library [42], which is based on the PyTorch framework.

For the statistical models AutoARIMA and AutoETS, the *statsforecast* library was used. These models differ from others in two key respects. Firstly, they include automated procedures for model selection and parameter estimation based solely on the training data, thereby eliminating the need for manual hyperparameter tuning. The algorithms internally evaluate a predefined model space and select the best-performing configuration as explained in section 2.2.3. As a result, their forecasting performance was directly evaluated without conducting a separate tuning phase.

Secondly, unlike the ML and DL models that are trained globally across all time series, AutoARIMA and AutoETS were trained locally, meaning one model per series. This is due to the structural assumptions and estimation procedures underpinning these classical statistical methods, which are inherently univariate and do not benefit from data pooling. The only parameter explicitly set for these models was the seasonal period, which was fixed to 12 to match the monthly frequency of the data.

For XGBoost and LightGBM, which do not natively account for temporal dependence, the mlforecast library automatically generated the required lagged features and statistical covariates. This level of automation not only streamlined the experimental design but also ensured that feature engineering procedures were applied consistently across modeling approaches. Furthermore, the ML models support both local an global training procedures. To determine the best approach, a simple cross-validation on the training set with light hyperparameter tuning was conducted. Initially, the outcome indicated no significant difference between the two approaches.

However, through further experimentation, it was determined that adding an additional categorical feature, representing the identifier of each series, proved necessary. After this, the results demonstrated superior performance for globally trained models, inline with expectations given the limited historical dataset of each time series.

The neural network architectures were implemented using NIXTLA's interface built on the PyTorch framework. This interface automates the transformation of raw inputs into PyTorch dataloaders and systematically configures model hyperparameters. Additionally, it facilitates the seamless integration of exogenous covariates into each networks training pipeline.

By default, all neural forecasting models are trained globally, allowing them to learn common patterns across series and leverage the increased volume of training examples, an advantage for more complex architectures.

By adopting this unified framework, a single, consistent dataset, containing historical order book observations and exogenous variables, could be provided to every model. Moreover, it permitted flexible model training, either fitting separate models to each series or fitting a single model across all series when supported by the respective algorithm.

Initial experiments using default hyperparameters revealed that forecasts were disproportionately influenced by extreme outliers, particularly from periods such as the COVID-19 pandemic, the 2021 semiconductor crisis, and increased inflation in 2022-2023. To address this issue, robust scaling techniques and alternative loss functions were evaluated via cross-validation. Results indicated that the use of Huber loss as the objective function without additional scaling produced the best performance. Huber loss applies a squared error penalty for residuals within a specified threshold and an absolute error penalty beyond that threshold, thereby balancing sensitivity to small residuals with resistance to outliers [43]. Accordingly, Huber loss was employed for all machine learning and deep learning models. Since XGBoost supports only the pseudo Huber loss, a smooth, differentiable approximation of Huber loss, this function was used for that model [44].

3.3.2 Hyperparameter Tuning

To enable consistent model comparisons, a standardized framework for hyperparameter optimization was developed. This framework ensures that all models are evaluated under comparable conditions.

Searching for the optimal hyperparameters of each model was executed based on temporal cross-validation. An expanding window strategy was employed, in which the training set grew progressively with each fold. The process began with an initial training window of 90 observations (equivalent to 7.5 years), which was expanded by 3 observations per fold, yielding a total of 7 folds. Each validation fold covered a fixed forecast horizon of 12 periods (see Figure 3.2)

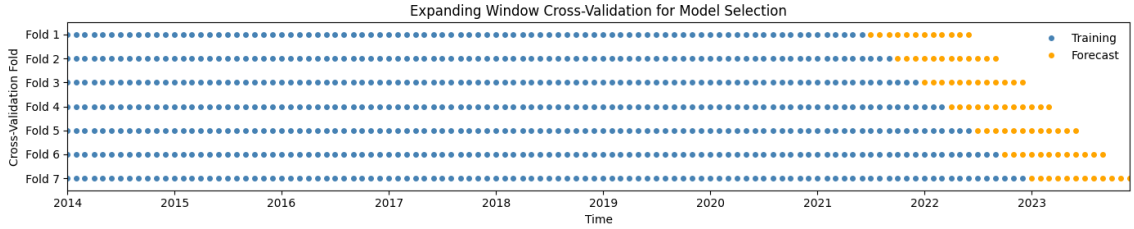

Figure 3.2: Visual representation of expanding window cross-validation.

Due to the substantial computational requirements and the expansive range of possible configurations, the hyperparameter search for ML and DL model was constrained to 100 iterations. When an exhaustive examination of every combination is impractical, two commonly adopted alternatives are random search and Bayesian optimization [45], [46]. In random search, a fixed number of configurations are sampled uniformly at random from the predefined parameter bounds. Although this method is straightforward to implement, its efficacy diminishes in high-dimensional spaces and it may miss regions containing near-optimal settings [47].

Bayesian optimization, by contrast, employs a probabilistic surrogate model of the objective function, often a Gaussian process or a Tree-structured Parzen Estimator, to predict performance across the search space [48], [49]. An acquisition function then identifies the most promising candidates by balancing exploration of unknown regions against exploitation of areas likely to yield improvements. Empirical studies have demonstrated that Bayesian approaches typically locate superior hyperparameter configurations more efficiently than random search [47]. Standard implementations of Bayesian optimization require continuous parameter domains, however most of the models utilized in this study require integer-valued or categorical hyperparameters. To address this limitation, the HyperOpt library was employed, which extends Bayesian optimization via Tree-structured Parzen Estimators and natively supports mixed parameter types [50], [51]. Lastly, since Bayesian methods benefit from an initial set of observations to train the surrogate model, HyperOpt is seeded with 16 randomly sampled configurations. The remaining 84 evaluations were then chosen iteratively by HyperOpt's acquisition function.

Each model was tuned independently under two scenarios:

- **Baseline**: trained using only historical target values.
- **Exogenous**: trained using exogenous variables, if supported.

The choice of parameter distributions was informed by empirical findings from similar use cases and the original papers of respective model. For the complete list of search spaces, refer to Appendix A.

3.4 Evaluation

This section describes the procedures used to evaluate the performance of the implemented forecasting models. The evaluation strategy was designed to be both

rigorous and aligned with real-world forecasting conditions, with a particular focus on how models perform across different forecasting horizons.

3.4.1 Expanding Window Forecasting and Multi-Prediction Coverage

To ensure robust and horizon-aware evaluation, a rolling origin setup with an expanding window was employed. At each training origin, the model was trained on all available data up to that point and used to produce a twelve-step-ahead forecast. This process was repeated sequentially across all months from January 2023 to November 2024, yielding 23 origin points in total.

From each origin month, the model predicted the next 12 months, but only predictions falling within the defined test period (January 2024 to December 2024) were retained for evaluation. Consequently, each of the 12 test points in 2024 received 12 separate forecasts, one from each of the 12 preceding origin months, resulting in 144 points for evaluation. This setup ensures that all test points are evaluated consistently across all forecast horizons from 1 to 12.

More formally, let y_t denote the actual value for month t, and let $\hat{y}_t^{(o)}$ be the forecast for month t made from origin o, such that $t = o + h$, where $h \in [1, 12]$. Each test point y_t for $t \in \{\text{Jan 2024}, ..., \text{Dec 2024}\}$ was thus predicted using all origin months $o \in \{t - 12, ..., t - 1\}$, corresponding to forecast horizons $h = 12, ..., 1$. For a visual representation of this evaluation setup, see Figure 3.3

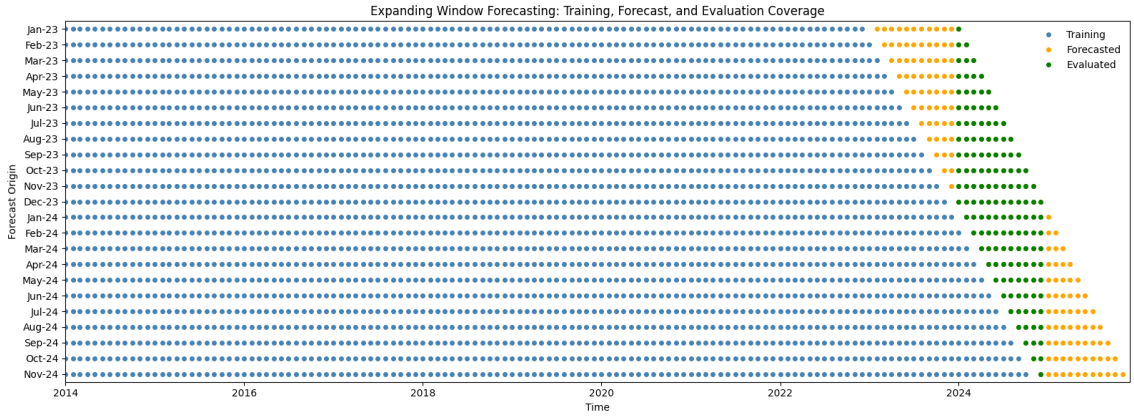

Figure 3.3: Visualization of the expanding window evaluation setup. Each row represents a forecast origin. Blue markers indicate the training period, orange markers represent the 12-month forecast horizon, and green markers denote forecasted values that fall within the 2024 evaluation period and are retained for error calculation.

This strategy offers several advantages:

- Each forecast is made strictly from historical data, respecting the causal nature of time-series forecasting.
- It mimics realistic deployment settings where models are retrained periodically.

- It provides a robust error distribution across horizons for each target month, enabling fair comparison of model performance at different lead times.

3.4.2 Baseline vs. Exogenous Data Evaluation

Model evaluation proceeded in multiple stages. Because some of the chosen models do not support exogenous regressors, while others accommodate only static, future, or past covariates, an initial step was to establish a baseline by assessing each model's performance using only historical order intake (i.e., with no exogenous variables). However, given that a few of the models demonstrate their strengths through the integration of exogenous information, an additional evaluation was conducted in which each model utilized the full range of exogenous data it is capable of handling.

3.4.3 Multi-Horizon Evaluation

Moreover, because this problem is framed as a multi-horizon forecasting task, it is interesting to examine how each model performs over different forecast horizons, particularly to discern which methods are more effective for short- and medium-term predictions. Accordingly, evaluations were performed for the following intervals:

- 1-3 months
- 4-6 months
- 7-9 months
- 10-12 months

3.4.4 Evaluation Metrics

To evaluate forecasting performance, two complementary metrics were employed: symmetric mean absolute percentage error (sMAPE) and root mean squared error (RMSE). These metrics were selected to provide distinct yet informative perspectives on model accuracy across different aggregation levels and demand magnitudes.

sMAPE was chosen as a percentage-based metric to enable performance comparisons between series with differing scales. While the traditional mean absolute percentage error (MAPE) is often used for this purpose, it becomes unstable when actual values approach zero, a common occurrence in some of the lower-demand series in this dataset. To address this, sMAPE was used instead, offering more stable behavior by normalizing the absolute error by the average of the actual and forecasted values:

$$\text{sMAPE} = \frac{1}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{(|y_t| + |\hat{y}_t|)/2}$$

where y_t denotes the actual value at time t, $\hat{y}_t$ the corresponding forecast, and n the number of observations.

In contrast, RMSE quantifies the absolute magnitude of forecasting errors in the same units as the target variable. It penalizes larger errors more heavily due to the squaring operation and is particularly useful in scenarios where accurately predicting high-volume markets or aggregate-level demand is more critical. As such, RMSE provides additional insight into practical performance implications for industrial planning and operations. It is defined as:

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

Together, sMAPE and RMSE offer a balanced view of model performance, capturing both relative and absolute error behaviors across a heterogeneous set of time series.

4

Results

This chapter presents the results obtained from evaluating the forecasting models described in the previous chapter. The results are structured to reflect the evaluation strategy, covering baseline comparisons, the use of exogenous data, and performance across different forecasting horizons. Lastly, representative forecast plots are presented to visually illustrate model behavior and highlight differences in forecast dynamics across model types and series.

4.1 Overall Model Performance

Table 4.1 presents the forecast performance of all implemented models without exogenous inputs, across the three regional datasets (Europe 30, Germany, and Spain). Metrics are averaged over a 12-month forecast horizon using sMAPE and RMSE, with lower values indicating better accuracy.

In the Europe 30 series, AutoETS achieved the best overall performance for the high-level "Total" series, reporting the lowest sMAPE (0.21) and RMSE (1212). AutoARIMA also performed strongly across several aggregated series. At the disaggregated level, LightGBM and XGBoost frequently delivered better results than the other models. For instance, LightGBM achieved the lowest RMSE in the PG1 series (1143), and both tree-based models performed well in the PG1 - PS2 and PG2 - PS2 subsegments, where model differences were especially pronounced. DL models such as TFT and TiDE struggled on average in this region, although N-HiTS reported solid results in specific subseries (e.g., PG1 - PS3).

For the Germany series, LightGBM emerged as the most accurate model overall, with the lowest RMSE in the "Total" series (155) and PG1 (155). AutoARIMA again showed strong sMAPE performance in several series, particularly PG2, where it achieved the lowest sMAPE (0.41). Neural models like TiDE and N-HiTS reported higher RMSEs across most subsegments, while tree-based models performed more consistently across all series. Disaggregated series such as PG1 - PS2 and PG2 - PS2 displayed large performance gaps, with LightGBM and XGBoost often outperforming others.

In the Spain series, SOFTS achieved the best RMSE in the high-level "Total" series (176), followed by TimeXer and XGBoost. Futhermore, TimeXer showed strong results in multiple subseries, matching or exceeding performance of other models in

low-variance segments such as PG1 - PS1. Meanwhile, other DL models such as TFT and TiDE were less competitive, although TiDE achieved the best sMAPE in PG1 - PS3.

Overall, the results suggest that neural forecasting models generally underperform compared to classical statistical and tree-based approaches when no exogenous inputs are available. Most neural models, including TFT, TiDE, and N-HiTS, yielded higher average error rates across the majority of series. An exception is SOFTS, which delivered competitive results in some disaggregated subseries, particularly in the Germany and Spain datasets.

The final row of Table 4.1 summarizes average performance across all series and regions. AutoARIMA reported the lowest average sMAPE (0.88), followed by LightGBM and SOFTS (both at 0.92). In terms of RMSE, XGBoost achieved the lowest overall average (360), closely followed by LightGBM (362) and AutoARIMA (367).

Table 4.1: Forecasting performance across selected datasets (12-step horizon). Metrics are averaged over all rolling predictions per test point. Best results per row are highlighted in bold.

	Series	AutoARIMA		AutoETS		LightGBM		XGBoost		TFT		TiDE		N-HiTS		SOFTS		TimeXer	
		sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE
Europe 30	**Total**	0.22	1241	**0.21**	**1212**	0.25	1242	0.22	1237	0.26	1401	0.26	1492	0.33	1774	0.24	1289	0.23	1271
	PG1	0.23	1230	0.22	1204	0.25	**1143**	**0.21**	1211	0.28	1397	0.26	1462	0.34	1778	0.24	1270	0.24	1247
	PG2	0.41	90	**0.36**	**79**	0.38	92	0.46	97	0.42	92	0.44	100	0.53	155	0.43	92	0.43	92
	PG3	0.69	6	**0.66**	5	0.85	6	0.67	**4**	0.83	8	0.82	7	0.87	7	0.72	7	0.72	7
	PG1 - PS1	0.23	1179	0.23	1265	0.29	1227	**0.22**	**1145**	0.29	1406	0.28	1463	0.33	1718	0.26	1279	0.26	1267
	PG1 - PS2	0.60	103	0.38	54	**0.21**	**21**	1.29	75	0.75	222	0.65	213	0.70	171	0.36	38	0.49	44
	PG1 - PS3	0.49	**96**	0.54	101	0.66	131	0.59	115	0.49	110	0.52	108	**0.48**	100	0.52	107	0.53	105
	PG2 - PS1	0.38	71	**0.32**	**64**	0.38	79	0.44	88	0.40	79	0.40	76	0.50	118	0.39	73	0.40	74
	PG2 - PS2	0.81	10	0.81	10	0.79	7	**0.67**	**6**	0.91	14	1.04	15	0.84	13	1.03	13	1.05	14
	PG2 - PS3	1.11	**31**	1.05	32	1.04	34	1.05	35	1.09	35	1.18	39	1.18	42	1.04	32	**1.03**	31
	PG3 - PS1	**0.00**	**0**	2.00	0	2.00	2	2.00	0	2.00	0	2.00	0	2.00	0	2.00	0	2.00	0
	PG3 - PS2	0.65	5	0.64	5	**0.57**	5	0.64	**4**	0.81	8	0.82	7	0.87	7	0.72	7	0.72	7
Germany	**Total**	0.35	180	0.39	209	**0.25**	**155**	0.28	164	0.39	216	0.37	213	0.45	291	0.33	180	0.36	202
	PG1	0.34	173	0.39	204	**0.26**	**155**	0.28	162	0.41	220	0.37	211	0.45	291	0.33	177	0.36	200
	PG2	**0.41**	3	0.53	5	0.41	**3**	0.43	4	0.53	5	0.62	6	0.63	9	0.61	6	0.63	7
	PG3	1.30	2	1.30	2	1.43	2	1.57	2	1.56	3	1.34	2	1.41	4	**1.09**	**2**	1.11	2
	PG1 - PS1	0.38	183	0.43	213	**0.26**	**152**	0.31	163	0.45	234	0.39	213	0.47	282	0.35	179	0.37	199
	PG1 - PS2	0.83	32	0.87	60	0.76	17	0.90	17	0.69	15	0.74	36	0.91	41	**0.67**	**14**	0.69	15
	PG1 - PS3	1.39	13	1.38	13	1.29	**12**	**1.26**	12	1.49	13	1.55	15	1.44	14	1.33	13	1.40	13
	PG2 - PS1	**0.46**	3	0.49	4	0.49	**3**	0.50	3	0.69	6	0.80	5	0.79	6	0.55	4	0.57	5
	PG2 - PS2	1.53	3	1.50	2	1.58	**1**	1.72	1	1.70	5	1.58	4	1.58	4	**1.46**	2	1.47	2
	PG3 - PS1	**2.00**	0	**2.00**	0	**2.00**	3	**2.00**	0	**2.00**	**0**	**2.00**	0	**2.00**	0	**2.00**	0	**2.00**	0
	PG3 - PS2	1.16	2	1.17	2	1.49	2	1.57	2	1.56	3	1.34	2	1.41	4	**1.09**	**2**	1.11	2
Spain	**Total**	0.49	213	0.48	209	0.50	223	0.40	193	0.47	217	0.49	220	0.53	249	**0.39**	**176**	0.41	184
	PG1	0.53	219	0.52	215	0.52	220	**0.42**	191	0.49	215	0.53	223	0.56	251	0.42	**181**	0.44	189
	PG2	0.74	12	0.68	11	**0.52**	**9**	0.58	9	0.73	13	0.68	12	0.83	20	0.64	10	0.64	10
	PG3	2.00	0	2.00	0	**1.86**	1	1.98	**0**	1.94	1	1.90	1	1.96	1	1.90	1	1.87	1
	PG1 - PS1	0.50	190	0.49	187	0.49	191	**0.39**	162	0.46	197	0.51	196	0.54	238	0.40	154	0.42	**162**
	PG1 - PS2	1.23	5	1.38	10	**0.88**	3	1.51	3	1.44	16	1.29	5	1.33	8	1.08	**3**	1.03	3
	PG1 - PS3	1.37	33	1.37	33	1.55	34	1.65	34	1.43	**32**	**1.35**	34	1.38	33	1.42	33	1.51	34
	PG2 - PS1	0.71	10	0.67	9	**0.50**	**7**	0.56	8	0.68	10	0.66	10	0.72	13	0.61	9	0.62	9
	PG2 - PS2	1.76	0	1.76	**0**	1.69	1	1.91	0	1.95	1	**1.65**	2	1.71	2	1.73	1	1.72	1
	PG2 - PS3	**1.47**	4	1.90	5	1.81	**4**	1.97	4	1.89	5	1.89	5	1.83	6	1.87	4	1.92	4
	PG3 - PS1	**2.00**	**0**	**2.00**	0	**2.00**	2	**2.00**	0	**2.00**	0	**2.00**	0	**2.00**	0	**2.00**	0	**2.00**	0
	PG3 - PS2	1.98	0	1.98	0	**1.86**	1	1.98	**0**	1.94	1	1.90	1	1.96	1	1.90	1	1.88	1
	Avg.	**0.88**	367	0.95	370	0.92	362	0.99	**360**	1.01	423	0.99	443	1.02	528	0.92	383	0.93	379

4.2 Overall Model Performance with Exogenous Variables

Table 4.2 presents the forecasting performance of all implemented models when exogenous variables are incorporated. Each model was trained and evaluated using an identical set of aligned exogenous inputs, where supported. Notably, AutoETS and SOFTS are omitted from this table, as they do not support exogenous feature

integration. As in previous sections, performance is reported using sMAPE and RMSE, averaged over a 12-month forecast horizon, with lower values indicating better accuracy.

In the Europe 30 series, AutoARIMA achieved the lowest sMAPE for the "Total" series (0.18), while LightGBM obtained the lowest RMSE (998). For most subseries within Europe 30, AutoARIMA, LightGBM and TimeXer were among the top-performing models in terms of RMSE, with XGBoost also ranking highly in some cases. DL models such as TFT, TiDE, and N-HiTS demonstrated moderate performance, with a few competitive results in disaggregated series.

In the Germany series, LightGBM again achieved the lowest RMSE across several series including "Total" (154), PG1 (153), and PG1 - PS1 (155). AutoARIMA also showed strong performance in selected cases, such as PG2 and PG1 - PS1, where it achieved the lowest sMAPE and RMSE. Neural models were occasionally competitive in narrow series but typically displayed higher RMSE values overall.

In the Spain series, TimeXer reported the lowest RMSE for the "Total" series (187), followed by AutoARIMA and LightGBM. LightGBM also ranked among the top performers in several disaggregated segments, including PG1 - PS2 and PG2 - PS1. While TFT and TiDE occasionally produced strong forecasts, especially in more stable, low-variance series, their overall performance was less consistent than that of the tree-based models.

Overall, the results suggest that neural forecasting models again underperform compared to classical statistical and tree-based approaches when exogenous inputs are available. Most neural models, including TFT, TiDE, and N-HiTS, achieved higher average error rates across series and regions. An exception is TimeXer, which delivered competitive results in some subseries, particularly in the Spanish series.

The final row in Table 4.2 summarizes average model performance across all datasets and series. LightGBM achieved the lowest average RMSE (308) and the lowest sMAPE (0.93). AutoARIMA and TimeXer followed closely, both with average sMAPE values of 0.93, and RMSEs of 331 and 410, respectively. These findings confirm that LightGBM, TimeXer, and AutoARIMA are the most consistently accurate models when exogenous inputs are available.

Table 4.2: Forecasting performance across selected datasets with exogenous variables (12-step horizon). Metrics are averaged over all rolling predictions per test point. Best results per row are highlighted in bold. All models were provided with the same set of aligned exogenous inputs during training and inference.

	Series	AutoARIMA		AutoETS		LightGBM		XGBoost		TFT		TiDE		N-HiTS		SOFTS		TimeXer	
		sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE	sMAPE	RMSE
Europe 30	**Total**	**0.18**	1073	-	-	0.21	**998**	0.24	1295	0.24	1369	0.27	1458	0.27	1575	-	-	0.25	1376
	PG1	**0.20**	1162	-	-	0.22	**994**	0.23	1222	0.25	1358	0.28	1440	0.28	1563	-	-	0.26	1365
	PG2	0.41	90	-	-	**0.39**	92	0.43	94	0.72	131	0.44	101	0.5	119	-	-	0.40	**87**
	PG3	0.68	5	-	-	**0.65**	5	0.66	**4**	0.69	7	0.72	8	0.70	6	-	-	0.74	6
	PG1 - PS1	**0.20**	**1040**	-	-	0.24	1051	0.23	1202	0.27	1356	0.28	1445	0.29	1539	-	-	0.27	1368
	PG1 - PS2	0.43	76	-	-	0.62	51	1.10	68	0.68	63	0.46	67	0.4	58	-	-	**0.34**	**35**
	PG1 - PS3	0.61	108	-	-	0.56	105	**0.47**	**96**	0.64	106	0.53	105	0.63	117	-	-	0.60	111
	PG2 - PS1	0.38	71	-	-	0.38	78	0.42	78	0.71	116	0.43	85	0.48	96	-	-	**0.38**	**71**
	PG2 - PS2	0.81	10	-	-	0.72	7	**0.67**	**6**	1.04	19	0.85	11	0.85	12	-	-	0.98	12
	PG2 - PS3	1.07	**31**	-	-	1.13	36	1.06	34	1.19	63	1.09	32	1.27	43	-	-	**1.04**	32
	PG3 - PS1	**2.00**	0	-	-	**2.00**	1	**2.00**	0	**2.00**	0	**2.00**	0	**2.00**	0	-	-	**2.00**	**0**
	PG3 - PS2	**0.63**	**4**	-	-	0.69	5	0.66	4	0.69	7	0.73	6	0.70	6	-	-	0.74	6
Germany	**Total**	0.34	178	-	-	**0.26**	**154**	0.40	214	0.36	194	0.4	229	0.44	253	-	-	0.36	195
	PG1	0.33	170	-	-	**0.26**	**153**	0.39	212	0.37	198	0.4	225	0.44	252	-	-	0.36	194
	PG2	**0.38**	**3**	-	-	0.42	4	0.45	4	0.64	7	0.56	7	0.59	7	-	-	0.58	6
	PG3	1.80	2	-	-	1.27	2	1.62	2	1.35	5	1.19	2	1.60	4	-	-	**1.17**	**2**
	PG1 - PS1	0.37	181	-	-	**0.27**	**155**	0.42	208	0.41	209	0.43	227	0.47	249	-	-	0.39	196
	PG1 - PS2	0.81	30	-	-	0.89	16	1.15	18	0.70	14	0.70	17	0.69	20	-	-	**0.62**	**14**
	PG1 - PS3	1.40	13	-	-	1.29	12	**1.27**	**12**	1.28	12	1.42	13	1.55	14	-	-	1.44	13
	PG2 - PS1	**0.46**	**3**	-	-	0.54	3	0.51	4	0.75	7	0.60	5	0.71	5	-	-	0.55	4
	PG2 - PS2	1.70	2	-	-	1.54	**1**	1.74	1	1.65	8	1.49	3	1.75	4	-	-	**1.48**	2
	PG3 - PS1	**2.00**	1	-	-	**2.00**	1	**2.00**	0	**2.00**	0	**2.00**	0	**2.00**	0	-	-	**2.00**	**0**
	PG3 - PS2	1.76	2	-	-	1.32	2	1.62	2	1.35	5	1.20	2	1.60	4	-	-	**1.17**	**2**
Spain	**Total**	0.47	204	-	-	0.47	216	0.58	238	0.51	223	0.49	219	0.52	228	-	-	**0.40**	**187**
	PG1	0.5	210	-	-	0.5	217	0.56	225	0.53	222	0.53	224	0.56	232	-	-	**0.43**	**192**
	PG2	0.64	10	-	-	**0.55**	**9**	0.58	9	0.72	10	0.72	12	0.74	14	-	-	0.64	10
	PG3	1.94	0	-	-	**1.87**	1	1.97	**0**	1.97	1	1.92	1	1.99	0	-	-	1.92	1
	PG1 - PS1	0.48	182	-	-	0.47	187	0.54	199	0.5	192	0.51	199	0.53	206	-	-	**0.41**	**165**
	PG1 - PS2	1.14	4	-	-	**0.93**	**3**	1.41	3	1.36	7	1.08	3	1.43	5	-	-	1.03	3
	PG1 - PS3	1.45	33	-	-	1.51	34	1.60	34	1.41	35	**1.40**	**33**	1.55	35	-	-	1.51	34
	PG2 - PS1	0.66	9	-	-	**0.54**	**7**	0.56	8	0.64	9	0.67	10	0.71	12	-	-	0.60	9
	PG2 - PS2	**1.78**	1	-	-	1.84	1	1.86	**0**	1.83	1	1.78	1	1.84	1	-	-	1.79	0
	PG2 - PS3	1.95	4	-	-	1.97	4	1.96	4	1.88	5	**1.86**	**4**	1.96	8	-	-	1.94	4
	PG3 - PS1	**2.00**	0	-	-	**2.00**	1	**2.00**	0	**2.00**	0	**2.00**	0	**2.00**	0	-	-	**2.00**	**0**
	PG3 - PS2	1.95	0	-	-	1.97	1	1.97	**0**	1.97	1	1.92	0	1.99	0	-	-	**1.92**	1
	Avg.	0.97	331	-	-	**0.93**	**308**	1.01	375	1.01	409	0.95	435	1.03	468	-	-	0.93	410

In Figure 4.1 below, the average RMSE across all datasets for each forecasting model is illustrated, comparing performance under two input configurations: historical data only (baseline) and historical data supplemented with exogenous variables. In general, the inclusion of exogenous inputs leads to a reduction in RMSE for most models, indicating improved forecasting accuracy. Notably, the best-performing model overall, LightGBM, achieved the lowest RMSE (308) when exogenous variables were included. AutoARIMA also exhibited a marked improvement, reducing RMSE from 367 to 331. Conversely, models such as XGBoost and TimeXer experienced a slight increase in RMSE when exogenous features were introduced, suggesting that the impact of additional inputs may vary depending on model architecture and how it leverages external information. Models that do not support exogenous variables, namely AutoETS and SOFTS, are excluded from this comparison.

Another notable observation from the figure is that neural forecasting models generally lag behind traditional statistical and tree-based methods in terms of average RMSE. With the exception of SOFTS and TimeXer, which are close to the performance of the leading models. Most neural architectures show consistently higher errors, even when exogenous inputs are included.

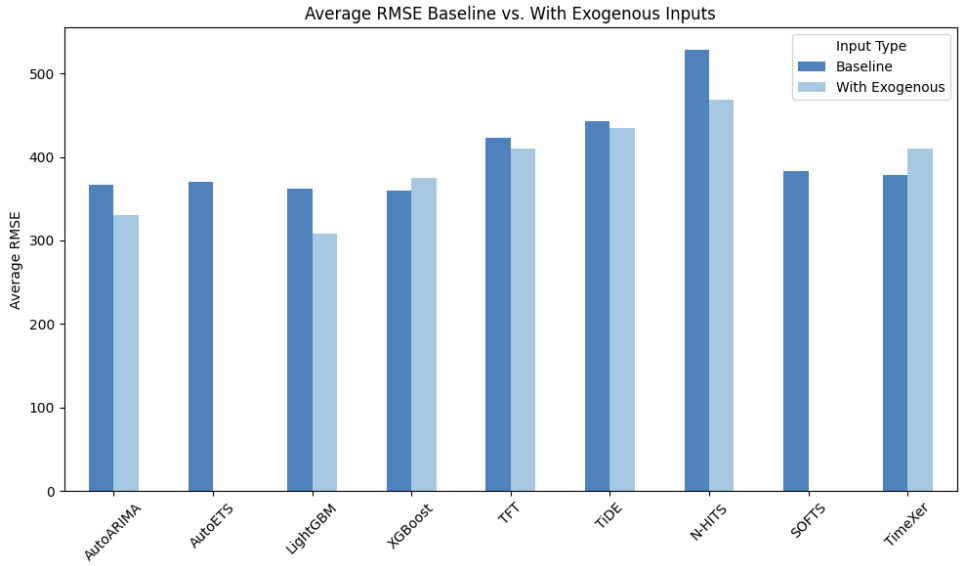

Figure 4.1: Average RMSE for each model with and without exogenous inputs. Models unable to handle exogenous variables (AutoETS and SOFTS) are omitted from the comparison.

4.3 Multi-Horizon Forecast Accuracy

To analyze how model performance evolves over different forecasting horizons, errors are grouped into four intervals: 1-3, 4-6, 7-9, and 10-12 months ahead. Table 4.3 summarizes the average forecasting performance in each interval, evaluated using sMAPE and RMSE. The table includes results for all models with both baseline and exogenous input configurations.

Error metrics tend to increase with longer forecasting horizons across most models, reflecting growing uncertainty further into the future. Without any exogenous variables, classical models AutoARIMA and AutoETS show limited adaptability over time, while machine learning models such as LightGBM and XGBoost tend to maintain lower RMSEs across horizons. Notably, AutoETS show the best performance on the short 1-3 month interval, while performing among the worst models on the longer 10-12 month interval.

Deep learning baseline models display more heterogeneous behavior. TiDE achieves competitive RMSEs at 1-3 months but its accuracy declines sharply thereafter, even though its sMAPE remains roughly constant, suggesting it captures relative patterns but struggles with scale on larger series. SOFTS yields moderate and nearly uniform errors across all horizons, which positions it among the better performers at 10-12 months despite weaker short-term results. N-HITS follows a similar pattern but starts from a higher error baseline, rendering it one of the least accurate models overall. TimeXer match the performance of the statistical and machine learning benchmarks at the 1-3 and 4-6-month intervals, yet its performance also deteriorates on longer horizons. In contrast, TFT stands out by improving with horizon length, ranking lowest in accuracy at 1-3 months but among the top performers at 10-12

months.

Notably, the inclusion of exogenous variables produces mixed effects across models. For instance, models such as AutoARIMA and LightGBM benefit from the additional information, particularly at shorter forecast horizons. LightGBM, in particular, demonstrates strong performance across all horizons, with minimal degradation over later intervals. In contrast, models like TiDE and N-HiTS show limited, or even negative, impact from exogenous inputs, especially in the 10-12 month range. The TFT model also benefits from exogenous inputs at shorter horizons, but its performance deteriorates over time, opposite to the baseline model, which improves with longer horizons. Interestingly, TimeXer with exogenous variables exhibits the reverse trend: it underperforms at short horizons but significantly outperforms the baseline in the 10-12 month interval.

Table 4.3: Forecast accuracy by horizon using sMAPE and RMSE. Values are grouped into four forecast intervals (1-3, 4-6, 7-9, and 10-12 months ahead). Bold values indicate best performance per interval.

Model	**1-3** sMAPE	**4-6** sMAPE	**7-9** sMAPE	**10-12** sMAPE	**1-3** RMSE	**4-6** RMSE	**7-9** RMSE	**10-12** RMSE
AutoARIMA (baseline)	**0.86**	**0.86**	**0.89**	**0.91**	345	361	368	391
AutoARIMA (exogenous)	0.95	0.95	0.97	0.98	305	**312**	348	380
AutoETS (baseline)	0.91	0.94	0.96	0.97	**284**	323	388	461
AutoETS (exogenous)	-	-	-	-	-	-	-	-
LightGBM (baseline)	0.92	0.91	0.92	0.92	342	363	369	375
LightGBM (exogenous)	0.92	0.91	0.93	0.95	309	**312**	**303**	**308**
XGBoost (baseline)	0.95	0.99	1.00	1.01	341	385	343	367
XGBoost (exogenous)	0.96	0.99	1.03	1.06	322	363	370	437
TFT (baseline)	1.04	1.00	1.02	0.99	513	445	346	365
TFT (exogenous)	1.00	1.00	1.03	1.00	317	390	484	429
TiDE (baseline)	0.98	0.99	1.00	0.98	358	504	458	438
TiDE (exogenous)	0.93	0.95	0.96	0.97	371	383	482	489
N-HITS (baseline)	0.99	1.03	1.03	1.05	489	583	517	519
N-HITS (exogenous)	1.00	1.03	1.03	1.06	362	397	442	627
SOFTS (baseline)	0.90	0.91	0.92	0.93	376	370	403	371
SOFTS (exogenous)	-	-	-	-	-	-	-	-
TimeXer (baseline)	0.91	0.92	0.93	0.96	344	371	373	422
TimeXer (exogenous)	0.93	0.93	0.94	0.94	405	423	436	370

A visual overview of these results is provided in Figure 4.2 below, displaying model-specific trends across the four forecasting intervals for both sMAPE and RMSE. The top row illustrates sMAPE values, while the bottom row shows RMSE. Dashed lines represent models trained on historical data only, while solid lines represent models trained with exogenous inputs. This visualization emphasizes how performance shifts over time and highlights which models benefit most from additional input features.

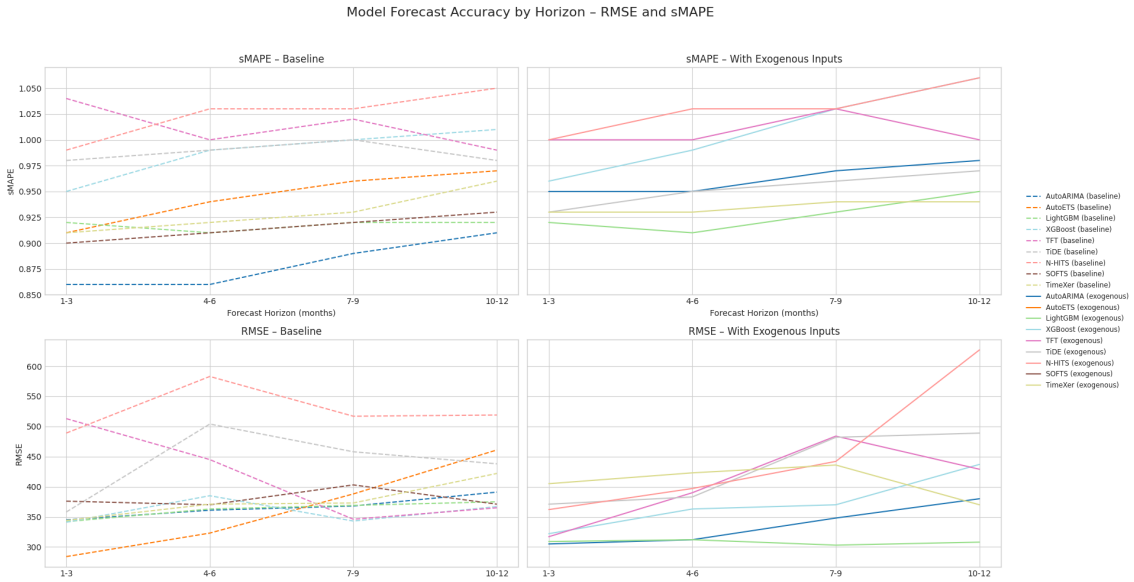

Figure 4.2: Forecast accuracy across four forecast horizons for all models, comparing baseline and with exogenous inputs. Top row reports sMAPE; bottom row reports RMSE. Dashed lines represent baseline models, while solid lines show results when exogenous inputs are included.

4.4 Visualization of Forecasting Results

To complement the tabular results, this section highlights representative forecast plots that illustrate model behavior across different regions and configurations. All visualizations share a common forecast cutoff date of 2023-12-01, with predictions made 12 months ahead. Each figure displays true and predicted volumes across four representative series, annotated with corresponding sMAPE and RMSE values.

While several statistical and machine learning models achieve strong sMAPE and RMSE values, their forecasting plots often reveal a more static behavior. In particular, the AutoETS model tend to produce nearly flat or gently sloped forecast lines. This is especially visible in Figure 4.5, where the predicted values remain largely constant across all four series. By contrast, AutoARIMA exhibits more dynamic behavior in some cases. For example, in Figure 4.3, the forecast for the "Europe 30" series shows a clear alignment with the observed seasonal pattern, capturing fluctuations more convincingly than AutoETS. Still, in lower-variance subseries, AutoARIMAs forecasts also tend toward flatness.

The tree-based models seem to fall into a similar pattern. While they achieve strong average error metrics (especially with exogenous inputs), their plots in Figures 4.6–4.9 frequently resemble smoothed horizontal lines.

In contrast, some neural models produce forecasts that more closely resemble real human-like extrapolations of observed trends. Notably, TFT (baseline) and TiDE (baseline), shown in Figures 4.10 and 4.12, respectively, display a distinct attempt to capture temporal structure. N-HiTS produce a mix of behavior, where the baseline

model sometimes follows patterns well, but in certain series its forecasts become erratic or lag behind. SOFTS and TimeXer, like AutoETS, frequently outputs flat lines, which explains its low and consistent error scores. Although, TimeXer with exogenous variables tend to overestimate predictions.

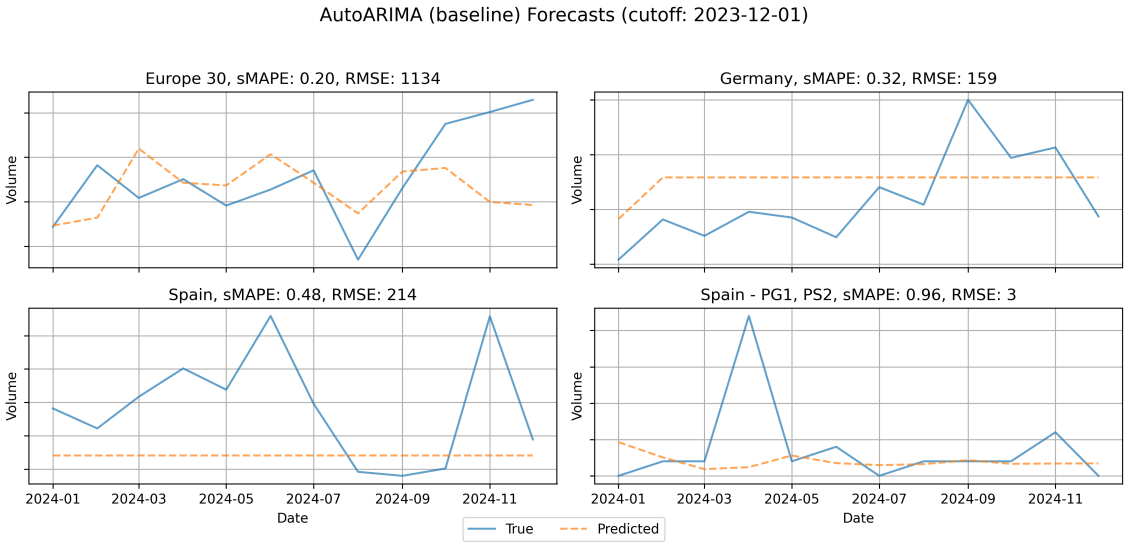

Figure 4.3: Forecast visualization for AutoARIMA using historical data only.

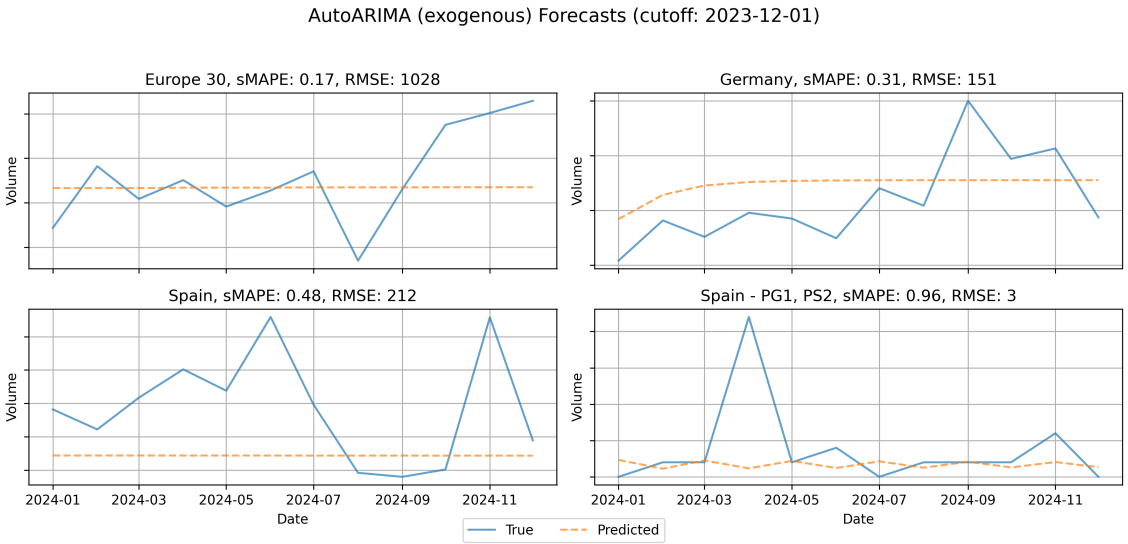

Figure 4.4: Forecast visualization for AutoARIMA with exogenous inputs.

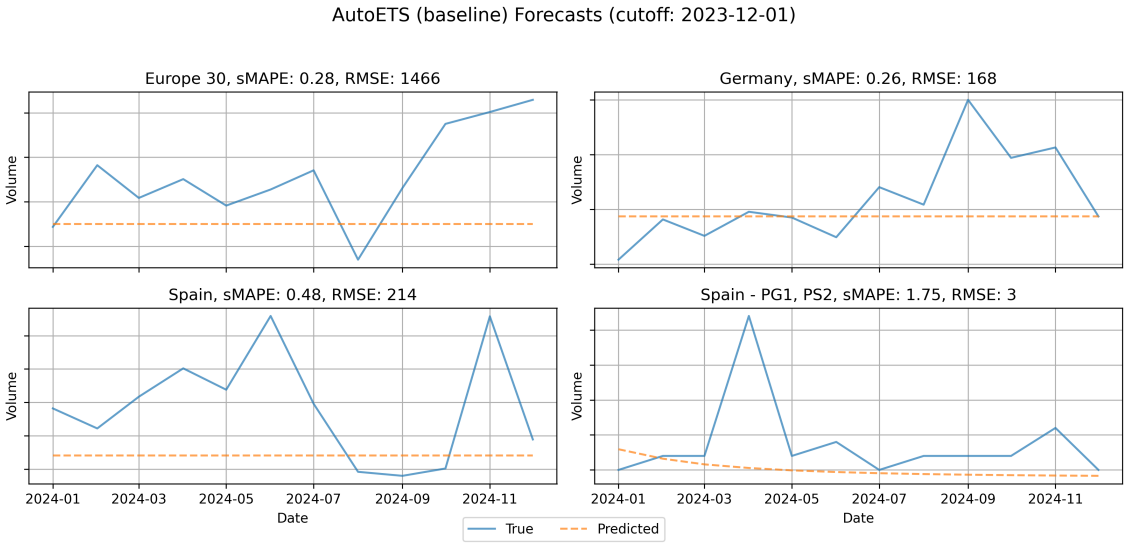

Figure 4.5: Forecast visualization for AutoETS (does not support exogenous inputs).

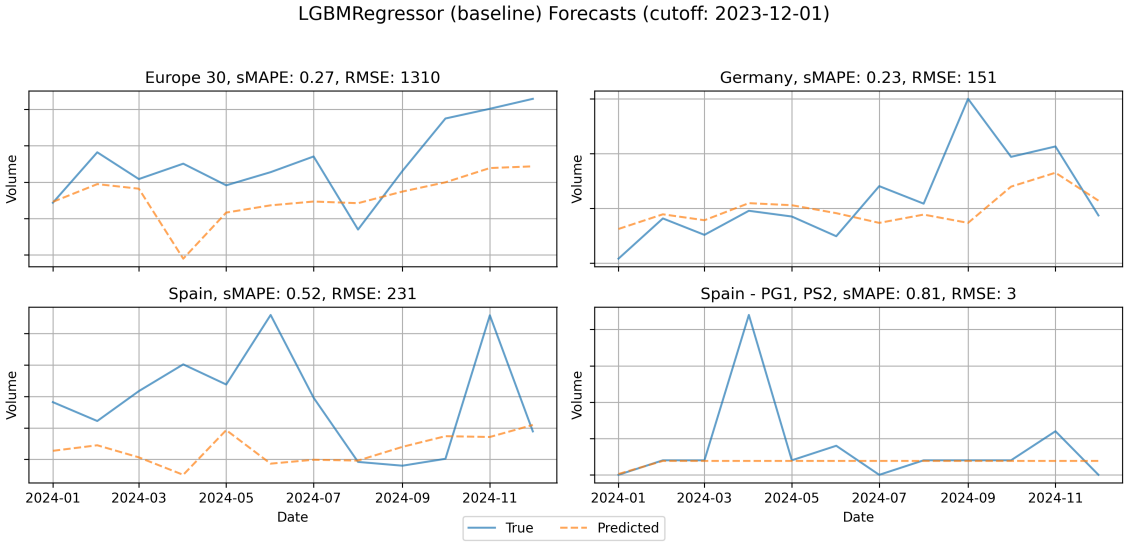

Figure 4.6: Forecast visualization for LightGBM using historical data only.

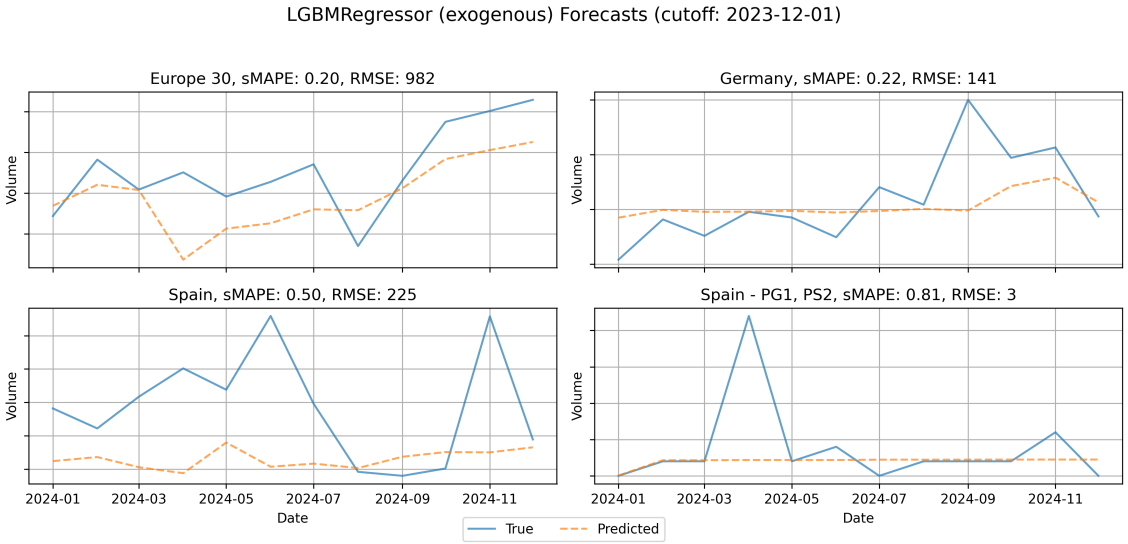

Figure 4.7: Forecast visualization for LightGBM with exogenous inputs.

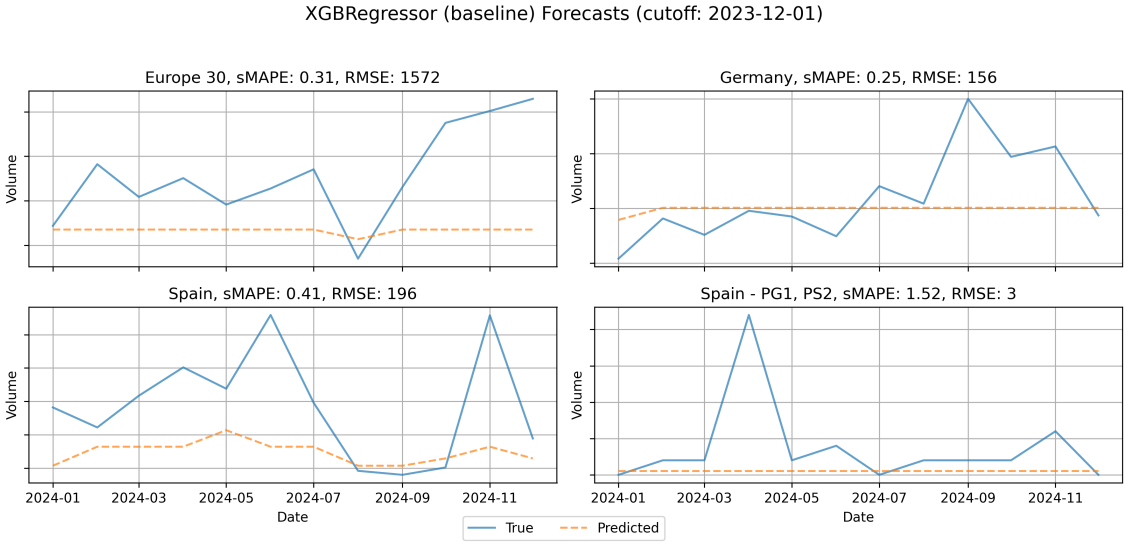

Figure 4.8: Forecast visualization for XGBoost using historical data only.

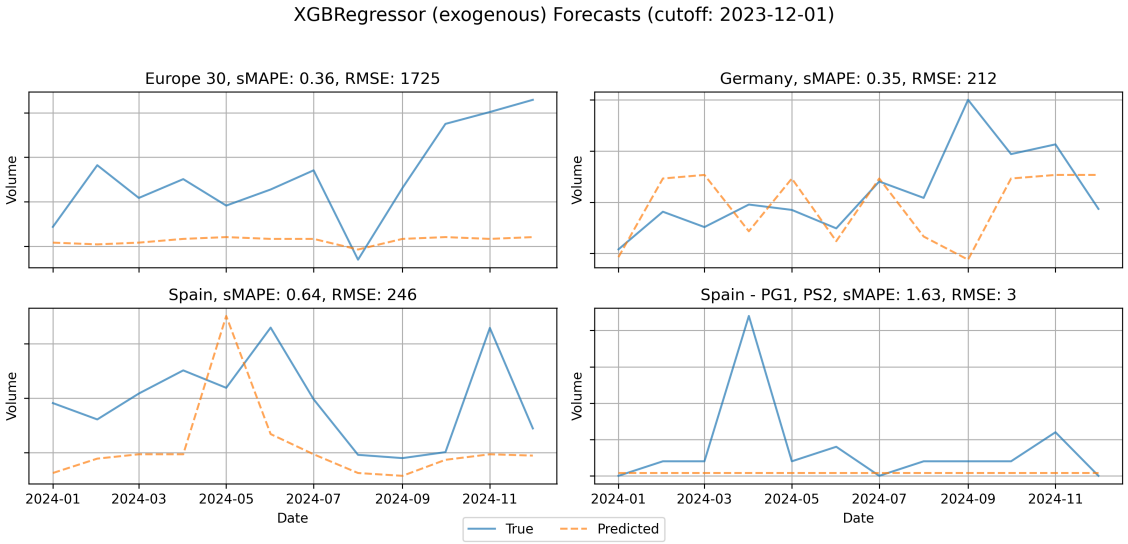

Figure 4.9: Forecast visualization for XGBoost with exogenous inputs.

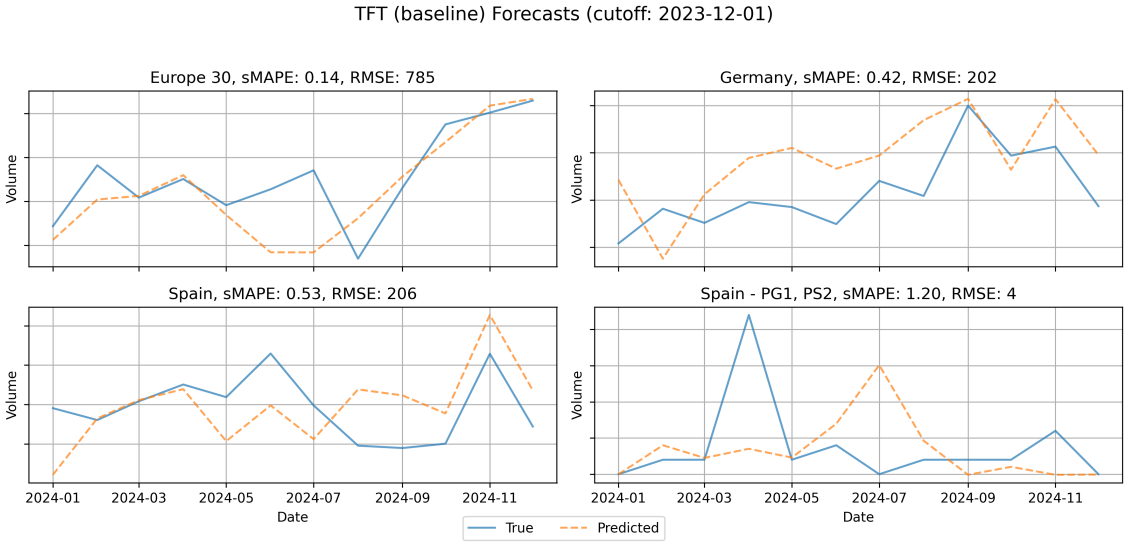

Figure 4.10: Forecast visualization for TFT using historical data only.

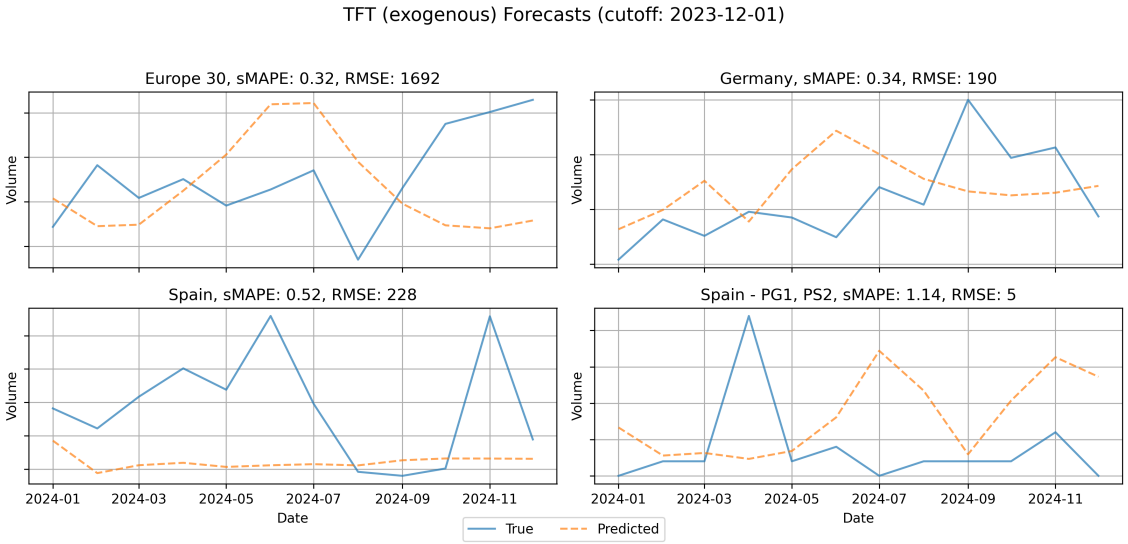

Figure 4.11: Forecast visualization for TFT with exogenous inputs.

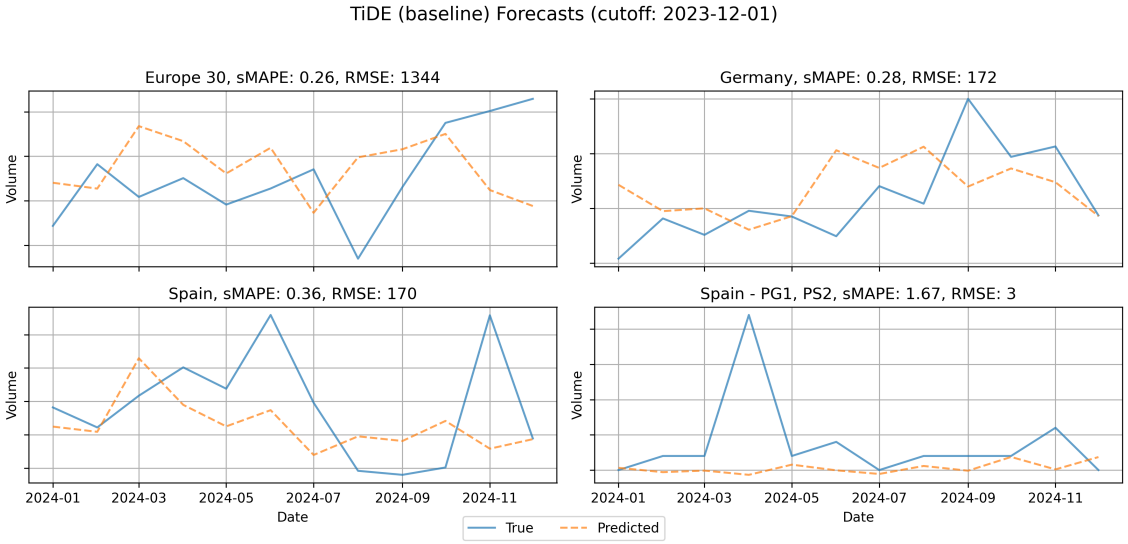

Figure 4.12: Forecast visualization for TiDE using historical data only.

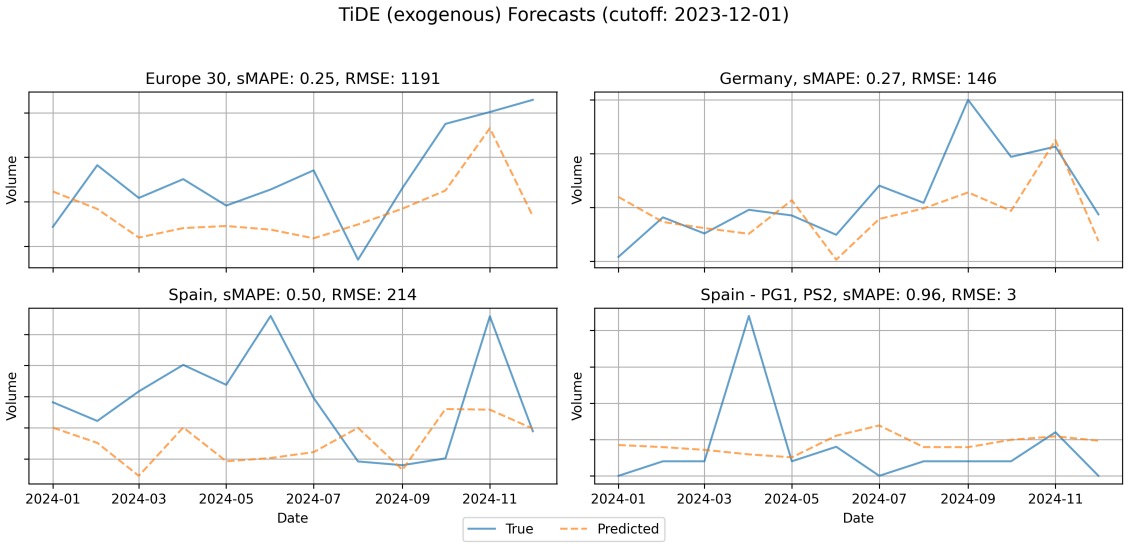

Figure 4.13: Forecast visualization for TiDE with exogenous inputs.

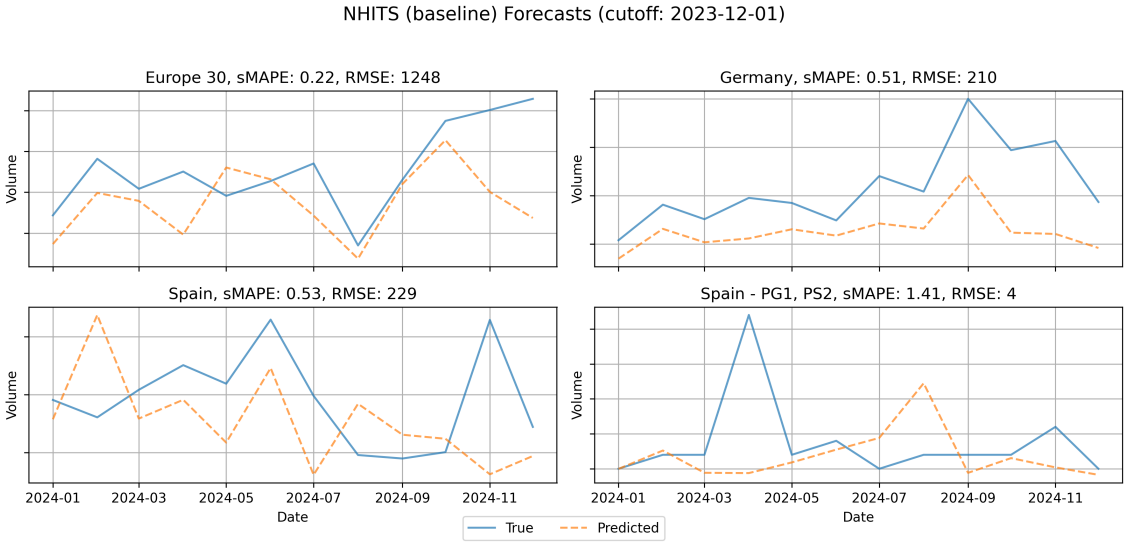

Figure 4.14: Forecast visualization for N-HiTS using historical data only.

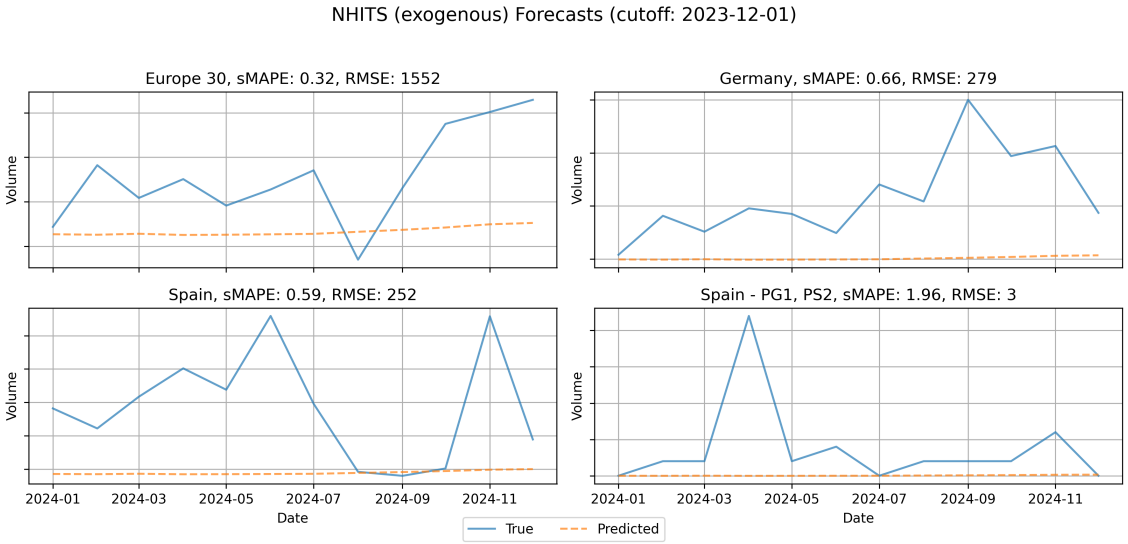

Figure 4.15: Forecast visualization for N-HiTS with exogenous inputs.

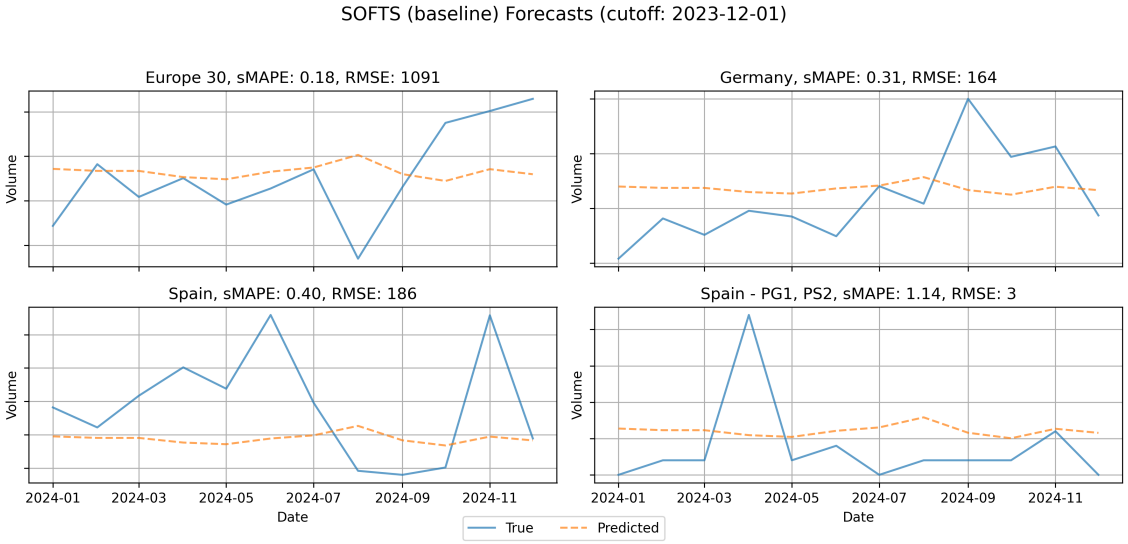

Figure 4.16: Forecast visualization for SOFTS (does not support exogenous inputs).

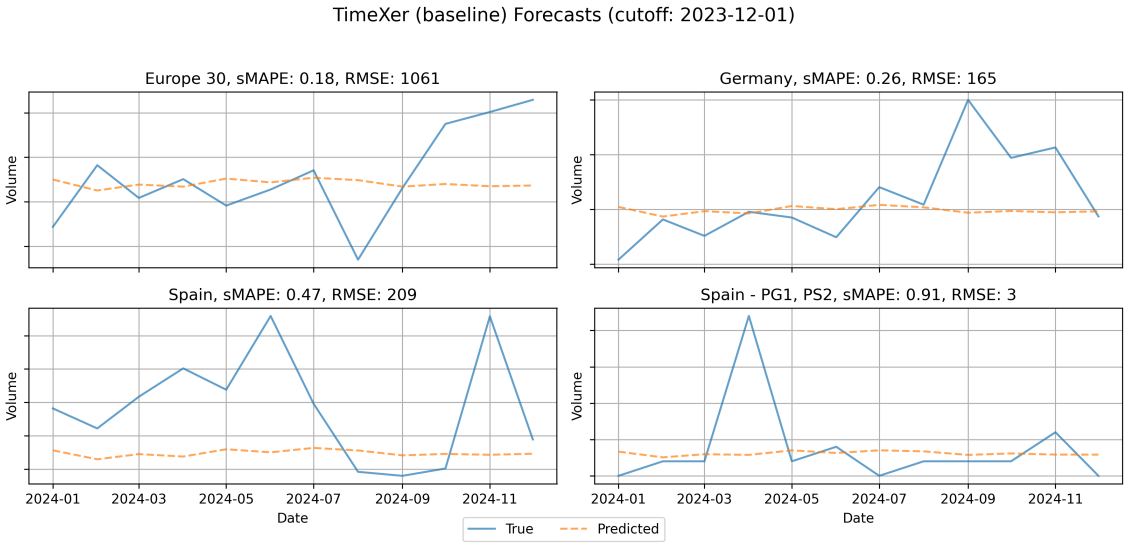

Figure 4.17: Forecast visualization for TimeXer using historical data only.

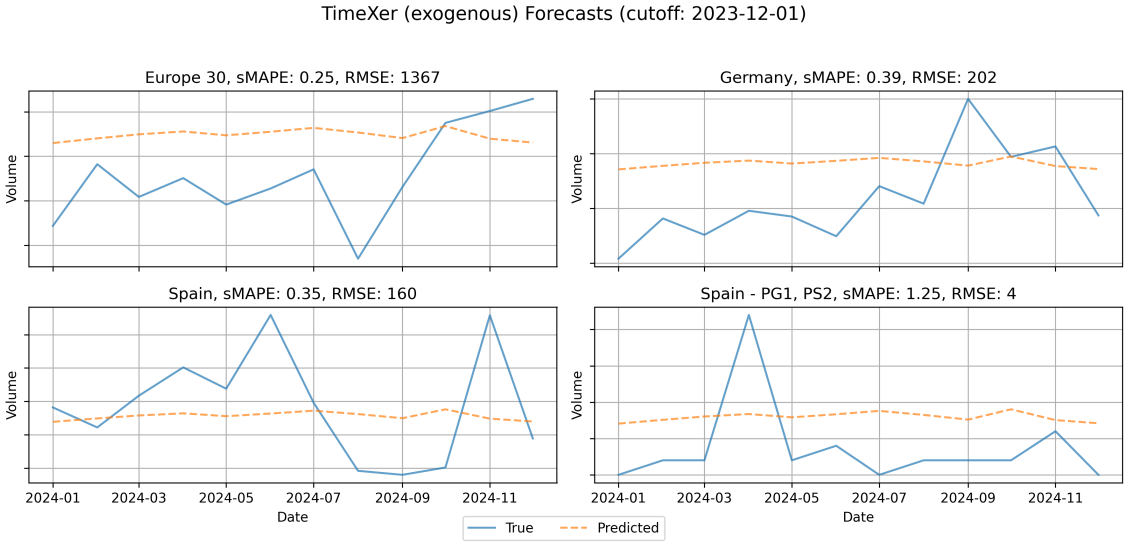

Figure 4.18: Forecast visualization for TimeXer with exogenous inputs.

5

Discussion

This chapter discusses the findings of the thesis in relation to the stated research questions and the broader context of time-series forecasting for industrial demand. The primary aim of this study was to evaluate how recently developed SoTA deep learning models perform compared to traditional statistical and machine learning methods when applied to a real-world industrial dataset. The results revealed a number of important insights, including performance differences across aggregation levels and forecast horizons, the role of exogenous variables, and practical considerations regarding model selection and business applicability. These findings are analyzed in detail in the sections that follow.

5.1 Model Performance in Industrial Setting

Studying the results of the baseline models, it is evident that the classical statistical and ML models generally outperformed the SoTA DL models. At the aggregated Europe 30 level, the traditional models consistently demonstrated superior performance, possibly due to their effectiveness at capturing stable aggregate-level patterns and trends without being overly sensitive to noise or short-term fluctuations. This aligns with prior forecasting literature, where simpler methods are often sufficient or even preferable.

However, at more disaggregated levels, LightGBM and XGBoost performed best, leveraging their ability to handle numerous correlated time series in a global model. Their strength at this granular level likely stems from their flexibility in capturing complex nonlinear relationships while also being efficient in learning with this relatively small dataset. Although, on the average scores across all series, the GBDT models perform only slightly better than the statistical models in terms of RMSE, AutoARIMA is superior in terms of sMAPE. This indicates that the GBDT models are more accurate in terms of total number of vehicles, while the AutoARIMA is better at generalizing the distribution across series.

The performance of the DL models was notably variable across different series, though two specific models, MLP-based SOFTS and Transformer-based TimeXer, stood out as relatively more consistent performers among the DL models. SOFTS, in particular, showed good performance on the seemingly difficult Spain dataset, surpassing the traditional models. Interestingly, SOFTS also performed best among the DL models on the Germany dataset, suggesting that the STAR fusion module

employed by SOFTS might be capturing correlations or common patterns shared between these markets.

A general observation from Table 4.1 indicates that on series with relatively high RMSE values (in the order of hundreds or thousands), the TFT, TiDE, and N-HiTS models noticeably lag behind other approaches. Intriguingly, these three DL models were distinct in their forecasting behaviors, producing more dynamic forecasts that did not resort to the smoothed, nearly straight-line predictions observed in the other models. This behavior, illustrated in Figures 4.3–4.18, suggests that while these models attempted, and to the human eye potentially succeeded, to capture richer temporal dynamics and complex demand fluctuations, they simultaneously became more susceptible to overfitting noisy or irregular historical patterns, ultimately reducing their predictive reliability in these volatile series.

5.2 The Role of Exogenous Variables

As illustrated in Figure 4.1, the inclusion of exogenous variables generally leads to improved model performance, as reflected by lower RMSE values across most models. While this trend is not universal, some models experience negligible or even negative effects, the overall results support the view that incorporating relevant external information can enhance forecast accuracy. LightGBM, in particular, achieved the lowest average RMSE of all models when exogenous variables were included.

The positive impact of exogenous variables is most evident in what would be considered the least complex models in the selection, AutoARIMA and LightGBM. However, for more complex DL models such as TFT, TiDE and TimeXer, the effect is more variable. These architectures may require more sophisticated feature selection mechanisms or larger datasets to extract meaningful patterns from exogenous inputs. In some cases, the addition of weak or noisy signals may even introduce overfitting or reduce generalization performance.

To ensure fair comparison, all models in this study were provided with the same set of exogenous variables, consisting of aligned historical values and corresponding foretasted values for inference. This controlled setup allows for a consistent evaluation of how exogenous inputs affect forecast performance across model types. However, it is important to recognize that models differ significantly in how they process and utilize exogenous information. For instance, TFT includes variable selection mechanisms through GRNs and VSNs, which allow the model to weigh the relevance of each input dynamically. Similarly, TimeXer introduces exogenous variables via a dedicated cross-attention mechanism that interacts with a learnable global token, enabling flexible integration of signals even when they vary in temporal resolution or alignment.

Despite these architectural differences, the models were not explicitly tuned to exploit exogenous inputs in a model-specific way. Moreover, static exogenous variables, such as product group or market region, were excluded from the experiments to maintain consistency, since not all models in the evaluation support static covariates. This design choice further limits the extent to which conclusions can be drawn

about the absolute effectiveness of each model's exogenous handling capability.

What the results do suggest, however, is that the inclusion of well-aligned exogenous variables can improve forecast accuracy. The best-performing models in this study, such as LightGBM and AutoARIMA, achieved their lowest error metrics when exogenous inputs were included. This indicates that exogenous information, when chosen appropriately, can provide significant predictive value. This finding aligns with the theoretical foundation presented in Section 2.1.3, which emphasizes that exogenous variables, particularly future or historically lagged regressors, can enhance model performance by introducing external signals that help anticipate shifts in the target variable.

Therefore, the findings highlight a key takeaway: selecting effective exogenous variables is likely more important than the specific method used to incorporate them. The process of identifying and validating suitable regressors should not be underestimated, it is highly dependent on the data domain, the forecasting horizon, and the model architecture. In practical applications, substantial effort should be allocated to exploring candidate exogenous signals, assessing their predictive value, and aligning them with the target variable in a way that each model can meaningfully leverage.

5.3 Model Performance Across Aggregation Levels and Forecasting Horizons

As previously outlined in Section 5.1, forecasts at higher aggregation levels generally demonstrate improved accuracy, likely due to the smoothing effect on localized fluctuations and inherent noise in the underlying data series. This smoothing effect presumably contributed to the superior performance of statistical models at the Europe-30 aggregation level. However, at the country-level series, identifying a singular best-performing model becomes more challenging. LightGBM generally demonstrates the strongest performance, however, for specific series (such as Spain Total, PG1, and PG1-PS1), SOFTS and TimeXer outperform LightGBM. Similarly, on the Europe 30 PG1-PS3 series, the N-HITS (baseline) model achieves the best performance in terms of sMAPE (comparable RMSE as AutoARIMA), though it lags behind significantly in terms of average scores.

Exploring model performance across varying forecasting horizons revealed clearer distinctions. AutoARIMA yielded the best performance across all horizons when evaluated by sMAPE, demonstrating robust generalization across series. In terms of RMSE, however, AutoETS dominated short-term horizons (1-3 months), while LightGBM with exogenous variables significantly outperformed other models on longer horizons. Notably, the performance of LightGBM (exogenous) remained stable even as the forecasting horizon extended.

One of the motivations behind the selection of models for this study was that DMS-based MLP and Transformer models would outperform classical statistical and ML methods utilizing IMF on long-term forecasting, reflecting the hypotheses of Zeng

et. al. [18]. The comparison of long-term forecasting methodologies did provide additional insights. Statistical and ML models utilizing IMF generally demonstrated superior accuracy compared to DMF approaches, supported by the theoretical expectations of IMF methods. However, the accuracy of IMF models typically degraded over longer horizons, most likely due to accumulating forecast errors from sequential predictions. Interestingly, this deterioration was not universally observed among DL models employing DMF. Only TFT (exogenous), TimeXer (baseline), and N-HITS (exogenouos) have similar performance degradation as the IMF models, while the rest show stable or even slightly improving performance at longer time-horizons. Nonetheless, despite this stability, the baseline errors for DL models were typically higher, suggesting that the relative benefit of DMF may not compensate for their inherently higher error levels.

Of course, it is worth emphasizing that the dataset used only contain about 120 data points per series which may be too small for DL models to accurately learn the underlying patterns, contributing to the general lack of performance. However, while that may be the case, this dataset represents a common real-world business problem, and thus the ability of models to accurately learn on small datasets is an important factor.

The absence of a universally superior model underscores a key finding: model effectiveness varies significantly across aggregation levels and forecasting horizons. Consequently, the empirical results support adopting a hybrid modeling strategy, wherein multiple models are strategically employed according to the specific aggregation level and forecast horizon. Such an approach ensures maximized predictive performance tailored to each forecast scenario.

5.4 Implications for Business

The quantitative analyses presented thus far indicate that classical statistical and ML models generally outperform the SoTA DL models in most scenarios, with only a few notable exceptions. However, as highlighted in Section 5.1 and illustrated in Figures 4.3–4.18, the superior models predominantly generate forecasts characterized by straight-line predictions, effectively predicting near-constant values across the forecast horizon.

While these straight-line forecasts yield the lowest error metrics when averaged over longer forecasting periods, they may provide limited actionable insight for practical business decisions. Specifically, from the perspective of non-technical business stakeholders, trust and interpretability are crucial factors influencing the adoption and effective utilization of predictive models. Straight-line forecasts, although statistically accurate on average, do not convey the timing of anticipated orders clearly, thereby leaving critical operational questions unanswered. Precisely timed predictions are essential for optimizing supply chain logistics, inventory management, and production scheduling, areas where knowing when orders will occur may be as important as knowing how many will occur.

In contrast, predictions from models like the TFT (baseline) (Figure 4.10), despite

their relatively lower overall accuracy, can offer valuable insights by capturing distinct trends and seasonal patterns. These forecasts can help business users visualize the temporal distribution of order intakes, enabling more informed and proactive decision-making. Clear identification of seasonal peaks and troughs would allow Company X to allocate resources efficiently, anticipate demand fluctuations, and improve overall responsiveness to market dynamics.

Moreover, presenting forecasts that clearly reflect business seasonality and market dynamics enhances stakeholder trust, facilitating greater model acceptance and integration into decision-making processes. Straight-line forecasts, while statistically better, might paradoxically undermine confidence by suggesting oversimplification or a lack of dynamic responsiveness to real-world business scenarios. Models illustrating realistic temporal fluctuations could, therefore, be considered more practically valuable despite their modestly higher average error metrics.

This insight raises important considerations regarding the adequacy of standard evaluation metrics and methodologies currently employed. Traditional accuracy metrics (for example, RMSE and sMAPE), as primarily used in this study, may not sufficiently capture the practical utility of forecasts within real-world industrial contexts. Therefore, future evaluations could benefit from incorporating additional qualitative or application-specific performance indicators, emphasizing the practical value of capturing temporal dynamics rather than strictly minimizing error.

Ideally, businesses such as Company X seek a balance between the high accuracy associated with the straight-line statistical models and the detailed temporal insights provided by the more complex DL models. Further research into hybrid forecasting approaches is therefore interesting. Potential strategies could involve blending the reliable average predictions of simpler models with temporal fluctuations derived from DL forecasts. Alternatively, hybrid models designed explicitly to leverage the complementary strengths of different forecasting approaches could provide better solutions. Exploring these avenues may significantly enhance both the accuracy and practical value of forecasting tools available to Company X.

6

Conclusion

This thesis set out to evaluate the forecasting performance of SoTA time series models on a real-world industrial dataset from Company X. The problem addressed involved forecasting the monthly order intake for commercial vehicles across different aggregation levels and forecast horizons.

The results indicate that classical statistical and GBDT models consistently outperformed deep learning models in terms of average error metrics. LightGBM and AutoARIMA in particular showed strong performance, especially when augmented with well-aligned exogenous inputs. Among the DL methods, SOFTS and TimeXer showed relatively stronger and more stable performance, particularly on more granular series. However, the overall predictive accuracy of DL models was likely hampered by the small dataset size, underscoring the data inefficiency of such architectures.

Furthermore, while the inclusion of exogenous variables generally improved model performance, this effect was model-dependent. Tree-based and statistical models benefited most, while DL models showed mixed results, highlighting the importance of both feature relevance and model architecture in leveraging exogenous inputs.

Importantly, while many of the top-performing models generated statistically accurate forecasts, they often lacked meaningful temporal variation, appearing as straight-line or smoothed trends. In contrast, certain neural models, such as TFT and TiDE, although less accurate on average, provided forecasts that better captured temporal structures like trends and seasonal shifts, which may be more actionable for operational decision-making.

In conclusion, the empirical findings suggest that no single model is universally optimal, but indications are that SoTA deep learning models do not outperform traditional statistical or machine learning methods in this specific industrial setting. Instead, model effectiveness varies significantly depending on the forecast horizon, aggregation level, and availability of relevant exogenous information. Therefore, a hybrid modeling strategy that tailors model choice to forecast context may provide the most practical value. Future work could explore model ensembling, incorporate domain-specific feature engineering, and investigate more data-efficient DL architectures or transfer learning approaches to better exploit small industrial datasets.

Bibliography

[1] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, 2020. arXiv: 1912.09363 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1912.09363.

[2] L. Han, X.-Y. Chen, H.-J. Ye, and D.-C. Zhan, *Softs: Efficient multivariate time series forecasting with series-core fusion*, 2024. arXiv: 2404.14197 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2404.14197.

[3] A. Das, W. Kong, A. Leach, S. Mathur, R. Sen, and R. Yu, *Long-term forecasting with tide: Time-series dense encoder*, 2024. arXiv: 2304.08424 [stat.ML]. [Online]. Available: https://arxiv.org/abs/2304.08424.

[4] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, and A. Dubrawski, *N-hits: Neural hierarchical interpolation for time series forecasting*, 2022. arXiv: 2201.12886 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2201.12886.

[5] J. Liu, Z. He, S. Lin, *et al.*, "Timexer: Bridging temporal patches and exogenous variables for multivariate forecasting," *arXiv preprint arXiv:2403.03670*, 2024. [Online]. Available: https://arxiv.org/abs/2403.03670.

[6] R. G. Brown, *Statistical Forecasting for Inventory Control*. McGraw-Hill, 1956.

[7] C. C. Holt, "Forecasting seasonal and trends by exponentially weighted moving averages," Carnegie Institute of Technology, Tech. Rep., 1957, O.N.R. Memorandum No. 52.

[8] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management Science*, vol. 6, no. 3, pp. 324–342, 1960. DOI: 10.1287/mnsc.6.3.324.

[9] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. John Wiley & Sons, 2015, ISBN: 9781118675021.

[10] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. arXiv: 1603.02754. [Online]. Available: http://arxiv.org/abs/1603.02754.

[11] G. Ke, Q. Meng, T. Finley, *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," Dec. 2017.

[12] *M5 forecasting - accuracy: Kaggle leaderboard*, https://www.kaggle.com/competitions/m5-forecasting-accuracy/leaderboard, Accessed: 10 April 2025.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[14] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[15] H. Zhou, S. Zhang, J. Peng, *et al.*, *Informer: Beyond efficient transformer for long sequence time-series forecasting*, 2021. arXiv: 2012.07436 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2012.07436.

[16] H. Wu, J. Xu, J. Wang, and M. Long, *Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting*, 2022. arXiv: 2106.13008 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2106.13008.

[17] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 17–23 Jul 2022, pp. 27 268–27 286. [Online]. Available: https://proceedings.mlr.press/v162/zhou22g.html.

[18] A. Zeng, M. Chen, L. Zhang, and Q. Xu, *Are transformers effective for time series forecasting?* 2022. arXiv: 2205.13504 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2205.13504.

[19] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. Melbourne, Australia: OTexts, 2021, Accessed on 2025-05-18. [Online]. Available: https://otexts.com/fpp3.

[20] P. Montero-Manso and R. J. Hyndman, "Principles and algorithms for forecasting groups of time series: Locality and globality," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1632–1653, 2021, ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2021.03.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207021000558.

[21] M. Marcellino, J. H. Stock, and M. W. Watson, "A comparison of direct and iterated multistep ar methods for forecasting macroeconomic time series," *Journal of Econometrics*, vol. 135, no. 1, pp. 499–526, 2006, ISSN: 0304-4076. DOI: https://doi.org/10.1016/j.jeconom.2005.07.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030440760500165X.

[22] G. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," *European Journal of Operational Research*, vol. 160, no. 2, pp. 501–514, 2005, Decision Support Systems in the Internet Age, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2003.08.037. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221703005484.

[23] Y. R. Sagaert, N. Kourentzes, S. De Vuyst, E.-H. Aghezzaf, and B. Desmet, "Incorporating macroeconomic leading indicators in tactical capacity planning," *International Journal of Production Economics*, vol. 209, pp. 12–19, 2019, The Proceedings of the 19th International Symposium on Inventories, ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2018.06.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925527318302603.

[24] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, "A state space framework for automatic forecasting using exponential smoothing methods," *International Journal of Forecasting*, vol. 18, no. 3, pp. 439–454, 2002. DOI: 10.1016/S0169-2070(01)00110-8.

[25] Y. R. Sagaert, N. Kourentzes, S. De Vuyst, E.-H. Aghezzaf, and B. Desmet, "Incorporating macroeconomic leading indicators in tactical capacity planning," *International Journal of Production Economics*, vol. 209, pp. 12–19, 2019, The Proceedings of the 19th International Symposium on Inventories, ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2018.06.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925527318302603.

[26] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting* (Springer Texts in Statistics), 2nd ed. Springer, 2002, ISBN: 9780387953519. DOI: 10.1007/b97391.

[27] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: The forecast package for r," *Journal of Statistical Software*, vol. 27, no. 3, pp. 1–22, 2008. DOI: 10.18637/jss.v027.i03.

[28] J. H. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting," *Annals of Statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[29] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002. DOI: 10.1016/S0167-9473(01)00065-2.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ISBN: 9780262035613. [Online]. Available: https://www.deeplearningbook.org/.

[32] M. Pérez-Enciso and L. M. Zingaretti, "A guide on deep learning for complex trait genomic prediction," *Genes*, vol. 10, no. 7, 2019, ISSN: 2073-4425. DOI: 10.3390/genes10070553. [Online]. Available: https://www.mdpi.com/2073-4425/10/7/553.

[33] Creative Commons, *Creative commons attribution 4.0 international public license*, 2013. [Online]. Available: https://creativecommons.org/licenses/by/4.0/.

[34] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. DOI: 10.1207/s15516709cog1402_1.

[35] L. Hu, J. Zhang, Y. Xiang, and W. Wang, "Neural networks-based aerodynamic data modeling: A comprehensive review," *IEEE Access*, vol. 8, pp. 90 805–90 823, 2020. DOI: 10.1109/ACCESS.2020.2993562.

[36] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-beats: Neural basis expansion analysis for time series forecasting," *arXiv preprint arXiv:1905.10437*, 2019. [Online]. Available: https://arxiv.org/abs/1905.10437.

[37] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013. DOI:

10.48550/arXiv.1301.3557. [Online]. Available: https://arxiv.org/abs/1301.3557.

[38] X.-Y. Chen, *Softs: Efficient multivariate time series forecasting with series-core fusion*, MIT License, 2024. [Online]. Available: https://github.com/Secilia-Cxy/SOFTS.

[39] *The mit license*, https://opensource.org/licenses/MIT, 1988.

[40] F. Garza, M. M. Canseco, C. Challú, and K. G. Olivares, *StatsForecast: Lightning fast forecasting with statistical and econometric models*, PyCon Salt Lake City, Utah, US 2022, 2022. [Online]. Available: https://github.com/Nixtla/statsforecast.

[41] Nixtla, *MLForecast: Scalable machine learning for time series forecasting*, https://nixtlaverse.nixtla.io/mlforecast/, Accessed: 2025-05-15, 2024.

[42] K. G. Olivares, C. Challú, A. Garza, M. M. Canseco, and A. Dubrawski, *NeuralForecast: User friendly state-of-the-art neural forecasting models.* PyCon Salt Lake City, Utah, US 2022, 2022. [Online]. Available: https://github.com/Nixtla/neuralforecast.

[43] P. J. Huber, "Robust estimation of a location parameter," *Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, Mar. 1964. DOI: 10.1214/aoms/1177703732. [Online]. Available: https://doi.org/10.1214/aoms/1177703732.

[44] J. T. Barron, "A general and adaptive robust loss function," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2019, pp. 4331–4339. DOI: 10.1109/CVPR.2019.00446.

[45] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. null, pp. 281–305, Feb. 2012, ISSN: 1532-4435.

[46] J. Snoek, H. Larochelle, and R. P. Adams, *Practical bayesian optimization of machine learning algorithms*, 2012. arXiv: 1206.2944 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1206.2944.

[47] R. Turner, D. Eriksson, M. McCourt, *et al.*, *Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020*, 2021. arXiv: 2104.10201 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2104.10201.

[48] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of Bayesian methods for seeking the extremum," *Towards Global Optimization*, vol. 2, no. 117-129, p. 2, 1978.

[49] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24, Curran Associates, Inc., 2011.

[50] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, Jun. 2013, pp. 115–123. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html.

[51] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2011, pp. 2546–2554. [Online]. Available: `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`.

A

Model Search Spaces and Configurations

Table A.1: Considered hyperparameters for XGBoost and optimal values for baseline and exogenous models.

Parameter	**Search Space**	**Optimal (Baseline)**	**Optimal (Exogenous)**
Learning rate	$\eta \in \log\mathcal{U}(10^{-3}, 10^{-1})$	0.058	0.034
Maximum tree depth	$d_{\max} \in \{3, 5, 7, 9, 12, 15\}$	5	5
Number of estimators	$N \in \{50, 75, \ldots, 800\}$	100	150
Row subsample ratio	$\lambda_{\text{row}} \in \mathcal{U}(0.5, 1.0)$	0.965	0.867
Feature subsample ratio	$\lambda_{\text{feat}} \in \mathcal{U}(0.5, 1.0)$	0.669	0.616
Min. child weight	$w_{\min} \in \{1, 2, \ldots, 10\}$	2	2
Minimum split loss	$\gamma \in \log\mathcal{U}(10^{-8}, 10)$	4.179	5.709
L1 regularization	$\alpha_{\ell_1} \in \log\mathcal{U}(10^{-8}, 10)$	$2.45{\cdot}10^{-3}$	$1.51{\cdot}10^{-3}$
L2 regularization	$\alpha_{\ell_2} \in \log\mathcal{U}(10^{-8}, 10)$	3.50	$4.80{\cdot}10^{-7}$
Tree method	tree_method $\in$ {`auto`, `gpu_hist`}	`gpu_hist`	`auto`
Lag configuration	$\mathcal{L} \in \{12, 24, 36, 48, 60, 72\}$	[1, 60]	[1, 60]

Table A.2: Considered hyperparameters for LightGBM and optimal values for baseline and exogenous models.

Parameter	**Search Space**	**Optimal (Baseline)**	**Optimal (Exogenous)**
Learning rate	$\eta \in \log\mathcal{U}(10^{-3}, 10^{-1})$	0.0504	0.0247
Number of leaves	$n_{\text{leaves}} \in \{20, \ldots, 150\}$	128	110
Maximum tree depth	$d_{\max} \in \{3, 5, 7, 9, 12, 15, \infty\}$	5	3
Number of estimators	$N \in \{100, 200, \ldots, 800\}$	325	700
Min. data per leaf	$n_{\min} \in \{5, 10, \ldots, 100\}$	75	40
Row subsample ratio	$\lambda_{\text{row}} \in \mathcal{U}(0.5, 1.0)$	0.716	0.561
Feature subsample ratio	$\lambda_{\text{feat}} \in \mathcal{U}(0.5, 1.0)$	0.503	0.734
L1 regularization	$\alpha_{\ell_1} \in \log\mathcal{U}(10^{-8}, 10)$	$1.28{\cdot}10^{-8}$	$1.04{\cdot}10^{-5}$
L2 regularization	$\alpha_{\ell_2} \in \log\mathcal{U}(10^{-8}, 10)$	$6.10{\cdot}10^{-3}$	$7.16{\cdot}10^{-6}$
Boosting type	boosting_type $\in$ {gbdt, dart, goss}	gbdt	goss
Huber delta	$\delta \in \log\mathcal{U}(0.1, 10)$	0.477	0.471
Lag configuration	$\mathcal{L} \in \{12, 24, 36, 48, 60, 72\}$	[1, 48]	[1, 48]

Table A.3: Considered hyperparameters for TFT and optimal values for baseline and exogenous models.

Parameter	Search Space	Optimal (Baseline)	Optimal (Exogenous)
Input window size	$L \in \{12, 18, \ldots, 72\}$	66	36
Hidden size	$d_{\text{hidden}} \in \{16, 32, \ldots, 256\}$	128	32
Attention heads	$H \in \{1, 2, 4, 8\}$	2	1
Max training steps	$\text{steps}_{\max} \in \{300, \ldots, 1500\}$	1200	1000
Learning rate	$\eta \in \log \mathcal{U}(10^{-4}, 10^{-1})$	$4.22 \cdot 10^{-4}$	$2.47 \cdot 10^{-3}$
Attention dropout	$\rho_{\text{attn}} \in \mathcal{U}(0.05, 0.3)$	0.266	0.121
RNN layers	$n_{\text{rnn}} \in \{1, 2\}$	1	1
Batch size	$B \in \{8, 16, 24, 32\}$	32	32
Huber delta	$\delta \in \log \mathcal{U}(0.1, 10)$	2.267	0.212

Table A.4: Considered hyperparameters for TimeXer and optimal values for baseline and exogenous models.

Parameter	Search Space	Optimal (Baseline)	Optimal (Exogenous)
Input window size	$L \in \{12, 18, \ldots, 72\}$	24	18
Hidden size	$d_{\text{hidden}} \in \{16, 32, \ldots, 1024\}$	256	32
Patch length	$P \in \{2, 4, 6, 8, 12, 24\}$	2	6
Attention heads	$H \in \{1, 2, 4, 8\}$	1	2
Encoder layers	$n_{\text{enc}} \in \{1, 2, 3, 4\}$	2	4
Feedforward dim	$d_{\text{ff}} \in \{16, 32, 64, 128\}$	128	32
Max training steps	$\text{steps}_{\max} \in \{300, \ldots, 1500\}$	1100	1000
Learning rate	$\eta \in \log \mathcal{U}(10^{-4}, 10^{-1})$	0.0248	0.0156
Dropout rate	$\rho \in \mathcal{U}(0.05, 0.3)$	0.267	0.138
Use normalization	`use_norm` $\in$ {`True, False`}	True	True
Batch size	$B \in \{8, 16, 24, 32\}$	32	32
Huber delta	$\delta \in \log \mathcal{U}(0.1, 10)$	0.326	2.208

Table A.5: Considered hyperparameters for TiDE and optimal values for baseline and exogenous models.

Parameter	Search Space	Optimal (Baseline)	Optimal (Exogenous)
Input window size	$L \in \{12, 18, \ldots, 72\}$	24	54
Hidden size	$d_{\text{hidden}} \in \{16, 32, 64, \ldots, 1024\}$	128	32
Encoder layers	$n_{\text{enc}} \in \{1, 2, 3\}$	1	3
Decoder layers	$n_{\text{dec}} \in \{1, 2, 3\}$	3	2
Decoder output dim	$d_{\text{out}} \in \{4, 8, 16, 32\}$	8	4
Temporal decoder dim	$d_{\text{temp}} \in \{32, 64, 128\}$	128	128
Max training steps	$\text{steps}_{\max} \in \{300, 400, \ldots, 1500\}$	1300	500
Learning rate	$\eta \in \log \mathcal{U}(10^{-5}, 10^{-1})$	$5.96 \cdot 10^{-5}$	$6.45 \cdot 10^{-5}$
Dropout rate	$\rho \in \mathcal{U}(0.05, 0.3)$	0.107	0.171
Layer normalization	`layernorm` $\in$ {`True, False`}	False	True
Batch size	$B \in \{8, 16, 24, 32\}$	32	32
Huber delta	$\delta \in \log \mathcal{U}(0.1, 10)$	0.475	0.472

Table A.6: Considered and fixed hyperparameters for N-HiTS and optimal values for baseline and exogenous models.

Parameter	Search Space	Optimal (Baseline)	Optimal (Exogenous)
Input window size	$L \in \{12, 18, \ldots, 72\}$	42	54
MLP units per stack	`mlp_units` $\in \{[64, 64], [128, 128], [256, 256], [512, 512]\}$	[512, 512]	[64, 64]
Pooling kernel sizes	$\mathbf{k} \in \{[2, 2, 2], [4, 4, 4], [8, 8, 8], [8, 4, 1], [16, 8, 1]\}$	[8,4,1]	[8,4,1]
Frequency downsampling	$\mathbf{f} \in \{[168, 24, 1], [24, 12, 1], [180, 60, 1], [40, 20, 1], [64, 8, 1]\}$	[64,8,1]	[168,24,1]
Number of blocks per stack	$\mathbf{n}_{\text{blocks}} \in \{1, 2, 3\}$	[1,1,1]	[1,1,1]
Dropout (basis layers)	$\rho \in \mathcal{U}(0.05, 0.3)$	0.093	0.051
Max training steps	$\text{steps}_{\text{max}} \in \{300, \ldots, 1500\}$	800	700
Learning rate	$\eta \in \log\mathcal{U}(10^{-5}, 10^{-1})$	0.00346	0.0448
Batch size	$B \in \{8, 16, 24, 32\}$	32	32
Huber delta	$\delta \in \log\mathcal{U}(0.1, 10)$	0.818	9.289
Stack types (fixed)	`stack_type` = identity	identity	identity
Number of stacks (fixed)	$S = 3$	3	3

Table A.7: Considered hyperparameters for SOFTS and optimal values for the baseline model. No exogenous variant was evaluated.

Parameter	Search Space	Optimal (Baseline)	Optimal (Exogenous)
Input window size	$L \in \{12, 18, \ldots, 72\}$	18	N/A
Hidden size	$d_{\text{hidden}} \in \{16, 32, \ldots, 1024\}$	64	N/A
Core dimension (STAD)	$d_{\text{core}} \in \{16, 32, \ldots, 1024\}$	64	N/A
Encoder layers	$n_{\text{enc}} \in \{1, 2, 3, 4\}$	3	N/A
Fully connected dim	$d_{\text{ff}} \in \{16, 32, 64, 128\}$	32	N/A
Max training steps	$\text{steps}_{\text{max}} \in \{300, \ldots, 1500\}$	1200	N/A
Learning rate	$\eta \in \log\mathcal{U}(10^{-4}, 10^{-1})$	0.0389	N/A
Dropout rate	$\rho \in \mathcal{U}(0.05, 0.3)$	0.0786	N/A
Use normalization	`use_norm` $\in \{$`True, False`$\}$	True	N/A
Batch size	$B \in \{8, 16, 24, 32\}$	16	N/A
Huber delta	$\delta \in \log\mathcal{U}(0.1, 10)$	0.872	N/A