

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCHOOL OF SCIENCE

Master's Degree Programme in Mathematics

Improving Heat-Load Forecasting with Deep Learning

Master's Thesis in Applied Mathematics

Supervisor:

Prof. Giovanni Paolini

Presented by:

Giovanni Lombardi

Company co-supervisors:

Simone Graziani

Daniele Zago

Federico Vitali

Academic Year 2024/2025

Contents

1	Introduction	1
1.1	Background and Motivation	2
1.2	The State of the Art in Time Series Forecasting	3
1.3	Scope and Objectives	5
1.4	Experimental Setup	6
1.5	Exploratory Data Analysis	7
1.6	Evaluation Metrics	11
2	Statistical Pipelines	17
2.1	MSTL-ETS Pipeline	17
2.1.1	Pipeline Overview	18
2.1.2	Seasonal-Trend decomposition using Loess	18
2.1.3	Data Pre-Processing	20
2.1.4	Two-Stage Seasonal Decomposition	22
2.1.5	Model Estimation and Component Forecasting	23
2.1.6	Hyperparameter Tuning	24
2.2	SARIMAX Pipeline	27
2.2.1	Pipeline Overview and Rationale	27
2.2.2	Exogenous Feature Engineering	29
2.2.3	Model Estimation	33
2.2.4	Hyperparameter Tuning	34
3	LSTM Pipeline	39
3.1	The LSTM Network	39
3.1.1	Motivation and Limitations of Standard RNNs	39
3.1.2	LSTM Architecture and Update Equations	40

3.2	The Seq2Seq LSTM Architecture	42
3.2.1	Architecture Overview	42
3.2.2	Design Rationale and Alternatives	42
3.2.3	Configurable Autoregressive Modes in the Decoder	44
3.3	Configuration and Hyperparameter Space	45
3.3.1	Model Architecture Hyperparameters	45
3.3.2	Data Loading and Preprocessing Hyperparameters	46
3.3.3	Training Configuration Hyperparameters	46
3.3.4	Feature Engineering Hyperparameters	49
3.4	Bayesian Optimisation with TPE	51
3.4.1	Bayesian Optimisation	51
3.4.2	Tree-Structured Parzen Estimator	54
3.4.3	Hyperparameter Optimisation Procedure	63
3.5	Multi-step Tuning	65
3.5.1	Preliminary Studies	68
3.5.2	Final Hyperparameter Tuning	78
4	Other Deep Learning Pipelines	83
4.1	TFT Pipeline	83
4.1.1	Model Description and Rationale	83
4.1.2	Hyperparameter Tuning	88
4.2	Chronos-2 Pipeline	90
4.2.1	Rationale for Including this Benchmark	90
4.2.2	Chronos-2 and its In-Context Learning Mechanism	91
4.2.3	Zero-shot Configuration	92
5	Comparative Assessment of Forecasting Models	95
5.1	Company models	96
5.2	Forecast Generation for Testing	97
5.3	Notes on Comparability and Threats to Validity	99
5.4	Series F1: Day-Ahead Forecast Evaluation	101
5.4.1	Computational Cost	101
5.4.2	Per-Window Errors	102
5.4.3	Bootstrap Assessment of Performance Improvement	107
5.4.4	Hour-Ahead Horizon Errors	110

5.4.5	Residual Analysis	111
5.5	Series F1: Week Ahead Forecast Evaluation	112
5.6	Series F2	113
5.6.1	Day-Ahead Forecasts	114
5.6.2	Week-Ahead Forecasts	115
5.7	Series F3 to F5	116
5.7.1	Day-Ahead Forecasts	116
5.7.2	Week-Ahead Forecasts	118
5.7.3	Interpretation of Results on Noisy Series	118
6	Conclusions	139
A	Training and Inference Times	147
B	Chronos-2 Pretraining Data Audit	151
C	Illustrative Winter Forecasts	155
D	Failure of the iid Bootstrap under Temporal Dependence	161
	Bibliography	167

Chapter 1

Introduction

This thesis investigates short-term thermal-load forecasting in the context of an internship at Optit¹, a Bologna-based company that develops decision-support systems leveraging optimisation, forecasting and data analytics across diverse sectors. The central objective is to develop and evaluate a range of forecasting approaches, spanning classical statistical methods and modern deep-learning architectures, as potential complements or replacements to the XGBoost and Support Vector Regression models currently deployed within Optit’s OptiEPTM platform, against which the proposed models are benchmarked. The evaluation is conducted on five synthetic heat-demand series across two operational horizons (day-ahead and week-ahead), with each model optimised independently per series and per horizon. Performance is assessed using a rolling cross-validation framework with separate seasonal analysis and bootstrap-based significance testing.

The remainder of this introductory chapter is structured as follows. It begins by situating the forecasting problem within the broader energy-production context and motivating the need for accurate demand predictions (Section 1.1), followed by an overview of the state of the art in time-series forecasting (Section 1.2). It then describes the objectives and scope of the study (Section 1.3) and outlines the experimental setup (Section 1.4). The chapter concludes with an exploratory analysis of the heat-demand dataset used in this work (Section 1.5) and an introduction to the evaluation metrics adopted throughout the thesis (Section 1.6).

¹<https://optit.net/>

1.1 Background and Motivation

This section situates the forecasting problem addressed in this thesis within the wider context of modern energy-production systems and outlines the motivations for improving demand-forecasting accuracy.

The management of modern energy production systems is becoming increasingly complex from both technical and organisational perspectives. Energy-service providers must pursue ambitious economic objectives, such as maximising operating margins and improving the performance of their asset portfolios, while making decisions that depend on a wide range of interconnected variables, including economic factors, technical constraints, market conditions and forecasting information.

Forecasts of energy demand, in particular, are among the most critical inputs to the planning process because they determine how effectively system operators, utilities and policymakers can balance supply and consumption across different time horizons [19]. Accurate short-term predictions (that is, up to a few days ahead) support real-time decisions such as dispatching generation units and maintaining grid stability, while medium-term forecasts (typically up to several months ahead) guide, for example, maintenance scheduling and market participation. Over longer horizons, reliable demand projections inform capital-intensive decisions, including investments in new generation assets, transmission capacity and energy-storage infrastructure, ensuring that system expansion remains aligned with future consumption patterns.

This need for accurate demand forecasting has become even more pronounced within the European Commission's broader strategy to decarbonise the energy sector. The revised Renewable Energy Directive (EU/2023/2413) commits the EU to a binding renewable-energy share of at least 42.5% by 2030². As highlighted in recent literature, the increasing penetration of weather-dependent resources of energy makes accurate forecasting indispensable because their variability affects system stability and operational efficiency [19]. Effective grid management relies on both generation and demand forecasts across different timescales. In this context, improving demand forecasts remains essential for supporting a reliable and efficient energy system as the EU advances its decarbonisation goals.

²https://energy.ec.europa.eu/topics/renewable-energy/renewable-energy-directive-targets-and-rules/renewable-energy-targets_en

In parallel, recent advances in digitalisation have accelerated the adoption of modelling and optimisation tools within energy-production systems. OptiEPTM³, a proprietary platform developed by Optit, operates in this context, offering end-to-end decision support for planning and trading activities. It constructs a *Digital Twin* of an energy system, whether a single plant or an entire portfolio, that encodes physical constraints (for example, capacities and ramp rates), operational characteristics (for example, efficiency curves and minimum up and down times), and economic structures (for example, fuel and maintenance costs, tariffs and market prices). By simulating system behaviour under alternative scenarios, the platform enables users to explore operational strategies, evaluate trade-offs and identify optimal plans. Central to this workflow is the forecasting module, which provides predictions of key variables such as thermal load. The quality of these forecasts directly affects the reliability and optimality of the plans produced by subsequent optimisation stages.

This thesis is motivated by the need to improve the forecasting component that supports OptiEPTM's operational planning workflow. The platform's current forecasting module is based primarily on classical machine-learning models such as XGBoost and Support-Vector Regression. These methods offer excellent computational efficiency and are well suited to real-time operational settings, but they may struggle to fully capture the non-linear dynamics, temporal dependencies, and long-range interactions that characterise heat-demand behaviour in complex building systems. These limitations motivate the exploration of complementary modelling approaches capable of enhancing predictive accuracy and robustness across diverse operating conditions.

1.2 The State of the Art in Time Series Forecasting

Deep neural networks, most notably large Transformer-based architectures, have driven major breakthroughs in areas such as natural language processing and computer vision, but their impact on time-series forecasting (TSF) has been far less consistent. Recent survey work on deep learning for TSF highlights a movement toward architectural diversification rather than convergence on a single dominant model family. A particularly notable trend in the literature is that simple architectures can often match or surpass the performance of more complex designs [42, 43].

³<https://optit.net/soluzioni/>

This has contributed to what the authors describe as a “renaissance” of architectures, in which both new and traditional modelling strategies are actively being revisited.

Empirical evidence therefore indicates that, depending on the dataset, horizon and data regime, complex deep learning architectures do not consistently outperform simpler alternatives. As a result, multiple modelling paradigms remain relevant and continue to be explored in parallel, such as [42]:

- *Classical and hybrid models*, including statistical approaches (e.g., ARIMA, ETS, state-space models) that capture linear dynamics and seasonality, as well as hybrid schemes that combine these foundations with machine-learning or deep-learning components to model non-linear effects, incorporate exogenous variables or enhance robustness in complex forecasting settings.
- *Fundamental deep-learning architectures*, such as MLPs, CNNs, RNNs, GNNs, and modern mixer-style or residual architectures (e.g., N-BEATS, N-HiTS).
- *Attention-based and Transformer models*, which use attention mechanisms to focus dynamically on the most relevant parts of the historical data and to capture long-range temporal dependencies (e.g. TFT, PatchTST).
- *Emerging non-Transformer models*, such as state-space architectures (e.g., MambaTS, C-Mamba), diffusion-based forecasters (e.g., Diffusion-TS), and other recent sequence-modelling approaches (e.g., frequency-domain MLPs), which model temporal dynamics through structured state equations or iterative generative processes that offer efficient long-range modelling and strong inductive biases for time series data.
- *Foundation models for time series*, which seek to leverage large-scale pre-training and cross-domain transfer to build general-purpose forecasting and representation-learning models (e.g., TimesFM, Chronos), trained on massive multi-domain corpora to enable zero-shot or few-shot forecasting across heterogeneous time-series tasks.

Given this landscape, selecting an appropriate set of models for evaluation requires considering both established baselines and recent advances in deep-learning architectures, a balance reflected in the models examined in this study.

1.3 Scope and Objectives

Building on the modelling trends outlined above and the forecasting needs discussed previously, this study focuses on developing and evaluating a set of alternative approaches for thermal-load prediction that may complement or improve on the existing methods of the company. To this end, we investigate a deliberately diverse set of modelling paradigms⁴, reflecting both classical statistical methods and more recent advances in deep learning. Specifically, we extend the modelling landscape in two main directions:

- *Statistical benchmarks*, represented here by MSTL-ETS and SARIMAX (Ch. 2). In particular, SARIMAX combines a linear regression component for exogenous effects with a SARIMA error process that captures autocorrelation structure and daily seasonality. MSTL-ETS, in contrast, is based on multi-seasonal decomposition followed by an ETS model. Although it does not exploit exogenous variables, we include it as a lightweight and interpretable baseline that is more informative than a naïve benchmark.
- *Modern deep-learning architectures*, including Long Short-Term Memory networks (LSTMs), the Temporal Fusion Transformer (TFT), and the recent foundation-style model Chronos-2 (Ch. 3–4). LSTMs are a fundamental class of sequence models; TFT augments sequence modelling with attention and variable-selection mechanisms; Chronos-2 is a pre-trained forecasting model that can be applied in a zero-shot manner, without task-specific training.

The goal is twofold:

1. to evaluate these models on the same forecasting tasks and under the same operational constraints as the existing platform, in order to assess whether they yield improved predictive accuracy and greater robustness to atypical or rapidly changing operating conditions;
2. to investigate whether such an improvement could be achieved while containing the computational overhead within the feasible operational limits of the forecasting platform.

⁴All code used in this thesis is available at <https://github.com/GioLombardiDev/Internship>

1.4 Experimental Setup

This section describes the dataset, forecasting tasks, and modelling setup considered in the study.

Synthetic heat-load dataset. This study examines heat-load forecasting for five sites, labelled F1 to F5. The datasets used are synthetic: the historical time series were derived from project data that were reworked to imitate realistic and representative behaviour, rather than originating from actual facilities. The series reflect different types of scenarios, including residential areas, industrial areas, and public buildings.

For each site, the dataset contains hourly heat-demand values from 19 July 2019 to 20 May 2025, together with exogenous meteorological variables (temperature, dew point, wind speed, air pressure, and humidity) generated to resemble typical measurements collected at such locations. The final year of data, from 20 May 2024 to 20 May 2025, is used as a held-out test period (see Chapter 5), while all earlier observations are used for training and validation.

Before modelling, a diagnostic analysis using Dynamic PCA (DPCA) [35] was performed, experimenting with several lag lengths to inspect the first two principal components for potential outliers or structural anomalies. No issues were detected.

A more detailed characterisation of the temporal patterns, seasonal structure, and relationships between the target and the exogenous variables is provided in the next section.

Perfect-foresight assumption. As is common in benchmarking studies of heat-load and energy-demand forecasting models, we assume that future meteorological inputs are available without error and therefore substitute the realised weather values in the synthetic dataset for the corresponding forecasts. This choice focuses the comparison on the forecasting models themselves, rather than conflating their errors with uncertainty in the weather forecasts. However, it yields optimistic performance estimates, particularly at the week-ahead horizon: while the assumption is often a reasonable approximation for day-ahead forecasting, forecast accuracy for temperature typically degrades with lead time, so real-world week-ahead accuracy may be lower.

Forecasting horizons. The analysis considers two forecasting horizons: day-ahead (24 hours) and week-ahead (168 hours). These horizons are commonly used in operational settings and correspond to typical short-term planning windows in

heat-load management. Day-ahead forecasts primarily rely on capturing short-term dynamics, whereas week-ahead forecasts require maintaining predictive accuracy over a longer horizon, across which uncertainty accumulates in both the demand process and the weather forecasts (although the latter is not reflected in our measured errors, since we substitute realised weather observations for forecasts under the perfect-foresight assumption).

Modelling strategy. We adopt a per-facility approach: for each model, optimisation is performed independently for each site. Forecast horizons are largely treated separately, but the week-ahead horizon is prioritised to address the more challenging task, which influences the day-ahead horizon and introduces partial coupling between horizons. Consequently, each model yields 10 fitted versions (5 facilities $\times$ 2 horizons), with facilities fully independent and horizons only partially so.

1.5 Exploratory Data Analysis

An exploratory analysis was carried out on the full dataset, in which we examined the main patterns in the target series and the correlations with exogenous variables, with the aim of identifying the features most suitable as primary predictors.

Trend and seasonality. Time series are often described in terms of systematic components such as *trend* (or *trend-cycle*) and *seasonal* components [e.g. 31, 11]. The trend component captures the long-term evolution of the series, reflecting persistent increases or decreases in its level over time, as well as medium-term fluctuations that do not occur at a fixed frequency. By contrast, the seasonal component captures patterns that repeat with a fixed, known period s , such as daily, weekly, or annual effects.

These components are commonly represented through *additive* or *multiplicative* decompositions, depending on whether seasonal fluctuations remain constant in magnitude or vary with the level of the series. In an additive decomposition, the series $\{y_t\}_{t \in \mathbb{Z}}$ is written as

$$y_t = T_t + S_t + R_t, \quad (1.1)$$

where T_t is the trend component, S_t is the seasonal component ($S_t \approx S_{t-s}$), and R_t

captures short-term irregular fluctuations. In a multiplicative decomposition,

$$y_t = T_t \cdot S_t \cdot R_t, \quad (1.2)$$

the seasonal and irregular components scale proportionally with the level of the series. More generally, a variance-stabilising transformation (e.g., Box–Cox; see Section 2.1.3) can interpolate between these two formulations, allowing intermediate behaviours to be represented.

In many practical applications, including the present study, a single time series may exhibit multiple seasonal patterns simultaneously (e.g., daily and weekly effects). These can be represented as separate components $S_t^{(1)}, S_t^{(2)}, \dots$, leading to a multi-seasonal decomposition. In the additive case:

$$y_t = T_t + \sum_{i=1}^m S_t^{(i)} + R_t.$$

This approach (implemented, for example, via the MSTL method described in Section 2.1.4) separates each seasonal pattern according to its own frequency, avoiding the conflation of higher-frequency effects into lower-frequency components. This improves interpretability and allows each component to be analysed and forecast independently.

In the following, we examine these components in the target series, highlighting recurring seasonal patterns across different timescales.

Visualisation of the target series. Figure 1.1 presents a four-week segment of the target series during a peak-demand period. This visualisation highlights the temporal dynamics observed during high-demand phases, revealing clear daily seasonality. Series F1 and F2 appear relatively regular and predictable, whereas F3 to F5 exhibit more irregular patterns. In addition, F3 and F5 display pronounced weekly seasonality. The amplitude of these seasonal fluctuations also tends to increase with the overall level of heat demand, suggesting that a multiplicative formulation (1.2), or alternatively a variance-stabilising transformation prior to additive modelling, is more appropriate in this setting.

Figure 1.2 presents weekly heat-demand profiles aggregated by season over one year of data, based on the following date ranges:

Autumn	20 September to 20 December
Winter	21 December to 20 March
Spring	21 March to 21 June

Shaded regions denote the 10th to 90th percentile bands. As expected, winter shows the highest demand. Autumn exhibits the greatest intra-seasonal variability: demand typically begins at low levels and rises steadily until late November. By contrast, spring begins after demand has already passed its peak and entered a declining phase, resulting in narrower variability bands and lower mean values than in autumn.

Variance behaviour. To improve the suitability of the series for statistical forecasting models, we apply a variance-stabilising transformation to the target variable before modelling, as described in later sections. Because many such transformations can excessively amplify values near zero, we impose a lower clipping threshold of 5 kWh. Fewer than 5% of observations fall below this level, even in summer. This threshold improves numerical stability and limits distortion at very low values while preserving the overall structure of the series.

Relationships with exogenous variables. We examined the pairwise relationships between the target variable and each exogenous feature using daily-aggregated data, obtained by averaging the hourly observations for each day. This reduces high-frequency noise and yields more interpretable patterns. For every pair of variables, Figure 1.3 presents:

- a scatterplot of their joint distribution based on daily averages;
- an overlaid linear regression line and a LOWESS curve [17] to highlight linear and potential non-linear trends;
- the corresponding Spearman (ρ_s) and Pearson (ρ_p) correlation coefficients.

Density plots for each variable appear along the diagonal.

Temperature shows the strongest association with the target ($\rho_s = -0.96$, $\rho_p = -0.92$), followed by dew point ($\rho_s = -0.92$, $\rho_p = -0.89$). Since the two variables are also highly correlated with each other ($\rho_s = 0.95$, $\rho_p = 0.95$), they provide largely overlapping information, making temperature the more natural choice as a primary predictor. The scatterplots of temperature, and similarly of dew point, against the target also reveal a clear saturation at higher temperatures: once temperature exceeds

a certain threshold, heat demand settles at its minimum level, corresponding to the summer period, when heating use is minimal.

Humidity displays a moderate positive correlation with the target, while air pressure and wind speed show weak or negligible associations, suggesting limited standalone predictive value. These observations, however, remain exploratory: pair-wise correlations do not capture multivariate interactions, temporal dependencies, or more complex non-linear effects.

When moving to daily aggregation, the strong relationships among temperature, dew point, and the target are still present, but they weaken substantially. This reduction is expected: at the daily scale, heat-load dynamics are more strongly shaped by behavioural factors, which introduce additional variability. Comparable patterns appear across all facilities, although the magnitude varies.

Table 1.1 summarises the correlations between temperature and heat demand for each facility, computed on both hourly data and daily means. The correlations are calculated using only the months from November to March across all years in the dataset, so as to exclude the largely flat-demand periods of the warmer seasons.

The particularly low hourly correlation for F2 is likely due to its pronounced night-time demand drops, even during the coldest periods (see Figure 1.2). At the daily scale, F3 and F5 exhibit the weakest associations with temperature. This is partly explained by their strong weekly seasonal patterns, where human activity influences average daily consumption more heavily.

Table 1.1: Spearman and Pearson correlations between heat demand and temperature for each site, based on hourly observations and daily averages. Correlations are computed using data from November to March to exclude periods of flat summer demand.

Corr.	Hourly					Daily				
	F1	F2	F3	F4	F5	F1	F2	F3	F4	F5
ρ_s	-0.53	-0.20	-0.56	-0.59	-0.34	-0.91	-0.90	-0.81	-0.87	-0.64
ρ_p	-0.56	-0.21	-0.55	-0.60	-0.37	-0.91	-0.89	-0.80	-0.87	-0.65

1.6 Evaluation Metrics

Model selection in this thesis primarily relies on unnormalised error metrics such as the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). For a forecast horizon of length h , these are defined as

$$\text{MAE} = \frac{1}{h} \sum_{t=T+1}^{T+h} |y_t - \hat{y}_t|, \quad \text{RMSE} = \sqrt{\frac{1}{h} \sum_{t=T+1}^{T+h} (y_t - \hat{y}_t)^2},$$

where y_t and $\hat{y}_t$ denote the observed and predicted heat demand at time t , and T is the *cut-off* time (that is, the last timestamp for which observations are available when the forecast is issued). Both metrics naturally place more weight on winter errors because they operate on the absolute scale of the series. During the coldest months, heat demand is higher and more variable, so absolute deviations are larger and contribute more to MAE and RMSE, with RMSE further amplifying occasional large errors. This property aligns with the operational priorities of the problem, where accurate forecasts during peak load conditions are far more critical than precision during the nearly flat summer months. MAE is straightforward to interpret, while RMSE provides an alternative that penalises larger discrepancies more heavily.

We also monitor the symmetric Mean Absolute Percentage Error (sMAPE) [33], a normalised metric defined as

$$\text{sMAPE} = \frac{100}{h} \sum_{t=T+1}^{T+h} \frac{|y_t - \hat{y}_t|}{|y_t + \hat{y}_t|/2}.$$

Its main advantage is that it expresses forecast errors relative to the magnitude of the series. However, sMAPE tends to over-penalise deviations during warm months: when demand is low, even small absolute fluctuations produce large relative errors. Since these periods are operationally unimportant and characterised by low signal-to-noise ratios, sMAPE can give an inflated impression of model underperformance despite negligible practical impact.

Normalising the error by the average of the target and the prediction rather than the target itself (in absolute value) partially mitigates the extreme penalties that occur when the true value unexpectedly approaches zero and the model over-forecasts. However, as discussed in detail by Goodwin and Lawton [26], this adjustment intro-

duces a different form of asymmetry: large over-forecasts are penalised less heavily than under-forecasts of the same absolute magnitude, which makes the metric difficult to interpret when large errors in both directions occur.

An alternative that is arguably easier to interpret (especially in winter) is the normalised MAE (NMAE), defined here as

$$\text{NMAE} = \frac{\sum_{t=T+1}^{T+h} |y_t - \hat{y}_t|/h}{\sum_{t \in \mathcal{T}} |y_t|/|\mathcal{T}|},$$

Here, $\mathcal{T}$ denotes a given reference period. In words, the NMAE scales the MAE over the forecast horizon by the average absolute level of the series during the reference period $\mathcal{T}$, yielding a dimensionless measure of forecast error. Given the strong annual seasonality of heat demand, we choose $\mathcal{T}$ as the set of timestamps from the same season as the forecast window. This yields an intuitive notion of relative error when demand is in a relatively stable regime (most notably in winter). However, in shoulder seasons the mean level can drift substantially over short periods, so a season-based normalisation may be less stable and therefore harder to interpret.

Finally, we considered using the Mean Absolute Scaled Error (MASE) [33], a scale-free metric widely used for cross-series comparison. MASE normalises forecast errors by the in-sample MAE of a seasonal naïve model. For a horizon h , and seasonal period s , it is given by

$$\text{MASE} = \frac{\sum_{t=T+1}^{T+h} |y_t - \hat{y}_t|/h}{\sum_{t=s+1}^T |y_t - y_{t-s}|/(T-s)},$$

where the denominator is the average in-sample absolute error of the seasonal naïve baseline with period s , which forecasts y_t by the lagged value y_{t-s} (in our case, the natural choice is $s = 24$ for hourly data with strong daily seasonality).

In our setting, however, the denominator becomes very small because the seasonal naïve model incurs almost no error during the extremely stable summer months, when demand is nearly flat. As a result, reasonable winter forecasting errors can appear disproportionately large when scaled by this quantity, making MASE values difficult to interpret. The same effect also distorts comparisons across sites: series with a longer flat warm period naturally yield a smaller scaling term and therefore mechanically higher MASE values, even when the underlying forecasting

performance is similar.

Additionally, the scaling term is sensitive to the dominant periodic structure of each series. Even if the denominator were computed using naïve errors restricted to winter, series with pronounced weekly seasonality would typically produce a larger seasonal naïve error (and hence a larger denominator) than series with weak or no weekly seasonality. This would mechanically generate smaller MASE values for series with strong weekly seasonality, reflecting differences in the baseline error rather than differences in forecasting quality. For these reasons, MASE was not adopted in this work.

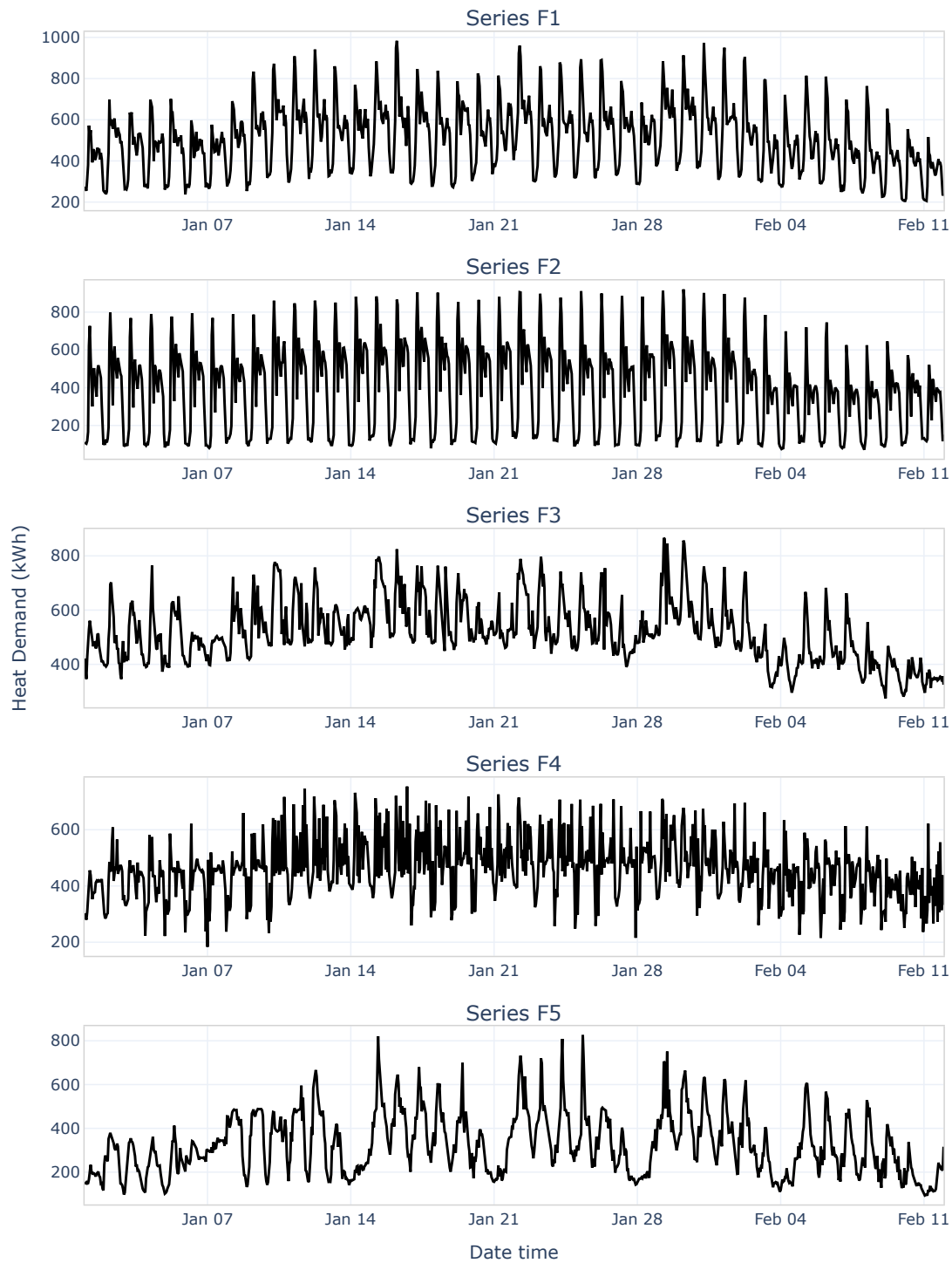


Figure 1.1: Target series for each site over a four-week peak-demand period.

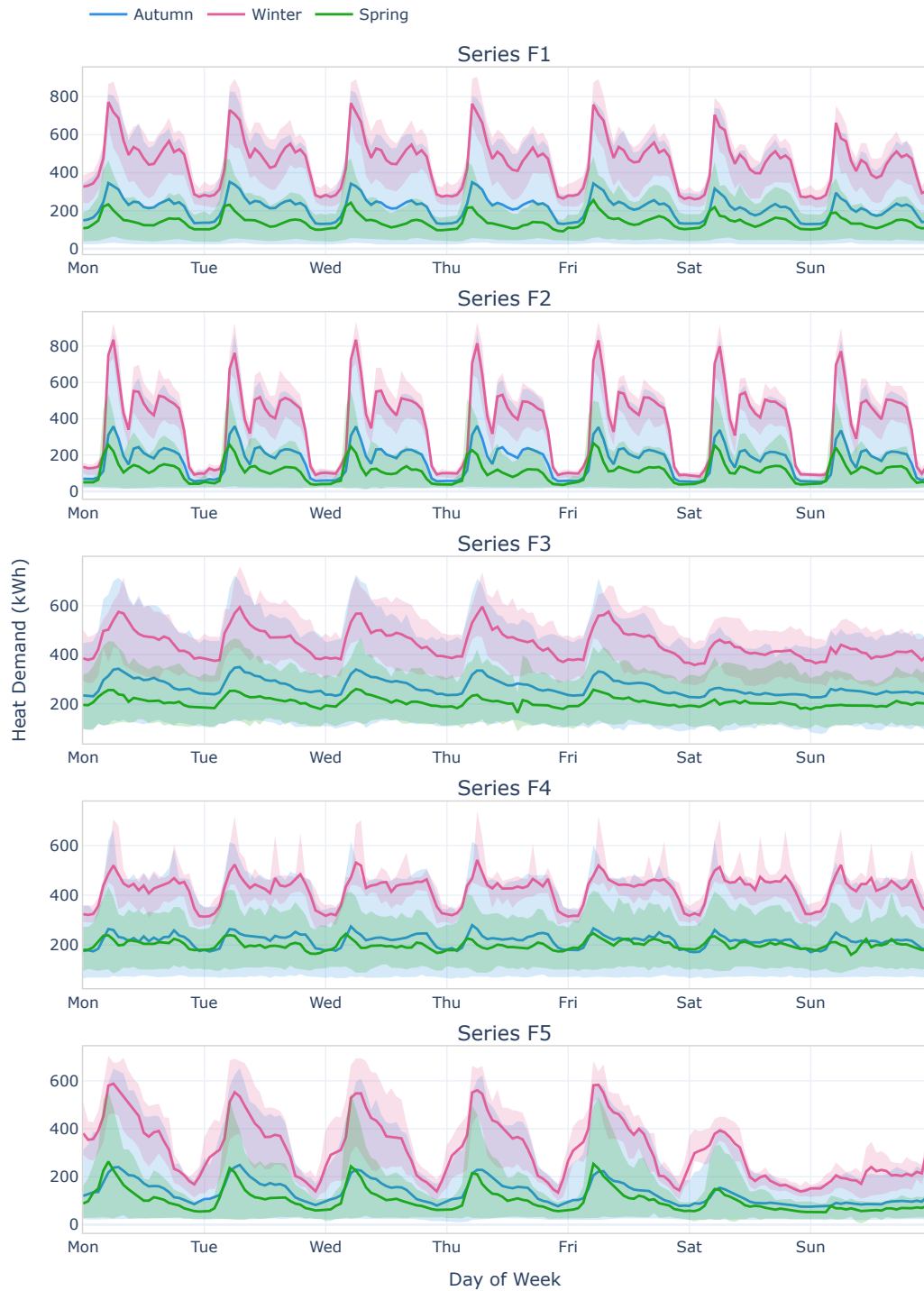


Figure 1.2: Weekly heat-demand profiles for each facility, aggregated by season (autumn, winter, spring). Solid lines represent the mean heat load for each hour of the week, while shaded ribbons denote the 80% variability range (10th–90th percentiles).

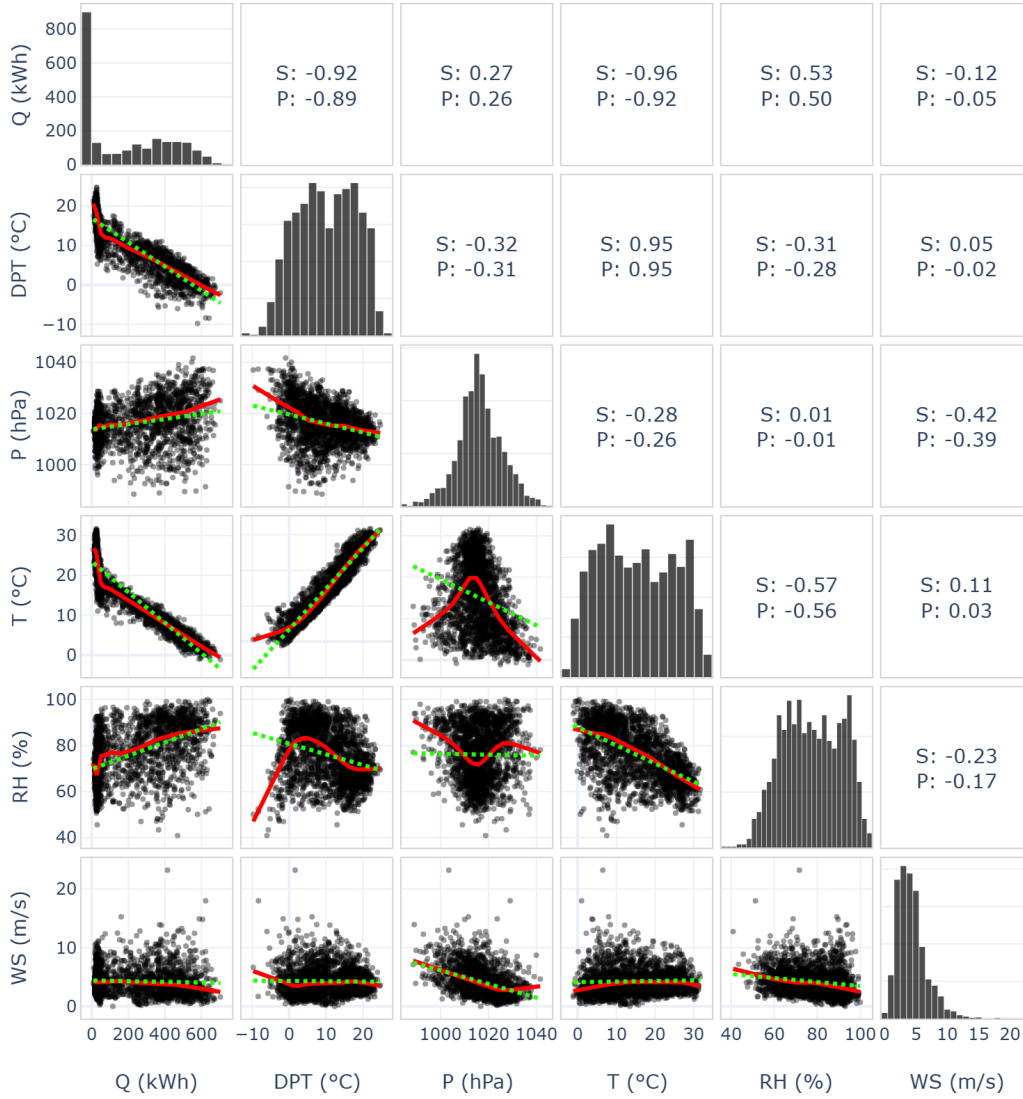


Figure 1.3: Pairwise relationships between heat demand and meteorological variables for Facility F1 (daily data). The lower triangle shows scatterplots with linear regression (green, dotted) and LOWESS smoothing (red). The diagonal panels display the marginal distributions, and the upper triangle reports Spearman and Pearson correlation coefficients. *Notes:* Q = thermal load, DPT = dew point, P = air pressure, T = temperature, RH = relative humidity, WS = wind speed.

Chapter 2

Statistical Pipelines

This chapter introduces two statistical forecasting pipelines that serve different purposes in the empirical study. We first describe MSTL-ETS, a fully univariate decomposition approach that separates annual, weekly, and daily seasonal structure and forecasts the remaining component using an ETS model (Section 2.1). While this pipeline is not intended to be a top-performing method in our setting, it provides a transparent and lightweight reference baseline that helps contextualise the gains achieved by more expressive models. We then develop the SARIMAX pipeline, which is designed as a competitive statistical benchmark: it combines a linear regression on engineered temperature features with a seasonal ARIMA error component to capture residual autocorrelation (Section 2.2). For both approaches, we detail the modelling choices and the tuning strategy used to obtain the final configurations.

2.1 MSTL-ETS Pipeline

This section introduces MSTL-ETS, a fully univariate decomposition-based forecasting pipeline that combines Multiple Seasonal-Trend decomposition using Loess (MSTL) [4] with an Exponential Smoothing State Space (ETS) model [31]. Annual seasonality is estimated from daily aggregates via classical decomposition and removed from the hourly series; MSTL then extracts weekly and daily seasonal components, while ETS forecasts the deseasonalised trend-remainder. The component forecasts are subsequently recombined to produce predictions on the original scale, yielding a transparent and lightweight reference baseline.

2.1.1 Pipeline Overview

The pipeline can be outlined as follows:

1. *Data pre-processing.* Each series is transformed to stabilise variance, with particular emphasis on the coldest months (typically from mid-November through late March, hereafter referred to as the *extended winter*), when heat demand is most variable and most operationally relevant.
2. *Annual decomposition.* The series is aggregated to the daily level and a yearly seasonal component is extracted. This annual component is later removed from the original hourly data.
3. *Hourly decomposition via MSTL.* MSTL is applied to the deseasonalised hourly series to extract weekly and daily seasonal components, together with a long-term trend and a remainder.
4. *Component forecasting.* Seasonal components are extrapolated using a naïve seasonal strategy, that is, by repeating the most recently estimated seasonal cycle over the forecast horizon, while the trend-plus-remainder is forecast using an ETS model.
5. *Recombination.* Forecasts of all components are summed to obtain predictions on the transformed scale, which are then back-transformed to the original units.

The following subsections describe each step in more detail.

2.1.2 Seasonal-Trend decomposition using Loess

The predictive accuracy of this pipeline depends critically on the quality of the extracted seasonal components. The MSTL algorithm used here relies on Seasonal-Trend decomposition using Loess (STL) [16] as its fundamental building block. Rather than providing a full technical account of the method, we present a brief overview to clarify its main ideas.

STL is a flexible procedure for decomposing a time series $Y_t, t = 1, \dots, n$, into trend T_t , seasonal S_t , and remainder R_t components using locally weighted regressions,

$$Y_t = T_t + S_t + R_t, \quad t = 1, \dots, n.$$

It was designed to be simple, robust to outliers, and applicable to a wide range of periodicities. Unlike classical decomposition methods, it allows the seasonal pattern to evolve gradually over time. Note that, in contrast to MSTL (described later), STL models a single seasonal component S_t .

The algorithm consists of two nested loops. Denote by $T_t^{(k)}$ and $S_t^{(k)}$ the trend and seasonal estimates at the k th step of the *inner loop*, initialised with $T_t^{(0)} = 0$ and $S_t^{(0)} = 0$. At this stage, the detrended series $Y_t - T_t^{(k)}$ is processed as follows. First, each cycle-subseries (that is, all observations occupying the same position within successive seasonal cycles) is smoothed locally using a first-degree LOESS fit [17]. In this step, a local linear regression is fitted around each time point, with observations weighted according to their proximity in time. The extent of this local neighbourhood, and therefore the smoothness of the resulting fit, is controlled by the seasonal window $n_{(s)}$. The resulting smoothed subseries are then reassembled into a full-length sequence $C_t^{(k+1)}$. A *low-pass filter*, implemented as a sequence of moving averages, is applied to $C_t^{(k+1)}$ to obtain a slowly varying trend of the raw seasonal estimate, denoted $L_t^{(k+1)}$. The updated seasonal component is then given by

$$S_t^{(k+1)} = C_t^{(k+1)} - L_t^{(k+1)}.$$

The trend estimate is subsequently updated by applying LOESS smoothing to the seasonally adjusted series $Y_t - S_t^{(k+1)}$.

In the *outer loop*, used only in the “robust” variant, robustness weights are computed from the remainder component and used to down-weight outliers and localised anomalies. Concretely, observations with large remainder values (relative to a robust scale estimate of the residuals) receive smaller weights, so that they contribute less to the LOESS smoothing steps in later inner-loop iterations. This yields a version of STL that is less sensitive to irregular spikes.

Although conceptually simple, STL exposes several parameters that (directly or indirectly) control the smoothness of the seasonal and trend components and the strength of the robustness iteration. Cleveland et al. [16] provide clear defaults for all parameters except the seasonal LOESS window $n_{(s)}$ and the decision of whether to use the robust option. These are consequently the only elements that strictly require manual tuning, and they determine our tuning strategy in the construction of the MSTL-ETS pipeline.

2.1.3 Data Pre-Processing

Variance Stabilisation

The raw heat-demand series exhibit clearly heteroscedastic behaviour: the variance increases markedly with the level of the series, particularly during periods of high demand in winter. Since MSTL assumes approximately constant variance when performing its additive decompositions, a variance-stabilising transformation is applied as a first step.

A simple and effective option in this context is the logarithmic transform, which is especially suitable given the steep rise in variability at high demand levels. Because demand values are clipped to remain above 5 kWh (see Section 1.5), there is no risk of numerical issues due to near-zero observations under the log.

As an alternative, we consider the Box–Cox transformation [9]:

$$\text{BoxCox}_\lambda(y) = \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \lambda \neq 0, \\ \ln(y), & \lambda = 0, \end{cases} \quad (2.1)$$

where λ controls the strength of the transformation. Values of λ close to 1 leave the data almost unchanged, apart from a constant shift of -1 , so the series remains essentially on its original scale. As λ tends towards 0, the transformation approaches the logarithm, providing progressively stronger variance stabilisation. Hence the Box–Cox family spans a continuum between an identity-like transformation and the log, allowing the degree of stabilisation to be matched to the level of heteroscedasticity in the data.

Because our forecasting models are most concerned with the cold regime, where both demand and variability are highest, we estimate λ using only observations from the *extended winter* period of each series. This period typically spans mid-November to late March, with exact boundaries chosen per facility using the available training data. Estimating λ in this way tailors the variance-stabilising transformation to the regime in which it is most needed.

This design choice aligns with the flexibility of MSTL: the method does not assume that seasonal amplitudes remain constant throughout the year and can accommodate gradual transitions between regimes. By stabilising variance during the extended-winter interval, while still fitting MSTL to the full series, we allow the

method to extract a more reliable representation of the winter seasonal structure, which is the period of greatest operational relevance. As the series shifts into the much flatter summer regime (or vice versa), MSTL adjusts the seasonal components smoothly so that the decomposition continues to reflect the underlying behaviour of the data.

Several methods exist for estimating λ . The *Guerrero estimator* selects the value that makes the *coefficient of variation*, defined as the ratio of the standard deviation to the mean, as stable as possible across predefined seasonal subsets. Because this criterion depends directly on variability within each subset, it can be sensitive to outliers or irregular fluctuations. The *maximum log-likelihood estimator*, in contrast, chooses the λ that maximises the (profile) Gaussian log-likelihood of the transformed data, re-estimating the mean and variance for every candidate value. In our experiments, this likelihood-based approach produced more stable λ estimates in the presence of extreme observations and is therefore adopted throughout this work.

Treatment of Summer Anomalies

After applying the logarithmic or Box–Cox transformation, a small number of unusually low values remain (particularly in series F3). These observations are substantially below the local mean and could, in principle, be imputed (e.g., by forward filling). In this work, however, we choose to retain them, for two main reasons:

- *Robustness of MSTL.* When run in robust mode, MSTL down-weights isolated abrupt deviations, limiting their influence on the estimated seasonal and trend components.
- *Seasonal context.* These dips occur almost entirely during the warm months. Because they take place well outside the high-load winter regime, any minor leakage into the seasonal or trend components is typically corrected long before winter demand rises again, as MSTL gradually readjusts to the approaching winter pattern.

By leaving these summer anomalies intact, we exploit the built-in robustness of MSTL without introducing additional modelling assumptions or bias through ad hoc imputation.

2.1.4 Two-Stage Seasonal Decomposition

The series exhibit three main seasonalities of interest: annual, weekly and daily. Applying MSTL directly to the hourly data in order to capture all three periods would be computationally expensive. To reduce this burden, we employ a two-stage decomposition strategy. First, the annual seasonality is extracted from daily-aggregated data. The resulting annual component is then removed from the hourly series, after which MSTL is applied only to estimate the weekly and daily seasonalities. This procedure retains all relevant periodic structure while substantially reducing computational cost.

Annual Seasonality from Daily Aggregates

Annual seasonality is estimated from the series aggregated at the daily frequency. We considered three methods for extracting the yearly pattern:

- *Classical Decomposition (CD)*. This is the simplest approach. The seasonal component is computed as the average of observations separated by one year, producing a perfectly periodic seasonal pattern.
- *Seasonal-Trend decomposition using Loess (STL)*, introduced in Section 2.1.2.
- *Robust STL*. A variant of STL that down-weights outliers during the iterative fitting procedure, also mentioned in Section 2.1.2.

Initial experiments showed that robust STL was not suitable in our setting. Given that the dataset covers only five annual cycles, the robust procedure struggled to distinguish between genuine seasonal structure and multi-day anomalies (e.g., unexpected cold spells). As a result, the robust weighting sometimes amplified rather than suppressed the influence of irregular episodes, inadvertently incorporating them into the seasonal component.

Standard STL and classical decomposition yielded very similar seasonal patterns. Considering the limited number of annual cycles and the higher tuning effort required by STL, we concluded that the additional complexity of STL was not justified. We therefore adopt classical decomposition for the annual component.

Daily and Weekly Seasonalities via MSTL

Once the annual component is estimated and removed from the hourly series, we apply MSTL to extract daily and weekly seasonal components. MSTL extends STL to multiple seasonal components by applying STL iteratively in a nested fashion, handling one seasonal period at a time from the highest-frequency component to the lowest. In our pipeline, the MSTL configuration is exposed through the parameters `periods_mstl`, `windows_mstl`, and the dictionary `stl_kwargs_mstl`. These parameters are passed directly to the corresponding arguments of `statsmodels`' MSTL class. Specifically,

- `periods_mstl` provides the list of seasonal periods used by MSTL, set to 24 (daily) and 168 (weekly) for the hourly data;
- `windows_mstl` specifies the LOESS smoothing window lengths for each seasonal component, governing the trade-off between flexibility and smoothness in the decomposition (corresponding to the seasonal window $n_{(s)}$ in the notation of Section 2.1.2).

Additional STL-related settings (such as trend smoothing and robustness parameters) are specified through `stl_kwargs_mstl` and passed unchanged to `statsmodels`, where they are *shared across all seasonal components*.

In this work, we set `robust=True` for all decompositions, as preliminary experiments consistently showed improved stability in the presence of sharp drops and other irregularities. Our tuning efforts therefore focus on the seasonal smoothing windows. This is in line with the original STL formulation, where the seasonal window (together with the robustness option) is the only parameter without a reliable automatic default, and with the MSTL paper, which emphasises the importance of this choice [16, 4].

2.1.5 Model Estimation and Component Forecasting

After decomposition, the series is represented as the sum of an annual seasonal component, weekly and daily seasonal components, a long-term trend, and a remainder term. Forecasting proceeds component-wise:

- *Seasonal components.* Annual, weekly, and daily seasonal components are forecast using a naïve seasonal strategy, that is, by repeating the most recently

estimated seasonal cycle over the forecast horizon. This choice reflects the assumption that seasonal patterns are stable or only slowly evolving over the forecast periods considered.

- *Trend plus remainder.* The deseasonalised series (the original series minus the annual, weekly, and daily seasonal components) is modelled using an Exponential Smoothing State Space (ETS) model, estimated via Nixtla's AutoETS.

In AutoETS, we specify `model="ZZN"`, which instructs AutoETS to let the data choose between additive or multiplicative error (first Z), and among several trend specifications (second Z), while omitting any seasonal component (N), since seasonality has already been accounted for through decomposition. Under this specification, AutoETS searches over all ETS configurations compatible with "ZZN", fits each candidate model by maximum likelihood, and selects the one with the lowest corrected Akaike Information Criterion (AICc) [29]. AIC is an information-theoretic model-selection criterion that ranks candidate models by combining a measure of in-sample fit (the maximised log-likelihood) with an explicit penalty for the number of estimated parameters. It can be derived as an approximately unbiased estimator (up to an additive constant) of the expected out-of-sample Kullback–Leibler discrepancy between a fitted candidate model and the true data-generating process [12]. AICc applies a finite-sample correction that increases the effective penalty for additional parameters when the sample size is limited, reducing the tendency of AIC to favour overly complex models.

Once the best ETS model is selected, forecasts are generated in the corresponding state-space framework. The forecasts of all components (annual, weekly, daily, and trend-plus-remainder) are then summed on the transformed scale, and the inverse variance-stabilising transformation (logarithm or Box–Cox) is applied to obtain predictions in the original units of heat demand.

2.1.6 Hyperparameter Tuning

Rolling-Origin Cross-Validation Design

The main hyperparameters of the pipeline are tuned by rolling-origin cross-validation. The objective is to evaluate alternative configurations across multiple forecast windows and to identify those that yield robust performance. Tuning is

carried out independently for each series (F1 to F5) and for each forecast horizon (day-ahead and week-ahead).

The evaluation period spans from mid-October 2023 to the end of March 2024, covering most of the cold regime for that year across all series. For each horizon, we define a sequence of evenly spaced rolling evaluation windows within this period (30 windows for day-ahead forecasting and 20 for week-ahead forecasting). In each fold, the model is trained on all observations up to (and excluding) the start of the evaluation window and evaluated on that window. Performance is measured using the error metrics introduced in Section 1.6, with a primary focus on MAE. This procedure is repeated across all windows and all hyperparameter combinations. For each configuration, we compute the mean and standard deviation of the metrics across folds and use these summary statistics to identify robust models. In parallel, we record computational time and favour configurations that yield clear efficiency gains over marginal improvements in accuracy.

Tuned and Fixed Hyperparameters

As mentioned in the previous sections, several aspects of the pipeline are kept fixed during tuning:

- Annual seasonality is always estimated using classical decomposition.
- Global MSTL settings (`stl_kwargs_mstl`) are kept at their default values in `statsmodels`, except for `robust=True`, which is enabled across all seasonal components.

Hyperparameter tuning thus focuses on the following components:

- *Data transformation* (`transform`). We compare a simple log transformation with a Box-Cox transformation in which λ is estimated by maximum log-likelihood using only the extended winter period.
- *Training horizon* (`train_horizon`). This is the number of most recent observations retained for MSTL decomposition and subsequent forecasting at the hourly level. Annual decomposition always uses the full training set.
- *Seasonal smoothing windows* (`windows_mstl`). This parameter specifies the LOESS smoothing spans for the two seasonal components in MSTL (daily and

weekly). It is a list of two integers, corresponding to the smoothing window lengths for the daily and weekly seasonalities, respectively.

Table 2.1 provides an overview of the hyperparameter ranges explored during tuning. Hyperparameter optimisation was carried out using a grid search.

Table 2.1: Hyperparameter search space for the MSTL-ETS pipeline.

Hyperparameter	Values / Range
transform	{“log”, “boxcox”}
train_horizon	{7, 11, 15, 19, 23} (<i>weeks of hourly data</i>)
windows_mstl	{[9, 11], [11, 15], [13, 17], [15, 19], [17, 21], [19, 23]}

Notes: when transform=“boxcox”, the Box–Cox parameter is estimated via maximum log-likelihood over the extended-winter portions of the training data. The parameter train_horizon is reported here in weeks rather than hours for readability. Robust MSTL decomposition (robust=True) is used in all configurations.

Tuning Results and Discussion

Across all series and horizons, the best-performing configurations share several common characteristics.

First, models consistently prefer longer training horizons and larger seasonal smoothing windows. In all cases, the selected window sizes exceed the default range of [11, 15]. Longer training horizons provide the decomposition algorithm with a broader temporal context, making it easier to down-weight isolated outliers or irregular fluctuations. Likewise, larger seasonal windows yield smoother and more stable seasonal patterns, reducing the tendency to overfit short-lived variations.

Second, the choice between a log transformation and a Box–Cox transformation has only a modest effect on forecasting performance. Although the Box–Cox transformation can stabilise variance slightly more effectively during the extended winter period, the flexibility of MSTL in modelling both trend and multiple seasonal components largely compensates for these differences. Consequently, both transformations yield similar performance across the error metrics considered.

Finally, performance differences across MSTL configurations are consistently small. When examining the five top-performing models for each series and forecast horizon (ranked by mean MAE), differences in mean MAE are typically within one

unit, while the corresponding standard deviations across folds range from 15 to 40. Although part of this variability may reflect the non-stationary behaviour of the series, the results indicate that MSTL is inherently robust to moderate changes in its configuration.

2.2 SARIMAX Pipeline

This section provides an overview of our SARIMAX pipeline, which serves as the main statistical benchmark in this project. The method combines a regression component based on engineered exogenous features with a seasonal ARIMA (SARIMA) model for the remaining dynamics. In contrast to the MSTL–ETS pipeline, which relies exclusively on the historical structure of the target series, SARIMAX incorporates meteorological information through a linear specification. The guiding idea is to represent heat demand as the sum of two interpretable components: a *deterministic* part explained by external drivers and fixed seasonal structure, and a *stochastic* remainder capturing residual autocorrelation and short-term fluctuations. The core model is implemented using StatsForecast¹ (Nixtla’s statsforecast Python package).

2.2.1 Pipeline Overview and Rationale

The SARIMAX Model

Given a target series y_t and a vector of exogenous regressors $\mathbf{x}_t$, we assume

$$y_t = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_t + \eta_t, \quad \eta_t \sim \text{SARIMA}(p, d, q) \times (P, D, Q)_s.$$

The regression term models the effect of the engineered features, while the SARIMA component accounts for the remaining serial dependence, including the dominant daily seasonality and short-lived deviations not explained by the regressors.

More explicitly, $\eta_t \sim \text{SARIMA}(p, d, q) \times (P, D, Q)_s$ means that η_t satisfies

$$\Phi(B^s) \phi(B) (1 - B)^d (1 - B^s)^D \eta_t = \Theta(B^s) \theta(B) \varepsilon_t, \quad \varepsilon_t \sim \text{WN}(0, \sigma^2),$$

where B is the backshift operator ($B^k \eta_t = \eta_{t-k}$), and $\{\varepsilon_t\}$ is a white-noise process, meaning that $\mathbb{E}[\varepsilon_t] = 0$, $\text{Var}(\varepsilon_t) = \sigma^2$, and $\text{Cov}(\varepsilon_t, \varepsilon_{t-k}) = 0$ for $k \neq 0$. The sequence

¹<https://nixtlaverse.nixtla.io/statsforecast>

$\{\varepsilon_t\}$ is often referred to as the *innovations*.

The non-seasonal and seasonal autoregressive (AR) and moving-average (MA) polynomials are given by

$$\begin{aligned}\phi(B) &= 1 - \phi_1 B - \dots - \phi_p B^p, & \theta(B) &= 1 + \theta_1 B + \dots + \theta_q B^q, \\ \Phi(B^s) &= 1 - \Phi_1 B^s - \dots - \Phi_P B^{Ps}, & \Theta(B^s) &= 1 + \Theta_1 B^s + \dots + \Theta_Q B^{Qs}.\end{aligned}$$

Equivalently, defining the differenced residual series

$$w_t = (1 - B)^d (1 - B^s)^D \eta_t,$$

yields $w_t \sim \text{SARMA}(p, q) \times (P, Q)_s$. Intuitively, after seasonal and non-seasonal differencing, the transformed residuals can be expressed as a linear combination of past values and current and past innovations, including seasonal lags.

Pipeline Overview

The pipeline can be outlined as follows:

1. *Data pre-processing*. We apply a variance-stabilising transformation to the target series (with emphasis on the cold regime), mirroring the motivation in the MSTL-ETS pipeline.
2. *Exogenous feature engineering*. Starting from the available meteorological inputs (primarily temperature), we construct a design matrix $\mathbf{x}_t$ by applying transformations and aggregations (e.g. rolling averages, lags, regime splitting) and by adding Fourier terms to represent weekly seasonality.
3. *Order selection and model fitting*. SARIMA orders are either fixed *a priori* or selected via StatsForecast's AutoARIMA stepwise procedure (described in Section 2.2.4), after which the SARIMAX model is fitted with the engineered regressors using StatsForecast's ARIMA.
4. *Forecast generation*. Multi-step forecasts are produced directly by the fitted SARIMAX model.
5. *Back-transformation*. Forecasts are mapped back to the original units by applying the inverse of the variance-stabilising transformation.

The following subsections describe some of these steps in more detail.

Rationale and modelling assumptions

A key motivation for SARIMAX is that heat demand is strongly driven by outdoor temperature during the heating season. Relative to purely univariate approaches, incorporating temperature allows the model to track demand changes that are difficult to infer from the recent history of the target alone (e.g. sudden cold spells). At the same time, SARIMAX remains comparatively interpretable: regression coefficients quantify the effect of each engineered feature, while the SARIMA component provides a parsimonious description of residual temporal dependence.

The main limitation of this approach is that exogenous effects enter through a linear regression with time-invariant coefficients. In our application, the relationship between temperature and heat demand is non-linear, with demand flattening above a threshold (see Figure 1.3, p. 16). To mitigate this, we include non-linear transformations of temperature (e.g. regime splitting and smoothed variants) among the regressors.

A second limitation is that the seasonal ARIMA component models a single seasonal period s (here daily, $s = 24$). When weekly structure is also present, we represent it through additional Fourier regressors. This effectively treats weekly seasonality as fixed and perfectly periodic in the transformed space, which is a simplification, but it provides a lightweight way to capture systematic day-of-week effects without introducing a second stochastic seasonal component.

2.2.2 Exogenous Feature Engineering

As mentioned earlier, the SARIMAX approach relies on carefully engineered features to represent seasonality and weather effects, since the regression component is linear and does not automatically capture non-linear relationships in the way many deep-learning models can. We therefore designed a feature-engineering procedure in which each exogenous series can be transformed through several optional steps before entering the regression component. The main feature-engineering control knobs are summarised in Table 2.2 and described below.

Table 2.2: Key configuration parameters controlling SARIMAX feature engineering and seasonal structure in the proposed pipeline. Type abbreviations: Opt=Optional, It=Iterable, Map=Mapping.

Name	Type	Description
with_exog	bool	Whether exogenous regressors are used in the regression component.
exog_vars	It[str]	Names of climate-based regressors to include.
threshold	float	Temperature threshold used to define the cold regime via the 4-day rolling mean (controls is_cold and cold/warm splitting).
drop_warm	bool	If True, drops most warm-regime regressors after splitting.
days_avgs	Opt[It[int]]	Rolling means over specified multiples of 24 hours for each climate regressor (low-frequency smoothing).
hours_avgs	Opt[It[int]]	Rolling means over the specified number of hours (high-frequency smoothing).
lags_exog	Opt[It[int]]	Lags (in hours) added for raw hourly regressors (only if drop_h_exog=False).
max_lag_d	Opt[It[int]]	For each daily-averaged regressor, adds lags at multiples of 24 hours up to the specified maximum.
max_lag_h	Opt[It[int]]	For each hourly-averaged regressor, adds lags at multiples of 1 hour up to the specified maximum.
drop_h_exog	bool	If True, drops raw hourly climate regressors after constructing aggregates (keeps only derived features).
use_peak_h	bool	If True, further splits cold-regime, high-frequency climate features into peak vs off-peak components.
peak_hours	Opt[It[int]]	Hour-of-day indices defining peak hours (applied on weekdays only).

Continued on next page

Name	Type	Description
kw	int	Number of weekly Fourier harmonics (period 168) added as regressors.
kw_c_only	bool	If True, removes warm-regime weekly Fourier terms after cold/warm splitting.
transform	Opt[str]	Target transform applied before fitting (e.g., boxcox_cold for winter-focused variance stabilisation).
lam_method	Opt[str]	Method used to select Box–Cox λ (e.g., log-likelihood based).
sar_kwargs	Map[str, Any]	Dictionary specifying the SARIMA residual component, including non-seasonal order (order), seasonal order (seasonal_order), and seasonal period (season_length).

Regime splitting by average temperature. A first key element is the use of temperature to identify two demand regimes: a *cold* regime $\mathcal{C}$ and a *warm* regime $\mathcal{W}$. The warm regime corresponds to the summer period in which demand is close to flat, while the cold regime covers the heating season, from autumn ramp-up to the spring ramp-down.

Let x_t denote a generic exogenous regressor and τ_t the outdoor temperature at time t . Conditional on the regime assignment, each regressor is split into two components so that the model can learn different linear relationships in cold and warm conditions:

$$\begin{aligned} x_{c,t} &= x_t \mathbb{1}(t \in \mathcal{C}), \\ x_{w,t} &= x_t \mathbb{1}(t \in \mathcal{W}), \end{aligned}$$

where $\mathbb{1}(\cdot)$ is the indicator function. In addition, we include

$$x_{\text{is_cold},t} = \mathbb{1}(t \in \mathcal{C}),$$

to allow for a regime-dependent intercept shift.

We define the cold regime using a 4-day rolling average of temperature (hence 96 samples):

$$\bar{\tau}_t = \frac{1}{96} \sum_{i=0}^{95} \tau_{t-i}.$$

Given a threshold τ_0 , we assign $t \in \mathcal{C}$ whenever $\bar{\tau}_t < \tau_0$, and $t \in \mathcal{W}$ otherwise.

The threshold τ_0 is tuned separately for each site to account for differences in plant characteristics and operating conditions. We perform a simple grid search over candidate values of τ_0 : for each candidate, we fit a linear regression of y_t on $(\tau_{c,t}, \tau_{w,t}, x_{\text{is_cold},t})$ and select the threshold that yields the best in-sample fit according to AICc.

Finally, because demand in the warm regime is typically close to flat, we often drop most warm-specific regressors to reduce model complexity and computational cost with negligible impact on accuracy.

Features derived from climate variables. Each weather variable can enter the regression either in raw form or after smoothing. In particular, we consider moving-average features to attenuate high-frequency fluctuations and retain slower trends. This can be beneficial because the within-day shape of heat demand is also driven by behavioural and operational factors, while slower variations are more plausibly explained by weather conditions.

We also allow lagged versions of both raw and smoothed variables. The motivation is that demand may respond with delay to changes in external conditions (for example due to thermal inertia), in which case including recent past values of the regressors can improve fit and forecast accuracy.

Separation into peak and off-peak hours. To better capture the operationally important morning peak, we optionally split raw exogenous variables (and other high-frequency aggregates, such as moving averages shorter than one day) into peak and off-peak components. Let $\mathcal{P}$ denote a set of peak hours (defined at the hourly level). For a regressor x_t , we construct

$$x_{p,t} = x_t \mathbb{1}(t \in \mathcal{P}), \quad x_{o,t} = x_t \mathbb{1}(t \notin \mathcal{P}).$$

The rationale is that the overall temperature–demand relationship is strongly non-linear at high frequency, but conditional on hour-of-day it can be closer to linear, making a piecewise linear specification more adequate.

Fourier features for weekly seasonality. Some series exhibit a pronounced weekly seasonal component. Since the seasonal ARIMA error component captures only one seasonal period s (here daily), we model weekly seasonality through additional regressors based on Fourier terms. Let $m = 168$ be the number of hours in a week and let $K \in \mathbb{N}$. For $k = 1, \dots, K$, we include

$$f_{k,t}^{\sin} = \sin\left(\frac{2\pi kt}{m}\right), \quad f_{k,t}^{\cos} = \cos\left(\frac{2\pi kt}{m}\right),$$

and apply the same cold/warm regime split to these terms.

Because Fourier terms are perfectly periodic while the magnitude of weekly fluctuations typically increases when demand is higher, we apply a winter-focused Box–Cox transformation to stabilise variance before fitting the model. Although transforming the response could in principle introduce curvature in the temperature–demand relationship, we found that this effect is limited in practice over the relevant winter range, so a linear temperature specification remains a reasonable approximation on the transformed scale. This, however, still effectively assumes that the weekly seasonal pattern is approximately stable in the transformed space, which is a strong simplification.

2.2.3 Model Estimation

Conditional on fixed hyperparameters (feature-engineering choices and SARIMA orders), model parameters are estimated using StatsForecast’s ARIMA implementation. The library supports three estimation methods (CSS, ML based on exact Gaussian maximum likelihood, and the hybrid CSS–ML), described below.

For any candidate parameter vector ϑ (collecting the regression coefficients and the ARMA parameters), define the differenced regression residual

$$w_t(\vartheta) := (1 - B)^d(1 - B^s)^D(y_t - \beta_0 - \boldsymbol{\beta}^\top \mathbf{x}_t).$$

Under the SARIMAX specification, $w_t(\vartheta)$ satisfies the seasonal ARMA relation

$$\Phi(B^s)\phi(B)w_t(\vartheta) = \Theta(B^s)\theta(B)\varepsilon_t,$$

for the innovation sequence $\{\varepsilon_t\}$.

Given ϑ , fitted innovations $\hat{\varepsilon}_t(\vartheta)$ can be computed recursively using the ARMA

recursion once a conditional initialisation for pre-sample quantities is fixed, for instance by conditioning on the first observed values and setting pre-sample innovations to zero. This yields a *conditional* sequence of prediction errors whose early terms depend on the chosen initialisation.

The conditional sum-of-squares (CSS) estimator [62] then minimises the sum of squared fitted innovations, typically after a cut-in t_0 to reduce sensitivity to the transient induced by the initialisation, using a numerical optimiser:

$$S_{\text{CSS}}(\vartheta) := \sum_{t=t_0}^n \hat{\varepsilon}_t(\vartheta)^2.$$

Alternatively, under the working assumption that ε_t are i.i.d. Gaussian, one can estimate ϑ by maximising the exact Gaussian log-likelihood, which corresponds to the ML method. This likelihood can be evaluated efficiently via a state-space formulation with Kalman filter recursions [e.g. 18] or via equivalent exact-likelihood algorithms for ARMA processes [e.g. 2].

In the StatsForecast API, the default option is the hybrid CSS-ML method, which first obtains starting values via CSS and then performs numerical optimisation of the Gaussian log-likelihood initialised at the CSS estimate. Both ARIMA and the AutoARIMA model described later default to this method, which we use throughout.

2.2.4 Hyperparameter Tuning

Throughout, we use temperature as the sole meteorological input. This choice is motivated by the fact that adding further climate variables typically triggers a large expansion of derived regressors (e.g. lags and rolling averages for each raw variable), substantially increasing fitting time, while offering no measurable benefit for the LSTM model developed in parallel, despite its greater flexibility.

Staged Procedure for Hyperparameter Tuning

Jointly optimising the feature-engineering choices for the exogenous regressors and the SARIMA orders of the residual component is computationally impractical, as it would require an expensive search over a large combinatorial space. We therefore adopt a staged procedure that decouples order selection from feature selection.

Stage 1. We first fix a simple and conservative set of temperature-based features

(raw temperature, a 1-day moving average, a small number of weekly Fourier terms, and minimal lagging) and run Nixtla’s AutoARIMA to select the residual orders that minimise the corrected Akaike information criterion (AICc). AutoARIMA performs a stepwise search over candidate ARIMA and seasonal ARIMA specifications and returns the best model found under the chosen information criterion.

Stage 2. Fixing the SARIMA orders obtained in Stage 1, we compare a grid of feature-engineering configurations (12 variants, from simpler to more expressive) using a rolling-origin cross-validation schedule on a held-out period (autumn 2023 to spring 2024). We evaluate models on a dense grid of windows, with a step size of 24 h between adjacent cut-offs, while restricting the number of refits to 15 to keep computation tractable. The selected configuration is the one with the best out-of-sample predictive accuracy under this validation protocol. Each SARIMAX model is implemented using `statsforecast`’s ARIMA, which allows fitting a user-specified order rather than searching over orders.

Stage 3. Finally, we re-run AutoARIMA using the feature configuration selected in Stage 2 to assess whether the preferred SARIMA orders change once the exogenous component is better specified, and we again validate the resulting models out-of-sample. In practice, this refinement often produced more complex specifications with improved AICc but worse forecasting accuracy under the validation schedule, consistent with overfitting to an in-sample criterion. We therefore retained the simpler orders selected in Stage 1.

To further reduce tuning time, we performed hyperparameter optimisation only for the week-ahead task. For the SARIMA residual component, the orders are selected via AutoARIMA by minimising AICc, that is, by an in-sample information criterion rather than a horizon-specific forecast-accuracy objective; consequently, order selection is not directly tied to whether the downstream evaluation is day-ahead or week-ahead. In contrast, the exogenous feature configuration affects the estimated regression component, which assumes a time-invariant linear relationship with constant coefficients. Features that encode relationships that persist over time (e.g. temperature effects and low-frequency seasonal structure) should therefore be informative across both horizons, since the same coefficient estimates enter every step of the forecast. Moreover, parameter estimation is driven by in-sample one-step

prediction errors through the likelihood, and therefore does not directly optimise a horizon-specific multi-step loss.

Nevertheless, horizon-specific optima are possible in principle: day-ahead accuracy may benefit from slightly more responsive, high-frequency features, whereas week-ahead performance may favour smoother representations. For consistency across model classes and to keep the tuning budget comparable, we adopt a similar strategy for the deep learning models reported later, tuning primarily on the week-ahead task.

How AutoARIMA Selects Orders

We use StatsForecast’s AutoARIMA implementation to select ARIMA and seasonal ARIMA orders.² The procedure follows the Hyndman–Khandakar stepwise strategy [32]: it first chooses the differencing orders and then performs a stepwise search over AR and MA orders by minimising AICc.

If the seasonal differencing order D is not supplied, the default behaviour is to decide whether to take a seasonal difference using the seas heuristic, which is based on an STL seasonal-strength measure. Conditioning on the chosen D , if the non-seasonal differencing order d is not supplied, it is selected via successive stationarity testing (the KPSS test [36] by default) applied to the seasonally differenced series, subject to an upper bound. When exogenous regressors are used, the differencing tests are applied to residuals from an initial linear regression of the target on the supplied regressors, in order to target the ARIMA error component rather than the deterministic part explained by $\mathbf{x}_t$.

Conditioning on (d, D) and the user-specified seasonal period s , AutoARIMA then searches over (p, q, P, Q) (and, when admissible, the inclusion of a constant or drift) by minimising AICc. The default search is stepwise, meaning that it explores the space of candidate models through local moves rather than fitting the full grid of possibilities.

In practice, we found that using the default starting specification ($p = q = 2$ and $P = Q = 1$ in the statsforecast implementation) often resulted in substantially longer stepwise-search runtimes and, under our rolling-origin validation protocol, poorer out-of-sample accuracy than starting from a zero-order model. We therefore

²Implementation details were verified against the official source code: <https://github.com/Nixtla/statsforecast>.

initialised the stepwise procedure from

$$\text{SARIMA}(0, d, 0) \times (0, D, 0)_s,$$

by setting the starting orders to zero. With this initialisation, the stepwise search consistently converged to simpler final models that achieved stronger forecasting accuracy on the validation windows.

Comment on Stage 2 tuning results

During Stage 2 feature tuning, we primarily considered MAE. A key observation is that most of the predictive gain is captured by a small set of exogenous regressors. In particular, the 1-day rolling average of temperature and Fourier terms for weekly seasonality were consistently important for good performance. Adding raw temperature also often improved accuracy, typically by a small but meaningful margin. Beyond these core components, additional regressors tended to yield only marginal improvements.

In a few cases, we nevertheless retained a more expressive configuration when it provided slightly better accuracy at the cost of only a few extra seconds of fitting time, as the main focus of this study is forecasting accuracy rather than scalability. That said, more time-efficient alternatives with only marginally higher average errors are possible, with runtime savings of up to 10–20 s relative to a roughly one-minute fit in some settings.

Chapter 3

LSTM Pipeline

To improve forecasting accuracy beyond classical benchmarks, we adopt an Encoder–Decoder architecture based on a Long Short-Term Memory (LSTM) network. This chapter begins with a brief overview of LSTMs and the specific Encoder–Decoder configuration employed (Section 3.1), followed by a description of the hyperparameters and tunable components of the LSTM pipeline developed in this project (Section 3.3). It then introduces Bayesian optimisation, focusing on the Tree-Structured Parzen Estimator (TPE) used for tuning deep learning networks, including LSTMs, in this thesis (Section 3.4). The chapter concludes with a discussion of the multi-step tuning methodology adopted for the LSTM pipeline (Section 3.5).

3.1 The LSTM Network

3.1.1 Motivation and Limitations of Standard RNNs

In contrast to feed-forward networks such as multilayer perceptrons (MLPs), recurrent neural networks (RNNs) [25] incorporate feedback connections that route activations from earlier time steps back into the model. This recurrent pathway allows the network’s internal state to carry temporal context forward and influence decisions at subsequent time steps.

Training standard RNNs with backpropagation through time (BPTT) is challenging because gradients are products of time-local Jacobians that reuse the same recurrent weights across time, which leads to vanishing or exploding magnitudes. Vanishing gradients, in particular, make conventional RNNs struggle to capture long-

range dependencies. In the early 1990s, research by Bengio et al. [7] formalised this difficulty, showing that standard RNNs cannot reliably learn dependencies beyond about 5–10 time steps in practice. As a result, RNN research languished for a period, as deeper networks were easier to train on non-sequential tasks.

In 1997, Long Short-Term Memory (LSTM) [27] networks were proposed to mitigate these issues. An LSTM replaces a plain recurrent hidden layer with memory blocks that contain a persistent *cell state* and multiplicative gates regulating information flow: the *input gate* controls how much new information is written to the cell; the *forget gate* (added in subsequent work [24]) attenuates or preserves the previous cell state; and the *output gate* determines how much of the cell state is exposed to downstream layers. In effect, LSTM gating allows gradients to propagate backwards through time more effectively, mitigating the vanishing gradient problem.

LSTM networks have since become a foundational component in deep learning for time series forecasting. Among the spectrum of deep learning models, LSTMs represent one of the simplest architectures capable of capturing sequential structure with minimal preprocessing or domain-specific feature engineering. Despite the emergence of more complex alternatives (such as those mentioned in Section 1.2), LSTMs continue to rank among the strongest baseline models in forecasting benchmarks [38], and are widely used as robust, interpretable, and well-supported architectures across domains.

3.1.2 LSTM Architecture and Update Equations

Given an input sequence $\mathbf{x}_{1:T} = (\mathbf{x}_1, \dots, \mathbf{x}_T)$, an LSTM iterates the following updates for $t = 1, \dots, T$:

$$\mathbf{i}_t = \sigma(W_{ix}\mathbf{x}_t + \mathbf{b}_{ix} + W_{ih}\mathbf{h}_{t-1} + \mathbf{b}_{ih}) \quad (3.1)$$

$$\mathbf{f}_t = \sigma(W_{fx}\mathbf{x}_t + \mathbf{b}_{fx} + W_{fh}\mathbf{h}_{t-1} + \mathbf{b}_{fh}) \quad (3.2)$$

$$\mathbf{g}_t = \tanh(W_{cx}\mathbf{x}_t + \mathbf{b}_{cx} + W_{ch}\mathbf{h}_{t-1} + \mathbf{b}_{ch}) \quad (3.3)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \quad (3.4)$$

$$\mathbf{o}_t = \sigma(W_{ox}\mathbf{x}_t + \mathbf{b}_{ox} + W_{oh}\mathbf{h}_{t-1} + \mathbf{b}_{oh}) \quad (3.5)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \quad (3.6)$$

Here, the W terms are weight matrices shared through time and the $\mathbf{b}$ terms are

bias vectors; σ is the sigmoid function; $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ are the input, forget, and output gate activations; $\mathbf{c}_t$ is the cell (or memory) state; and $\mathbf{h}_t$ is the exposed hidden (or output) state. The operator $\odot$ denotes element-wise multiplication. This is the standard non-peephole LSTM used in common libraries. We write two biases per gate (one paired with the input-to-hidden transform and one with the hidden-to-hidden transform) to match PyTorch's parameterisation for cuDNN compatibility; mathematically, using a single bias is equivalent.

All gate activations have the same dimensionality as the hidden vector $\mathbf{h}_t$ and the cell state $\mathbf{c}_t$; this shared dimensionality is commonly referred to as the LSTM's hidden size. Let D be the input dimension and H the hidden size. Then

$$W_{ix}, W_{fx}, W_{cx}, W_{ox} \in \mathbb{R}^{H \times D}, \quad W_{ih}, W_{fh}, W_{ch}, W_{oh} \in \mathbb{R}^{H \times H},$$

and the biases are in $\mathbb{R}^H$.

The equations above show that the LSTM maintains two complementary representations: the cell state $\mathbf{c}_t$, a persistent internal memory, and the hidden state $\mathbf{h}_t$, a task-facing output. It also employs three multiplicative gates that control how information is written, retained, and revealed.

Input gate $\mathbf{i}_t$. Determines how much new candidate content is written into memory at step t . Each component of $\mathbf{i}_t \in [0, 1]^H$ scales the corresponding component of the candidate update $\mathbf{g}_t$.

Forget gate $\mathbf{f}_t$. Controls retention or decay by scaling the previous cell state $\mathbf{c}_{t-1}$ before it is carried forward. When the components of $\mathbf{f}_t \in [0, 1]^H$ are close to 1, information is preserved; when they are close to 0, it is forgotten.

Output gate $\mathbf{o}_t$. Regulates how much of the memory in $\mathbf{c}_t$ is exposed to the rest of the network via $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$, allowing information to be stored without being revealed at every step.

Functionally, $\mathbf{h}_t$ is a gated, squashed view of the memory cell $\mathbf{c}_t$, whereas $\mathbf{c}_t$ itself is the durable substrate on which information is accumulated, preserved, or decayed over time. In practice, $\mathbf{h}_t$ is the working output used to make predictions: in sequence tagging it is mapped to the current label y_t ; in one-step forecasting $\mathbf{h}_t$ serves as features to predict the next value y_{t+1} ; and in Encoder–Decoder models

(e.g., for translation) one passes $\mathbf{h}_T$ (or all past hidden states when using attention) to a decoder that generates the target sequence.

3.2 The Seq2Seq LSTM Architecture

3.2.1 Architecture Overview

To generate multi-step-ahead forecasts for the next h time steps following the last observed time point T , we employ an LSTM-based encoder–decoder architecture. Both encoder and decoder are LSTMs. The encoder processes a fixed lookback window of length k , comprising the target series and exogenous variables from $T - k + 1$ through T , and compresses the relevant history into the final hidden and cell states $(\mathbf{h}_T, \mathbf{c}_T)$. The decoder is initialised with these states and, for each step $t = T + 1, \dots, T + h$, conditions on the corresponding future covariates $\mathbf{x}_t$ (e.g., calendar or weather features) to update its hidden and cell states $(\mathbf{h}_t, \mathbf{c}_t)$. A linear (or shallow MLP) projection maps $\mathbf{h}_t$ to the forecast $\hat{y}_t$. Optionally, the decoder may operate autoregressively by feeding its previous prediction $\hat{y}_{t-1}$ (together with $\mathbf{x}_t$) back as input at time t . We also allow stacking multiple LSTM layers in both encoder and decoder (with a shared hidden size) to obtain a deeper model. A schematic of the single-layer variant is provided in Figure 3.1.

3.2.2 Design Rationale and Alternatives

The encoder–decoder architecture introduced above, also referred to as the sequence-to-sequence (seq2seq) architecture, was originally proposed in 2014 for neural machine translation to handle input and output sequences of potentially different lengths [56, 14]. Traditional RNNs could be trained for sequence prediction tasks, such as next-word prediction, but they were not designed to generate an entire output sequence directly. The new architecture solved this by breaking the task into two phases: an encoder RNN reads the input sequence and compresses its information into a fixed-size context vector, and a decoder RNN unfolds this vector into the output sequence. Encoder–Decoder models quickly found broad use beyond translation, for tasks like speech-to-text, text summarisation, and conversational response generation. In all cases, the ability to encode variable-length context and decode to variable-length outputs in an end-to-end fashion was a major advantage.

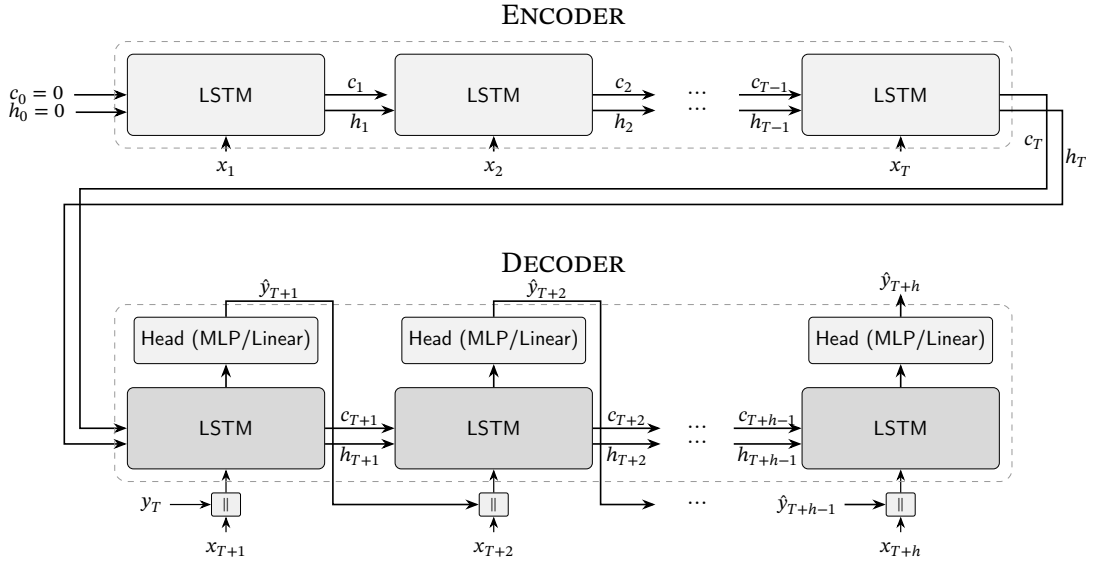


Figure 3.1: Encoder–decoder LSTM architecture used in this study. For simplicity of notation, the encoder length is shown as $k = T$.

The success of encoder–decoder RNNs in NLP naturally inspired their application to time series forecasting, where there is also a need to map input sequences to output sequences of potentially different lengths in a time-aware manner. Although, in forecasting, the length of the prediction horizon is typically fixed and known in advance, the model must still learn how to encode historical context and generate temporally aligned future outputs, often while conditioning on known future inputs. Early demonstrations showed that LSTM-based seq2seq models could outperform traditional models or vanilla RNNs by better capturing the temporal structure in the data [64, 44]. The seq2seq architecture fits multi-step time-series forecasting well: it accommodates mismatched input and output lengths, integrates exogenous variables known in advance in a natural way, and supports autoregressive decoding to build coherent forecast trajectories.

The use of an LSTM as a decoder is not the only viable design choice in encoder–decoder forecasting architectures: MLP-based decoders are also common. For example, Wen et al. [61] propose a two-stage decoder composed of a *global* and a *local* MLP. The global MLP takes as input the encoder’s summary vector and the entire sequence of future-known covariates, and outputs a set of horizon-specific context vectors (one for each of the h forecast steps) alongside a horizon-agnostic context shared across all horizons. The *local* MLP is then applied independently at each future time step:

it takes as input the corresponding future covariates, the associated horizon-specific context, and the horizon-agnostic context, and produces the prediction for that horizon. Unlike LSTM-based decoders, which rely on sequential recurrence and feed their own predictions into subsequent steps, this architecture generates all forecasts in parallel. In many implementations, this can be computationally convenient, since forecasts for all future time steps are produced in parallel. MLP-based decoders therefore offer a practical alternative for multi-step forecasting, particularly when informative future covariates are available and can help compensate for the lack of feedback from previous time steps.

We ultimately opted for an LSTM decoder as it provides a natural mechanism for multi-step forecasting: the same recurrent cell is reused at every prediction step, so adjusting the forecasting horizon only affects how long the decoder is unrolled rather than requiring architectural modifications, as would be the case for an MLP-based decoder.

3.2.3 Configurable Autoregressive Modes in the Decoder

To support different trade-offs between forecast consistency, training complexity, and computational efficiency, we implemented a configurable mechanism enabling various levels of autoregression (AR) within the decoder. The supported decoding strategies are as follows:

Step-wise AR mode. The decoder receives, at each time step t , the previously predicted target value $\hat{y}_{t-1}$ (except for $t = T + 1$, in which case it receives the true target value y_{t-1}). This allows the model to condition sequentially on its own outputs, capturing fine-grained temporal feedback. This is the mode depicted in Figure 3.1.

Non-AR mode. The decoder does not take previous predictions as inputs; instead, each forecasted value is produced using only the decoder’s recurrent state and the known future covariates. Temporal information is still propagated through the hidden and cell states, but there is no explicit autoregressive feedback via past outputs. This simplifies training and reduces computational cost, though it may limit the model’s ability to capture short-range dependencies within the forecast horizon.

Block AR mode. The decoder at time t is provided with the lagged value $\hat{y}_{t-b}$, enabling inference in blocks of length b rather than one step at a time (as in the step-wise AR mode). This approach is particularly useful for long-horizon forecasts, where recurring daily patterns can be leveraged efficiently. When the forecast horizon is shorter than the block length, the method reduces to a static lag-based strategy.

These alternative configurations offer flexibility across multiple axes: while non-autoregressive decoding is computationally efficient and easier to train, autoregressive modes may yield improved performance in settings where the target exhibits strong short-term autocorrelations.

3.3 Configuration and Hyperparameter Space

This section outlines the key components of the pipeline configuration, including model architecture, feature design, preprocessing, and training hyperparameters. While some of these hyperparameters are standard across deep learning applications and will be mentioned only briefly (or omitted altogether), others are more specific to our forecasting setup and warrant detailed discussion.

3.3.1 Model Architecture Hyperparameters

Table 3.1 summarises the main hyperparameters controlling the model’s architecture.

Of particular note is the head parameter, which determines the type of projection used to map the LSTM decoder output to the final forecast. When set to “linear”, a single linear transformation is applied at each forecast step. When set to “mlp”, a single-hidden-layer MLP is used, with hidden dimensionality equal to its input size and with ReLU activation. In both cases, the projection head is applied identically at each forecast step, with weights shared across time.

When configured with a positive probability, dropout is applied at multiple points in the architecture: between LSTM layers (when $\text{num_layers} \geq 2$), before the projection layer when using a linear head, and after the activation function when using an MLP head. These regularisation mechanisms help mitigate overfitting, particularly in models with higher capacity.

Table 3.1: Model-level hyperparameters that specify the architecture and decoding behaviour of the LSTM encoder–decoder in Figure 3.1.

Name	Type	Description
input_len	int	Length of the encoder context window.
output_len	int	Forecast horizon length.
hidden_size	int	Dimensionality of LSTM hidden states.
num_layers	int	Number of stacked LSTM layers in encoder and decoder.
head	str	Output head type: “linear” or “mlp”.
dropout	float	Dropout probability.
use_ar	str	AR mode in decoder (cf. Section 3.2.3): “none” (non-AR), “prev” (step-wise AR), or “24h” (block-wise AR).

3.3.2 Data Loading and Preprocessing Hyperparameters

Table 3.2 summarises the main hyperparameters controlling data preprocessing.

Each sample in the dataset used by the LSTM consists of a sequence of length $k + h$, where k is the length of the lookback window processed by the encoder, while h is the forecast horizon.

To facilitate effective training, we normalise the input variables so as to stabilise gradient-based optimisation by bringing all features to comparable scales. We experimented with both *z-score standardisation* (standardising to zero mean and unit variance) and *min-max normalisation* (rescaling to a fixed range, typically $[0, 1]$).

In addition, we tested two normalisation strategies:

Global: scaling parameters (e.g., mean and standard deviation) are computed on the training portion of the dataset and then applied consistently to all samples across training, validation, and test sets.

Per-sample: each sample is normalised independently, using statistics computed only over its encoder window (past k steps). These statistics are then applied to both the encoder and decoder portions of the sample. This mode ensures that each forecast is conditioned only on information available at prediction time and can adapt to varying local dynamics.

3.3.3 Training Configuration Hyperparameters

Table 3.3 summarises the main hyperparameters controlling training.

Table 3.2: Data loading and preprocessing hyperparameters that control how sliding-window samples are generated and how input series are normalised before being passed to the LSTM model.

Name	Type	Description
stride	int	Step size between consecutive sliding windows. A stride of 1 means windows are generated hourly; 24 corresponds to daily windows.
batch_size	int	Number of samples per batch during training and evaluation.
norm_mode	str	Normalisation strategy: “none” applies no normalisation, “global” uses dataset-wide statistics from the training set, and “per_sample” normalises each encoder window independently.
norm_type	str	Type of normalisation: either “zscore” or “minmax”.

The training mechanisms used here that are most specific to the autoregressive decoder (and, more generally, to recurrent models) are *teacher forcing* (TF) [63] and the related training scheme *scheduled sampling* [6].

When the decoder operates in an autoregressive (AR) mode, meaning it recursively uses its own past predictions as inputs, TF consists in supplying the ground truth target value y_{t-1} at each time step t during training, in place of the model’s own previous prediction $\hat{y}_{t-1}$. In the case of block autoregression with block size b , this generalises to replacing $\hat{y}_{t-b}$ with the corresponding true value y_{t-b} . Teacher forcing is widely used to simplify and stabilise the optimisation of recurrent decoders, and has been repeatedly reported to speed up or even enable convergence in practice [63, 34, 41]. At the same time, relying on ground truth values that are unavailable at inference time introduces a mismatch between training and deployment conditions, commonly referred to as exposure bias. Scheduled sampling addresses this by gradually reducing the use of teacher forcing during training in an attempt to narrow the gap between training and inference.

In this work, we adopt a simple decreasing teacher forcing schedule in the spirit of the original scheduled sampling by Bengio et al. [6], tuning only the rate at which the forcing probability decays. Similar decreasing TF schemes have been employed in RNN-based sequence-to-sequence models for spatio-temporal prediction [23, 46]. Alternative curricula in which the teacher forcing ratio increases over training have recently shown promising results in time series settings [57]. However, a systematic comparison of such strategies falls outside the scope of the present study and is left

Table 3.3: Training-related hyperparameters controlling optimisation, regularisation and early stopping in the LSTM forecasting pipeline. Type abbreviation: Opt=Optional.

Name	Type	Description
learning_rate	float	Initial learning rate for the optimiser.
n_epochs	int	Maximum number of training epochs.
patience	int	Number of consecutive epochs without improvement tolerated before early stopping.
es_rel_delta	float	Minimum relative improvement in validation loss required to reset early stopping.
tf_drop_epochs	int	Number of epochs over which the teacher forcing probability decays to zero.
use_lr_drop	bool	Whether to drop learning rate after a specified number of epochs.
lr_drop_epoch	Opt[int]	Specific epoch at which to drop the learning rate; if None, drop occurs when teacher forcing ends.
lr_drop_factor	float	Multiplicative factor for reducing the learning rate.
grad_clip_max_norm	float	Maximum allowed gradient norm for gradient clipping.
weight_decay	float	L2 regularisation coefficient (weight decay).

for future work.

In our implementation, we apply TF at the batch level: with a given probability, an entire training batch is either processed with TF or without it. This allows efficient parallel computation, since all sequences in the batch follow the same mode and can thus be executed at the same time, which is important when training on GPUs. We use a linear schedule to decrease the TF probability over the course of training. This schedule is governed by the parameter `tf_drop_epochs`, which defines the number of epochs over which the TF probability decays to zero. The initial 20% of this interval is treated as a warmup phase, during which the TF probability is fixed at 1.0. After the warmup, the TF probability decreases linearly, reaching zero after 90% of the total `tf_drop_epochs` have elapsed. Beyond this point, TF remains disabled (probability zero), except for periodic “spikes” in which a TF probability of 0.3 is temporarily reintroduced every three epochs as a heuristic safeguard against training instabilities arising from fully autoregressive decoding.

3.3.4 Feature Engineering Hyperparameters

The feature engineering process defines how raw input variables are transformed and expanded to provide richer temporal context and improved predictive signals to the model. The configuration is highly customisable, allowing control over the inclusion of lags, rolling statistics, time features, and seasonality indicators. Table 3.4 summarises all feature-related hyperparameters supported by the pipeline. The following paragraphs describe the main configuration options and their intended roles within the forecasting setup.

The model can process both *endogenous variables*, which include past values of the target series and transformations of these values, and *exogenous variables*, which comprise weather measurements, features derived from these measurements such as lags or rolling averages, and time-based indicators.

Weather exogeneous variables. Each of the five available meteorological variables (specifically temperature, dew point, wind speed, pressure, and humidity) can be selected from the auxiliary data. These exogenous inputs are assumed to be available in advance at prediction time, for example through weather forecasts, and can therefore be incorporated throughout the model to inform both the encoding and decoding stages.

Lags and Rolling Averages. To strengthen the model’s capacity to learn both seasonal structure and longer term dynamics, the configuration allows the use of explicit lags and rolling means for endogenous and exogenous variables. Hour-based lags, for example 24 or 168 hours, provide the model with direct access to past observations at relevant cycle lengths and therefore support the learning of seasonal patterns. Rolling averages smooth short term fluctuations and highlight longer term behaviour, which helps the model learn trend components. As already introduced for the SARIMAX model, the rolling average of a variable x_t over the last w observations is defined as

$$\bar{x}_t^{(w)} = \frac{1}{w} \sum_{i=0}^{w-1} x_{t-i}.$$

Although, in principle, a sufficiently expressive model could learn to reconstruct such lagged or smoothed features internally (given a long enough input window) explicitly providing them often surfaces key periodic and trend information more directly. This intuition is supported by findings in the literature [15, 65], where

decomposing time series into trend and seasonal components improves forecasting accuracy. In our context, lags and moving averages play a comparable role: they highlight recurring patterns and long-term variations without requiring explicit signal decomposition.

We did not include explicit signal decompositions for practical considerations. To avoid data leakage, any such decomposition would need to be recomputed for every encoder window rather than applied once to the full dataset. A more significant difficulty arises in the decoder, where autoregression requires the decomposition to be recalculated at every decoding step using the model predictions. This repeated processing is costly in time, especially because the target series often exhibits several seasonal patterns, for example daily and weekly cycles. Capturing these structures would require decomposition techniques that are more advanced than standard STL or CD (introduced in Section 2.1.4), which would further increase the computational burden.

Nonetheless, we acknowledge that explicit decompositions may provide further performance improvements. Approaches inspired by MSTL-LSTM architectures, for example applying the decomposition only to the encoder inputs in order to avoid repeated computations during decoding, represent an interesting direction for future work.

Time and Calendar Features. Cyclical time features such as hour-of-day, day-of-week, and month-of-year are incorporated to help the model capture structured temporal regularities. These features are encoded using sine and cosine transformations to reflect their periodic nature and ensure continuity at boundaries (e.g., midnight to midnight or Sunday to Monday). For example, if $h_t \in \{0, \dots, 23\}$ denotes the hour-of-day for the timestamp indexed by t , it is encoded by

$$\text{hod}_t^{\sin} = \sin\left(\frac{2\pi h_t}{m}\right), \quad \text{hod}_t^{\cos} = \cos\left(\frac{2\pi h_t}{m}\right),$$

where $m = 24$ is the cycle length in hours.

An additional temporal indicator, referred to as weekday-sat-sun (WSS), distinguishes between weekdays, Saturdays, and Sundays using cyclical encoding. This feature aims to further help the model differentiate among distinct weekly regimes.

A binary holiday indicator was not included, as a visual inspection of the target series did not reveal marked deviations attributable to holidays. Nonetheless, such a

feature could be easily integrated and might offer marginal gains, particularly for improving forecast accuracy around holiday periods.

Differencing. As an alternative to modelling absolute target values, the pipeline optionally supports replacing the target with its 24-hour difference

$$\Delta_t y = y_t - y_{t-24}.$$

This seasonal differencing strategy is motivated by the idea that neural networks can struggle to capture periodic seasonal patterns accurately in time series forecasting tasks [15]. By modelling the difference between consecutive daily values, the model focuses on learning residual adjustments to the seasonal cycle rather than reconstructing the cycle itself from scratch. This can simplify the learning task, especially when daily seasonality is strong and relatively stable. In our experiments, this approach proved beneficial for the SARIMAX model, and we therefore explored its use in the LSTM pipeline as well.

3.4 Bayesian Optimisation with TPE

3.4.1 Bayesian Optimisation

Bayesian Optimisation (BO) is a sequential, model-based methodology designed for the global optimisation of objective functions that are expensive to evaluate and lack analytical expression. As presented by Brochu et al. [10], BO provides a probabilistic framework for efficiently locating the extremum of a black-box function $f : \mathcal{X} \rightarrow \mathbb{R}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ denotes the search space.

In this framework, the unknown function f is modelled as a random function endowed with a prior probability distribution, typically a Gaussian Process (GP),

$$f \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')),$$

where $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ is the mean function and $k(\mathbf{x}, \mathbf{x}') = \text{Cov}(f(\mathbf{x}), f(\mathbf{x}'))$ is the covariance (or kernel) function under the GP prior.

This prior encodes initial beliefs about the smoothness and general behaviour of f . Each evaluation of the true function, composing a dataset of noisy observations

Table 3.4: Feature-engineering hyperparameters specifying which exogenous variables, lagged terms, rolling averages and time-based encodings are included as inputs to the LSTM model. Type abbreviation: Tup=Tuple.

Name	Type	Description
exog_vars	Tup[str, ...]	Names of whether exogenous variables, selected from {temperature, dew_point, wind_speed, pressure, humidity}.
hour_averages	Tup[int, ...]	Rolling means (in hours) computed for both endogenous and exogenous features. For example, (24, 168) adds daily and weekly averages.
endog_hour_lags	Tup[int, ...]	Lags (in hours) applied to endogenous features.
include_exog_lags	bool	Whether to apply the same lags defined in endog_hour_lags to exogenous variables.
use_differences	bool	If true, the target becomes a 24-hour difference (i.e., $y_t - y_{t-24}$).
time_vars	Tup[str, ...]	Time and calendar features to add, chosen among: hod (hour-of-day), dow (day-of-week), moy (month-of-year), and wss (weekday/weekend classification). All features are encoded using sine and cosine pairs.

$\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, is used to update the posterior belief over f :

$$f \mid \mathcal{D}_n \sim \mathcal{GP}(m'(\mathbf{x}), k'(\mathbf{x}, \mathbf{x}')),$$

where $m'(\mathbf{x})$ and $k'(\mathbf{x}, \mathbf{x}')$ denote the posterior mean and covariance functions, respectively, derived from the prior and the observed data. In the standard GP regression setting used in BO, these quantities admit closed-form expressions obtained by conditioning the prior on the observations (see, e.g., Rasmussen and Williams [52]). Together, they define the updated GP surrogate (and its predictive distribution), yielding a progressively more informative representation of the objective landscape as additional evaluations are collected.

The optimisation process proceeds sequentially: at iteration n , the current probabilistic model guides the selection of the next evaluation point $\mathbf{x}_{n+1} \in \mathcal{X}$ by maximising an *acquisition function*. The acquisition function quantifies the expected benefit

of evaluating the objective at a candidate point, balancing two competing objectives:

Exploitation: which prioritises regions that the surrogate model predicts to yield favourable objective values, for example reflected in the posterior mean $m'(\mathbf{x})$ being low in a minimisation setting.

Exploration: which prioritises regions where the model uncertainty is large, reflected in a high posterior variance $k'(\mathbf{x}, \mathbf{x})$, since such areas may contain better optima.

The evaluation of the function at $\mathbf{x}_{n+1}$ yields a new noisy observation

$$y_{n+1} = f(\mathbf{x}_{n+1}) + \varepsilon_{n+1}, \quad \varepsilon_{n+1} \sim \mathcal{N}(0, \sigma_\varepsilon^2),$$

where the noise variance σ_ε^2 is treated as a model hyperparameter. This new observation is added to $\mathcal{D}_n$,

$$\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{(\mathbf{x}_{n+1}, y_{n+1})\},$$

thereby updating the surrogate model.

Common choices for the acquisition function include the Expected Improvement (EI) and the Probability of Improvement (PI) criteria [3]. Assuming the task is to minimise f , the PI function selects the next candidate that maximises the probability of improving upon the current best (lowest) objective value y^* :

$$\text{PI}_{y^*}(\mathbf{x}) = \mathbb{P}(Y \leq y^* \mid \mathbf{x}, \mathcal{D}_n) = \int_{-\infty}^{y^*} p(y \mid \mathbf{x}, \mathcal{D}_n) dy, \quad (3.7)$$

where $p(y \mid \mathbf{x}, \mathcal{D}_n)$ denotes the posterior predictive distribution under the GP prior and the assumption of Gaussian observation noise:

$$p(y \mid \mathbf{x}, \mathcal{D}_n) = \int_{-\infty}^{\infty} p(y \mid f(\mathbf{x})) p(f(\mathbf{x}) \mid \mathbf{x}, \mathcal{D}_n) df(\mathbf{x}),$$

with $p(y \mid f(\mathbf{x})) = \mathcal{N}(y; f(\mathbf{x}), \sigma_\varepsilon^2)$ and $p(f(\mathbf{x}) \mid \mathbf{x}, \mathcal{D}_n) = \mathcal{N}(f(\mathbf{x}); m'(\mathbf{x}), k'(\mathbf{x}, \mathbf{x}))$.

The EI criterion extends this idea by weighting the potential improvement by its magnitude:

$$\text{EI}_{y^*}(\mathbf{x}) = \mathbb{E}[(y^* - Y)_+ \mid \mathbf{x}, \mathcal{D}_n] = \int_{-\infty}^{y^*} (y^* - y) p(y \mid \mathbf{x}, \mathcal{D}_n) dy, \quad (3.8)$$

where $(\cdot)_+ = \max(\cdot, 0)$.

In practice, it is common to introduce a small positive margin ξ and compute the probability (or expectation) of achieving an improvement over $y^* + \xi$ rather than y^* , that is, using $\text{PI}_{y^*+\xi}$ and $\text{EI}_{y^*+\xi}$. This relaxation makes the improvement condition less stringent and encourages a more exploratory behaviour.

While the PI criterion mainly targets points that are likely to outperform the current best observation, it can become overly exploitative by repeatedly sampling near known good regions. In contrast, the EI criterion also accounts for the potential magnitude of improvement. Thus, EI balances selecting points that are likely to yield small improvements over y^* with selecting points that have a lower chance but the potential for much larger improvements. As a result, EI naturally encourages a more explorative sampling behaviour.

By iteratively refining the surrogate and optimizing the acquisition function, Bayesian Optimisation achieves a principled trade-off between exploration and exploitation. This enables efficient convergence toward the global optimum while requiring only a relatively small number of function evaluations; a property that makes BO particularly suitable for applications where each evaluation of f is computationally or experimentally costly.

3.4.2 Tree-Structured Parzen Estimator

The Tree-structured Parzen Estimator (TPE) algorithm, introduced by Bergstra et al. [8], is a variant of Bayesian Optimisation designed to handle complex, hierarchical (tree-structured) search spaces. It has become one of the most widely used methods for hyperparameter optimisation, forming the core of frameworks such as Hyperopt and Optuna. In what follows, we focus on the Optuna implementation of TPE, which is the version used in our experiments. We rely on Optuna v4.5.0, whose TPE design matches the formulation analysed by Watanabe [60] and discussed below.

As introduced in Section 3.4.1, Bayesian Optimisation uses a probabilistic surrogate model (e.g., a Gaussian Process) of the unknown objective and an acquisition function to select subsequent evaluations. TPE retains the sequential, model-based spirit of BO, but modifies the surrogate-modelling and sampling strategy to better accommodate high-dimensional, conditional, discrete or mixed parameter spaces.

This is reflected in the name of the algorithm itself:

- *Tree-structured* refers to the ability of the method to handle hierarchical or conditional search spaces, where the relevance of certain parameters depends on others. For example, a hyperparameter may be active only when a particular model component is selected. These dependencies are represented as branches in a tree, allowing the algorithm to navigate complex configuration spaces efficiently.
- *Parzen estimator* refers to the use of non-parametric kernel density estimators (KDEs) [53], also known as Parzen estimators. In this approach, the probability density underlying a set of observations is estimated by placing a smooth kernel function on each data point and summing these contributions to obtain a continuous density estimate. The method is *non-parametric* because it does not assume that the underlying distribution belongs to any fixed parametric family, allowing the estimate to adapt flexibly to the observed data.

Below, we describe TPE following the work by Watanabe [60]. All the mathematical derivation described below is rigorous when $\mathcal{X} = \mathbb{R}^d$ for some d (i.e., all hyperparameters are continuous and unconditional). Otherwise, the same steps still hold *in substance*, but one should replace densities by the more correct notion of Markov kernels on the corresponding spaces.

For simplicity, we assume that the set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is already sorted so that $y_1 \leq \dots \leq y_n$.

While classical BO relies on modelling the latent objective function f through a probabilistic prior $p(f)$, the TPE directly models the joint distribution of observations and configurations, $p(\mathbf{x}, y | \mathcal{D}_n)$. The joint distribution can be expressed via the chain rule:

$$p(\mathbf{x}, y | \mathcal{D}_n) = p(\mathbf{x} | y, \mathcal{D}_n) p(y | \mathcal{D}_n).$$

In the TPE framework, the terms on the right-hand side are modelled separately.

A first key assumption made by the TPE model is the following:

$$p(\mathbf{x} | y, \mathcal{D}_n) = \begin{cases} p(\mathbf{x} | \mathcal{D}_n^{(\ell)}), & \text{if } y \leq y^\gamma, \\ p(\mathbf{x} | \mathcal{D}_n^{(g)}), & \text{if } y > y^\gamma, \end{cases} \quad (3.9)$$

where:

- the top-quantile parameter $\gamma \in (0, 1)$ depends on n (for example, in Optuna the default is $\gamma = 0.1$, clipped to ensure that $\mathcal{D}_n^{(\ell)}$ contains at most 25 samples);

- y^γ denotes the empirical γ -quantile of the observed objective values, defined as

$$y^\gamma = y_{n^{(\ell)}}, \quad n^{(\ell)} = \lfloor \gamma n \rfloor = \max\{k \in \mathbb{N} \mid k \leq \gamma n, \},$$

using the fact that $\{y_i\}_{i=1}^n$ is sorted in non-decreasing order;

- $\mathcal{D}_n^{(\ell)}$ is the subset containing the best (lowest) γ fraction of observations, $\mathcal{D}_n^{(\ell)} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n^{(\ell)}}$, and $\mathcal{D}_n^{(g)}$ is its complementary set,

$$\mathcal{D}_n^{(g)} = \mathcal{D}_n \setminus \mathcal{D}_n^{(\ell)}.$$

The marginal distribution of the objective values is modelled as a uniform empirical distribution over the observed samples:

$$p(y \mid \mathcal{D}_n) = \frac{1}{n} \sum_{i=1}^n \delta(y - y_i),$$

where $\delta(\cdot)$ denotes the Dirac delta function. Hence,

$$\int_{-\infty}^{y^\gamma} p(y \mid \mathcal{D}_n) dy = \frac{n^{(\ell)}}{n} \approx \gamma. \quad (3.10)$$

In the remainder, we will assume for simplicity that the two coincide.

We also introduce the shorthand notation

$$\ell(\mathbf{x}) = p(\mathbf{x} \mid \mathcal{D}_n^{(\ell)}), \quad g(\mathbf{x}) = p(\mathbf{x} \mid \mathcal{D}_n^{(g)}),$$

where the explicit dependence on $\mathcal{D}_n^{(\ell)}$ and $\mathcal{D}_n^{(g)}$ is omitted for brevity.

The assumption (3.9) allows for a significant simplification of the EI and PI over y^γ (from equations (3.8) and (3.7), respectively). Following the derivations in Bergstra et al. [8] and Watanabe [60], we obtain the following result.

Proposition 3.4.1 (TPE acquisition function). *Under the notations introduced in Sections 3.4.1–3.4.2, we have*

$$\text{EI}_{y^\gamma}(\mathbf{x}) \propto \text{PI}_{y^\gamma}(\mathbf{x}) \propto \frac{\ell(\mathbf{x})}{p(\mathbf{x} \mid \mathcal{D}_n)}, \quad (3.11)$$

for all $\mathbf{x} \in \mathcal{X}$ such that $p(\mathbf{x} \mid \mathcal{D}_n) > 0$, with proportionality constants in each expression

that are strictly positive. Consequently, if g is positive, for a fixed threshold y^γ , the maximisers of both the EI and the PI coincide and are given by

$$\arg \max_{\mathbf{x} \in \mathcal{X}} \frac{\ell(\mathbf{x})}{g(\mathbf{x})}.$$

Proof. We begin by evaluating the EI over the threshold y^γ . All probability densities below are conditional on the observed data $\mathcal{D}_n$, which we omit for brevity. By definition (3.8),

$$\begin{aligned} \text{EI}_{y^\gamma}(\mathbf{x}) &= \int_{-\infty}^{y^\gamma} (y^\gamma - y) p(y \mid \mathbf{x}) \, dy \\ &= \int_{-\infty}^{y^\gamma} (y^\gamma - y) \frac{p(\mathbf{x} \mid y) p(y)}{p(\mathbf{x})} \, dy \quad (\text{Bayes' theorem}) \\ &= \frac{\ell(\mathbf{x})}{p(\mathbf{x})} \int_{-\infty}^{y^\gamma} (y^\gamma - y) p(y) \, dy. \end{aligned}$$

The integral is strictly positive and independent of $\mathbf{x}$, hence

$$\text{EI}_{y^\gamma}(\mathbf{x}) \propto \frac{\ell(\mathbf{x})}{p(\mathbf{x})}.$$

For the PI criterion (3.7),

$$\begin{aligned} \text{PI}_{y^\gamma}(\mathbf{x}) &= \int_{-\infty}^{y^\gamma} p(y \mid \mathbf{x}) \, dy \\ &= \int_{-\infty}^{y^\gamma} \frac{p(\mathbf{x} \mid y) p(y)}{p(\mathbf{x})} \, dy \\ &= \frac{\ell(\mathbf{x})}{p(\mathbf{x})} \int_{-\infty}^{y^\gamma} p(y) \, dy, \end{aligned}$$

The integral is again strictly positive and independent of $\mathbf{x}$, hence

$$\text{PI}_{y^\gamma}(\mathbf{x}) \propto \frac{\ell(\mathbf{x})}{p(\mathbf{x})}.$$

Next, we expand the marginal density $p(\mathbf{x})$:

$$\begin{aligned} p(\mathbf{x}) &= \int_{-\infty}^{\infty} p(\mathbf{x} | y) p(y) dy \\ &= \ell(\mathbf{x}) \int_{-\infty}^{y^\gamma} p(y) dy + g(\mathbf{x}) \int_{y^\gamma}^{\infty} p(y) dy \\ &= \gamma \ell(\mathbf{x}) + (1 - \gamma) g(\mathbf{x}). \quad (\text{equation (3.10)}) \end{aligned}$$

Substituting this expression into the ratio in (3.11) yields

$$\frac{\ell(\mathbf{x})}{p(\mathbf{x})} = \frac{\ell(\mathbf{x})}{\gamma \ell(\mathbf{x}) + (1 - \gamma) g(\mathbf{x})} = \begin{cases} 0, & \text{if } \ell(\mathbf{x}) = 0, \\ \left[\gamma + (1 - \gamma) \frac{g(\mathbf{x})}{\ell(\mathbf{x})} \right]^{-1}, & \text{otherwise.} \end{cases}$$

Therefore, both EI and PI are increasing functions of the ratio $\ell(\mathbf{x})/g(\mathbf{x})$ on the set

$$\mathcal{X}_+ = \{\mathbf{x} \in \mathcal{X} \mid g(\mathbf{x}) > 0\}.$$

Hence, they share the same maximisers on that set, namely

$$\arg \max_{\mathbf{x} \in \mathcal{X}_+} \frac{\ell(\mathbf{x})}{g(\mathbf{x})}. \quad \square$$

In light of this result, $r(\mathbf{x})$ serves as the acquisition function in TPE. Specifically, at each optimisation step, TPE samples several candidate configurations from $\ell(\mathbf{x})$ (controlled by the `n_ei_candidates` parameter in Optuna, with default value 24) and selects the one that maximises $r(\mathbf{x})$ among them.

The two densities $\ell(\mathbf{x})$ and $g(\mathbf{x})$ are estimated via Parzen estimators. Before describing their structure, note that the hyperparameter space $\mathcal{X}$ can be expressed as the product

$$\mathcal{X} = \prod_{j=1}^d \mathcal{X}_j, \quad \mathcal{X}_j \subset \mathbb{R} \text{ or a finite (possibly non-numerical) set,}$$

where each $\mathcal{X}_j$ represents the domain of the j th hyperparameter. This domain may be an interval in $\mathbb{R}$ for continuous variables, an interval in $\mathbb{N}$ for integer variables, or any finite set for categorical ones. Accordingly, each configuration $\mathbf{x} \in \mathcal{X}$ can be rep-

resented as $\mathbf{x} = (x_1, \dots, x_d)^\top$, where x_j denotes the value of the j th hyperparameter. Likewise, an observed configuration $\mathbf{x}_i$ is written as $\mathbf{x}_i = (x_{i1}, \dots, x_{id})^\top$.

In the *univariate* setting (`multivariate=False`, the default in Optuna), each parameter dimension is modelled independently. The density estimates for the promising and non-promising regions are given by

$$\ell(\mathbf{x}) = \prod_{j=1}^d \left(w_0 p_j(x_j) + \sum_{i=1}^{n^{(\ell)}} w_i k_j(x_j, x_{ij} \mid b_j^{(\ell)}) \right), \quad (3.12)$$

$$g(\mathbf{x}) = \prod_{j=1}^d \left(w_{n+1} p_j(x_j) + \sum_{i=n^{(\ell)}+1}^n w_i k_j(x_j, x_{ij} \mid b_j^{(g)}) \right), \quad (3.13)$$

where the weights $\{w_i\}_{i=0}^{n+1}$ are updated at each iteration, k_j denotes the kernel function associated with the j th parameter, $b_j^{(\ell)}$ and $b_j^{(g)}$ are the corresponding bandwidths, and p_j is a non-informative prior. This formulation is called *univariate* because the joint model factorises across dimensions. Sampling a configuration from $\ell(\mathbf{x})$ amounts to sampling each parameter x_j independently from its one-dimensional marginal,

$$\ell_j(x_j) = w_0 p_j(x_j) + \sum_{i=1}^{n^{(\ell)}} w_i k_j(x_j, x_{ij} \mid b_j^{(\ell)}),$$

and maximising the acquisition ratio $r(\mathbf{x})$ is equivalent to maximising it in each coordinate separately, since

$$r(\mathbf{x}) = \prod_{j=1}^d \frac{\ell_j(x_j)}{g_j(x_j)}.$$

A key advantage of this factorised form is that it naturally supports tree-structured (conditional) search spaces: parameters are sampled in a hierarchical order, where higher-level (parent) variables are drawn first, and subsequent parameters are sampled only if their defining conditions are met. This makes the univariate TPE particularly well suited for hyperparameter spaces with conditional dependencies.

In contrast, the *multivariate* configuration (`multivariate=True`) does not assume

independence of the d dimensions. In this case, the Parzen estimators are defined as

$$\ell(\mathbf{x}) = w_0 \prod_{j=1}^d p_j(x_j) + \sum_{i=1}^{n^{(\ell)}} w_i \prod_{j=1}^d k_j(x_j, x_{ij} \mid b_j^{(\ell)}), \quad (3.14)$$

$$g(\mathbf{x}) = w_{n+1} \prod_{j=1}^d p_j(x_j) + \sum_{i=n^{(\ell)}+1}^n w_i \prod_{j=1}^d k_j(x_j, x_{ij} \mid b_j^{(g)}), \quad (3.15)$$

where each mixture component corresponds to an observed configuration $\mathbf{x}_i$ (except for the non-informative prior), and its contribution is given by a product of one-dimensional kernels centred at the corresponding coordinates x_{ij} . Unlike in the univariate case, the mixture is taken over full observations rather than per-dimension marginals. This means that correlations between parameters can be represented: each component of the mixture encodes a joint configuration of hyperparameters, allowing $\ell(\mathbf{x})$ and $g(\mathbf{x})$ to assign higher probability to regions where promising parameter combinations tend to co-occur. Note, however, that within each mixture component the coordinates are treated independently, since the component density factorises as a product of one-dimensional kernels.

This configuration does not support conditional search spaces, as all parameters are sampled jointly and cannot be treated independently. However, Optuna provides a group option that restores this flexibility by applying the multivariate estimator separately to maximal unconditional subsets of hyperparameters, while treating different groups as independent. We do not further examine this variant, since it was not required for our experiments.

How the choices for these hyperparameters are handled in TPE is described below.

Weights. In Optuna, each Parzen estimators ℓ and g assigns larger relative weight to more recent trials, so newer information dominates the density. To express the weighting scheme precisely, we introduce the following notation.

- Let $h_i \in \{\ell, g\}$ indicate whether trial i , corresponding to observation $(\mathbf{x}_i, y_i)$, belongs to the good set $\mathcal{D}_n^{(\ell)}$ or the bad set $\mathcal{D}_n^{(g)}$.
- Let $m_i = |\mathcal{D}_n^{(h_i)}|$ denote the size of that set.
- Within each set, order trials by increasing trial index (oldest to newest), and let $r_i \in \{1, \dots, m_i\}$ be the rank of trial i inside its own set.

- Let T_{new} be a positive integer representing the number of most recent trials that receive full weight (in Optuna's default, $T_{\text{new}} = 25$).
- Let $\lambda > 0$ be a positive real number representing the (unnormalised) prior weight (Optuna's `prior_weight`).

Then, at each optimisation step, the unnormalised weights are defined as

$$\tilde{w}_i = \begin{cases} \lambda, & i = 0 \text{ or } i = n + 1, \\ 1, & r_i > m_i - T_{\text{new}}, \\ \frac{1}{m_i} + \left(\frac{r_i - 1}{m_i - T_{\text{new}}} \right) \left(1 - \frac{1}{m_i} \right), & r_i \leq m_i - T_{\text{new}}. \end{cases}$$

A more recent weighting scheme proposed by Song et al. [55] suggests using weights w_i in $\ell(x)$ proportional to $y' - y_i$, while keeping uniform weights for $g(x)$. We did not investigate this variant.

Kernel functions. The choice of kernel for hyperparameter x_j primarily depends on its type.

For a continuous hyperparameter with domain $\mathcal{X}_j = [L_j, R_j]$, the kernel k_j is defined as a truncated Gaussian kernel:

$$k_j(x, x_{ij} | b_j) = \frac{\mathbb{1}(x \in [L_j, R_j])}{Z_j(x_{ij})} g(x, x_{ij} | b_j), \quad b_j \in \mathbb{R}^{>0}, \quad x \in \mathcal{X}_j,$$

where $\mathbb{1}$ denotes the indicator function for the specified set, g is the Gaussian kernel, and $Z_j(x_{ij})$ is a normalisation constant ensuring the kernel integrates to one over $[L_j, R_j]$, that is:

$$g(x, x_{ij} | b_j) = \frac{1}{\sqrt{2\pi b_j^2}} \exp\left(-\frac{|x - x_{ij}|^2}{2b_j^2}\right), \quad Z_j(x_{ij}) = \int_{L_j}^{R_j} g(x, x_{ij} | b_j) dx.$$

If the hyperparameter is modelled in logarithmic space, the same formulation applies to the log-transformed variable.

For an integer-valued hyperparameter with domain $\mathcal{X}_j = \{L_j, L_j + q, \dots, R_j\} \subset \mathbb{Z}$, where $q \in \mathbb{N}$ is the step size between consecutive values in the grid, the kernel k_j is

defined as a discretised Gaussian kernel:

$$k_j(x, x_{ij} | b_j) = \frac{\mathbb{1}(x \in \mathcal{X}_j)}{W_j(x_{ij})} \int_{x - \frac{q}{2}}^{x + \frac{q}{2}} g(z, x_{ij} | b_j) dz, \quad b_j \in \mathbb{R}^{>0}, \quad x \in \mathcal{X}_j,$$

where $W_j(x_{ij})$ is a normalisation constant ensuring $\sum_{x \in \mathcal{X}_j} k_j(x, x_{ij} | b_j) = 1$.

For a categorical hyperparameter with $C \in \mathbb{N}$ possible categories (with no inherent ordering), TPE uses the Aitchison–Aitken kernel

$$k_j(x, x_{ij} | b_j) = \begin{cases} 1 - b_j, & \text{if } x = x_{ij}, \\ \frac{b_j}{C-1}, & \text{if } x \in \mathcal{X}_j \setminus \{x_{ij}\}, \\ 0, & \text{otherwise,} \end{cases} \quad b_j \in [0, 1).$$

For the non-informative priors, a Gaussian distribution $\mathcal{N}((R+L)/2, (R-L)^2)$ is employed for numerical parameters defined over $[L, R]$, while a uniform distribution $U(\{1, \dots, C\})$ is used for categorical parameters with C possible classes. These priors serve two purposes: they help mitigate overexploitation by ensuring a baseline level of exploration, and they guarantee that $g(\mathbf{x}) > 0$ for all configurations, thereby allowing $r(\mathbf{x})$ to be evaluated everywhere in the search space.

Bandwidths. For a numerical hyperparameter with domain $[L, R]$ (either continuous or integer-valued), the bandwidth is computed in Optuna as

$$b = \frac{R-L}{5} N^{-1/(d+4)},$$

where d denotes the dimension of the search space and N the number of observations (specifically, $n^{(\ell)}$ for the density $\ell(x)$ and $n - n^{(\ell)}$ for $g(x)$). For a categorical hyperparameter with $C \in \mathbb{N}$ possible categories, the bandwidth is determined as

$$b = \frac{C-1}{N+C}.$$

Unlike alternative implementations (such as the one used in Hyperopt), this formulation makes the bandwidth depend solely on the number of samples and the dimensionality of the space, ensuring a consistent scaling across trials. The bandwidth is decreased gradually as the number of trials increases, promoting broader exploration in the early stages and progressively sharper exploitation around promis-

ing regions later on.

To further prevent overexploitation, when `consider_magic_clip=True` (the default in Optuna), the bandwidth is clipped according to the heuristic

$$b_{\text{new}} = \max(b, b_{\text{min}}), \quad b_{\text{min}} = \frac{R - L}{\min(100, N)}.$$

This safeguard maintains a minimum degree of kernel smoothness.

3.4.3 Hyperparameter Optimisation Procedure

This section describes how hyperparameter optimisation was carried out throughout the project, primarily using Bayesian optimisation via the TPE sampler and, mostly for preliminary experiments, grid search.

Each optimisation experiment was organised as an *Optuna study*, which repeatedly evaluates candidate hyperparameter configurations. Each evaluation is a *trial*: the study proposes a configuration, runs the training procedure (and, when applicable, validation), records the resulting objective value, and then uses all past trials to guide subsequent proposals. Every study involves three key components:

- a *suggester* (sampler), which at the beginning of each trial proposes a configuration to be tested;
- an *objective function*, which evaluates the configuration and returns an *objective value* (e.g. a validation loss); and
- a *pruner*, which monitors intermediate results and terminates unpromising trials early based on partial learning curves (e.g. validation loss after a few epochs).

In our setup, the suggester was either the `TPESampler`, which implements the TPE algorithm discussed in the previous section, or the `GridSampler`, which exhaustively iterates over all possible configurations. The pruner was typically disabled for grid search studies, whereas for TPE-based studies a mild pruning strategy was applied using the `PercentilePruner`.

TPE settings. We initially adopted Optuna’s default TPE settings, except for enabling `multivariate=True`, which is generally recommended for capturing parameter interactions. After the first preliminary study, which showed signs of over-

exploitation (see Section 3.5.1), we increased the number of initial random trials (`n_startup_trials`) from 10 to 25% of the total to encourage broader exploration. At the same time, we increased the number of candidate samples drawn from the modelled likelihood (`n_ei_candidates`) from 24 to 40, aiming to refine candidate selection following the extended exploratory phase.

Objective function. For a given configuration, the objective function trained the model initialised with that configuration on all data available up to, but excluding, November 2023, using the L1 loss. During training, the MAE was computed after each epoch over the period from 2023-11-01 to 2024-04-01 (inclusive), using sliding windows shifted by one time step. These intermediate MAE values were reported to Optuna and monitored by the pruner for possible early termination. Training concluded when either the early-stopping criterion was satisfied, the maximum number of epochs was reached, or the pruner interrupted the trial. The final objective value returned to Optuna corresponded to a summary statistic derived from the recorded intermediate validation MAEs, typically the best intermediate MAE or an average of the MAEs around the best epoch.

Validation design and justification. While rolling-origin cross-validation would provide a more robust performance estimate, it would also substantially increase computational cost given the number of models tuned in this study. In our setting, the training and validation periods exhibit similar seasonal patterns and comparable overall levels, suggesting that no major regime shifts occur over the evaluation horizon. This mitigates the main drawback of omitting cross-validation, namely the risk that the performance estimate is biased by structural changes in the target series. Under these conditions, using a single, relatively long validation window provides a reasonable and computationally efficient measure of out-of-sample accuracy.

Pruner settings. When pruning was enabled, we used a PercentilePruner with a 60% percentile threshold, a patience of three epochs, and a warm-up period of seven epochs. This configuration ensured that only clearly underperforming trials were terminated prematurely.

Rationale for using Optuna. We employed Optuna’s implementation of the TPE estimator rather than alternative versions due to its ease of use and the strong empirical performance reported by Watanabe [60]. In their comprehensive ablation and benchmarking study, Optuna’s TPE (v4.0.0) showed particularly robust behaviour on

discrete and categorical search spaces, largely attributed to its bandwidth selection heuristic and magic clipping. Although newer enhanced variants such as MOTPE and c-TPE can outperform Optuna’s TPE on certain continuous benchmarks, the study confirmed that the Optuna defaults remain highly competitive, especially for heavily discrete or conditional hyperparameter spaces. In our experiments, the search space is predominantly categorical (approximately 85% of the parameters in the main tuning phase), which makes this choice well justified.

3.5 Multi-step Tuning

As discussed in Section 3.3, the pipeline involves a large number of hyperparameters, making it impractical to perform an exhaustive joint optimisation over all of them. Instead, our approach concentrates the tuning effort on the hyperparameters most relevant to the *model* (see Section 3.3.1) and to the *training process* (see Section 3.3.3).

The overall tuning procedure is organised into two main stages:

Preliminary studies (broad exploration across series). This first stage aims to obtain sensible initial settings for the pipeline’s hyperparameters, select strong feature related hyperparameters (see Section 3.3.4), and define suitable ranges for the subsequent search.

Final model tuning (per-series optimisation). The second phase involves per-series optimisation to develop tailored models for each time series and for both daily and weekly forecast horizons. At this stage, the feature-related hyperparameters identified in the preliminary study are fixed, and the tuning effort is redirected toward model- and training-related hyperparameters. As discussed later, we ultimately chose to perform extensive tuning only for the week-ahead horizon, while conducting a partial retuning for the daily case. This decision was made to reduce computational effort and to maintain greater methodological consistency with the SARIMAX pipeline.

Each stage consisted of several Optuna studies, conducted as detailed in Section 3.4.3.

Reporting convention. In the following, whenever we report

$$\text{mean objective } \mu \pm \sigma, (n = \dots),$$

the symbols μ and σ denote the mean and standard deviation of the objective values across a given subset of n trials. If the subset contains values $\{y_1, \dots, y_n\}$, these quantities are defined as

$$\mu = \frac{1}{n} \sum_{i=1}^n y_i, \quad \sigma^2 = \frac{1}{n} \sum_{i=1}^n (y_i - \mu)^2.$$

In some cases, we also report bootstrapped 95% confidence intervals for the sample mean of the objective values, denoted by

$$CI_{95\%}^* = \dots, (n = \dots).$$

These confidence intervals rely on the assumption that the trials within each subset can be treated as approximately independent and identically distributed. This is only an approximation, since the trials arise from distinct yet related hyperparameter configurations. Even with this simplification, the standard approach still offers a sound and interpretable estimate of the average performance and its associated uncertainty, allowing for meaningful comparisons across subsets.

fANOVA decomposition and importances. To obtain insight into the importance of individual hyperparameters, one often inspects fANOVA importances [30, 28]. Let $\mathbf{X} = (X_1, \dots, X_d)^\top$ be a random configuration on $\mathcal{X}$ with distribution ν . We take $\nu = \bigotimes_{j=1}^d \nu_j$, where each ν_j is uniform on the domain $\mathcal{X}_j$ of hyperparameter j , so that the coordinates are independent under ν (more general settings are discussed in Hooker [28]). For any subset $U \subseteq \{1, \dots, d\}$, write $\mathbf{x}_U = (x_j)_{j \in U}$ and $\mathbf{X}_U = (X_j)_{j \in U}$, and denote the corresponding marginal measure by $\nu_U = \bigotimes_{j \in U} \nu_j$. We also denote the complement of U by $-U = \{1, \dots, d\} \setminus U$.

For any U , define the marginal prediction (marginal mean response)

$$a_U(\mathbf{x}_U) = \mathbb{E}_\nu[f(\mathbf{x}_U, \mathbf{X}_{-U})] = \int f(\mathbf{x}_U, \mathbf{x}_{-U}) d\nu_{-U}(\mathbf{x}_{-U}).$$

Thus a_U can be interpreted as a coarse prediction of the objective value based only

on the coordinates in U , obtained by averaging over the remaining hyperparameters.

Functional ANOVA represents f as a sum of effects indexed by subsets,

$$f(\mathbf{x}) = \sum_{U \subseteq \{1, \dots, d\}} f_U(\mathbf{x}_U), \quad (3.16)$$

where each f_U captures the effect associated with the coordinates in U beyond what is explained by strict subsets, defined recursively by

$$f_U(\mathbf{x}_U) = \begin{cases} \mathbb{E}_v[f(\mathbf{X})] & \text{if } U = \emptyset, \\ a_U(\mathbf{x}_U) - \sum_{V \subsetneq U} f_V(\mathbf{x}_V) & \text{otherwise.} \end{cases}$$

Hence, f_U refines a_U by subtracting the contributions already explained by strict subsets $V \subsetneq U$.

Under the product-measure assumption, these functions have zero mean and are orthogonal (see, e.g., Hooker [28]):

$$\mathbb{E}_v[f_U(\mathbf{X}_U)] = 0 \quad \text{for all } U \neq \emptyset, \quad \mathbb{E}_v[f_U(\mathbf{X}_U) f_W(\mathbf{X}_W)] = 0 \quad \text{for } U \neq W.$$

Consequently, the decomposition (3.16) yields a variance decomposition

$$\text{Var}_v(f(\mathbf{X})) = \sum_{U \subseteq \{1, \dots, d\}} \mathbb{V}_U, \quad \text{where } \mathbb{V}_U = \text{Var}_v(f_U(\mathbf{X}_U)).$$

Intuitively, $\mathbb{V}_U$ measures the variability attributable to the joint effect of the hyperparameters in U , after removing all lower-order effects.

Accordingly, the first-order (single-hyperparameter) fANOVA importance of coordinate j is based on $\mathbb{V}_{\{j\}}$. We report the normalised first-order importances

$$I_j = \frac{\mathbb{V}_{\{j\}}}{\sum_{k=1}^d \mathbb{V}_{\{k\}}},$$

so that $\sum_{j=1}^d I_j = 1$. (If instead one normalises by $\text{Var}_v(f(\mathbf{X}))$, the resulting first-order shares need not sum to one in the presence of interactions.)

In practice, f is replaced by a surrogate $\hat{f}$ fitted to the observed trials. In our experiments, $\hat{f}$ is a random forest, and we estimate the fANOVA variance components from $\hat{f}$ using the efficient random-forest procedure of Hutter et al. [30]. Concretely, we

use Optuna’s built-in fANOVA evaluator¹ to compute the corresponding importance scores from the completed study.

3.5.1 Preliminary Studies

The preliminary studies were conducted in several stages and focused exclusively on the *24-hour-ahead* forecasting task.

Objective value. All Optuna studies shared the same objective value: the *best validation MAE* achieved during training over the cold semester of 2023/2024. The decision to use the best validation MAE, rather than the final value, was motivated by the fact that in later tuning phases the number of training epochs will itself be optimised to terminate near the point where validation performance is expected to peak. However, since this metric does not capture the smoothness of the validation curve, the sampler may occasionally favor configurations that produce noisy validation trajectories with sharp, isolated MAE minima. To mitigate this effect, we manually inspected the intermediate validation MAE curves across epochs for each trial.

Step 0: Manual Exploratory Trials

The process began with a set of quick, manually run experiments on series F1. In these tests, the model, training, and data related hyperparameters were fixed to simple default settings (for example, 64 hidden units, an encoder window length of 168, a single LSTM layer, and a decoder operating in AR mode). Temperature was used as the only climate-based exogenous variable, along with standard calendar features.

These initial hand-run trials led to the following observations:

- `norm_mode="global"` clearly outperformed both alternatives, `"per_sample"` and `"None"`. In addition, the choice between `"zscore"` and `"minmax"` normalisation did not produce any noticeable difference in performance.
- Enabling `hour_averages` or `use_cold_season` (i.e., setting them to any value other than `None` or `False`) either degraded performance or had negligible effect.

¹Specifically, `optuna.importance.FanovaImportanceEvaluator`.

- `use_differences=False` consistently yielded better results than the alternative.
- Including both 24-hour and 168-hour lags of the target variable systematically improved performance.

Based on these insights, we established updated default values for feature construction and normalisation, which are now encoded in the configuration classes. These defaults served as the starting point for the subsequent optimisation steps.

Step 1: Initial Tuning of Model and Training Parameters

The first Optuna study (recorded as `preliminary_study_v1_F1`) was conducted on series F1. Its goal was to explore a broad set of modelling and training parameters while keeping features and normalisation fixed to the defaults determined in Step 0. The search space, together with the relevant fixed parameters and enforced constraints, is displayed in Table 3.5.

Table 3.5: Search space for Step 1 (first preliminary Optuna study for tuning the LSTM pipeline). The column *Type* indicates the Optuna/TPE sampling family used for each hyperparameter (categorical or log-uniform continuous). Fixed settings and the excluded heavy configuration are reported below the search space.

Hyperparameter	Type	Values / Range
<code>head</code>	Categorical	{“linear”, “mlp”}
<code>hidden_size</code>	Categorical	{32, 64, 128}
<code>num_layers</code>	Categorical	{1, 2}
<code>dropout</code>	Categorical	{0.0, 0.15, 0.3}
<code>input_len</code>	Categorical	{72, 168}
<code>batch_size</code>	Categorical	{64, 128}
<code>learning_rate</code>	Float (log)	[0.0001, 0.005]
<i>Fixed:</i> <code>n_epochs=25</code> (with early-stopping), <code>use_ar=“prev”</code> , <code>norm_mode=“global”</code> , <code>tf_drop_epochs=15</code> , <code>lr_drop_epoch=15</code> .		
<i>Constraint:</i> exclude computationally heavy configurations in which the choices <code>hidden_size=128</code> , <code>num_layers=2</code> , and <code>input_len=168</code> occur together.		

Sampler behaviour and convergence. The study comprised roughly 200 trials. The optimisation trace showed early concentration on a small set of promising configurations, indicating overexploitation and reduced exploration. As a result,

some hyperparameter combinations were sampled frequently and are therefore well supported empirically, whereas others were visited only sparsely, limiting the reliability of conclusions. The optimisation history also suggested convergence after about 30 trials. While this behaviour is consistent with the observed overexploitation, the fact that about 93% of completed trials obtained a best MAE within the interval $[20.29, 22]$ indicates that the search was operating close to a performance floor: many configurations approached ~ 20 MAE but did not improve beyond that threshold.

Loss dynamics. Validation trajectories were typically noisy during the first 10–15 epochs and then stabilised for runs exceeding 15 epochs ($\sim 88\%$ of completed trials). Two mechanisms likely contributed to this behaviour:

- the scheduled reduction of teacher forcing in the early epochs, which can transiently destabilise validation metrics; and
- a learning-rate halving after epoch 15, which promotes subsequent stabilisation.

Model complexity. Performance remained relatively stable across model sizes:

- *Light models* ($< 40\text{k}$ parameters): mean objective 22.08 ± 1.64 ($n = 14$); the smallest configuration (1 layer, 32 hidden units; $\sim 10\text{k}$ parameters) showed mild underfitting.
- *Heavy models* ($> 160\text{k}$ parameters): mean objective 21.12 ± 0.72 ($n = 60$).
- *Intermediate sweet spot* ($\sim 40\text{k}$ – 110k parameters; e.g., 64 hidden units, 1–2 layers): mean objective 21.03 ± 0.73 ($n = 91$).

These results identified an architecture with 64 hidden units and two layers as a strong candidate.

Regularisation and learning rate. Analysis of one- and two-dimensional binned performance surfaces for dropout and learning_rate showed generally stable results across most configurations. The marginal over dropout and model size showed that zero dropout yielded the best overall results, while the only clearly suboptimal combination (mean objective 23.15 ± 2.25) involved high dropout (0.3) together with a medium or small parameter count ($< 110\text{k}$). Learning rates around 3×10^{-3} produced slightly higher mean objectives and greater variability across both one- and two-dimensional analyses, indicating that intermediate learning rates were more robust.

Decoder head. The choice between “linear” and “mlp” heads did not yield systematic performance differences; fANOVA assigned this factor minimal importance. This indicates that predictive capacity resided primarily in the recurrent backbone, which is reasonable given that the MLP head increased the parameter count by less than 1%. Accordingly, we fixed head=“linear” in subsequent studies to concentrate search effort on more influential dimensions.

Input length. Using an input length of three days or one week yielded comparable results across one- and two-dimensional marginals. The three-day input window often achieved a slightly better mean objective, suggesting that additional contextual information did not substantially benefit the model. This is likely due to the inclusion of lagged target values as model inputs (via `endog_hour_lags=(24, 168)`), which allowed the network to access information from preceding seasonal periods even with shorter input windows. A subsequent study examined the interaction between these two hyperparameters in more detail.

Conclusions. Across 200 trials spanning 132 feasible categorical configurations (some of which were under-sampled) and an additional continuous parameter (the learning rate), the highest-performing settings generally coincided with the most extensively explored regions, lending credibility to the principal findings. Some configurations remained insufficiently sampled, and conclusions for those should be viewed as provisional. Overall, the evidence supported the following defaults: input length 72 (when lag features are present), 64 hidden units with 1–2 layers, batch size 64, low dropout, learning rate $\sim 7 \times 10^{-4}$, and a linear head. These insights provided a solid foundation for the subsequent steps.

Step 2: Climate-Based Exogenous Variables

This step examined whether adding climate related variables beyond temperature, specifically humidity, wind speed, and air pressure, could improve predictive accuracy. Two configurations were compared: a temperature only setup (T) and a setup including temperature plus humidity, wind speed, and pressure (THPW). Dew point was excluded because its strong collinearity with temperature made it unlikely to provide additional predictive value.

For each target series, we ran one Optuna study for each configuration (T or THPW), performing four repeats per configuration with different random seeds. All other hyperparameters were fixed to the defaults identified in Step 1. Model

complexity was not adjusted between configurations, since the parameter increase was modest ($\sim 2\%$ on $\sim 100k$ total parameters) and thus unlikely to affect model capacity or regularisation.

The results are summarised below for each series.

- F1** (preliminary_study_v2_F1). T consistently outperformed THPW in all but one repeat. The single exception was caused by an unusually early activation of the early stopper, which truncated training and compromised the result. In every trial that lasted a reasonable number of epochs, T achieved lower error. This indicates that the additional variables provided no benefit and, if anything, slightly degraded performance.
- F2** (preliminary_study_v2_F2). Results for T and THPW were virtually indistinguishable, with differences smaller than the variability across seeds. Variance across repeats exceeded variance across configurations, indicating that stochastic effects in training had a greater impact than the choice of exogenous inputs. Early stopping typically occurred around 10 epochs, while validation loss remained noisy likely due to the decay of teacher forcing, which can destabilise validation as the model transitions to fully autoregressive prediction. A safer protocol would therefore enforce a minimum of 10–15 epochs before early stopping, ensuring exposure to a few epochs without teacher forcing. Despite these caveats, the study clearly shows that THPW offers no advantage over T.
- F3** (preliminary_study_v2_F3). No trials stopped prematurely; all lasted at least 13 epochs. Here, the evidence strongly favored T: median objective was 29.35 for T versus 31.30 for THPW, a gap well beyond run-to-run variability ($\text{std} < 0.35$). fANOVA attributed roughly 93% of the variance to the exogenous setting and only 7% to the repeat index, confirming that temperature alone yields superior performance.
- F4** (preliminary_study_v2_F4). All trials ended very early (5–10 epochs), likely reflecting the noisy nature of this series. Validation loss converged quickly to near-minimum values, triggering early stopping prematurely and obscuring differences between T and THPW. A second study, enforcing a minimum of 16 epochs before early stopping (preliminary_study_v2_F4_second), confirmed that THPW offered no advantages: validation loss remained flat or worsened

slightly in both configurations despite the extended training. This reinforces the conclusion that the additional variables do not contribute meaningful signal.

F5 (preliminary_study_v2_F5). Again, T outperformed THPW. The gap in median objective was ~ 1.6 points (37.57 vs. 39.18), well above run-to-run variability. fANOVA results aligned with this finding. Early stopping often occurred relatively early (median ~ 11 epochs), consistent with the noisier nature of this series, where overfitting emerges faster. Nonetheless, the advantage of T is clear and robust.

Conclusions. Across all five series, using temperature alone was consistently as effective, or even more effective, than including the additional variables. Incorporating humidity, wind speed, and air pressure never improved accuracy and, in the cases of F3 and F5, slightly degraded it. As a result, the temperature only configuration was adopted as the default exogenous input for subsequent studies.

A possible explanation for the weaker performance of the full THPW configuration in F3 and F5 lies in their pronounced *weekly seasonality* (these are the only series where it is strongly present, as shown in Figure 1.2 on p. 15). Although the inclusion of additional variables did not significantly increase the parameter count, it substantially expanded the feature space (e.g., 14 vs. 11 decoder inputs), potentially diverting model capacity from the most informative temporal patterns. In these series, the model already needed to reconcile both daily and weekly cycles, creating complex interactions between lagged inputs and time features. The added variables likely amplified this complexity, slightly but consistently degrading overall performance.

Step 3: Input Length vs. Lag Features

Step 3 focused on comparing short vs. long input windows (`input_len`), both when including lagged values of the target (and optionally of the exogenous variables) and when excluding them.

In theory, the LSTM's cell state is intended to preserve contextual information across the entire lookback horizon. In practice, however, issues such as vanishing gradients and the inherent challenges of training recurrent architectures often constrain the model's capacity to maintain and retrieve fine-grained temporal dependencies. Given prior evidence that the 24-hour and 168-hour lags are strongly correlated with the current target value, it is reasonable to assume that explicitly including

these lags as input features can help the model recognise and exploit them more effectively, while also alleviating the encoder’s memory burden. Conceptually, this mirrors the rationale behind attention mechanisms, which allow the decoder to access all previous states and dynamically weight them by relevance [47]. Here, we test whether explicitly providing lagged features yields better forecasting performance than relying on the LSTM to infer such relationships implicitly from the raw input window.

Because each study required several hours to complete, we ran this experiment only on two representative series:

- **F1**, one of the most visually regular series, where results are likely to be cleaner; and
- **F3**, one of the two series (F3 and F5) with strong weekly-seasonal patterns, providing contrast to F1.

For these and subsequent experiments, we set `include_exog_lags=True`. Preliminary trials across all series indicated consistently better performance when lagged exogenous variables were included. This outcome is intuitively reasonable, since providing lagged values of both the target and the exogenous variables enriches the temporal context and allows the model to capture persistent relationships between the target and temperature over time.

Search space. For each configuration listed in Table 3.6, we conducted three seeded trials.

Table 3.6: Search space for Step 3 (third preliminary Optuna study for tuning the LSTM pipeline). The column *Type* indicates the Optuna/TPE sampling family used for each hyperparameter (categorical in this case). Fixed settings are reported below the search space.

Hyperparameter	Type	Values / Range
<code>input_len</code>	Categorical	{24, 72, 168, 504}
<code>endog_hour_lags</code>	Categorical	{(), (168,), (168, 24)}
<code>hidden_size</code>	Categorical	{64, 128}
<i>Fixed:</i> <code>num_layers=2</code> , <code>head="linear"</code> , <code>dropout=0.1</code> , <code>lr=7e-4</code> , <code>batch_size=64</code> , <code>norm.mode="global"</code> , <code>n_epochs=25</code> (with early-stopping).		

Results. In the following, we summarise study results by series. We denote 7D1D, 7D, N the (partial) configurations with, respectively, `endog_hour_lags=(168, 24)`, `(168,)` and `()`.

- F1** (`preliminary_study_v3_F1`). Results show a clear preference for the 7D1D configuration. Mean objective values worsen progressively as lags are removed: 20.74 for 7D1D (bootstrapped $CI_{95\%}^*=[20.47, 21.10]$, $n = 24$), 22.30 for 7D ($CI_{95\%}^*=[22.04, 22.61]$, $n = 24$), 24.25 for N ($CI_{95\%}^*=[23.81, 24.72]$, $n = 24$). fANOVA confirms this effect, attributing $\sim 90\%$ of importance to the lag parameter. The standard deviations show the same upward trend (0.70, 0.80, and 1.15, respectively), indicating that including lagged features not only improves accuracy but also enhances training stability and the consistency of the results.
- F3** (`preliminary_study_v3_F3`). The pattern is even more pronounced here: the lag parameter dominates almost entirely, with $\sim 96\%$ fANOVA importance. Including the 7D1D lags leads to markedly better performance compared to other settings, leaving little ambiguity in the result.

Conclusion. The results consistently support the initial hypothesis: explicitly providing the key lagged features ($t - 24$ and $t - 168$) yields superior performance across all input window lengths. Even with long lookback horizons, models incorporating these lags as inputs outperform those relying solely on the LSTM’s internal memory to infer temporal dependencies. Consequently, we adopted the 7D1D configuration as the default setting for all subsequent studies.

Step 4: Effect of the Autoregressive Variable

In this step, we compared autoregressive (AR) and non-autoregressive (NAR) decoding strategies (described in Section section 3.2.3 on page 44) on all five series, fixing all other hyperparameters to sensible defaults based on the previous steps.

In AR mode, the decoder receives the previous predicted target $\hat{y}_{t-1}$ as part of its input at time t . We considered removing this recursive variable for the following reasons.

- *Computational overhead per horizon.* With the AR variable enabled, the decoder must be called explicitly h times, where h is the forecast horizon, because

each prediction depends on the one produced immediately before it. This sequential dependency prevents batching of decoder inputs and makes both the forward and backward passes substantially slower. In practice, under the configuration used in this study, epochs with unit teacher forcing probability (that is, full AR decoding) ran up to four times slower than epochs with full teacher forcing, which require only a single decoder call per pass.

- *Scalability and extension to longer horizons.* AR decoding scales poorly with the forecast horizon: extending the output window to a week (using similar configurations as above), a single training epoch with AR decoding took more than 20 minutes on GPU, compared to only ~10 seconds without AR feedback. This makes hour-by-hour AR decoding impractical for longer horizons. The non-AR variant, in contrast, provides a scalable route to weekly forecasts: we predict in 24-hour blocks and feed each block's output back as a lagged input to the subsequent decoder. This *block-autoregressive* mode (Section 3.2.3) replaces the unavailable true 24-hour lag over multi-day horizons with a model-generated proxy, while retaining the empirical advantage of that feature observed in the previous step. Exogenous lags remain applicable, as forecasts for the exogenous variables are assumed to be available up to seven days in advance.
- *Simpler training dynamics.* Removing the AR variable also simplifies training, as no teacher forcing schedule is required. An additional consequence is that the model is trained directly under the same conditions used during validation. This alignment between training and inference can reduce the number of epochs needed for convergence, further shortening overall training time.

However, removing the AR input may slightly reduce accuracy, as AR decoding enforces step-to-step temporal consistency: each prediction conditions on the models own previous output. Without this feedback, long-horizon forecasts can become more susceptible to drift. Two factors can mitigate this effect:

- *Recurrent state retains recent context.* Even without explicitly feeding back the last prediction, the LSTM's hidden and cell states propagate information across time steps. With sufficient capacity, these states capture short-term dynamics from recent observations, partially compensating for the missing AR signal.

- *Strong lags and exogenous features.* In our configuration, explicit target lags and informative exogenous variables provide direct links to recent history and external drivers. These inputs reduce dependence on self-feedback and help maintain temporal coherence across the forecast horizon.

Search space. We conducted four seeded trials for both AR and NAR configurations across all target series, keeping all other parameters fixed to the defaults identified in previous steps. For example: `input_len=72`, `hidden_size=64`, `num_layers=2`, `head="linear"`, `dropout=0.1`, `lr=7e-4`, `batch_size=64`, `norm.mode="global"` and `n_epochs=25` (with early stopping).

Results. The relevant studies (labeled `preliminary_study_v4_F*`, for series `*`) yielded the following findings.

F1–F2 For these series, accuracy differences between AR and NAR configurations were negligible: mean objective values differed by about ~ 0.1 (approximately one standard deviation), and fANOVA attributed $\sim 75\%$ of the variance to the repeat index and only $\sim 25\%$ to the AR factor. In contrast, runtime was nearly four times faster without AR (averaging $\sim 1'30''$ per trial versus $\sim 4'–6'$ with AR). Validation loss curves for NAR were smoother and stabilised earlier, although the best epochs remained close (median 14 vs. 17 on F1; 13 vs. 14 on F2).

F3–F5 On these series as well, NAR showed no measurable loss in accuracy while converging more quickly and with smoother validation curves. AR models, slowed by teacher forcing, required additional epochs to stabilise. Average runtimes were $\sim 1'20''$ per trial without AR versus $\sim 3'–4'$ with AR (lower than on the first series due to earlier early-stopping triggers), again yielding a three to four times speed-up.

Conclusions. Across all five series, removing the AR variable had no discernible effect on accuracy but consistently reduced training time by a factor of 3–4, with smoother and earlier convergence. These results strongly support adopting the NAR configuration as the default for subsequent experiments.

Step 5: Narrowing Model Complexity per Series ID

As the final stage of the preliminary studies, we conducted small exploratory experiments for each series ID (named `preliminary_study_v5_F*`, for series `*`). In

these tests, we fixed a simple baseline configuration (low dropout, an average learning rate, and a single LSTM layer) and varied only the `hidden_size` across a broad range, repeating each setup twice for robustness.

The aim was to establish an appropriate target level of model complexity for the final studies. Although comparisons between narrower and wider architectures are not strictly comparable without jointly tuning other hyperparameters (such as dropout or learning rate), this procedure was sufficient to identify configurations that were clearly underfitting or noticeably over-parameterised. The latter typically reached their minimum loss within the first epoch and showed pronounced overfitting after only three to four epochs.

This step was crucial because the optimal model capacity is strongly dependent on the signal-to-noise characteristics of each series. Since these ratios vary substantially across IDs, establishing a sensible complexity range for each series ensured that subsequent tuning would focus on models with adequate representational capacity.

3.5.2 Final Hyperparameter Tuning

Experimental Setup

In the final tuning phase, we focused exclusively on model and training hyperparameters, keeping all other design choices fixed according to the preliminary studies. Tuning was carried out on the 7-day-ahead forecasting model, and the resulting configurations were then adapted to the 24-hour-ahead task with minimal additional adjustments. Search spaces for the 7-day-ahead forecasting model are illustrated in Table 3.7.

Choice of Tuning Target. We used the 7-day model as the primary tuning target for two main reasons:

- *More complex training dynamics.* In the 7-day-ahead forecasting task, the decoder is block-autoregressive, and predictions for each day are recursively fed back to generate subsequent days. This requires an explicit teacher-forcing schedule, which is absent in the 24-hour model. Optimising the 7-day configuration therefore ensures that both autoregressive dependencies and teacher-forcing dynamics are tuned coherently. From this configuration, a robust 24-hour variant can be obtained by replacing the autoregressive input with a 24-hour lag of the target and re-tuning only a few training-related parameters

Table 3.7: Main hyperparameter search space used in the final tuning phase (week-ahead model) for the LSTM pipeline. The *Type* column indicates the Optuna/TPE sampling family (categorical, integer, or log-uniform continuous). The *Applicability* column indicates whether a hyperparameter was tuned or fixed for each series, and reports series-specific ranges when these differ. Parameters marked as *Derived* are deterministically set from other choices. Fixed settings shared across all studies are reported below the search space.

Hyperparameter	Type	Values / Range (typical)	Series / Applicability
use_ar	Categorical	{“none”, “24h”}	F1–F3: both explored F4–F5: fixed to “24h”
num_layers	Categorical	{1, 2}	F1–F4: searched F5: fixed to 1
hidden_size	Int	Maximum span across series: [8, 128]	F1: [32,128], step 16 F2: [16,112], step 16 F3: [8,32], step 8 F4: [8,24], step 4 F5: [8,32], step 4
dropout	Float	[0.0, 0.3]	All series
learning_rate	Float (log)	$[1 \times 10^{-4}, 2 \times 10^{-3}]$	All series
lr_drop_epoch	Categorical	{4, 7}	All series
use_weight_decay	Categorical	{False, True}	F3 only
weight_decay	Float (log)	$[1 \times 10^{-6}, 1 \times 10^{-3}]$	F3 only
tf_drop_epochs	Derived	Equal to lr_drop_epoch	Active whenever use_ar=“24h”
<i>Fixed across studies:</i> input_len=72, head=“linear”, norm_mode=“global”, batch_size=64, include_exog_lags=True, n_epochs=25 with early stopping (es_start_epoch=5, es_rel_delta=0.5%), use_lr_drop=True with factor=0.3.			

(primarily learning rate, dropout, and the number of epochs). In contrast, going from a 24-hour to a 7-day model would require designing the entire teacher-forcing schedule from scratch.

- *Methodological consistency.* This strategy also mirrors the one adopted for SARIMAX, where only the 7-day horizon was tuned and then reused for the 24-hour task (see Section 2.2.4), ensuring a better methodological consistency

between the statistical baselines and the LSTM model.

Optimisation protocol. Hyperparameters were optimised *per series* using a Tree-structured Parzen Estimator (TPE), with a forecast horizon of one week and approximately 200 trials per study. When both `use_ar="none"` and `use_ar="24h"` were tested, we ran two separate studies (hence ~ 400 trials in total). All Optuna studies were conducted as described in Section 3.4.3.

For each series, after completing the studies for the two `use_ar` settings, we retained one or two promising configurations from each and selected a final candidate by performing roughly ten repeated runs per candidate, which also informed the choice of the training epoch budget. Once the final week-ahead model was identified, the same architecture was reused for the 24-hour task (by setting `output_len=24`), and only the learning rate, dropout, and number of epochs were re-tuned for this horizon.

Objective value. The optimisation target was defined as the *mean validation MAE computed over a small window centered on the best epoch*, rather than the single best value. This formulation favours models with smoother and more stable training dynamics, reducing the risk of selecting configurations that perform well only at isolated epochs. This is particularly important because validation loss was not monitored during the final testing phase (all data except the test data was used for training).

Findings Across Series

Across all series, several consistent patterns emerged. The following summarises the most salient observations.

Learning rate. The learning rate consistently had the largest influence on performance, with very low values leading to slow or stagnant optimisation and very high values causing unstable training. Across series, a relatively narrow band of intermediate learning rates (around 1×10^{-3}) produced the best trade-off between speed and stability.

Model complexity. The optimal model capacity varied substantially across series, likely reflecting differences in signal-to-noise ratio and temporal structure. For series F1 and F2, models with approximately 110k parameters yielded the best results, suggesting that these series contain more informative temporal dependencies that

benefit from a larger representational capacity. In contrast, for the noisier series F3, F4, and F5, smaller models (with roughly 12k, 4k, and 1.5k parameters, respectively) performed best. In such cases, excessive capacity tends to promote overfitting to noise rather than capturing meaningful temporal patterns, while compact architectures generalise more reliably and converge more stably.

The use of two LSTM layers was almost never advantageous and only improved performance for the cleanest series, where training was comparatively easier. This indicates that the hierarchical representations enabled by deeper networks may introduce unnecessary complexity and unstable gradients when the data do not support such depth.

Regularisation. regularisation effects were broadly consistent. Dropout values close to zero were preferred in nearly all cases, with a single exception (series F3) where a higher value (around 0.25) improved stability. When included in the tuning, weight decay had only a minor influence. This was most likely because generalisation was already supported by other mechanisms, such as structural regularisation through limited model capacity and stochastic regularisation via dropout.

Autoregressive vs. Non-Autoregressive Configurations. For each of the first three series, we tuned two models: one operating in block-autoregressive (BAR) mode and one in non-autoregressive (NAR) mode. NAR models, which remove the feedback of past predictions, trained faster (approximately 20% faster) and did not require a teacher-forcing schedule. However, BAR models consistently achieved lower objective values, typically by about 0.5–2 points, which corresponds to an improvement of roughly 2–8% over total mean objective values of approximately 25 for F1–F2 and 35 for F3. This difference was confirmed as statistically significant by bootstrapped confidence intervals. As BAR models provided higher accuracy with only moderate additional computational cost, we adopted them as the default configuration for subsequent series (F4 and F5). Nonetheless, NAR models remain a practical alternative when computational efficiency is the primary consideration.

Teacher forcing and learning-rate scheduling. In the autoregressive studies, the learning-rate decay and teacher-forcing schedule were typically coupled, with the learning-rate drop epoch coinciding with the end of the teacher-forcing decay. For most series, varying this drop epoch had only a minor effect on performance. The main exception was series F4, where a slower teacher-forcing decay improved validation MAE, while the precise timing of the learning-rate drop remained largely

irrelevant. The final F4 configuration therefore adopted a slow teacher-forcing schedule, such that the model never trained in a fully autoregressive regime. Although this behaviour may appear counterintuitive, empirical results were stable across random seeds. This likely indicates that the model encountered difficulties as teacher forcing diminished, deviating from previously good performance once the guidance from true targets was reduced.

Adaptation to the 24-hour task. After identifying a well-performing weekly BAR configuration for each series, we derived the corresponding 24-hour model by reusing the same architecture. For each series, we then conducted a compact follow-up study focused on fine-tuning the learning rate, dropout, and training duration. These experiments identified optimal learning rate and dropout values that were largely consistent with those obtained during the weekly tuning.

The main difference concerned the training duration: in most cases, the optimal number of epochs was approximately twice as large for the 24-hour-ahead prediction task. This reflects the greater inherent difficulty of the weekly forecasting task, where models tended to converge and overfit more rapidly, thus benefiting less from extended training. Epoch counts were determined by selecting the point at which additional training produced only marginal improvements in validation MAE while maintaining stable performance across repeated runs. In some cases, slightly more time-efficient configurations were also acceptable.

Chapter 4

Other Deep Learning Pipelines

This chapter presents the two additional deep learning pipelines considered in this thesis: the Temporal Fusion Transformer (TFT) and Chronos-2. It begins with a description of the TFT pipeline, outlining the model architecture and the rationale for its adoption, followed by a discussion of the hyperparameter-tuning strategy and the main findings from the tuning experiments (Section 4.1). It then introduces the Chronos-2 pipeline, focusing on the motivation for including it as a zero-shot benchmark, its in-context learning mechanism, and the configuration study used to define its deployment in our experiments (Section 4.2).

4.1 TFT Pipeline

This section presents the second deep learning pipeline developed for heat load forecasting. The pipeline is built around the Temporal Fusion Transformer (TFT) [39] model, provided by the Python library Darts¹. It comprises data normalisation, minimal feature engineering, model training and forecasting with TFT, followed by denormalisation of the predicted values.

4.1.1 Model Description and Rationale

We begin by outlining the TFT model and the motivation for adopting it. For a comprehensive description of the architecture, we refer to the original publication by Lim et al. [39], where TFT was first introduced. Here, we summarise its principal

¹<https://unit8co.github.io/darts/>

components in order to clarify the considerations that guided our choice of this model.

The TFT can be understood as a deliberately enhanced encoder–decoder LSTM architecture in which each additional component is designed to remedy a specific limitation of the underlying recurrent model. Its decoder operates without feeding back its own predictions as inputs for subsequent time steps (the non-AR configuration discussed in Chapter 3), matching the setting used for the day-ahead LSTM experiments, whereas the week-ahead task employed the block-AR decoder. The core seq2seq LSTM remains responsible for modelling local temporal structure, yet TFT surrounds it with mechanisms for conditioning on static covariates, filtering inputs through variable selection, and fusing information through static enrichment, attention and gated residual connections. These components enable the model to handle heterogeneous data, capture long-range temporal relationships, and yield interpretable internal signals. Its architecture is summarised in Figure 4.1.

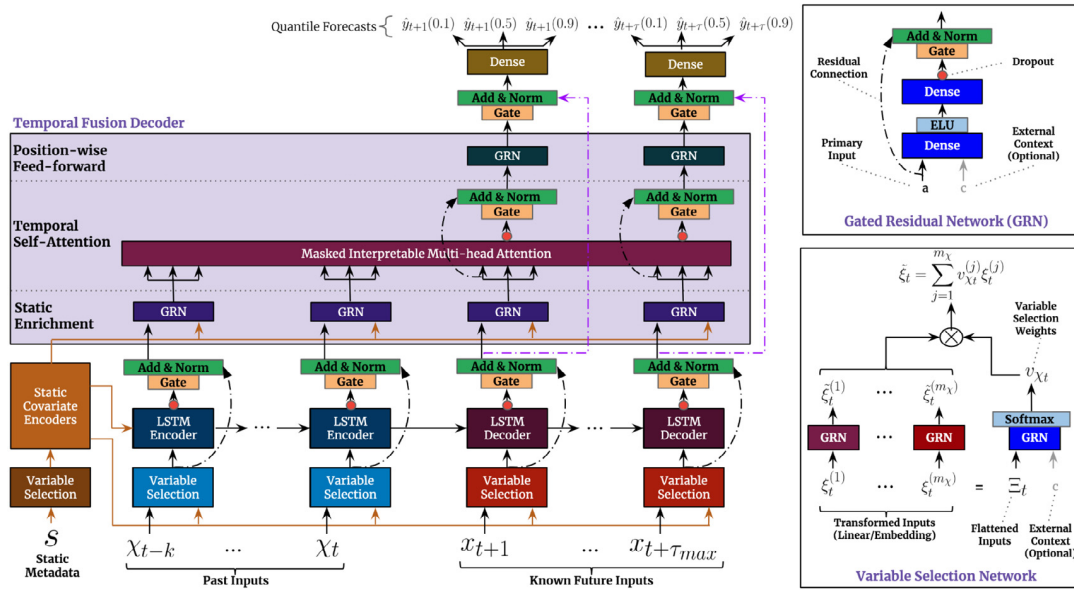


Figure 4.1: TFT architecture, reproduced from [39] (Fig. 2).

Gated Residual Networks

Throughout the architecture, Gated Residual Networks (GRNs) play a structural role. A GRN, represented in the top-right box of the figure, provides a controlled way to introduce non-linearity while keeping optimisation stable. Functionally, it

interpolates between a simple linear transformation and a shallow MLP, depending on how strongly the non-linear path is activated. This flexibility allows the model to express richer interactions when useful, but also to fall back to a near linear mapping when the data is not complex enough to justify deeper transformations. The residual connection and normalisation inside the GRN ensure that information and gradients flow reliably, even when multiple such blocks are stacked.

Static Covariate Encoders

Static attributes such as category, location or demographic indicators are handled through dedicated static encoders. These networks transform the static inputs into a collection of context vectors using separate GRNs, which serve as global conditioning signals throughout the architecture. They are used to initialise the hidden and cell states of both encoder and decoder LSTMs, to inform variable selection for non-static variables, and later enrich the temporal representations. Conceptually, this converts the LSTM from a model with a generic initial state into one whose dynamics are explicitly tailored to the entity being forecast.

This design makes TFT naturally suited for training on multiple related time series (for example, it could be trained jointly on series F1 to F5). Indeed, in the original paper by Lim et al. [39], the model was evaluated on datasets containing from tens to several hundreds of individual series. Our use case differs, as we operate in a data-scarce setting. This is likely the main limitation when applying a complex and sophisticated architecture such as TFT: higher model capacity typically requires larger datasets to be fully exploited, even though the internal gating mechanisms partially mitigate this requirement.

Variable Selection Networks

A major extension of TFT compared with a vanilla encoder–decoder LSTM is the set of variable selection networks, shown in the bottom right box of the figure. Rather than presenting all input features to the LSTM in a raw concatenated form, each group of static, past and future variables is first embedded or linearly transformed into a common latent space. Each embedding is then transformed individually through a GRN. In parallel, the full set of raw embeddings, optionally combined with external context derived from static covariates, is processed by another GRN that produces a set of soft selection weights. These weights are applied to the GRN-transformed

variable embeddings to obtain a contextually weighted representation at every time step. In this way, variable selection becomes an explicit operation, allowing irrelevant or noisy features to be downweighted on an instance-wise basis. This reduces the burden on the LSTM and provides direct interpretability of variable-importance patterns.

The Encoder–Decoder LSTM Core

After variable selection, the selected temporal features are processed by the encoder–decoder LSTM block. Within TFT, this recurrent component serves mainly as a local pattern extractor, capturing short- to medium-range temporal dependencies and incorporating ordering information implicitly, without the need for positional encodings. Since its initial state is conditioned on static context vectors, the LSTM learns entity-specific dynamics without requiring large amounts of data to infer these patterns from scratch.

The outputs of the LSTM are then combined with static context in a dedicated static enrichment layer. This step enables the model to reinterpret local temporal patterns in light of global entity information, maintaining the influence of static covariates even at deeper stages of the network.

Interpretable Multi-Head Attention

To model long-range temporal dependencies, TFT introduces an interpretable multi-head self-attention layer applied to the enriched temporal features.

The self-attention mechanism in the TFT is applied along the temporal axis. This enables the model to directly connect distant time steps within the input window, allowing it to capture long-range dependencies, complex lag structures, and periodic patterns that would be difficult for an LSTM to retain solely through its recurrent hidden state (a limitation also observed in Section 3.5.1).

In the *interpretable* multi-head attention mechanism introduced in the TFT, the key architectural modification consists of sharing the value projection across all attention heads. Unlike standard multi-head attention [59], where each head employs its own independent value projection, TFT uses a single shared set of value representations for every head. Although the attention weights are computed independently for each head (via head-specific query and key projections), all heads attend to the same underlying value vectors. Consequently, the attention weights are

averaged across heads, yielding a single attention distribution over past time steps for each lead hour in the forecast horizon.

Accordingly, at each cut-off time T , and for each forecast horizon step $\tau \in \{T + 1, \dots, T + h\}$, the model produces a single attention weight for each past time step j in the processed window:

$$\alpha(T, j, \tau), \quad j \leq \tau$$

These weights quantify the relative importance of each past time step j when producing the forecast for time τ . For instance, $\alpha(T, j, T + 1)$, with $j \leq T + 1$, identifies the historical time points that are most influential for one-step-ahead prediction. By aggregating these weights across different cut-off times T , one may analyse systematic temporal patterns in the model’s reliance on past observations, as demonstrated in Section 7 of Lim et al. [39].

Following this attention mechanism, further gating and residual connections regulate the contribution of the attention layer. The final fused representation is then mapped to quantile estimates for each forecasting horizon, allowing the model to produce multi-horizon probabilistic predictions without assuming a specific error distribution. While the TFT paper trains its predictive quantiles using the standard quantile-regression loss, our focus is solely on point forecasts. We therefore employ an L1 loss, which is consistent with the training objective used for the other models.

Final Considerations

Overall, the TFT can be interpreted as a carefully engineered decomposition of the forecasting problem. Variable selection modules filter and weight the relevant inputs, static encoders condition the temporal dynamics, the LSTM component captures short-term sequential structure, and the attention mechanism models longer-range dependencies. Gating layers regulate the contribution of each module, ensuring that the model adapts its complexity to the data. This design explicitly addresses several limitations of a plain encoder–decoder LSTM while retaining its strengths, resulting in a model that is both expressive and more interpretable than typical deep forecasting architectures.

For these reasons, TFT represents a natural benchmark for comparison with our LSTM-based pipeline. It should be noted, however, that the current experimental

setup, where models are trained independently for each time series, is not ideal for a global architecture such as TFT, whose parameterisation and representation power are intended to benefit from multi-series training. This limitation is not unique to TFT; it is a general characteristic of complex, highly parameterised neural forecasting models.

4.1.2 Hyperparameter Tuning

Tuning Strategy

Hyperparameter tuning for the TFT model followed the same overall approach used for the LSTM pipeline. Tuning was performed on a per-series basis for both day-ahead and week-ahead forecasting. The main search phase was conducted on the week-ahead task. Once a satisfactory configuration was identified, only the learning rate, dropout, and number of training epochs were adjusted for the day-ahead task, while all architectural choices were kept fixed.

Because of the considerable computational cost of the TFT and the relatively limited performance gains observed (see Chapter 5), tuning was ultimately restricted to series F1.

Main Tuning Phase

To keep the search tractable, we fixed several modelling choices in advance and focused on a reduced hyperparameter space. These fixed components arose from different motivations.

Some were guided directly by insights from the LSTM study, such as constraining the model to a single LSTM layer, which had generally proved preferable to deeper recurrent stacks. Other settings were chosen to ensure consistency across models. In particular, temperature was retained as the sole exogenous climate-related feature, and the same calendar features used for the LSTM were adopted for the TFT: hour-of-day, day-of-week, month-of-year, and weekday-Saturday-Sunday separation with cyclical encoding.

Additionally, following the design rationale of the TFT, we did not include lags of either the target or temperature, since the self-attention mechanism is expected to capture long-range temporal dependencies without such explicit inputs. Instead,

the lookback window was set to 10 days to provide a historical context comparable to that of the LSTM encoder.

A final group of parameters was fixed primarily for computational reasons. Normalisation used min-max rescaling in the interval $[0, 1]$. Although z-score normalisation (used for the LSTM) was briefly tested, it showed comparable effects, so the default min-max option was adopted. Gradient clipping with maximum norm 10 was applied throughout to prevent excessively large gradients from destabilising training.

The hyperparameters effectively explored during tuning are listed in Table 4.1. As in the final LSTM search, optimisation was performed using Optuna’s TPE sampler with `multivariate=True` and a warm-up phase increased to 25% of trials to favour exploration. A mild Percentile Pruner and early stopping were also used. Each trial trained the full pipeline on all data up to (but excluding) the cold semester of 2023/2024, and validation was carried out on that semester. The objective value corresponded to the mean validation MAE computed over a small window centered on the best epoch.

Table 4.1: Search space for the TFT model.

Hyperparameter	Type	Values / Range
hidden_size_idx	Integer	$[0, 4]$
hidden_size	Derived	$\{20, 40, 80, 120, 160\}$ (mapped from hidden_size_idx)
dropout	Float	$[0.0, 0.6]$
num_attention_heads	Categorical	$\{1, 4\}$
batch_size	Categorical	$\{64, 128\}$
learning_rate	Float (log)	$[1 \times 10^{-4}, 1 \times 10^{-2}]$

Training Results

The TFT exhibited a strong robustness to architectural variations. fANOVA analysis attributed less than 10% of the total importance to `num_attention_heads` and `hidden_size_idx`, while approximately 75% was assigned to `dropout` and about 15% to `learning_rate`. Examination of one-dimensional bins revealed a clear degradation in performance for high dropout values (above 0.5) and for very low learning rates (below 3×10^{-4}). These trends persisted when inspecting two-dimensional

bins. Such behaviour is expected: excessively high dropout reduces the effective capacity of the model and slows convergence, while very low learning rates hinder the optimiser from escaping shallow minima, preventing the model from reaching well-generalising configurations.

Overall, the tuning procedure indicates that the TFT is comparatively robust to architectural hyperparameters and is governed primarily by optimisation-related ones, likely due to the extensive use of gating mechanisms. The final configurations obtained through this process were used in the testing phase described in Chapter 5.

4.2 Chronos-2 Pipeline

In this section, we provide a concise overview of the foundation model Chronos-2 [1] and describe how it was employed as a *zero-shot* benchmark in our study, that is, without any task-specific training or fine-tuning on our dataset. All experiments were conducted using the official `chronos-forecasting` implementation provided by the authors. This setup allows us to assess to what extent a large pre-trained model can exploit exogenous information in a purely inference-based setting.

We further evaluate Chronos-2 in its *full cross-learning* mode, in which information sharing across sites is enabled, and compare it with a per-site deployment where cross-site information sharing is disabled.

4.2.1 Rationale for Including this Benchmark

Unlike the previously considered models, Chronos-2 is designed as a general-purpose forecaster that can be applied in a zero-shot setting. This makes it conceptually different from the other models considered in this thesis. In particular, models such as LSTM and TFT are trained to adapt their parameters to the specific forecasting task and dataset, exploiting plant- and dataset-specific information, whereas Chronos-2 aims to provide strong forecasts “as is” across diverse, previously unseen forecasting problems. From an operational perspective, models of this type are attractive because they can be deployed with minimal adaptation effort, substantially reducing the modelling and maintenance overhead.

The main motivation for including Chronos-2, relative to earlier foundation models, is its reported strength on covariate-informed forecasting tasks [1]. This is closely aligned with our use case, where weather-based exogenous drivers play a central

role in achieving accurate heat-demand forecasts. The Chronos-2 technical report presents results on multiple comprehensive evaluation suites; notably, it reports leading performance on *fev-bench*, which places particular emphasis on multivariate and covariate-informed tasks. The report further indicates that the largest performance margins arise on tasks involving covariates, and it includes energy-domain case studies where incorporating dynamic covariates yields substantial improvements over a univariate deployment. Taken together, these findings motivate its inclusion as a zero-shot benchmark in a setting where exogenous information is known to be critical.

The main limitation of including this model is that, because Chronos-2 is very recent, its pretraining corpus overlaps temporally with our evaluation period. If any pretraining series were sufficiently related to our target plants, this could in principle confer an unfair advantage, as the model might implicitly encode information that is “from the future” relative to the cut-offs defining our test windows (a form of leakage). We discuss this concern, together with our evaluation schedule, in Chapter 5. To investigate the risk, we performed a data audit of the Chronos-2 pretraining sources (Appendix B), which suggests that such leakage is unlikely in our setting, although it cannot be excluded with certainty.

4.2.2 Chronos-2 and its In-Context Learning Mechanism

At a high level, the Chronos-2 model pipeline includes input normalisation, a patch-based tokenisation and embedding of the input sequence, a transformer stack, and a quantile forecasting head that outputs a predictive distribution over the horizon. In this section, we focus on the transformer stack, since it is the main component enabling Chronos-2 to incorporate multivariate information and covariates at inference time.

A central concept in Chronos-2 is that of a *group*. The model formalises relatedness through an integer group identifier assigned to each input series (including both targets and covariate streams). Series that share the same group identifier are processed jointly and are permitted to exchange information during inference.

Depending on the application, a group may correspond to a single series (univariate inference), the set of variates of a multivariate series (multivariate forecasting), a target series together with its associated covariates (covariate-informed forecasting), or, more generally, any collection of related series intended to share information

within the model. In our context, the natural grouping places each plant’s target together with its weather and calendar covariates in the same group. The full cross-learning configuration instead allows information sharing across multiple plants by assigning them a common group identifier.

Within the transformer stack, Chronos-2 alternates between two attention operations.

Time attention. The first is *time attention*, which is standard self-attention applied along the temporal axis. After patching, each variate is represented as a sequence of patch embeddings, and time attention allows each patch to attend to other patches of the same variate.

Group attention. The second mechanism is *group attention*, which constitutes the distinctive architectural feature of Chronos-2. At a fixed patch index, group attention performs attention across all series that share the same group identifier. In other words, attention is computed across parallel series within a group, rather than across time within a single series. This enables information sharing across multiple variates without concatenating them into a single long sequence, thereby supporting multivariate and covariate-informed tasks within a unified transformer architecture.

Intuitively, the alternation between time attention and group attention allows the model to combine temporal information extracted within each input dimension with information coming from other dimensions in the same group.

4.2.3 Zero-shot Configuration

Although the Chronos-2 implementation in `chronos-forecasting` supports fine-tuning, we use the model exclusively in zero-shot mode. We nevertheless performed a small configuration study to determine a default specification for the main experiments. Specifically, we compared:

- using temperature as the sole exogenous driver versus using all five meteorological covariates (temperature, dew point, wind speed, humidity, and air pressure), and

- per-site information sharing (i.e. groups defined at the site level) versus the full cross-learning setting, in which all series across sites are treated as belonging to a single shared group.

In all cases, we enriched the input with calendar-based features (the same ones used for the other deep learning models). We evaluated the resulting four configurations using rolling cross-validation on the extended winter 2023/2024 period.

Overall, the performance differences were small. However, across all series and for both forecast horizons, the full cross-learning configuration consistently resulted in a slight degradation in accuracy.

Using all covariates rather than temperature alone did not provide a clear or consistent benefit. While it occasionally improved average accuracy (the largest gain was observed for F1, at approximately 0.5 MAE points for day-ahead forecasts), in other cases it slightly worsened performance (e.g. for F5).

Given the absence of a clear and robust advantage, we opted for the more parsimonious specification using temperature as the sole exogenous driver. Beyond empirical performance, this choice also reduces practical dependencies, as incorporating additional covariates would require reliable forecasts for all meteorological variables. Furthermore, including all covariates slightly increases inference time (from approximately 6 to 8 seconds on CPU).

Chapter 5

Comparative Assessment of Forecasting Models

This chapter provides a comparative evaluation of the forecasting models developed in the previous sections, tuned jointly by forecast horizon and series, alongside the company models and the additional off-the-shelf, general-purpose benchmark Chronos-2.

Models are evaluated using a rolling-origin cross-validation procedure on the final year of data, which is held-out and used exclusively for this final evaluation. This set-up mimics deployment conditions by reproducing the operational workflow, with models retrained weekly and forecasts generated using only the data available at each origin. The analysis spans three seasons, namely autumn, winter, and spring, capturing the main phases of the heat-demand cycle: the mid-autumn ramp-up, the winter peak, and the spring decline. Comparisons focus on three aspects: forecast accuracy (across noise levels and both forecast horizons), computational cost, and residual behaviour.

Computational Environment. All tests for the models implemented by the company (XGBoost and SVMR) were conducted on a laptop equipped with an Intel Core i7-13620H processor of the thirteenth generation (10 cores, 16 threads, 2.40 GHz base frequency) and 32 GB of RAM, running Windows 11 Pro. All other models were trained and tested on a virtual machine equipped with an Intel Xeon CPU @ 2.00 GHz (2 cores, 4 threads), an NVIDIA Tesla T4 GPU with 16 GB of VRAM, and a Debian-based Linux system (kernel 5.10) running CUDA 12.4.

5.1 Company models

The analysis includes not only the models developed in this project but also those utilised in OptitEPTM's current forecasting module. The main operational model is an XGBoost model [13], complemented by a second model based on Support Vector Machine Regression (SVMR) [58].

XGBoost is an optimised gradient boosting framework that constructs an additive ensemble of regression trees in sequence, with each tree correcting the residual errors of the previous ones.

SVMR is a kernel-based regression method that estimates the function best fitting the data by maximising the margin around the regression hyperplane. Through kernel transformations, it can capture non-linear relationships while maintaining good generalisation.

The SVMR approach adopted by the company is implemented as an ensemble, with different models specialised for specific seasons or characteristic phases of heat demand. This strategy allows the system to adapt to regime changes without relying on a single global model.

A key distinction between these methods and the recurrent neural architectures used in this work lies in how temporal information is represented and learned. Recurrent neural networks process observations in their natural temporal order and maintain internal states that evolve as new data arrive, enabling them to model sequential dependencies such as multi-day cycles, gradual drifts and long-range interactions. Tree-based and kernel methods, by contrast, do not maintain temporal memory. They operate on fixed-length feature vectors, so any temporal structure must be encoded explicitly through feature engineering, for example with lagged values, rolling summaries or calendar variables. Such features provide snapshots of the recent past, but the model does not learn the dynamics linking successive time steps.

Although implementation details are proprietary, both company models rely on engineered features derived from the history of heat demand together with past and future temperature values.

5.2 Forecast Generation for Testing

The evaluation of all forecasting models is carried out separately for each target series using a rolling cross-validation procedure over one full year of data, from May 2024 to May 2025. This setup emulates a realistic operational scenario in which models are periodically retrained as new observations become available and forecasts are issued on a moving temporal window.

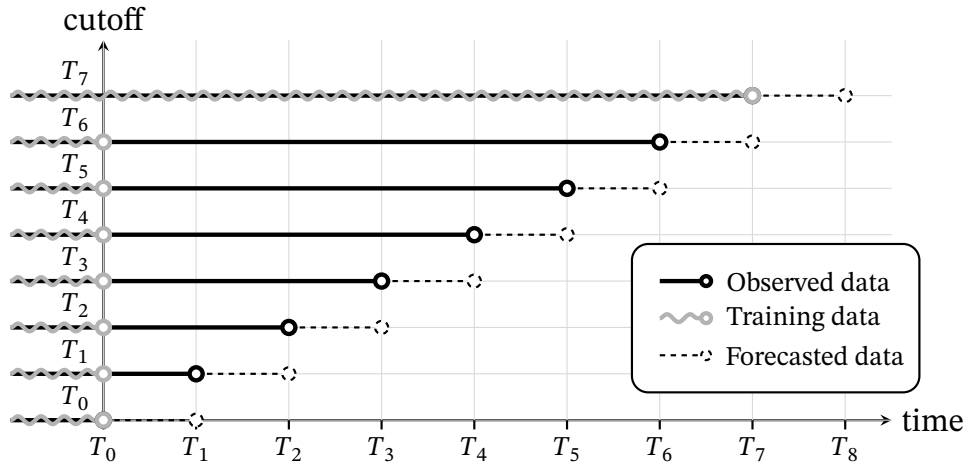
The number of forecast windows depends on the prediction horizon. Successive windows are adjacent, so that each model produces one forecasted value at every timestamp. Put differently, the step between successive cut-offs (that is, the datetimes marking the inclusive end of the available contextual data) is equal to the forecast horizon.

Each model is retrained once every seven days, matching the company’s usual deployment practice. For the day-ahead horizon, the most recently retrained model is reused on the intermediate days within the same week. This design balances computational efficiency with adaptability to evolving data. This retraining schedule does not apply to Chronos-2, which we use off the shelf with fixed weights, that is, without any additional training or fine-tuning in our pipeline.

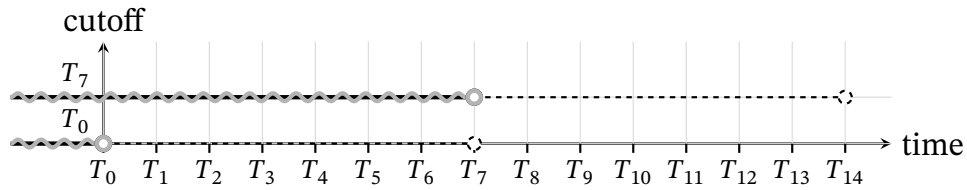
This forecast generation process is illustrated in Figure 5.1 for a representative week of data, from the cut-off at time T_0 to the cut-off one week later, T_7 .

Implementation details. The first cut-off is set at 23:00 on Sunday, 2024-05-19. Models are trained on all data up to and including this cut-off, or, where indicated by the tuning results, using a shorter model-specific estimation window (for example, the year preceding the cut-off). Subsequent steps depend on the forecast horizon:

- *Daily horizon.* The model generates forecasts for the next day. The following cut-off is at 23:00 on Monday, 2024-05-20, when the model is not refit but simply incorporates the newly available data as additional context. Cut-offs then advance every 24 hours, with retraining occurring each Sunday at 23:00.
- *Weekly horizon.* The model generates forecasts for the subsequent week. The next cut-off is at 23:00 on Sunday, 2024-05-26, when the model is retrained and used to forecast the following week. Cut-offs then advance every seven days, with retraining each Sunday at 23:00.



(a) Day-ahead



(b) Week-ahead

Figure 5.1: Rolling cross-validation forecast generation with adjacent windows, illustrated over a representative week from cut-off T_0 to $T_7 = T_0 + 1$ week, where $T_i = T_0 + i$ days. Solid segments denote observed data available up to each cut-off, wavy segments indicate the training history used when a model is retrained, and dotted segments show the issued forecasts. (a) Day-ahead: cut-offs advance daily and the same model is reused between weekly retraining points. (b) Week-ahead: cut-offs advance weekly.

5.3 Notes on Comparability and Threats to Validity

Although all models are evaluated under the same rolling-origin protocol described above, a few methodological differences can make the comparison appear more clear-cut than it truly is. This section summarises the main nuances that should be kept in mind when interpreting the results.

End-to-end optimisation differs across model families. The deep learning models LSTM and TFT are trained end-to-end for the day-ahead and week-ahead forecasting task, that is, by optimising a loss computed over the full forecast window. By contrast, the company baselines (XGBoost and SVMR), as implemented in the current pipeline and in their standard single-step formulation, are not configured to optimise a loss over the entire horizon. Instead, they are trained to minimise a loss for hour-ahead predictions and are subsequently applied recursively to generate the required multi-horizon forecasts. This recursive strategy is known to be susceptible to error accumulation across the forecast window [5], which can disadvantage such models in long-horizon evaluations.

Importantly, this is not a fundamental limitation of tree-based or kernel methods. Using an ensemble of single-step models, one can construct multi-horizon forecasting strategies, for example the Direct strategy [5], where a different model is fitted for each lead time (e.g., each hour in the forecast window), thus avoiding purely recursive roll-outs. Moreover, recent work has proposed extensions of gradient-boosted decision trees to genuinely multi-output settings, where a single boosted model predicts multiple targets jointly [66, 45]. Consequently, results for XGBoost (and SVMR) should be interpreted with the caveat that these baselines could be strengthened by tailoring them directly to multi-hour horizons. However, such alternative formulations are not guaranteed to outperform the purely recursive strategy used here, and they can substantially increase computational cost, for instance by requiring one model per lead time in the Direct strategy.

The statistical models constitute a further case: they are estimated by maximum likelihood rather than by directly minimising a forecast error metric. While this implies a mismatch between the training objective and the evaluation metrics reported in this chapter, maximum-likelihood estimation is the standard and principled fitting procedure for these model classes.

Finally, Chronos-2's pretraining objective is explicitly multi-horizon: it min-

imises a quantile regression loss averaged over all forecast steps, with the forecast length (number of output patches) randomly sampled during training, and with an explicit post-training stage that increases the maximum supported context length and horizon [1]. Consequently, for any chosen horizon it produces the full set of multi-step outputs simultaneously in one forward pass, rather than via recursive roll-outs. In the released Chronos-2 checkpoint, this corresponds to supporting prediction lengths up to 1024 time steps (and context lengths up to 8192)¹, which is much longer than the 168 steps needed for week-ahead forecasts.

Potential overlap between Chronos-2 pretraining data and the test period.

A second important note concerns the pretraining data of Chronos-2. Since Chronos-2 was released in October 2025, it is possible that its pretraining corpus overlaps with our evaluation period (summer 2024 to summer 2025), which could introduce an unfair advantage if the model had been exposed to information correlated with the target series. To assess this risk, we inspected the training data description in Ansari et al. [1]. The authors provide a complete list of the real univariate datasets used during pretraining (Table 6 in their Appendix A). For each dataset, we checked whether it is a fixed historical benchmark compiled before 2024 or a resource that is continuously updated and may include observations from 2024 onwards. When recent updates were possible, we further inspected the dataset content to evaluate whether it could plausibly act as a proxy for our targets, including by checking for overlap in entities, geography, or other identifiers that might induce spurious correlations. The results of this analysis are reported in Appendix B and support the conclusion that, in practice, the risk of leakage should be marginal. Nevertheless, the possibility of some degree of leakage cannot be entirely excluded. From an operational perspective, this concern can be addressed by re-running the evaluation on a more recent holdout period (e.g., winter 2025/2026), which removes any potential overlap with Chronos-2 pretraining data.

CPU runtimes are not directly comparable across the two environments. In particular, XGBoost can utilise multi-threaded CPU execution, so its training and inference times can scale materially with the number of available CPU threads. Since the laptop provides 10 cores/16 threads whereas the virtual machine exposes only 2 cores/4 threads, measured CPU times may differ substantially between setups even with identical data and hyperparameters. As a result, the reported training-time gap

¹<https://huggingface.co/amazon/chronos-2>

Table 5.1: Training and inference times for all models on series F1 for day-ahead forecasts. Each value reports the mean with the standard deviation in parentheses, computed over 8 runs after 2 warm-up runs, except for the TFT on the CPU, where only 3 post-warm-up runs were used to reduce computational cost. All models were trained on data up to, but not including, 2025-01-01, with in-sample lengths determined individually according to the tuning results.

Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
Naive24h	– (–)	– (–)	8.34 (0.53)	– (–)
XGBoost	< 1 (–)	– (–)	– (–)	– (–)
SVMR	< 1 (–)	– (–)	– (–)	– (–)
MSTL-ETS	8.79 (0.09)	– (–)	25.11 (1.64)	– (–)
SARIMAX	48.65 (0.52)	– (–)	180.56 (35.56)	– (–)
LSTM	264.20 (14.77)	61.90 (0.75)	5.26 (0.28)	3.62 (0.67)
TFT	1679.16 (6.25)	761.74 (2.88)	64.40 (2.24)	87.87 (3.24)
Chronos-2	– (–)	– (–)	6288.34 (330.17)	487.52 (56.36)

between the two company models (XGBoost and SVMR) and the remaining models may appear substantially larger than it would under a shared hardware environment.

5.4 Series F1: Day-Ahead Forecast Evaluation

We begin by analysing the results for the day-ahead forecasts for series F1.

5.4.1 Computational Cost

Alongside predictive accuracy, the computational cost of training and evaluating each model is an important practical consideration. Table 5.1 summarises the training and inference times measured in our setting.

A substantial disparity is observed across models. The classical machine learning methods (XGBoost and SVMR) train extremely quickly, even relative to the statistical approaches (MSTL-ETS and SARIMAX). As expected, the deep learning architectures constitute the most computationally intensive group. The LSTM takes about 4 minutes 30 seconds on the CPU and roughly 1 minute on the GPU, whereas the TFT is considerably more demanding, requiring almost 30 minutes on the CPU and around 12 minutes 30 seconds on the GPU. Training times for Chronos-2 are

not reported because the model is used zero-shot, hence no training is required to produce forecasts.

It is worth noting that, in many operational environments (including the one considered here), model fitting is performed only once per week for each series, so the incremental training cost (for example, fitting an LSTM compared with XGBoost) is incurred relatively infrequently. In our setting, however, training is executed independently for each series and each forecast horizon. At larger scales, this can compound into several hours of additional computation per retraining cycle.

The table also reports inference times. The slightly higher latency of Naive24h compared with the LSTM is likely attributable to implementation overhead in Statsforecast’s `SeasonalNaive`, which copies the previous day’s observations but still incurs non-trivial execution overhead. While Chronos-2 has no training time, it exhibits the highest inference latency, reflecting the cost of large-model forward passes during prediction (particularly on CPU).

For brevity, computational times for the other series are not reported in the main text; the complete results are provided in Appendix A. Although the absolute values vary across tasks, the relative ordering of the models remains highly consistent. For the deep learning approaches, training times are generally lower, occasionally reaching as little as half of the time reported here, since both the number of epochs and the network capacity are typically reduced for the later, and often noisier, series, with reduced capacity in particular acting as an implicit form of regularisation.

5.4.2 Per-Window Errors

This section examines how forecasting accuracy varies over individual day-ahead evaluation windows. Table 5.2 reports summary statistics for each season, presenting mean and standard deviation for the MAE, RMSE and sMAPE metrics (introduced in Section 1.6), together with the *win rate* (denoted “wrate”), defined as the percentage of evaluation windows for which a model attains the lowest error among all models. The tables are organised by ascending mean MAE to facilitate comparison. Figure 5.2 complements these results by displaying, for each cut-off, how much the MAE of the competing models differs from that of the LSTM benchmark. Positive values indicate poorer performance relative to the LSTM. Together, these results provide a detailed picture of how the models behave across seasons.

Table 5.2: Per-window error statistics for the *day-ahead* forecast horizon on *F1*. Each table is ordered by ascending mean MAE.

(a) Autumn (92 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	16.23 (9.21)	26	20.54 (11.78)	34	8.55 (6.78)	28
Chronos-2	18.24 (12.09)	22	23.11 (14.94)	15	8.60 (6.05)	21
XGBoost	18.55 (9.58)	10	23.66 (11.98)	10	9.98 (6.54)	8
SARIMAX	20.11 (11.81)	12	25.19 (14.36)	12	10.07 (7.32)	14
TFT	22.36 (12.73)	16	27.83 (15.63)	15	11.76 (8.90)	16
SVMR	23.75 (13.67)	3	29.52 (16.81)	3	14.43 (13.17)	2
Naive24h	26.19 (21.54)	5	33.16 (26.20)	3	11.25 (7.22)	4
MSTL-ETS	29.87 (20.22)	5	37.63 (24.27)	8	13.24 (8.14)	7

(b) Winter (90 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	20.22 (7.35)	37	25.59 (9.66)	39	4.57 (1.44)	38
Chronos-2	22.58 (8.52)	26	28.09 (10.83)	27	5.11 (1.85)	27
SARIMAX	24.82 (10.26)	16	31.09 (12.84)	11	5.55 (1.97)	16
XGBoost	25.65 (9.60)	10	32.19 (11.81)	12	5.89 (2.20)	8
TFT	26.44 (10.04)	3	32.21 (11.34)	4	5.97 (1.88)	3
SVMR	28.37 (8.73)	3	35.69 (11.37)	3	6.54 (2.02)	4
Naive24h	40.87 (22.20)	4	49.94 (24.89)	2	9.29 (4.73)	4
MSTL-ETS	45.29 (18.76)	0	58.43 (23.60)	1	9.93 (4.06)	0

(c) Spring (59 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	8.07 (5.54)	37	10.49 (7.19)	32	5.80 (1.91)	41
Chronos-2	10.20 (7.32)	14	12.90 (9.34)	22	7.25 (2.64)	14
TFT	10.71 (6.47)	14	13.93 (8.92)	10	8.52 (4.08)	10
XGBoost	10.97 (7.47)	14	14.15 (9.45)	10	7.62 (2.40)	14
SVMR	14.63 (7.14)	3	17.98 (8.67)	5	13.13 (6.68)	2
SARIMAX	15.11 (10.11)	14	18.44 (12.24)	15	10.51 (5.15)	14
Naive24h	16.49 (15.16)	5	20.66 (18.70)	5	10.06 (4.90)	7
MSTL-ETS	20.60 (15.81)	0	27.14 (19.63)	0	13.72 (6.00)	0

Performance Evaluation: LSTM

The LSTM model is consistently more accurate on average than the others across all seasons and across all evaluated metrics. In winter, the most operationally critical period, LSTM achieves the best performance in roughly 40% of the 90 forecast windows, compared with close to 30% for Chronos-2, while every other benchmark remains below 16% (Table 5.2). This ordering is reinforced by pairwise comparisons: LSTM outperforms Chronos-2 in MAE on 63% of windows and in sMAPE on 60%, and it exceeds SARIMAX on 74% of windows in MAE and 72% in sMAPE, with comparable margins against XGBoost. The reported standard deviations also tend to be slightly lower for LSTM, suggesting more stable performance over time and reduced sensitivity to particular forecast windows. Taken as a whole, the evidence indicates that LSTM surpasses the other models in accuracy on series F1. The statistical significance of this difference is assessed in Section 5.4.3.

Although the relative improvement of LSTM over the company model XGBoost may appear substantial, the effect becomes more modest when viewed in relation to the scale of the series. For instance, the LSTM's mean winter MAE is roughly 20% lower than XGBoost's; however, expressed relative to the series magnitude (as reflected by sMAPE), the gain is only about 1% of the series level. Put differently, the LSTM's forecasts are on average about 1% closer to the true series at each timestamp than those produced by XGBoost. Whether this improvement is worthwhile, especially given the longer training time required by the LSTM, depends on the accuracy requirements of the specific application.

Performance Evaluation: SARIMAX

SARIMAX performs well during winter, where it ranks third only after LSTM and Chronos-2, the two consistently best models. However, its accuracy deteriorates during shoulder seasons, especially in spring.

For SARIMAX, this pattern is expected. The model divides the year into two regimes, warm and cold, based on a rolling average of temperature, and it stabilises the variance of the cold period through a targeted Box-Cox transformation. This coarse regime specification can create abrupt changes in behaviour between the two states. When the true transition in demand is gradual or occurs later than the temperature transition, for example due to social or operational factors, the model can become temporally misaligned with the actual dynamics of heating demand.

Moreover, the regression coefficients associated with the cold period, which include weekly Fourier terms and temperature related regressors for that regime, are mainly estimated to fit the dominant peak season. As a consequence, they are not necessarily well adapted to the shorter transitional intervals.

These features are not accidental. The design was chosen with the aim of prioritising accuracy during the high demand and operationally critical winter period. Approaches that attempt to improve performance during the transitions might do so at the cost of accuracy in this main regime. For this reason, the observed weakness in the transitional periods reflects not only the structure of the model but also a deliberate trade-off that favours stable and reliable winter forecasts.

Nonetheless, this highlights a limitation of SARIMAX, and of many classical linear time-series models, in that they typically assume the data can be rendered (approximately) stationary by a small set of simple transformations (for example differencing) and that the governing relationships (lag dynamics and covariate effects) remain stable over time. Introducing temperature as an exogenous regressor and applying a Box–Cox transformation can help with level shifts and variance stabilisation, but they do not fully address structural breaks, regime changes, or time-varying effects. By contrast, models such as XGBoost or deep neural networks are far more expressive and can learn nonlinear and interacting relationships between calendar variables, temperature, and the target without those relationships being explicitly specified. This additional flexibility can lead to better performance during transition periods.

Performance Evaluation: MSTL-ETS

MSTL-ETS exhibits weak performance, even when compared with the simple Naive24h baseline. This behaviour can be traced to structural aspects of both the target series and the model itself. The main contributing factors are outlined below.

1. In winter, when heat demand is high and relatively stable, most day-to-day variation is driven by short-term temperature changes. The Naive24h model adjusts immediately to such changes by repeating the most recent daily pattern. In contrast, MSTL-ETS forecasts the next day by extrapolating seasonal and trend components estimated from several preceding days. This reduces its ability to respond to rapid fluctuations and generally results in higher forecast errors.

2. MSTL-ETS is retrained only once per week, reusing the same estimated seasonal patterns and ETS-based trend forecaster throughout the entire week. This restricts its ability to adapt to evolving conditions. While such a retraining strategy may be acceptable for more complex models, it represents a major limitation for a simple decomposition-based approach.
3. Reliable estimation of daily and weekly seasonal components is challenging because these patterns drift over time and are easily contaminated by irregular variation. When the decomposition misattributes noise or one-off events to the seasonal structure, MSTL-ETS can extrapolate an incorrect component into the forecast. This effectively reinforces spurious patterns and amplifies noise rather than smoothing it. In these cases, the simple Naive24h strategy of repeating the most recent observed profile can be more robust, because it avoids imposing potentially misestimated structure on the next day's trajectory.

Figure 5.3 illustrates the first two points discussed above. In this example, the model fails to anticipate an overall increase in heat demand, which is likely linked to a decrease in temperature. It adjusts to the higher morning peaks only in the forecasts for the following week, yet by that time both demand and temperature have already returned to their previous levels.

Overall, these results indicate that the decomposition based framework of MSTL-ETS, which relies on smoothly evolving seasonal components, is not well suited to short term heat demand forecasting, where rapid adaptation to small but frequent fluctuations driven by exogenous factors is essential. MSTL-ETS can still offer a meaningful advantage over the naïve baseline when the series displays strong weekly seasonality, but it generally provides little or no improvement in other situations.

Performance Evaluation: TFT

The LSTM consistently outperforms the TFT, even though the TFT extends an LSTM-based forecasting backbone with attention, gating mechanisms and variable selection networks. In our setting, this suggests that greater architectural expressiveness does not necessarily translate into better generalisation. Before extensive tuning, the TFT produced noticeably smoother forecasts than the other models, consistent with an overly conservative fit that under-represented short-term variability. After tuning, the TFT's predictions became much closer in shape to those of the LSTM,

and the performance gap narrowed to about 2 sMAPE percentage points. Taken together, these results imply that the TFT is more sensitive to hyperparameter choices and regularisation than the simpler LSTM: without careful calibration it can settle into a high-bias, over-smoothed regime, whereas the LSTM captures the dominant temporal dynamics more reliably under default settings. In this setting, the TFT's additional capacity therefore translates primarily into a higher tuning burden rather than a clear accuracy gain.

A further consideration is that TFT is typically used as a global model trained on many related time series. This is the setting adopted in the original paper in which the architecture was introduced [40]. Training on multiple related series allows the model to exploit shared temporal patterns and to make effective use of its large capacity. When only a single series, or a very small number of series, is available, the model cannot benefit from this structure. In this regime, the larger parameter set is harder to estimate reliably, increasing sensitivity to hyperparameters and reducing the likelihood of a clear accuracy gain.

5.4.3 Bootstrap Assessment of Performance Improvement

Care is required when interpreting statistical significance. The standard deviations reported in Table 5.2 should not be viewed as those of independent and identically distributed observations, since the per-window errors exhibit temporal dependence and non-stationary behaviour. Temporal dependence arises because consecutive evaluation windows are contiguous in time, so a model that performs poorly on one window often performs poorly on the next. This dependence may be further reinforced by the fact that models are re-estimated only once per week. In addition, the distribution of the per-window errors may vary substantially over time, for instance due to shifts in the level or variability of the target series, changes in the behaviour of the exogenous variables, or evolving relationships between them. These forms of non-stationarity and temporal dependence make simple i.i.d.-based approximations inappropriate. Consequently, the significance of the observed performance differences cannot be assessed reliably with a standard t -test and must instead be evaluated using inference methods that account for both temporal dependence and potential non-stationarity in the data.

To assess statistical significance, we employ the Circular Block Bootstrap (CBB) method [49]. Let $\{X_t\}_{t \in \mathbb{Z}}$ denote a *strictly stationary* process, that is, a process whose

finite-dimensional distributions are invariant to time shifts: for any $k \in \mathbb{N}$, any indices $t_1, \dots, t_k \in \mathbb{Z}$, and any shift $h \in \mathbb{Z}$, one has

$$(X_{t_1}, \dots, X_{t_k}) \stackrel{d}{=} (X_{t_1+h}, \dots, X_{t_k+h}).$$

We observe a sample $\{X_t\}_{t=1}^n$, which in our setting corresponds to the sequence of observed loss differences of a given model relative to the reference model LSTM for a specified metric. For $i = 1, \dots, n$, define the circular block of length ℓ by

$$B_{i,\ell} = (X_i^{(c)}, \dots, X_{i+\ell-1}^{(c)}), \quad X_t^{(c)} = X_{((t-1) \bmod n)+1}.$$

The CBB resample $\{X_t^*\}_{t=1}^n$ is constructed by drawing i.i.d. starting points I_k uniformly from $\{1, \dots, n\}$, for $k = 1, 2, \dots$, and concatenating the blocks $B_{I_k, \ell}$ until n observations are obtained. In our case, the statistic of interest is

$$T_n = \bar{X}_n = \frac{1}{n} \sum_{t=1}^n X_t.$$

By the bootstrap principle, its sample distribution is approximated by that of

$$T_n^* = \frac{1}{n} \sum_{t=1}^n X_t^*,$$

taken under the conditional probability

$$\mathbb{P}^*(\cdot) = \mathbb{P}(\cdot \mid X_1, \dots, X_n).$$

Since the exact conditional distribution of T_n^* is unavailable, it is common practice to approximate it using Monte Carlo replications

$$T_n^{*(b)}, \quad b = 1, \dots, B.$$

We used $B = 10\,000$ bootstrap replications. The associated 95% percentile confidence interval is

$$[T_n^*(0.025), T_n^*(0.975)],$$

where $T_n^*(q)$ denotes the empirical q quantile of the empirical distribution of T_n^* computed from the B resamples under the conditional measure $\mathbb{P}^*$.

To test $H_0 : \mu = 0$ for the marginal mean $\mu = \mathbb{E}[X_1]$ of the stationary process, we compute the two sided bootstrap p -value

$$\hat{p}_n = \frac{1}{B} \sum_{b=1}^B \mathbb{1}(|T_n^{*(b)} - T_n| \geq |T_n|),$$

where $T_n^{*(b)}$ denotes the b th CBB replicate of the sample mean.

Theoretical guarantees for the CBB, including the block length selection method used here, typically require strict stationarity of the underlying process [37, 50]. To align with these assumptions, we restrict attention to a central cold regime period in which the loss differences appear approximately stationary (see Figure 5.2) and do not exhibit structural changes. The selected period spans 2024-11-10 to 2025-03-17.

To further support the stationarity assumption beyond visual inspection, we applied two complementary tests: DF-GLS [21] (null hypothesis of a unit root, which intuitively means shocks have persistent effects so the series need not revert to a stable long-run mean) and KPSS [36] (null hypothesis of stationarity, here specified around a constant mean). These tests probe stationarity in the standard weak (covariance) sense and do not directly verify strict stationarity. All loss-differential series were broadly consistent with stationarity under these checks, with a small number of exceptions: XGBoost on F1 marginally failed for RMSE and sMAPE; and SARIMAX and SVMR on F3, as well as SVMR on F5, likely due to residual weekly seasonality in the loss differentials, which can both violate the constant-mean stationarity assumption and affect the reliability of DF-GLS and KPSS when seasonal components are not explicitly modelled.

The block length b is a key parameter in block bootstrap procedures. We select it using the Python function `optimal_block_length` from the package `recombinator`, which implements the data driven rule of Politis and White [50] with the correction of Patton et al. [48]. This chooses b by minimising the asymptotic mean squared error of the CBB estimator of the long run variance, with the unknown constants estimated via plug in. The resulting choice adapts automatically to the temporal dependence in the data.

Figure 5.4 reports the resulting intervals for all models and series. All confidence intervals lie well away from zero, indicating that the performance improvements are statistically meaningful. This is consistent with the fact that all two sided p -values are below 0.001, even after Holm–Bonferroni adjustment for multiple comparisons.

In our application, the data-driven block length selector consistently returned values of either 2 or 3. Robustness checks using fixed block lengths between 1 and 10 lead to the same conclusion, with all resulting p -values close to zero.

The CBB analysis therefore provides strong evidence that the LSTM achieves systematically lower predictive errors than the competing models. These findings are robust to the choice of block length and to multiple testing adjustments, indicating that the observed performance gains cannot be attributed to sampling variability alone.

It is important, however, to interpret these results alongside the variability observed in the per-window statistics (see Table 5.2). The standard deviations reported in the tables are often large relative to the corresponding differences in mean errors. This means that, when only a handful of daily windows are inspected, the natural day-to-day fluctuations can easily overshadow the underlying performance gap, making the models appear practically similar over short stretches. At the same time, the bootstrap analysis shows that the average loss differences are statistically significant, even when accounting for the temporal dependence present in the sequence of window-level errors. Taken together, these findings indicate that the advantage of the LSTM becomes reliably visible only once performance is aggregated across a sufficiently long run of daily windows, whereas over a small number of windows the short-term variability can make competing models appear nearly equivalent.

5.4.4 Hour-Ahead Horizon Errors

To examine whether model accuracy varies systematically across the day, forecasting errors were grouped by forecast horizon h . Each group collects all errors associated with timestamps that lie h hours ahead of their respective cut-off. Since forecasts are always initiated at midnight in our setup, a horizon h corresponds to hour $h - 1$ of the day, allowing the errors to be interpreted as grouped by hour of day.

Figure 5.5 summarises the results. Panel (a) displays the average hourly heat-demand profile for each month, with shaded bands representing the 80% empirical percentile interval. Panel (b) reports MAE and sMAPE values by forecast horizon for each model. The shaded regions show 95% bootstrap percentile intervals for the mean error, computed as described in Section 5.4.3. During winter, per-hour errors are approximately stationary, making the bootstrap procedure appropriate.

The MAE curves exhibit a clear diurnal pattern, with the largest errors occurring

during the morning demand peak. However, when normalising by the level of the series (as is approximately achieved by sMAPE), the profiles are comparatively stable across hours, with only a mild increase around 13:00 and during the night-time hours. For the most accurate models, the confidence intervals frequently overlap, indicating that no single model dominates at any particular hour. Nevertheless, the LSTM tends to achieve among the lowest errors throughout the day.

The most notable anomaly is MSTL-ETS. During the typical midday decline in heat demand, its errors are almost twice those of the naïve baseline. A plausible explanation is that MSTL more readily captures the two daily peaks and the nightly trough, since these patterns remain relatively consistent, whereas the transitional midday hours exhibit greater variability and a less well defined structure. This makes it more difficult for MSTL to identify stable daily patterns during those hours and can occasionally lead it to amplify spurious fluctuations, resulting in poor forecasts.

It is also worth noting that the overall degradation in accuracy as h increases is relatively small: for most models, the errors at $h = 24$ are almost indistinguishable from those at $h = 1$.

Overall, hour-ahead forecast performance is fairly stable across the day once differences in series magnitude are taken into account. The main criticality revealed by this per-hour analysis is the pronounced midday weakness of MSTL-ETS. The strongest models generally display comparable accuracy at any given hour, with only minimal degradation as the forecast horizon increases.

5.4.5 Residual Analysis

Because heat demand behaves relatively consistently during the winter months, without the regime shifts typically seen in transitional seasons, Auto-Correlation Function (ACF) plots provide a useful way to assess how model errors relate across different lags. Lower autocorrelation values indicate that fewer systematic structures remain in the residuals. An ideal model would yield residuals that are statistically indistinguishable from white noise, with no significant autocorrelations across the lags examined, suggesting that no systematic linear temporal structure remains in the errors. Throughout this section, residuals are defined as $r_t = y_t - \hat{y}_t$.

Figure 5.6 presents the ACFs for a selection of models. The LSTM shows the weakest overall temporal dependence, with autocorrelations approaching the 95% confidence bounds for the ACF under white-noise assumptions by around lag 12,

although occasional peaks remain at larger lags. This behaviour is reasonable given that forecasts are generated in 24-hour blocks, so some degree of positive autocorrelation at shorter lags is expected. Noticeably, the LSTM is the only model that does not show a pronounced autocorrelation at lag 24. The larger autocorrelations, particularly around lag 24, for the other models are broadly consistent with the rankings in Table 5.2 based on mean MAE. The TFT exhibits the strongest autocorrelation patterns, especially at daily seasonal lags. Despite the inclusion of temporal covariates such as hour of day and day of week, it appears not to have captured the daily seasonal structure as effectively as the other models.

Histogram plots of the residuals show broadly symmetric distributions across models, although small systematic biases remain. For SARIMAX and LSTM the mean residuals are around -1.5 , while XGBoost and Chronos-2 show larger negative biases of approximately -5 , and TFT around -7 . Under the convention $r_t = y_t - \hat{y}_t$, these negative mean residuals indicate a slight tendency to overforecast. Such biases are non-negligible when compared with the mean MAE values (approximately 25 to 30), but they remain small relative to the overall level of the series, which is on the order of a few hundred. An inspection of per-window mean error (ME) suggests that these biases may be driven by a small number of windows containing large deviations, corresponding to isolated instances in which all models overforecasted.

5.5 Series F1: Week Ahead Forecast Evaluation

We next examine the performance of all models on the week ahead forecasting task for series F1. Because windows are non-overlapping, the number of evaluation windows per season is limited (13 in autumn and winter, and 8 in spring). This increases uncertainty in per-season summaries, but some patterns in Table 5.3 are consistent across metrics and seasons.

Results broadly mirror the day-ahead setting. Across seasons and error measures, the LSTM attains the lowest mean errors and, in most cases, the highest win rates. Relative to the day-ahead horizon, LSTM win rates are often larger, suggesting that the performance gap between LSTM and competing models increases for week-ahead forecasts.

Chronos-2, which was a close second at the day-ahead horizon, degrades at the week-ahead horizon, especially in spring and autumn. Its win rates can remain

competitive, yet its mean errors are frequently worse than TFT. Together with the comparatively large standard deviations, this mismatch suggests high variability: the model performs well on a subset of windows but can fail badly on others. Given the small number of windows per season, a few atypical weeks can disproportionately affect the mean, so part of this degradation may be driven by limited-sample variability rather than a systematic loss of skill at longer horizons.

Conversely, TFT appears to benefit from the longer horizon. While it ranked below several competitors at the day-ahead horizon in autumn and winter, it now tends to place second across most metrics.

A likely explanation is the training objective. As discussed in Section 5.3, LSTM and TFT are the only models in our evaluation that are optimised end-to-end for the multi-step week-ahead task, that is, they are trained to minimise a loss computed over the full forecast window. During training, these models explicitly face long-horizon phenomena such as error accumulation and can learn to mitigate them. By contrast, XGBoost and SVMR are trained for one-step-ahead predictions and then applied recursively, and the statistical baselines are fitted to in-sample dynamics rather than directly to a multi-step forecasting loss.

The statistical significance of these improvements is assessed using the same Circular Block Bootstrap procedure described in Section 5.4. Focusing on an evaluation period of 19 weekly windows covering the coldest months, the bootstrap confidence intervals for the mean loss differential (alternative model minus LSTM) are strictly positive for all metrics and all competitors. Although the sample size is modest, these results indicate that the observed gains are unlikely to be explained by sampling variability alone.

5.6 Series F2

From series F2 onwards, TFT is omitted due to its prohibitively high computational cost relative to the modest performance gains observed on F1. Overall, results for F2 largely mirror those for F1, with the main exception that Chronos-2 is less competitive on this series, particularly in winter. LSTM again provides the most accurate forecasts across seasons and metrics. The improvements are modest relative to the intrinsic window-to-window variability of the series, yet they are consistent and statistically significant under the block bootstrap test.

5.6.1 Day-Ahead Forecasts

Per Window-Errors

Table 5.4 on page 125 reports per-window errors by season. In winter, three cut-offs are excluded due to missing values. The key observations are summarised below.

LSTM remains the most accurate model. LSTM is the leading method across all seasons and metrics, achieving the lowest mean MAE, RMSE, and sMAPE. Relative to the strongest competing model in each season, its mean MAE is about 15% lower (e.g., 17.25 vs 20.31 in autumn, 18.99 vs 22.10 in winter, and 12.56 vs 14.87 in spring). Its win rates are also consistently high: the MAE win rate is 43% in autumn and rises to 51% and 49% in winter and spring, respectively. As in F1, improvements in sMAPE are comparatively small (typically one to three percentage points). Moreover, standard deviations of per-window errors often exceed mean differences between leading models, indicating that window-to-window variability is larger than the average performance gap.

Chronos-2 does not outperform XGBoost. While Chronos-2 ranked close to LSTM on F1, the picture changes on F2: it ranks fourth in autumn and winter (by mean MAE) and second only in spring. Its win rates are not consistently higher than those of XGBoost; in winter, Chronos-2 is clearly less competitive across all metrics, whereas in autumn and spring the comparison is mixed. This highlights that the performance of a pre-trained, general-purpose forecasting model may not transfer reliably across series with different dynamics.

MSTL-ETS and SVMR underperform. MSTL-ETS is consistently weaker than the Naive24h baseline across seasons and metrics, reflecting the same adaptability limitations observed for F1. The Naive24h baseline performs comparatively better on F2, plausibly because the series exhibits a more stable daily structure, as suggested by the narrower morning variability bands in Figure 1.2 on page 15. SVMR also lags behind the stronger models and does not provide a competitive alternative.

Bootstrap Analysis

The day-ahead bootstrap analysis in Figure 5.7 on page 123 follows the same Circular Block Bootstrap protocol used for F1. Confidence intervals for the mean

loss differences ($model - LSTM$) are strictly positive across competing models and metrics, and the corresponding two-sided p -values are below the reporting precision, indicating that the LSTM advantage on F2 is unlikely to be explained by sampling variability alone. As in F1, the conclusions are robust to different choices of block length, with all reasonable specifications yielding the same qualitative outcome.

Hour-Ahead Horizon Errors and Residual Structure

An inspection of winter per-hour-ahead errors reveals patterns similar to series F1. The LSTM attains the lowest mean error throughout the day, although the margin relative to the other best-performing benchmarks is not statistically significant under block bootstrap.

The winter ACF diagnostics for series F2 and the best-performing models are shown in Figure 5.8. As in the day-ahead error analysis, we removed three cut-offs due to missing values. Overall, the patterns are broadly consistent with those observed for F1: the LSTM residual autocorrelation decays most rapidly. However, for F2 the LSTM exhibits a statistically significant negative autocorrelation at lag 24 (approximately -0.1), suggesting an alternating day-to-day error pattern, consistent with a tendency to over-correct relative to the previous day's forecast error. By contrast, Chronos-2 shows persistent positive autocorrelation at seasonal lags (multiples of 24 hours), indicating that daily seasonal structure remains more strongly in its residuals than for the other models.

These peaks are informative about error structure, but they do not necessarily imply poor predictive performance. Even small systematic effects at a given lag can yield statistically significant autocorrelations while having a limited impact on mean error aggregates, as in the case of the LSTM here.

5.6.2 Week-Ahead Forecasts

Moving to the week-ahead results, summarised in Table 5.5 on page 127, we observe broadly similar patterns to F1, with some deviations in spring.

In autumn and winter, LSTM maintains a clear lead, and its advantage is more pronounced at this longer horizon. In winter, LSTM attains high win rates across metrics (77% for RMSE and 92% for sMAPE, with 69% for MAE).

The remaining models largely preserve their day-ahead ordering. The main

exception is spring, where XGBoost achieves the lowest mean MAE. Given the small number of windows ($n = 8$), this ranking is sensitive to individual windows; inspection of the per-window errors suggests that LSTM's mean is influenced by one high-error window, despite relatively strong win rates. This does not undermine the broader finding that, overall, LSTM is the most reliable performer across seasons at the week-ahead horizon.

5.7 Series F3 to F5

Series F3 to F5 are increasingly challenging to forecast, in the sense that normalised error levels tend to increase overall compared to the first two series, especially for F5. In this setting, the scope for additional gains from plant-specific models (that is, all models except Chronos-2) becomes more limited: while LSTM generally remains one of the strongest plant-specific approaches, its advantages over the best competing benchmarks in this group are often modest. In contrast, as the series become more difficult, Chronos-2 increasingly attains an accuracy advantage, most clearly in winter and for day-ahead forecasts.

5.7.1 Day-Ahead Forecasts

Per-Window Errors and Bootstrap Results

Tables 5.7 to 5.9 (pp. 132–134) report per-window statistics for day-ahead forecasts by season. Across these series, LSTM typically ranks second behind Chronos-2 and outperforms the other plant-specific models, although its margin over the strongest plant-specific benchmarks is usually small. This is consistent with the bootstrap analysis: the confidence intervals constructed using the procedure in Section 5.4.3 and summarised in Figure 5.11 for the mean loss differential (model – LSTM) often include zero for XGBoost and SARIMAX.

In contrast, Chronos-2 ranks first in most day-ahead settings, with the exception of spring on F3 where LSTM remains slightly better on average. Its advantage is particularly evident in winter across F3–F5, whereas in the shoulder seasons gains are typically smaller (often around half a percentage point in sMAPE). The largest improvements occur on F5: in winter, Chronos-2 reduces MAE by approximately 17% relative to the best alternative, corresponding to an sMAPE reduction of around

2 percentage points.

Table 5.6 (p. 128) summarises block-bootstrap tests for pairwise comparisons of Chronos-2 against LSTM and XGBoost. The underlying (two-sided) raw p -values are computed using the same resampling procedure described in Section 5.4.3, but with Chronos-2 (rather than LSTM) as the reference model, that is, by testing the null hypothesis that the mean loss differential is zero. Recall that, for a candidate model m , reference model r and error metric ℓ , the mean loss differential is

$$\bar{\Delta} = \frac{1}{|\mathcal{S}|} \sum_{t \in \mathcal{S}} (\ell_t^{(m)} - \ell_t^{(r)}),$$

where $\ell_t^{(m)}$ denotes the window-level error of model m for the forecast issued at cut-off t , and $\mathcal{S}$ is the set of cut-offs used in the test. As detailed in Section 5.4.3, we restrict the bootstrap test to a central subperiod of the test span (winter plus a small portion of autumn) to obtain approximately stationary loss differentials. For each site (F3 to F5), we treat all tests for that site as a single multiple-testing family, pooling across competing models and metrics, and apply the Holm–Bonferroni procedure to obtain adjusted values p_{holm} . With the exception of F3 (where the improvement over LSTM is not significant), Chronos-2 achieves statistically significant improvements over these competitors in MAE and sMAPE, and on F5 in RMSE as well, as indicated by the corresponding Holm-adjusted p -values falling below 0.05.

Finally, SARIMAX continues to underperform during the mid-seasons relative to LSTM, Chronos-2 and XGBoost, a pattern especially evident on F3. One plausible explanation is its comparatively rigid treatment of weekly seasonality through fixed Fourier terms in the Box–Cox-transformed space, which may limit adaptability when seasonal structure varies over time.

Residual Structure

Figure 5.9 (p. 129) shows ACF plots of winter residuals for the day-ahead forecasts on series F3–F5. Across series, all models display non-negligible autocorrelation at some lags, with prominent peaks often occurring at daily lags (multiples of 24 h) and frequently exceeding the 95% white-noise bounds.

For F4, residual autocorrelations are generally weaker than for the other series. A plausible explanation is that F4 contains large, irregular demand spikes that are not well captured by any model (see Figure C.4); when forecast errors are dominated

by such unpredictable events, the residual sequence can appear more noise-like even if point accuracy is not necessarily higher.

5.7.2 Week-Ahead Forecasts

Week-ahead forecasts present a less stable picture, which is expected given the markedly smaller number of forecast windows. As shown in Tables 5.10 to 5.12 (pp. 135–137), LSTM and XGBoost generally remain among the strongest approaches, although individual seasonal rankings can change and occasional reversals are observed.

At this longer horizon, Chronos-2 typically loses its day-ahead advantage and often exhibits higher mean errors, consistent with the degradation observed on the earlier series at week-ahead horizons. Nevertheless, there are notable exceptions. In winter of F4, Chronos-2 achieves the lowest errors by a wide margin, while LSTM underperforms. In addition, Chronos-2 remains competitive in specific settings (for example, winter of F5, where the leading models are closely matched), highlighting the increased sensitivity of seasonal summaries when sample counts are small.

5.7.3 Interpretation of Results on Noisy Series

Taken together, these results suggest that relative model strengths shift as series difficulty increases. For day-ahead forecasts, performance among the main plant-specific models (LSTM, XGBoost, and SARIMAX) largely converges. This pattern indicates that, on noisier series, refinements to plant-specific modelling choices tend to yield limited improvements. By contrast, Chronos-2's relative advantage increases as difficulty rises, which is consistent with the view that large-scale pre-training, or more generally transfer and cross-series learning, can be beneficial when single-series data are noisy and contain limited signal.

This interpretation should be treated as suggestive rather than definitive. Stronger plant-specific architectures, richer features, or improved training procedures could still yield meaningful gains within the plant-specific regime. Moreover, as discussed in Section 5.3, Chronos-2 was pre-trained on data that overlap our evaluation period, but from other series that are likely unrelated to the target plants, according to our data audit. Such temporal overlap does not in itself constitute leakage, but it could still confer an advantage if some pre-training series are linked to the target plants

and therefore carry information about their future demand, which cannot be ruled out entirely.

A visual inspection of forecasts (Appendix C) provides an additional perspective. Chronos-2 typically produces smoother predictions than most alternatives. Together with its higher average day-ahead accuracy on the noisier series, this suggests that some plant-specific models may be overly responsive to high-frequency fluctuations (and thus prone to fitting noise), whereas Chronos-2 may place more emphasis on lower-frequency structure learned during pre-training.

At the same time, Chronos-2 tends to struggle at longer horizons. In the week-ahead setting it is often outperformed by LSTM, and in many cases also by XGBoost, with only a small number of exceptions.

To make the difficulty trend more explicit, Figure 5.10 reports, for winter, each model's NMAE for each series, for both day-ahead (top) and week-ahead (bottom) predictions. We focus on NMAE rather than sMAPE because NMAE is more stable and more interpretable for cross-series comparisons. In particular, sMAPE can be inflated when demand approaches zero. For example, on F2, a sharper nocturnal drop in demand (Figure 1.2, p. 15) leads to disproportionately high sMAPE values.

For the day-ahead case, we also mark the best-performing model (lowest NMAE) for each series with a star when its improvement over all other models is statistically significant under the block bootstrap test described in Section 5.4.3. Statistical significance is assessed using block-bootstrap p-values for pairwise comparisons based on mean loss differences, adjusted for multiple testing using the Holm–Bonferroni procedure. A yellow star indicates significance against all competing models across all metrics considered (MAE, sMAPE, and RMSE), whereas a blue star indicates significance across all metrics except RMSE. Since NMAE is a positive rescaling of MAE (within each series), the MAE-based significance results also apply to NMAE. Significance tests were carried out only for the day-ahead case, as the week-ahead evaluation contains too few observations (approximately 15 rolling windows) to support reliable statistical inference.

The resulting bar plots suggest that, for day-ahead forecasts, LSTM tends to perform best on the comparatively easier series (F1 and F2), while Chronos-2 gains a relative advantage as difficulty increases. At the week-ahead horizon, results are less stable. LSTM typically leads, followed by XGBoost, with a notable reversal on F4 in winter where Chronos-2 becomes the best performer.



Figure 5.2: MAE differences per cut-off relative to LSTM baseline across autumn, winter and spring on series F1 for the day-ahead horizon. Positive values indicate worse performance than LSTM.

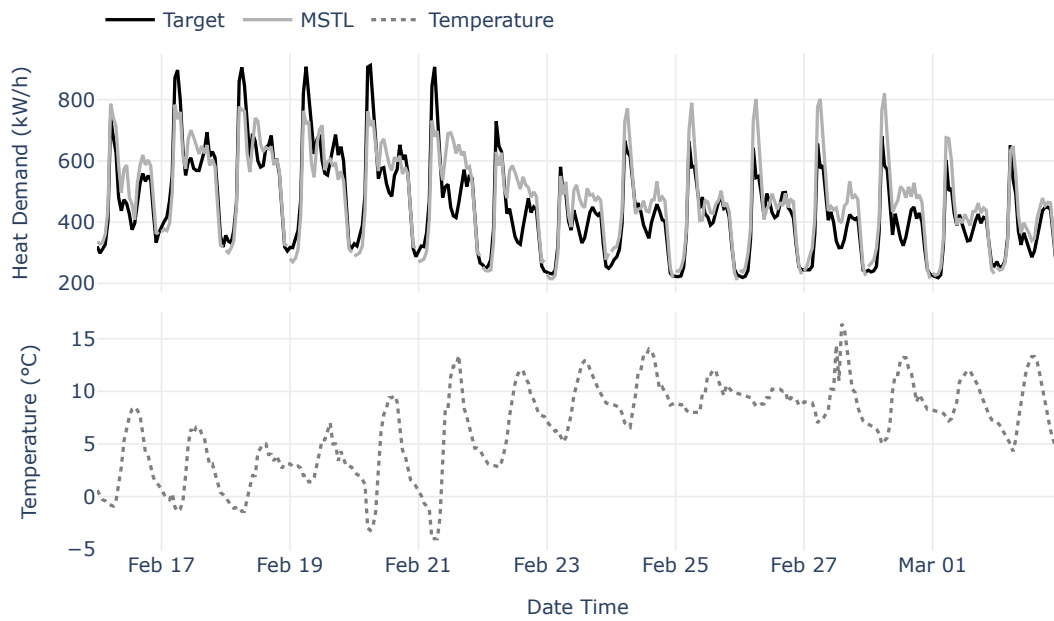


Figure 5.3: Day-ahead forecasts produced by the MSTL-ETS model for F1 during selected winter days. The figure illustrates a delayed response of the model to target fluctuations associated with temperature variations.

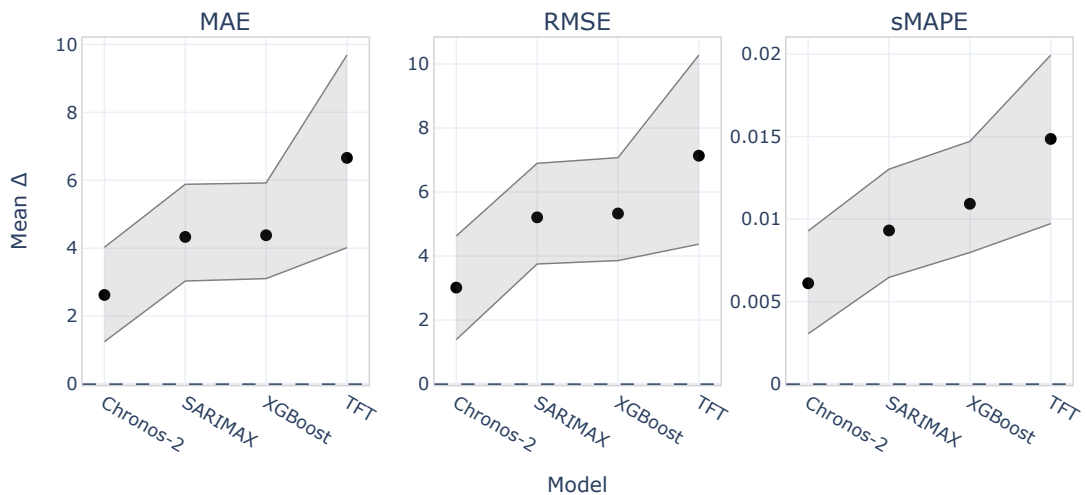
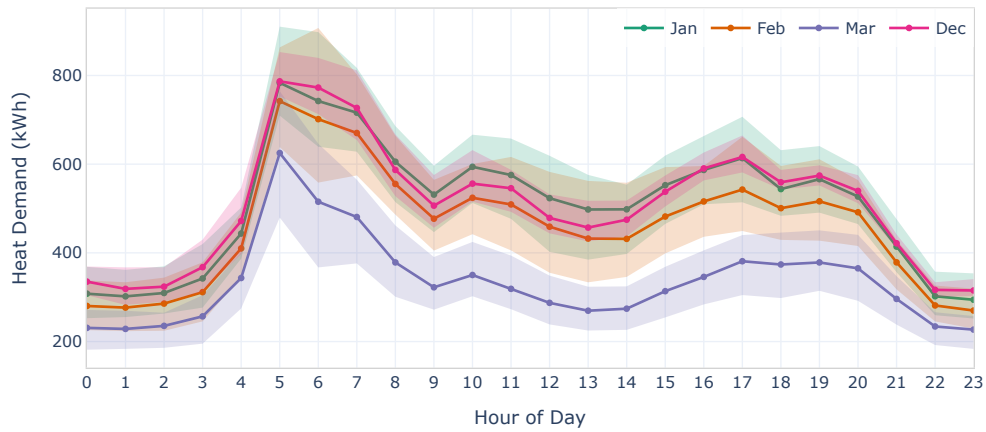
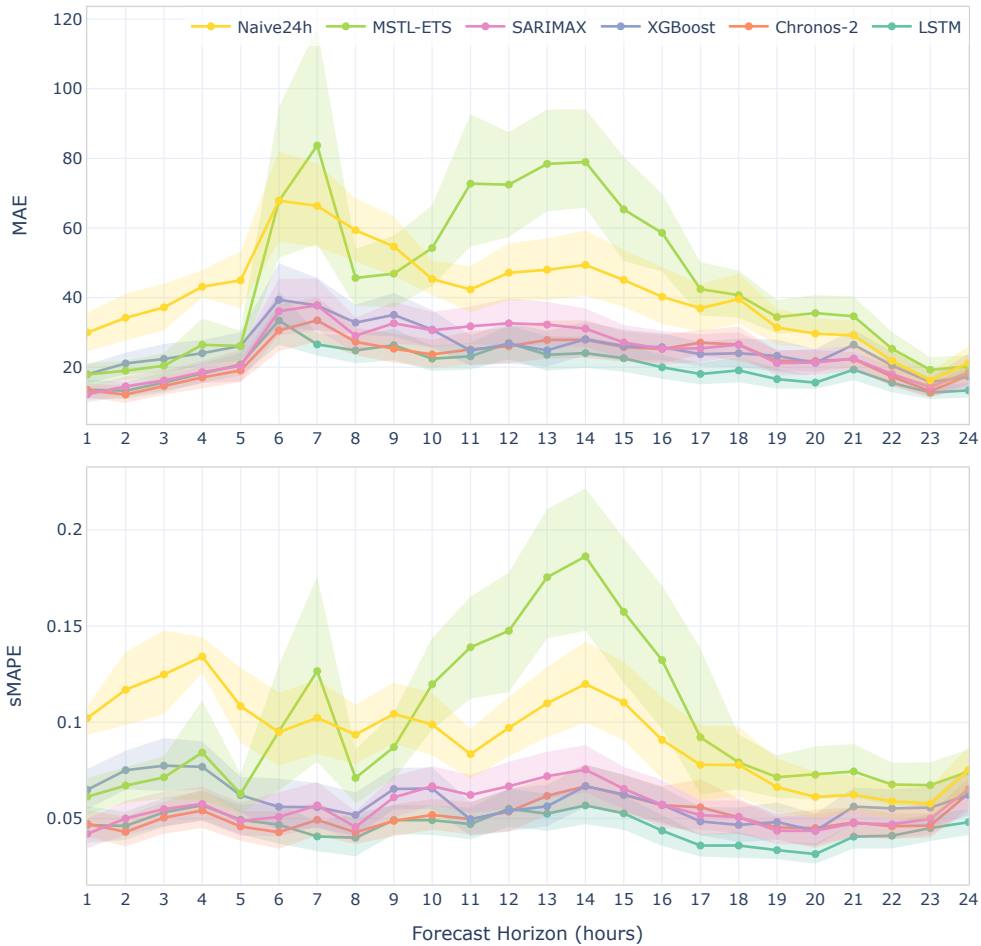


Figure 5.4: Mean performance differences ($model - LSTM$) for MAE, RMSE and sMAPE across all models. Point estimates are shown with 95% bootstrap confidence intervals. Positive values indicate that the LSTM achieves lower error.



(a)



(b)

Figure 5.5: Hourly heat demand and forecast errors for winter day-ahead predictions on F1, grouped by hour-ahead lead time, for selected models. (a) Average hourly heat demand profiles by month, with 80% variability bands. (b) MAE and sMAPE by forecast horizon, with 95% bootstrap percentile intervals.

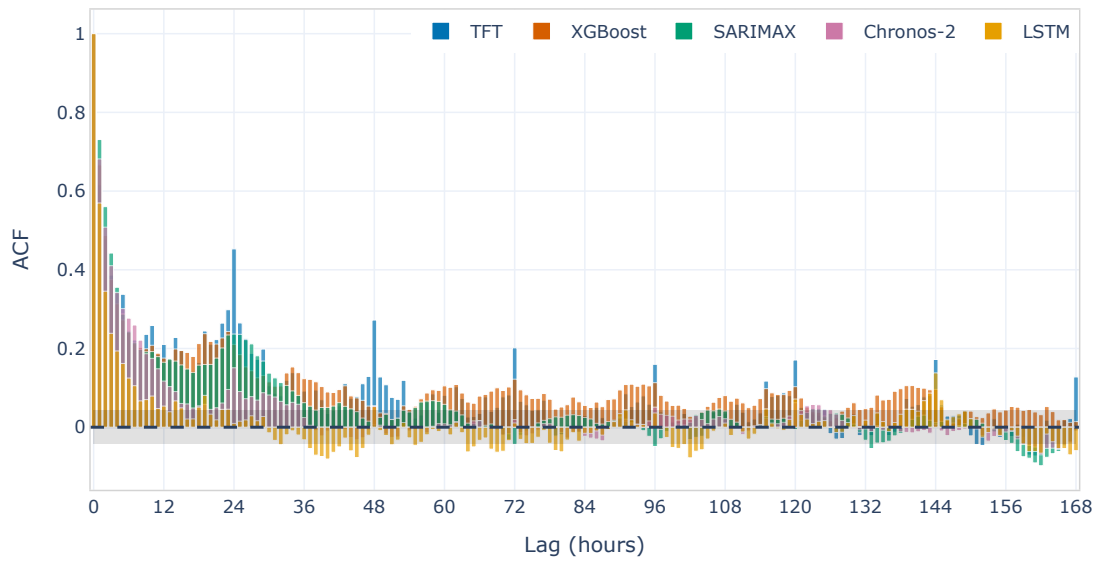


Figure 5.6: ACF of residuals for winter day-ahead forecasts on series F1 for TFT, XGBoost, SARIMAX, LSTM, and Chronos-2, shown for lags up to 168 hours. The shaded grey band denotes the 95% confidence interval for the ACF under white-noise assumptions, providing a reference for identifying statistically significant autocorrelations.

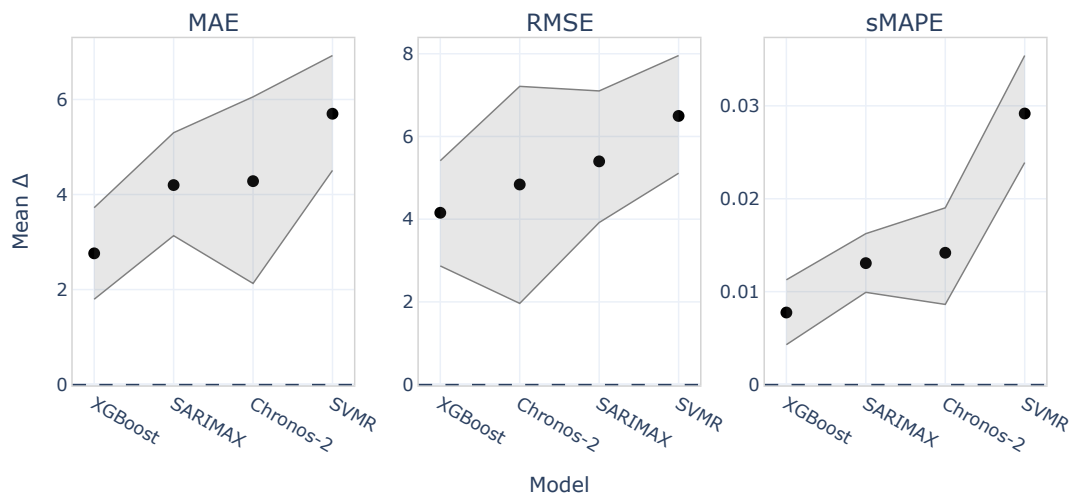


Figure 5.7: Mean performance differences ($model - LSTM$) for selected metrics and models on series F2 and day-ahead forecasts. Point estimates are shown with 95% bootstrap confidence intervals. Positive values indicate that the LSTM achieves lower error.

Table 5.3: Per-window error statistics for the *week-ahead* forecast horizon on *F1*. Each table is ordered by ascending mean MAE.

(a) Autumn (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	22.34 (11.49)	23	28.49 (12.96)	31	13.70 (14.88)	31
TFT	25.14 (13.48)	31	31.67 (14.59)	31	14.64 (14.36)	23
XGBoost	26.89 (13.00)	15	34.51 (14.38)	8	15.38 (14.29)	15
SARIMAX	30.07 (17.21)	0	37.96 (19.47)	0	16.79 (14.56)	0
Chronos-2	31.77 (25.48)	23	39.84 (28.07)	15	16.18 (14.61)	23
MSTL-ETS	36.39 (23.05)	8	46.71 (26.25)	15	16.70 (10.23)	8
SVMR	42.05 (16.37)	0	51.12 (18.25)	0	31.51 (34.38)	0
Naive24h	49.00 (35.78)	0	60.16 (40.39)	0	22.53 (15.87)	0

(b) Winter (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	24.70 (6.89)	54	31.99 (10.20)	54	5.74 (1.44)	46
TFT	28.11 (7.75)	8	36.16 (10.74)	8	6.62 (1.84)	8
Chronos-2	30.81 (10.97)	23	39.06 (13.09)	23	7.01 (2.57)	31
SARIMAX	32.01 (12.72)	15	40.40 (14.78)	15	7.38 (2.66)	15
XGBoost	32.43 (9.65)	0	41.80 (13.17)	0	7.52 (2.19)	0
SVMR	38.58 (10.79)	0	48.34 (13.56)	0	9.26 (2.46)	0
MSTL-ETS	53.75 (12.31)	0	69.52 (14.14)	0	12.36 (3.13)	0
Naive24h	57.57 (24.06)	0	71.25 (27.89)	0	13.28 (5.42)	0

(c) Spring (8 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	10.81 (7.12)	38	14.58 (10.06)	50	8.17 (1.88)	50
TFT	11.86 (7.71)	50	16.26 (11.12)	38	8.51 (1.05)	38
XGBoost	15.26 (9.77)	0	20.45 (13.40)	0	10.76 (2.95)	0
Chronos-2	18.42 (9.17)	0	24.14 (12.58)	0	14.73 (8.61)	0
Naive24h	25.37 (17.82)	0	31.36 (21.99)	0	18.12 (10.39)	0
MSTL-ETS	25.70 (16.23)	0	33.02 (21.19)	0	18.24 (6.32)	0
SARIMAX	25.72 (15.34)	12	30.92 (18.44)	12	19.58 (11.57)	12
SVMR	28.24 (7.86)	0	33.95 (8.11)	0	27.02 (14.88)	0

Table 5.4: Per-window error statistics for the *day-ahead* forecast horizon on *F2*. Each table is ordered by ascending mean MAE. In winter, 3 cut-offs are excluded due to missing values in the data.

(a) Autumn (92 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	17.25 (8.49)	43	23.07 (11.94)	36	13.86 (9.04)	34
XGBoost	20.31 (10.78)	10	26.43 (14.44)	13	16.14 (12.38)	12
SARIMAX	20.99 (14.43)	14	27.51 (18.68)	13	15.54 (12.50)	21
Chronos-2	21.53 (13.40)	15	28.14 (17.55)	15	15.00 (9.88)	17
SVMR	23.08 (11.11)	7	29.27 (13.95)	11	22.45 (18.95)	4
Naive24h	24.51 (15.17)	4	31.51 (19.48)	4	17.69 (11.20)	4
MSTL-ETS	40.75 (33.27)	7	53.19 (41.64)	8	25.98 (21.10)	8

(b) Winter (87 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	18.99 (7.18)	51	26.16 (13.51)	45	6.38 (2.05)	41
XGBoost	22.10 (9.69)	21	30.93 (16.92)	23	7.28 (2.68)	22
SARIMAX	23.86 (8.68)	8	32.25 (14.75)	11	7.88 (2.56)	7
Chronos-2	24.18 (9.07)	10	32.00 (14.46)	8	8.05 (2.88)	15
SVMR	24.40 (8.75)	3	32.57 (15.04)	6	9.02 (3.20)	5
Naive24h	30.21 (12.07)	6	40.23 (19.39)	7	10.09 (4.00)	5
MSTL-ETS	37.51 (15.85)	1	50.65 (21.52)	0	10.60 (4.41)	6

(c) Spring (59 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	12.56 (10.15)	49	17.00 (15.02)	49	15.14 (7.69)	46
Chronos-2	14.87 (11.53)	17	19.48 (16.62)	15	18.13 (8.59)	17
XGBoost	15.53 (12.37)	14	21.20 (17.72)	17	18.78 (9.17)	12
SVMR	18.11 (11.21)	3	23.62 (15.83)	7	26.06 (14.36)	3
Naive24h	18.81 (15.01)	3	25.00 (20.57)	3	20.67 (9.07)	5
SARIMAX	19.45 (14.64)	12	26.13 (20.37)	7	19.87 (9.38)	14
MSTL-ETS	37.43 (29.70)	2	51.35 (39.46)	2	34.12 (17.11)	3

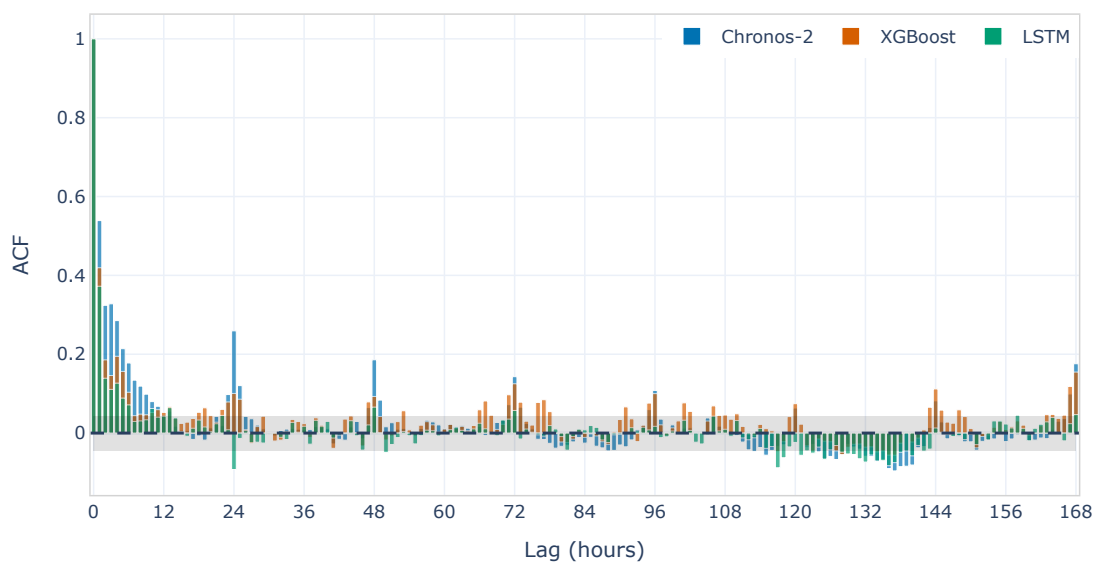


Figure 5.8: ACF of residuals for winter day-ahead forecasts on series F2 for XGBoost, LSTM, and Chronos-2, shown for lags up to 168 hours. The shaded grey band denotes the 95% confidence interval for the ACF under white-noise assumptions, providing a reference for identifying statistically significant autocorrelations. Three cut-offs were excluded due to missing values.

Table 5.5: Per-window error statistics for the *week-ahead* forecast horizon on *F2*. Each table is ordered by ascending mean MAE.

(a) Autumn (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	29.54 (17.00)	54	39.66 (22.33)	46	24.45 (23.70)	46
XGBoost	32.14 (27.26)	15	42.27 (33.93)	31	24.20 (26.10)	15
SARIMAX	33.38 (29.71)	15	43.32 (36.10)	0	25.98 (29.15)	8
Chronos-2	39.71 (33.94)	8	51.53 (41.76)	8	26.89 (28.38)	15
SVMR	41.62 (22.62)	0	52.25 (28.09)	8	37.97 (33.37)	8
MSTL	42.27 (28.55)	0	56.08 (36.10)	0	26.17 (19.52)	0
Naive24h	42.98 (35.81)	8	54.67 (43.45)	8	30.92 (28.49)	8

(b) Winter (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	24.37 (11.00)	69	38.04 (28.85)	77	7.87 (3.16)	92
XGBoost	31.15 (11.00)	15	49.47 (28.81)	8	9.36 (2.91)	8
Chronos-2	33.91 (8.68)	0	48.85 (25.69)	8	10.83 (3.45)	0
SARIMAX	35.05 (12.02)	8	50.10 (26.60)	0	12.06 (3.93)	0
SVMR	40.48 (12.90)	0	55.92 (25.22)	8	15.30 (5.39)	0
Naive24h	46.51 (14.82)	0	66.44 (31.29)	0	15.42 (3.91)	0
MSTL	49.06 (15.12)	8	67.98 (28.10)	0	15.13 (6.12)	0

(c) Spring (8 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
XGBoost	22.53 (16.08)	12	33.18 (23.81)	0	23.05 (8.89)	12
LSTM	26.73 (25.63)	62	37.62 (35.69)	62	26.39 (14.55)	62
Chronos-2	30.18 (19.04)	12	43.59 (27.51)	12	33.40 (20.80)	12
SVMR	31.27 (11.92)	0	41.63 (18.20)	12	47.96 (26.22)	0
SARIMAX	32.69 (23.39)	0	45.60 (33.43)	0	31.12 (14.44)	12
Naive24h	35.42 (29.41)	0	48.68 (39.93)	12	31.85 (13.60)	0
MSTL	38.93 (29.42)	12	53.53 (40.34)	0	36.71 (14.05)	0

Table 5.6: Two-sided block bootstrap comparisons against Chronos-2 for series F3–F5. Each entry reports the mean loss differential $\bar{\Delta}$ of the specified model relative to Chronos-2, with the Holm-adjusted p -value in parentheses, $\bar{\Delta}(p_{\text{holm}})$. Positive values of $\bar{\Delta}$ indicate superior performance of Chronos-2. sMAPE values are reported in percentage points.

(a) Series F3

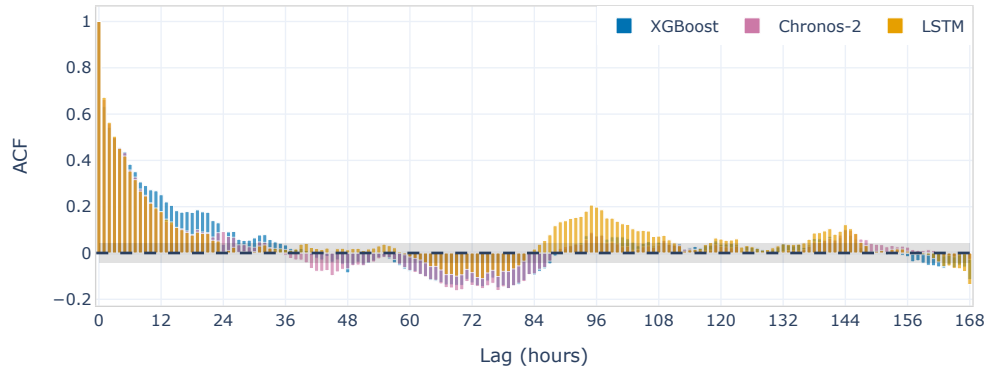
Model	MAE	RMSE	sMAPE
	$\bar{\Delta}(p_{\text{holm}})$	$\bar{\Delta}(p_{\text{holm}})$	$\bar{\Delta}(p_{\text{holm}})$
LSTM	2.83 (0.066)	2.70 (0.066)	0.6 (0.066)
XGBoost	2.95 (0.024)	2.82 (0.024)	0.8 (0.003)

(b) Series F4

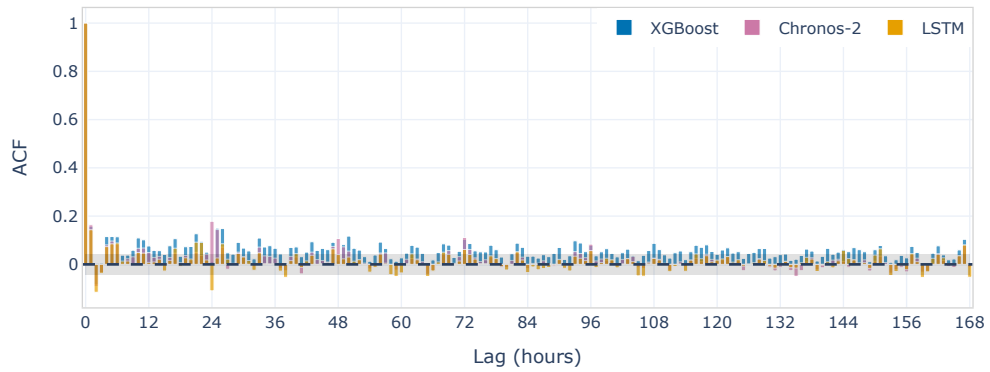
Model	MAE	RMSE	sMAPE
	$\bar{\Delta}(p_{\text{holm}})$	$\bar{\Delta}(p_{\text{holm}})$	$\bar{\Delta}(p_{\text{holm}})$
LSTM	3.07 (0.001)	1.07 (0.286)	0.8 (0.000)
XGBoost	3.91 (0.000)	1.13 (0.253)	0.9 (0.000)

(c) Series F5

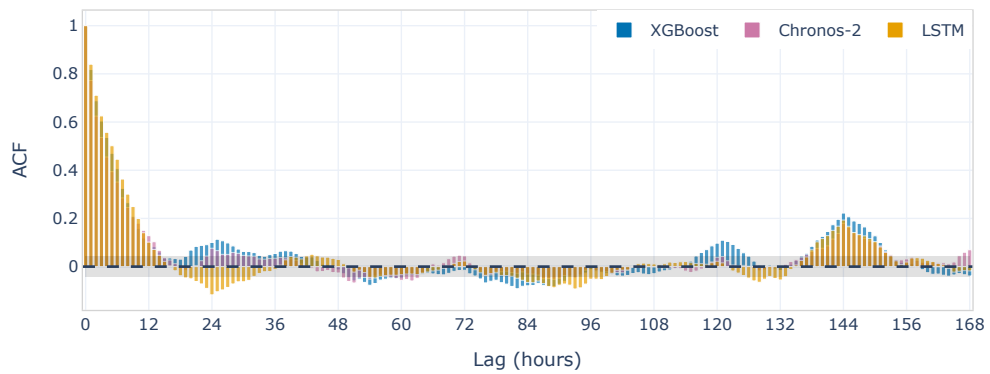
Model	MAE	RMSE	sMAPE
	$\bar{\Delta}(p_{\text{holm}})$	$\bar{\Delta}(p_{\text{holm}})$	$\bar{\Delta}(p_{\text{holm}})$
LSTM	5.32 (0.020)	6.39 (0.020)	1.6 (0.020)
XGBoost	7.45 (0.000)	8.16 (0.001)	2.6 (0.000)



(a) Series F3

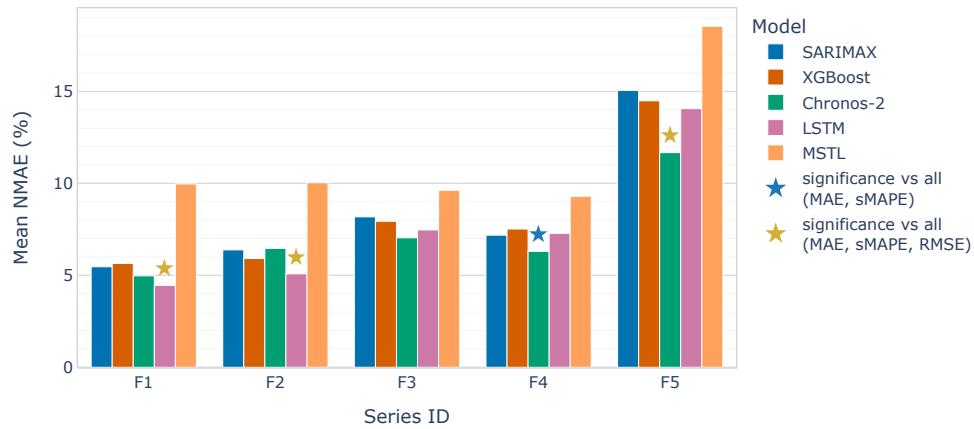


(b) Series F4

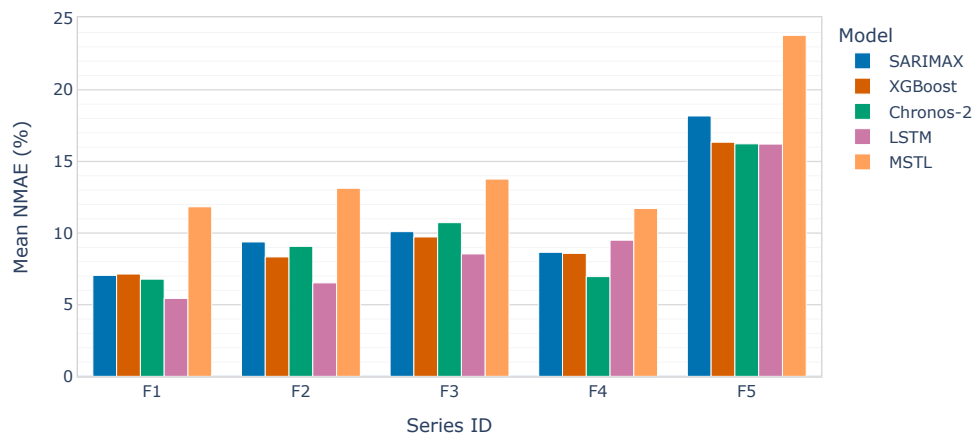


(c) Series F5

Figure 5.9: ACF of residuals for the XGBoost, LSTM and Chronos-2 models on series F3 (a), F4 (b), and F5 (c), over lags up to 168 hours. The shaded grey band denotes the 95% confidence interval for the ACF under white-noise assumptions, providing a reference for identifying statistically significant autocorrelations.



(a) Day-ahead



(b) Week-ahead

Figure 5.10: Winter NMAE by series and model for day-ahead (top) and week-ahead (bottom) forecasts. For day-ahead forecasts, stars mark the lowest-NMAE model for each ID when its improvement over all other models is statistically significant under the block bootstrap test (Holm-Bonferroni corrected): yellow indicates significance across MAE, sMAPE and RMSE, while blue indicates significance across all metrics except RMSE.

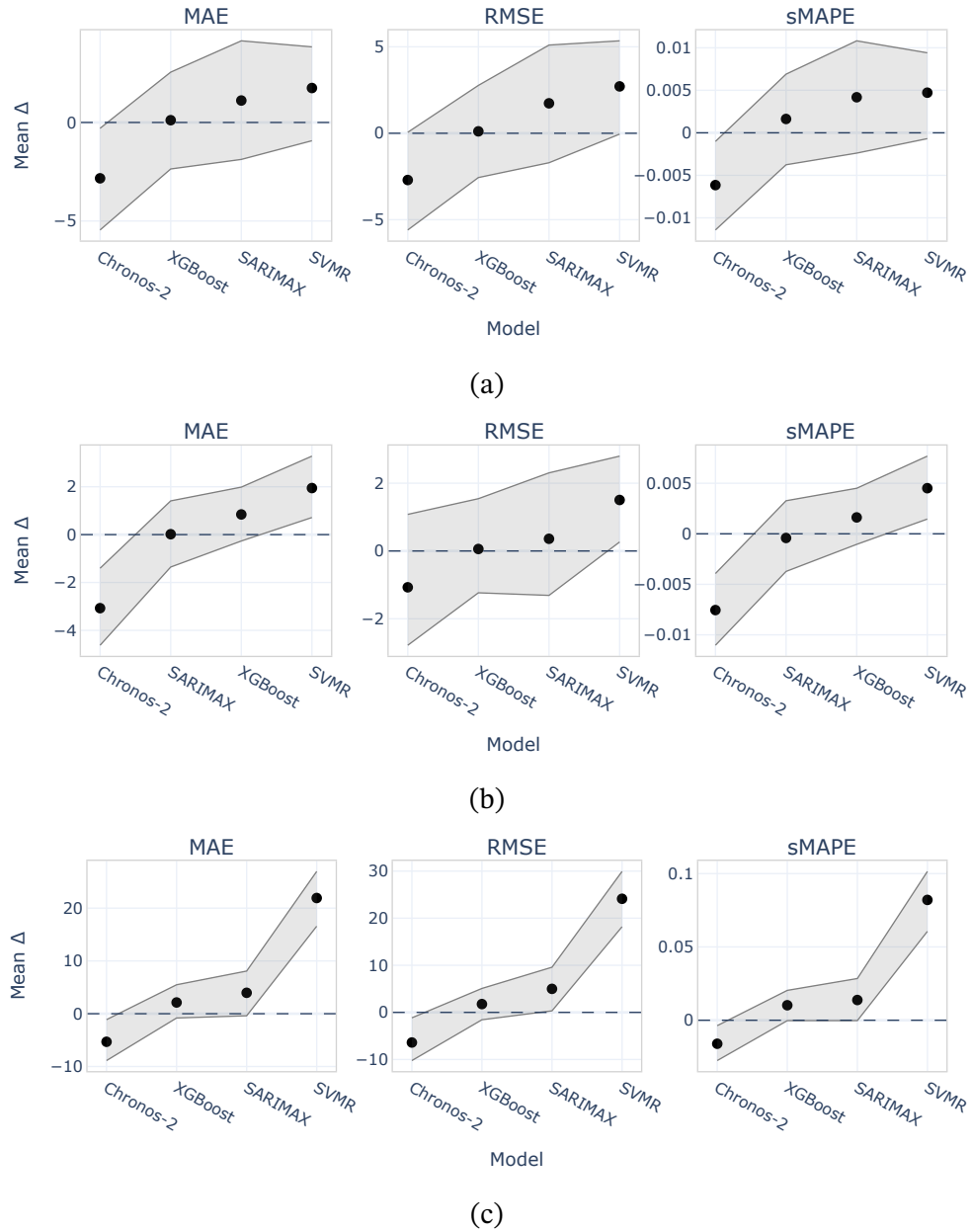


Figure 5.11: Mean performance differences ($model - LSTM$) for selected metrics and models on series F3-F5 and day-ahead forecasts. Point estimates are shown with 95% bootstrap confidence intervals. Positive values indicate that the LSTM achieves lower error.

Table 5.7: Per-window error statistics for the *day-ahead* forecast horizon on *F3*. Each table is ordered by ascending mean MAE.

(a) Autumn (92 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	30.68 (20.59)	22	39.09 (26.34)	23	12.49 (10.44)	21
LSTM	32.96 (22.62)	22	41.24 (28.43)	22	13.11 (11.15)	21
SVMR	33.85 (20.99)	17	42.92 (26.81)	17	14.13 (11.53)	21
XGBoost	36.37 (18.50)	10	44.57 (23.19)	13	17.75 (16.61)	10
MSTL-ETS	38.70 (33.72)	12	48.77 (37.28)	11	14.63 (8.16)	11
SARIMAX	38.84 (21.37)	10	47.54 (26.01)	8	17.80 (14.91)	8
Naive24h	39.29 (25.84)	8	51.27 (32.90)	7	15.64 (12.12)	10

(b) Winter (90 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	31.29 (18.77)	26	39.53 (22.50)	27	6.80 (3.79)	26
LSTM	33.20 (18.73)	19	41.47 (22.73)	20	7.27 (3.76)	18
XGBoost	35.29 (17.44)	12	43.42 (19.84)	12	7.87 (3.71)	11
SVMR	35.55 (16.21)	10	44.49 (19.88)	8	7.92 (3.45)	10
SARIMAX	36.36 (16.18)	20	45.22 (19.83)	19	8.12 (3.54)	22
MSTL-ETS	42.78 (22.54)	8	52.81 (25.57)	8	9.42 (4.45)	8
Naive24h	47.71 (24.71)	6	60.61 (30.23)	7	10.54 (5.17)	6

(c) Spring (59 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	17.44 (11.10)	32	22.09 (14.31)	31	8.97 (5.09)	27
Chronos-2	17.90 (13.26)	29	22.22 (15.98)	27	9.10 (5.93)	29
SVMR	21.17 (12.08)	12	26.38 (15.38)	10	11.05 (6.28)	14
XGBoost	22.30 (15.58)	7	27.19 (18.05)	14	11.15 (6.21)	12
MSTL-ETS	23.41 (13.22)	8	28.03 (15.80)	10	11.84 (4.82)	7
Naive24h	25.95 (18.13)	7	32.60 (21.93)	5	12.47 (6.40)	7
SARIMAX	30.01 (18.82)	5	34.75 (20.42)	3	15.34 (8.81)	5

Table 5.8: Per-window error statistics for the *day-ahead* forecast horizon on *F4*. Each table is ordered by ascending mean MAE.

(a) Autumn (92 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	25.15 (21.43)	45	35.72 (29.63)	39	9.33 (5.88)	46
LSTM	26.09 (19.51)	21	35.55 (27.05)	16	9.76 (4.51)	18
XGBoost	26.91 (20.85)	11	36.59 (28.12)	20	10.29 (5.29)	9
SARIMAX	28.76 (23.82)	5	38.56 (30.77)	9	11.08 (6.61)	7
SVMR	29.20 (19.05)	5	38.69 (26.02)	5	12.20 (6.81)	5
MSTL-ETS	29.68 (23.13)	7	40.01 (30.70)	8	11.19 (5.91)	8
Naive24h	31.62 (26.34)	7	44.65 (35.87)	3	12.14 (7.41)	8

(b) Winter (90 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	26.59 (14.48)	48	39.22 (22.32)	41	5.96 (2.61)	52
SARIMAX	30.30 (16.09)	13	41.13 (22.85)	14	6.80 (2.92)	12
LSTM	30.70 (13.82)	12	41.51 (20.04)	16	6.98 (2.45)	10
XGBoost	31.71 (15.57)	2	41.41 (20.33)	8	7.16 (2.79)	1
SVMR	32.87 (14.84)	6	43.33 (20.07)	8	7.50 (2.74)	7
Naive24h	34.18 (19.11)	16	50.56 (29.48)	10	7.62 (3.52)	16
MSTL-ETS	39.21 (17.83)	3	54.87 (25.80)	3	8.80 (3.25)	2

(c) Spring (59 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	15.51 (11.78)	41	19.67 (15.09)	39	8.34 (5.61)	39
LSTM	16.76 (12.62)	24	21.27 (16.07)	27	8.70 (5.34)	24
SVMR	18.65 (11.76)	10	23.57 (14.58)	5	10.37 (5.97)	10
XGBoost	19.10 (17.44)	3	24.14 (20.56)	3	9.91 (8.09)	2
Naive24h	20.93 (17.34)	7	26.96 (22.61)	7	10.61 (6.77)	8
MSTL-ETS	21.43 (14.83)	5	26.39 (17.37)	3	10.91 (5.50)	5
SARIMAX	23.10 (18.76)	10	27.28 (21.15)	15	12.15 (9.63)	12

Table 5.9: Per-window error statistics for the *day-ahead* forecast horizon on *F5*. Each table is ordered by ascending mean MAE.

(a) Autumn (92 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	26.24 (26.70)	34	36.18 (37.18)	35	15.60 (15.17)	36
LSTM	26.27 (23.22)	14	35.99 (32.68)	12	18.06 (14.34)	13
XGBoost	28.61 (25.97)	12	38.21 (35.16)	15	17.88 (14.16)	13
SARIMAX	30.81 (27.57)	14	41.00 (37.62)	12	18.93 (14.83)	13
MSTL-ETS	39.22 (37.60)	5	50.30 (47.82)	7	23.46 (19.10)	5
SVMR	42.59 (32.36)	7	54.01 (40.55)	5	33.54 (24.03)	7
Naive24h	47.21 (59.60)	14	63.32 (73.40)	14	24.22 (24.19)	13

(b) Winter (90 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	36.33 (20.09)	30	45.04 (24.16)	33	12.25 (6.02)	24
LSTM	43.78 (32.64)	26	54.21 (36.64)	20	14.45 (9.94)	24
XGBoost	45.10 (28.80)	10	55.30 (33.21)	10	14.99 (8.49)	10
SARIMAX	46.86 (26.79)	12	58.19 (31.99)	12	15.65 (8.48)	17
MSTL-ETS	57.72 (34.62)	12	70.88 (40.44)	13	19.37 (11.81)	12
SVMR	63.36 (28.94)	2	75.53 (31.60)	3	22.52 (12.62)	3
Naive24h	89.34 (64.67)	8	107.97 (73.41)	8	29.79 (21.33)	9

(c) Spring (59 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	16.71 (13.32)	32	23.06 (19.97)	32	21.80 (17.75)	27
LSTM	17.14 (12.66)	20	23.39 (17.39)	17	22.44 (18.20)	24
XGBoost	17.25 (11.47)	22	23.43 (15.61)	22	22.43 (18.80)	14
SARIMAX	23.40 (15.53)	5	31.17 (21.25)	8	27.47 (18.74)	14
SVMR	28.28 (19.68)	7	37.08 (26.17)	7	37.20 (24.73)	5
MSTL-ETS	32.29 (26.23)	2	45.04 (35.43)	2	36.64 (20.05)	3
Naive24h	33.25 (35.54)	12	44.99 (46.22)	12	32.40 (22.28)	14

Table 5.10: Per-window error statistics for the *week-ahead* forecast horizon on *F3*. Each table is ordered by ascending mean MAE.

(a) Autumn (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
XGBoost	46.88 (16.42)	0	57.55 (19.57)	0	24.11 (20.95)	0
LSTM	47.45 (21.61)	15	58.39 (26.83)	15	22.17 (19.00)	15
SVMR	48.73 (20.11)	23	60.88 (25.87)	23	21.67 (15.96)	23
SARIMAX	51.93 (23.53)	8	61.87 (25.69)	15	26.02 (21.81)	8
MSTL-ETS	52.60 (26.10)	0	65.12 (29.31)	8	22.49 (17.23)	0
Chronos-2	53.73 (34.49)	38	64.86 (36.45)	38	25.48 (23.57)	38
Naive24h	57.12 (37.55)	15	69.86 (39.92)	0	25.36 (21.56)	15

(b) Winter (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	37.97 (13.31)	46	48.05 (15.55)	54	8.58 (2.46)	46
XGBoost	43.24 (12.60)	23	53.32 (14.32)	23	9.97 (2.97)	23
SARIMAX	44.88 (16.46)	15	55.77 (18.26)	8	10.25 (3.34)	15
Chronos-2	47.67 (23.63)	15	59.88 (25.93)	15	10.58 (4.84)	15
SVMR	49.45 (14.13)	0	60.89 (15.94)	0	11.24 (3.20)	0
MSTL-ETS	61.20 (23.11)	0	75.64 (27.92)	0	13.90 (5.29)	0
Naive24h	72.42 (21.29)	0	90.75 (25.38)	0	16.90 (5.52)	0

(c) Spring (8 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
MSTL-ETS	30.83 (19.10)	12	37.24 (21.55)	12	16.16 (9.53)	12
LSTM	32.55 (24.41)	38	38.46 (26.22)	25	17.47 (14.87)	38
XGBoost	32.91 (23.07)	0	40.04 (25.13)	12	17.62 (12.98)	0
SVMR	37.89 (12.44)	12	44.78 (11.64)	12	20.96 (9.01)	12
Chronos-2	40.78 (28.23)	25	48.00 (28.75)	25	22.42 (16.45)	25
SARIMAX	42.23 (24.63)	12	48.60 (26.17)	12	21.60 (10.52)	12
Naive24h	42.40 (27.35)	0	51.04 (31.28)	0	21.56 (11.66)	0

Table 5.11: Per-window error statistics for the *week-ahead* forecast horizon on *F4*. Each table is ordered by ascending mean MAE.

(a) Autumn (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
XGBoost	31.55 (23.54)	31	42.92 (30.11)	23	12.01 (6.17)	31
LSTM	32.48 (24.23)	15	43.34 (30.62)	23	12.11 (5.85)	15
SARIMAX	36.27 (31.09)	23	48.18 (36.68)	23	14.25 (9.87)	23
SVMR	37.42 (19.07)	8	49.10 (25.25)	8	16.93 (9.05)	8
Chronos-2	40.49 (39.61)	23	53.06 (45.54)	23	15.12 (14.16)	23
MSTL-ETS	42.21 (31.38)	0	54.33 (38.08)	0	15.73 (8.49)	0
Naive24h	47.16 (43.61)	0	60.46 (49.97)	0	18.33 (15.58)	0

(b) Winter (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
Chronos-2	29.36 (11.20)	69	44.61 (18.07)	46	6.73 (1.89)	69
XGBoost	36.20 (15.09)	0	46.98 (17.31)	31	8.36 (2.86)	0
SARIMAX	36.51 (15.67)	8	48.61 (19.93)	8	8.35 (2.67)	8
Naive24h	38.80 (14.44)	15	58.49 (22.90)	8	9.03 (3.02)	15
SVMR	38.89 (13.82)	0	50.94 (17.25)	0	9.11 (2.80)	0
LSTM	40.08 (13.42)	8	51.48 (15.96)	8	9.59 (3.19)	8
MSTL-ETS	49.42 (18.80)	0	68.21 (25.79)	0	11.44 (3.48)	0

(c) Spring (8 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	26.59 (16.40)	25	31.74 (19.05)	25	13.96 (5.26)	25
XGBoost	27.61 (26.05)	0	33.40 (28.23)	0	14.35 (12.37)	0
SVMR	30.65 (19.45)	0	36.46 (20.67)	0	16.71 (9.30)	0
SARIMAX	31.26 (28.50)	25	36.23 (30.23)	25	16.34 (14.70)	25
Chronos-2	32.04 (29.36)	38	38.65 (32.02)	38	17.54 (14.70)	38
MSTL-ETS	34.31 (26.93)	0	40.37 (30.17)	0	17.24 (11.31)	0
Naive24h	35.31 (36.17)	12	43.74 (40.25)	12	16.84 (15.28)	12

Table 5.12: Per-window error statistics for the *week-ahead* forecast horizon on *F5*. Each table is ordered by ascending mean MAE.

(a) Autumn (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	32.46 (24.88)	23	44.94 (34.09)	23	21.68 (8.64)	23
XGBoost	36.42 (30.92)	15	49.14 (39.67)	23	22.73 (11.99)	23
SARIMAX	37.24 (32.78)	15	50.85 (41.36)	15	24.01 (15.08)	8
Chronos-2	39.91 (40.49)	23	54.44 (50.22)	23	24.60 (22.79)	23
MSTL-ETS	55.29 (50.31)	0	72.67 (62.60)	0	29.24 (13.50)	0
SVMR	60.12 (34.04)	8	74.87 (41.15)	8	43.81 (18.96)	8
Naive24h	86.86 (80.91)	15	114.48 (103.36)	8	40.69 (27.11)	15

(b) Winter (13 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	50.46 (23.45)	38	67.50 (31.85)	23	16.17 (6.35)	31
Chronos-2	50.51 (23.26)	23	66.64 (32.58)	31	16.45 (6.33)	31
XGBoost	50.86 (21.31)	23	66.53 (27.04)	31	16.75 (5.60)	15
SARIMAX	56.57 (21.32)	0	73.46 (30.81)	0	18.87 (5.68)	8
SVMR	72.37 (19.33)	8	88.84 (20.94)	8	25.79 (5.48)	0
MSTL-ETS	74.08 (45.75)	8	92.79 (51.59)	8	25.88 (16.96)	15
Naive24h	145.11 (33.80)	0	185.87 (39.78)	0	51.58 (8.39)	0

(c) Spring (8 forecast windows)

Model	MAE		RMSE		sMAPE (%)	
	mean (std)	wrate	mean (std)	wrate	mean (std)	wrate
LSTM	20.64 (12.25)	38	29.99 (18.49)	25	25.75 (13.53)	38
XGBoost	23.90 (14.25)	0	33.37 (19.64)	25	28.81 (15.99)	12
Chronos-2	27.43 (17.36)	12	39.70 (27.78)	12	31.84 (13.17)	12
SARIMAX	28.81 (13.79)	0	40.14 (20.47)	0	34.78 (18.34)	0
MSTL-ETS	38.28 (33.64)	25	51.02 (42.62)	12	42.16 (16.62)	12
SVMR	44.09 (11.52)	0	57.28 (18.69)	0	51.88 (22.63)	0
Naive24h	49.91 (42.31)	25	73.93 (62.75)	25	44.06 (14.26)	25

Chapter 6

Conclusions

This thesis has examined the problem of short-term heat-demand forecasting and compared a range of statistical, machine-learning and deep-learning approaches on five heat-demand series. The models were evaluated in a realistic operational framework for both day-ahead and week-ahead horizons and were benchmarked against the forecasting tools currently used within OptiEPTM.

Summary of Contributions

The contributions of this work fall into three main areas.

1. *A unified evaluation framework.* A complete and operationally consistent evaluation pipeline was developed. It included rolling cross-validation over one full year of data, horizon-specific retraining strategies, separate assessment across seasons, per-window and per-hour-ahead error analysis, residual structure diagnostics and a bootstrap-based significance procedure designed to respect temporal dependence.
2. *Implementation and tuning of statistical and deep-learning models.* MSTL-ETS, SARIMAX, LSTM and TFT models were implemented for short-term heat-demand forecasting, using horizon-specific and series-specific configurations. The TFT was applied only to series F1 and was subsequently excluded, having shown comparatively low accuracy in this single-series data regime combined with very high computation time. The LSTM underwent a multi-step tuning process, including systematic feature and decoder ablations, which provided

insight into the usefulness of different inputs and offered a principled approach for adapting the model to new series with limited tuning effort. Additionally, the foundation model Chronos-2 was used zero-shot to provide a general-purpose benchmark.

3. *A detailed comparison with company models.* The thesis provided a rigorous and quantitative comparison between the currently deployed XGBoost and SVMR models and the above mentioned models. This clarified where the existing methods perform well, where they fall short and the extent to which the evaluated statistical and deep learning models can yield robust improvements under realistic operating conditions.

Main Empirical Findings

This section summarises the main empirical findings of the final comparative evaluation. Table 6.1 provides a qualitative overview of the main models across noise regimes and forecast horizons.

Table 6.1: Qualitative model performance overview from the final evaluation. Accuracy is summarised separately for low-noise (F1–F2) and high-noise (F3–F5) series, for day-ahead and week-ahead horizons, and the remaining columns report qualitative assessments of interpretability and operational efficiency.

Model	Accuracy				Interpret.	Oper. efficiency
	Low noise		High noise			
	Day	Week	Day	Week		
LSTM	High	High	Med.	Med.	Med.–Low	Med.–Low
Ch-2	Med.	Med.–Low	High	Med.–Low	Low	Med.
SAR	Med. [†]	Med.–Low [†]	Med. [†]	Med.–Low [†]	High	Med.
XGB	Med.	Med.	Med.	Med.	Med.	High

[†] SARIMAX degrades sharply in transitional periods or under weekly seasonality.

Accuracy as a function of series difficulty. On the two most structured series (F1 and F2), the LSTM is the most reliable performer across seasons and for both horizons. At the day-ahead horizon, its advantage over all competitors is supported

by the Circular Block Bootstrap analysis on a central cold subperiod (2024-11-10 to 2025-03-17), where loss differentials are approximately stationary. Under MAE, RMSE, and sMAPE, the bootstrap confidence intervals for mean loss differentials (competitor minus LSTM) lie strictly above zero, indicating that the observed gains reflect systematic performance differences rather than sampling variability. At the same time, per-window summaries show that day-to-day fluctuations are large relative to mean gaps, so the LSTM advantage becomes clearly visible only after aggregating performance across many windows.

As series difficulty increases (F3–F5), day-ahead performance among the *plant-specific* models tends to converge. In this regime, the LSTM typically remains one of the strongest plant-specific approaches, but its margins over the best competing baselines (notably XGBoost, and in some winter settings SARIMAX) are often modest and frequently not distinguishable under the bootstrap confidence intervals.

Against this backdrop, Chronos-2 becomes more prominent. It is a close competitor to the LSTM for day-ahead forecasts on F1, but it is less competitive on F2 and tends to degrade at the week-ahead horizon, where its performance becomes more variable. Conversely, on the more difficult series (especially F4 and F5), Chronos-2 typically achieves the best day-ahead accuracy, most clearly in winter. Block-bootstrap comparisons using Chronos-2 as the reference indicate that its gains are statistically significant on these hardest series, while being weaker and not always significant on F3. This pattern is consistent with the hypothesis that large-scale pre-training and cross-series transfer can be particularly beneficial when single-series data are noisy and contain limited exploitable structure, though, as discussed in Section 5.3, Chronos-2 was pre-trained on datasets that overlap our test period in time, which could give it an unfair advantage if any of those series are related to the target plants.

Week-ahead performance. At the week-ahead horizon, the number of evaluation windows is much smaller, so seasonal summaries are more sensitive to a handful of atypical weeks. On series F1 and F2, the LSTM generally remains the most accurate model and often attains higher win rates than at the day-ahead horizon. The TFT, evaluated only on F1 due to computational cost, tends to improve in relative terms at the week-ahead horizon. These patterns are consistent with a key methodological distinction: LSTM and TFT are trained end-to-end to optimise a loss over the full forecast window, whereas the company baselines (XGBoost and SVMR) are trained in

a single-step formulation and then applied recursively, making them more exposed to error accumulation at longer horizons. The statistical baselines constitute a further case, as they are estimated by maximum likelihood rather than by directly minimising the forecasting losses reported in the evaluation.

For series F3 to F5, week-ahead results are less stable overall: LSTM and XGBoost are typically among the most accurate models, while Chronos-2 often loses its day-ahead advantage, with a notable exception in winter for F4 where Chronos-2 achieves the best performance.

Practical magnitude of the improvements. In absolute terms (MAE, RMSE), the leading model often achieves a noticeable reduction relative to the strongest alternative. In some winter day-ahead settings, the MAE reduction is on the order of 15% to 20%. This leading model is typically the LSTM on the more structured series (F1–F2) and Chronos-2 on the noisiest series (notably F4–F5). However, once errors are normalised by the series magnitude (sMAPE, and similarly NMAE for cross-series comparisons), the separation between the best and the closest competitors is usually modest. Across most settings, differences between the top models are often on the order of one to two sMAPE percentage points, and only occasionally approach roughly three points. Thus, while the best model is consistently identifiable in aggregate, the incremental improvement in relative terms is frequently modest.

Behaviour of the classical baselines. Among the plant-specific baselines, XGBoost is often the strongest competitor to the LSTM, particularly on the structured series. SARIMAX performs well in winter yet deteriorates in the shoulder seasons, consistent with its coarse regime specification and the difficulty of maintaining temporal alignment during gradual transitions. Despite this limitation, SARIMAX offers the highest interpretability among the strongest models, with a transparent decomposition into autoregressive dynamics and explicitly parameterised covariate effects, which can be valuable in operational settings. MSTL-ETS is generally ill suited to this deployment set-up: its decomposition-based forecasts adapt too slowly to short-term, temperature-driven fluctuations and the weekly retraining schedule further limits responsiveness, leading it to underperform even a Seasonal naïve baseline in most settings.

Computational costs and operational trade-offs. Computational cost introduces an additional trade-off. The company models (XGBoost and SVMR) are extremely fast to train, even relative to the statistical baselines, training in less than a

second on CPU. The LSTM training requires up to one minute on GPU, while the TFT is markedly more expensive and was therefore not evaluated beyond F1. Chronos-2 requires no training in our pipeline (zero-shot), but it exhibits the highest inference latency among the tested approaches, reflecting the cost of large-model forward passes. Weekly retraining means that training overhead is incurred infrequently for each series; however, when training is performed independently for each series and each horizon, these costs compound at scale and can amount to several additional hours per retraining cycle.

Residual diagnostics. Residual diagnostics broadly reinforce the error-based findings. Focusing on winter, the strongest models also tend to exhibit weaker residual autocorrelation, suggesting that less systematic linear temporal structure remains in their errors. At the same time, several approaches retain clear seasonal dependence at daily lags (multiples of 24 h), for example TFT on F1 and Chronos-2 on F2. Importantly, these ACF patterns are informative about residual structure, but they do not translate one-to-one into differences in average accuracy, because the remaining dependence can be of relatively small magnitude and have limited impact on aggregate error metrics.

Practical Implications

Taken together, these findings suggest the following implications for operational forecasting within OptiEPTM.

For the more structured series (F1–F2), the LSTM provides the most consistent accuracy gains over the current company benchmark. Whether this justifies replacing XGBoost in production depends on the trade-off between accuracy and operational cost.

For the noisier series (F3–F5), the day-ahead results indicate that Chronos-2 can be highly competitive and often the most accurate model, whereas at the week-ahead horizon its performance is less reliable and is frequently matched or exceeded by LSTM and XGBoost (with notable exceptions such as winter in F4). Since Chronos-2 is used zero-shot, it avoids training cost but has higher inference latency than XGBoost, so deployment again requires balancing accuracy gains against computational overhead.

Finally, to address the comparability concern discussed in Section 5.3, an op-

erational safeguard is to re-run the evaluation on a more recent holdout period (e.g., winter 2025/2026), which removes any potential temporal overlap between the original test window and Chronos-2’s pretraining data.

Limitations

A few aspects should be kept in mind when interpreting the results and translating them to deployment.

First, the evaluation uses realised (historical) temperature rather than operational temperature forecasts. In practice, forecast errors in temperature will propagate into demand forecasts, especially at the week-ahead horizon, and relative model robustness to such input noise was not assessed here.

Second, model comparisons are affected by differences in optimisation objectives and multi-step forecast construction. LSTM and TFT are trained end-to-end to minimise a loss over the full forecast window, whereas XGBoost and SVMR are trained one-step ahead and then rolled out recursively to obtain multi-step predictions, which can exacerbate error accumulation at longer horizons. Statistical baselines are fitted by maximum likelihood rather than by directly minimising the reported error metrics. These differences may influence rankings, particularly at longer horizons.

Third, Chronos-2 was released in October 2025, and parts of its pretraining corpus overlaps in time with the evaluation period. Our audit suggests leakage risk is likely marginal, but it cannot be fully excluded; re-running the evaluation on a more recent holdout period would remove this concern.

Finally, reported CPU runtimes are indicative rather than strictly comparable for the company models (XGBoost and SVMR) against the others, since benchmarks were obtained across two different hardware environments with different CPU resources.

The last three issues are discussed in more detail in [Section 5.3](#).

Future Work

Several promising directions for further research emerge from this study. Incorporating temperature-forecast uncertainty, either through probabilistic weather inputs or explicit modelling of forecast errors, would provide a more realistic assessment of

operational performance.

A second direction is predictive uncertainty quantification. The current analysis focuses on point forecasts, but many operational decisions require information about forecast confidence or risk. Techniques such as bootstrap-based predictive intervals, quantile regression, Bayesian approaches, or probabilistic sequence models could provide uncertainty bands that reflect both data noise and model variability, offering operators a fuller picture of forecast reliability across horizons and regimes.

Model design also offers clear opportunities. Global training across multiple related series and transfer-learning approaches could help neural models exploit shared structure and mitigate the limited data available for individual sites. The particularly strong zero-shot performance of Chronos-2 on the most challenging series (F4–F5) suggests that, when signal is weak, leveraging information from related series can be more effective than purely plant-specific learning.

Finally, deployment-focused improvements could be explored. These include assessing whether training on the full historical record is necessary or whether comparable performance can be achieved using only the most recent years of data, as well as adopting tuning strategies that prioritise simplicity and computation time over marginal accuracy gains. Hybrid architectures that combine classical models with recurrent components, such as SARIMA–LSTM or XGBoost–LSTM hybrids, may offer a favourable balance between interpretability, speed, and accuracy.

Final Remarks

In conclusion, this thesis has shown that short-term heat-demand forecasting benefits from model choices that reflect both series difficulty and forecast horizon. In our evaluation, the LSTM emerges as the most reliable option on the more structured series, while Chronos-2 can be highly competitive on the most challenging series, particularly for day-ahead forecasts. From a deployment perspective, these accuracy gains should be balanced against practical constraints such as computation, scalability, and interpretability. Overall, the proposed evaluation framework and the empirical findings provide a sound basis for improving OptiEPTM’s forecasting module and for motivating further investigation into deep-learning based approaches to heat-demand forecasting.

Appendix A

Training and Inference Times

In this appendix, we report detailed training and inference times per model. In all tables, each value reports the mean with the standard deviation in parentheses, computed over 8 runs after 2 warm-up runs, except for the TFT on the CPU, where only 3 post-warm-up runs were used to reduce computational cost (where applicable). All models were trained on data up to, but not including, 2025-01-01, with in-sample lengths determined individually according to the tuning results.

Training and inference times for Naive24h, XGBoost, SVMR, and Chronos-2 are reported only for series F1, as these times do not vary materially across series. TFT times are reported only for F1 because the model was not developed for the other series, as described in the main text.

Table A.1: Training and inference times for all models on series F1, for both the day-ahead prediction task and the week-ahead one. Each value reports the mean with the standard deviation in parentheses, computed over 8 runs after 2 warm-up runs, except for the TFT on the CPU, where only 3 post-warm-up runs were used to reduce computational cost.

DAY-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
Naive24h	– (–)	– (–)	8.34 (0.53)	– (–)
XGBoost	< 1 (–)	– (–)	– (–)	– (–)
SVMR	< 1 (–)	– (–)	– (–)	– (–)
MSTL-ETS	8.79 (0.09)	– (–)	25.11 (1.64)	– (–)
SARIMAX	48.65 (0.52)	– (–)	180.56 (35.56)	– (–)
LSTM	264.20 (14.77)	61.90 (0.75)	5.26 (0.28)	3.62 (0.67)
TFT	1679.16 (6.25)	761.74 (2.88)	64.40 (2.24)	87.87 (3.24)
Chronos-2	– (–)	– (–)	6288.34 (330.17)	487.52 (56.36)
WEEK-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
Naive24h	– (–)	– (–)	16.67 (1.46)	– (–)
XGBoost	< 1 (–)	– (–)	– (–)	– (–)
SVMR	< 1 (–)	– (–)	– (–)	– (–)
MSTL-ETS	8.59 (0.08)	– (–)	38.97 (0.97)	– (–)
SARIMAX	49.01 (0.82)	– (–)	221.07 (82.24)	– (–)
LSTM	385.44 (3.97)	68.17 (0.59)	9.75 (0.55)	5.73 (0.29)
TFT	1786.28 (2.28)	716.61 (0.57)	81.40 (1.42)	85.69 (4.14)
Chronos-2	– (–)	– (–)	7207.97 (498.20)	444.64 (63.50)

Table A.2: Training and inference times for selected models on series F2, for both the day-ahead prediction task and the week-ahead one. Each value reports the mean with the std in parentheses, computed over 8 runs after 2 warm-up runs.

DAY-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	5.61 (0.05)	– (–)	25.89 (2.27)	– (–)
SARIMAX	55.72 (0.95)	– (–)	201.51 (57.43)	– (–)
LSTM	133.15 (2.01)	33.70 (0.25)	4.43 (0.48)	3.44 (0.13)
WEEK-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	4.13 (0.06)	– (–)	38.15 (1.80)	– (–)
SARIMAX	55.76 (0.49)	– (–)	214.71 (81.52)	– (–)
LSTM	454.46 (8.64)	62.77 (0.45)	7.71 (0.36)	6.30 (0.44)

Table A.3: Training and inference times for selected models on series F3, for both the day-ahead prediction task and the week-ahead one. Each value reports the mean with the std in parentheses, computed over 8 runs after 2 warm-up runs.

DAY-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	8.52 (0.10)	– (–)	25.47 (1.23)	– (–)
SARIMAX	42.06 (0.30)	– (–)	203.72 (1.27)	– (–)
LSTM	102.06 (1.05)	51.40 (0.71)	4.00 (0.37)	3.33 (0.12)
WEEK-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	8.39 (0.05)	– (–)	51.60 (36.64)	– (–)
SARIMAX	41.81 (0.29)	– (–)	215.91 (4.67)	– (–)
LSTM	161.29 (1.52)	58.48 (0.35)	6.49 (0.31)	5.55 (0.15)

Table A.4: Training and inference times for selected models on series F4, for both the day-ahead prediction task and the week-ahead one. Each value reports the mean with the std in parentheses, computed over 8 runs after 2 warm-up runs.

DAY-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	7.00 (0.05)	– (–)	24.90 (2.07)	– (–)
SARIMAX	5.62 (0.08)	– (–)	190.36 (68.40)	– (–)
LSTM	61.51 (0.83)	41.78 (0.84)	4.02 (0.09)	3.40 (0.10)
WEEK-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	5.54 (0.05)	– (–)	37.76 (1.26)	– (–)
SARIMAX	5.60 (0.06)	– (–)	211.05 (80.98)	– (–)
LSTM	52.98 (1.29)	21.60 (0.10)	6.73 (0.39)	5.56 (0.06)

Table A.5: Training and inference times for selected models on series F5, for both the day-ahead prediction task and the week-ahead one. Each value reports the mean with the std in parentheses, computed over 8 runs after 2 warm-up runs.

DAY-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	4.09 (0.07)	– (–)	26.37 (1.84)	– (–)
SARIMAX	55.72 (0.44)	– (–)	203.59 (58.41)	– (–)
LSTM	96.90 (1.55)	64.84 (1.43)	3.53 (0.15)	3.32 (0.10)
WEEK-AHEAD				
Model	Train Time (s)		Inference Time (ms)	
	CPU	GPU	CPU	GPU
MSTL-ETS	8.28 (0.03)	– (–)	38.38 (1.44)	– (–)
SARIMAX	55.61 (0.33)	– (–)	210.58 (81.64)	– (–)
LSTM	36.62 (0.33)	18.55 (0.25)	7.25 (0.72)	5.58 (0.20)

Appendix B

Chronos-2 Pretraining Data Audit

In this appendix, we report, for each real dataset used in Chronos-2 pretraining (cf. Table 6 in Ansari et al. [1]), the year of the last available observation and a high-level description. From a leakage perspective, the relevant datasets are those whose coverage extends to 2024 or later. However, based on their content and provenance, these datasets appear unlikely to have a meaningful relationship with our target series, which represent heat consumption for residential areas, industrial areas, or public buildings in Italy.

Table B.1: Real datasets cited for Chronos-2 pretraining (dates as in our inspection).

Dataset		End year	Description (high-level)
Electricity		2014	Electricity consumption time series for multiple customers in Portugal.
KDD Cup (2018)		2018	Hourly air-quality measurements collected at monitoring stations located in Beijing and London.
M4		2018	Univariate time series collection from the M4 forecasting competition, spanning multiple domains and sampling frequencies.
Mexico City Bikes		2026	Operational demand data from a bike-sharing system in Mexico City (service usage over time).

Continued on next page

Dataset	End year	Description (high-level)
Melbourne Pedestrian Counts	2026	Hourly pedestrian counts recorded by a city-wide network of sensors in Melbourne.
Solar	2006	Synthetic U.S. solar photovoltaic plant generation and forecast data for 2006.
Taxi	2025	Taxi demand time series for New York City (trip activity aggregated over time).
UberTLC	2015	A public NYC dataset of ride-hailing pickups (Uber and similar services) released via NYC TLC records requests.
USHCN	2026	Station-based climate records (temperature and precipitation) from the U.S. Historical Climatology Network (USHCN), collected at weather stations across the 48 contiguous United States (CONUS), that is, all U.S. states except Alaska and Hawaii.
WeatherBench	2018	A cleaned subset of ERA5 (historical global weather data) used to train and test machine-learning models for global weather forecasting.
Wiki	2026	Daily Wikipedia page-view time series (web traffic / popularity dynamics).
Wind Farms	2020	Wind power production time series from multiple wind farms in Australia (generation traces over time).
Temperature–Rain	2017	Weather station time series including temperature and rainfall-related measurements (Australia-wide coverage).
London Smart Meters	2014	Half-hourly household electricity consumption readings from London smart meters (large customer panel).

Continued on next page

Dataset	End year	Description (high-level)
Alibaba Cluster Trace (2018)	2018	Data-centre workload traces (jobs/tasks and resource usage) used for systems and capacity forecasting benchmarks.
Azure VM Traces (2017)	2017	Virtual machine workload traces from Microsoft Azure (telemetry for capacity and demand forecasting).
Borg Cluster Data (2011)	2011	Cluster workload traces from Google’s Borg ecosystem, used in resource-usage forecasting studies.
LargeST (2017)	2017	Large-scale California traffic benchmark with thousands of road sensors, providing spatio-temporal traffic time series and sensor metadata.
Q-Traffic	2017	Beijing road-network traffic dataset from Baidu Map, providing road-segment traffic speed time series plus road-network structure and direction-query auxiliary data.
Buildings 900K	–	Synthetic building energy time series from simulated buildings (BuildingsBench) ¹

¹ Although Table 6 in Ansari et al. [1] labels all entries as “real univariate datasets”, Buildings 900K is in fact a simulated (synthetic) dataset as described in Emami et al. [22].

Appendix C

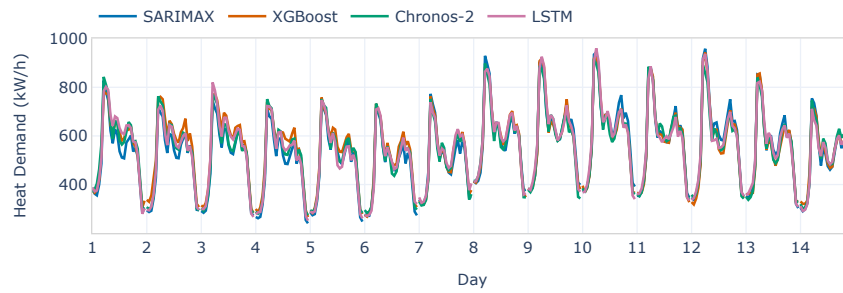
Illustrative Winter Forecasts

This appendix provides a visual illustration of typical forecasts during winter, the most critical period for heat-demand forecasting. The aim is simply to give a visual impression of model behaviour over a short two-week excerpt. We show example forecasts from the best-performing models studied in this thesis, together with the strongest company benchmark, XGBoost. Alongside the forecasts, we plot the residuals (defined as prediction minus ground truth) to assess how closely predictions match the ground truth.

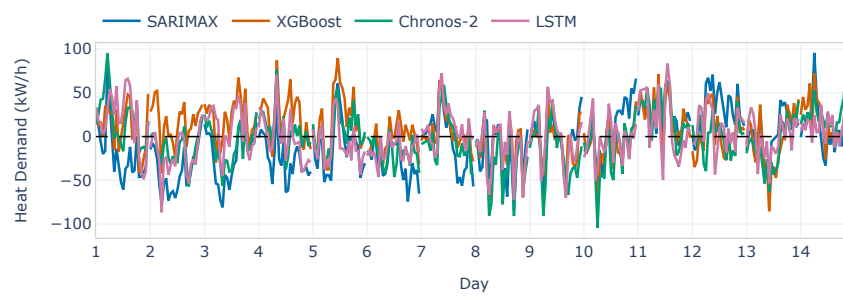
For each target series, Figures C.1 to C.5 report, over the same two-week winter excerpt,

- forecasts and residuals for the day-ahead prediction task (Subfigures a.–b.);
- forecasts and residuals for the week-ahead prediction task (Subfigures c.–d.).

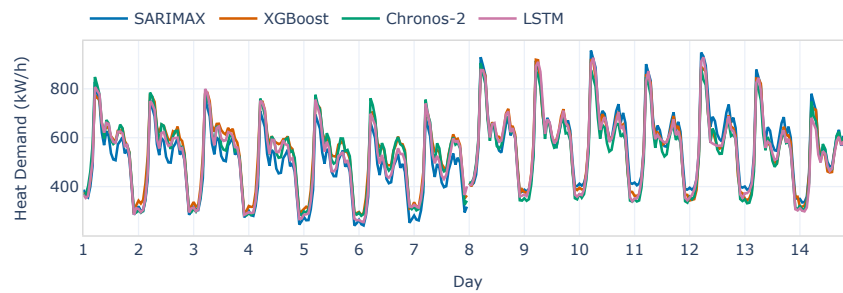
The x-axis is shown in relative day units, since the purpose is illustrative rather than tied to specific dates. In the day-ahead setting, forecasts are issued at midnight each day, corresponding to the vertical grid lines. Trained models are retrained once per week in accordance with the schedule described in Section 5.2, that is, immediately before the first forecast day of each week within the excerpt (days 1 and 8). In the week-ahead setting, forecasts are issued once per week, at midnight at the start of those same days.



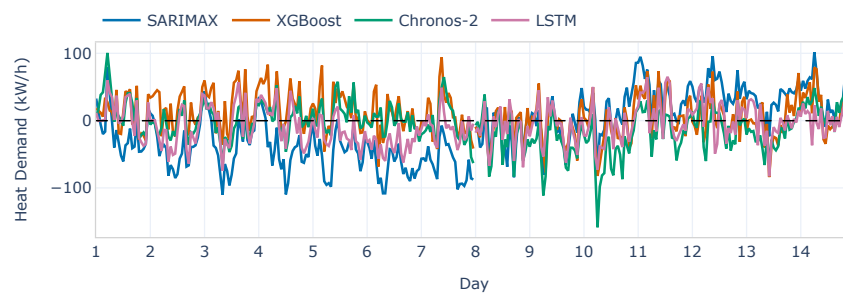
(a) Day-ahead forecasts



(b) Day-ahead residuals

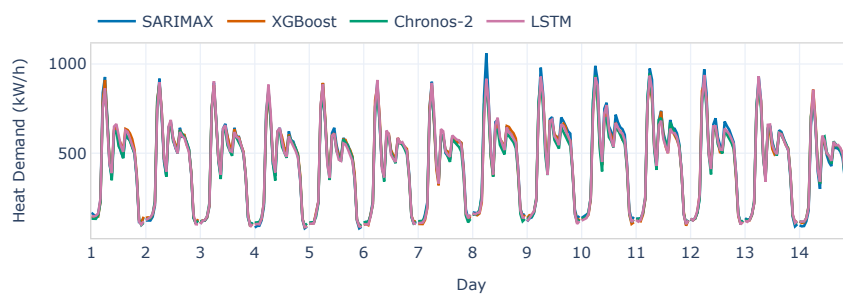


(c) Week-ahead forecasts

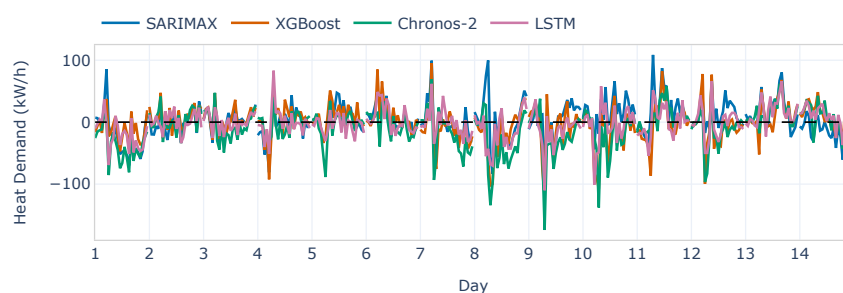


(d) Week-ahead residuals

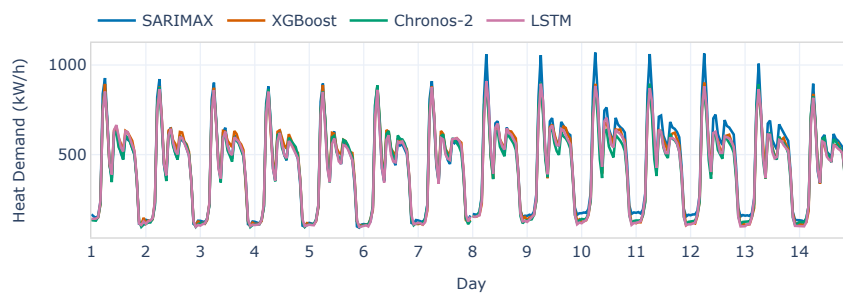
Figure C.1: Two-week winter excerpt for series F1: day-ahead forecasts (a) and residuals (b); week-ahead forecasts (c) and residuals (d).



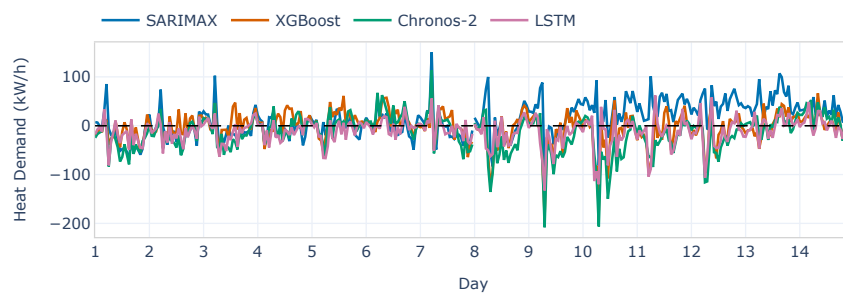
(a) Day-ahead forecasts



(b) Day-ahead residuals

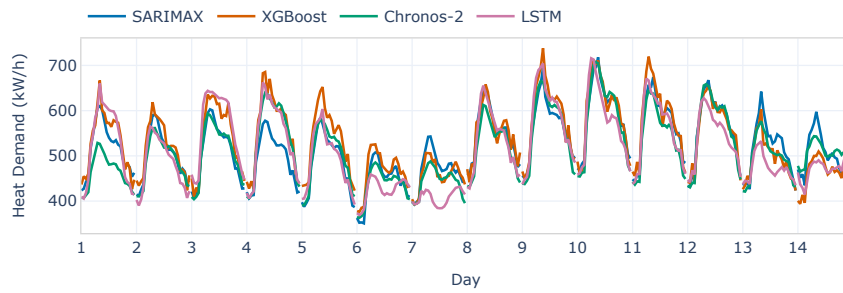


(c) Week-ahead forecasts

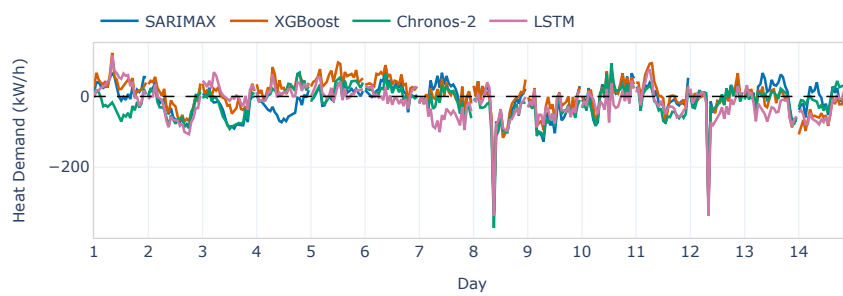


(d) Week-ahead residuals

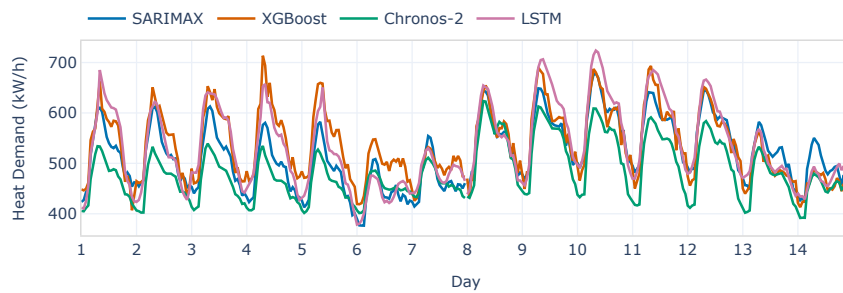
Figure C.2: Two-week winter excerpt for series F2: day-ahead forecasts (a) and residuals (b); week-ahead forecasts (c) and residuals (d).



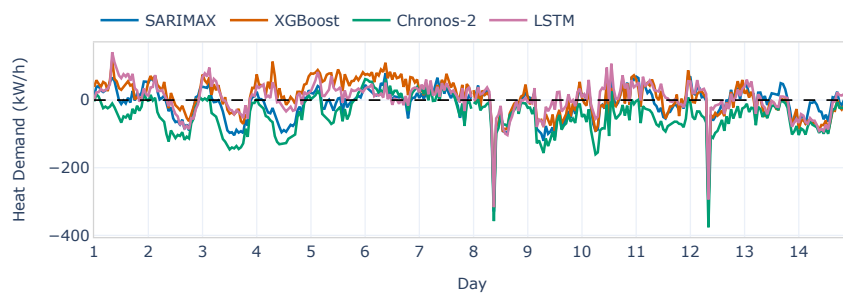
(a) Day-ahead forecasts



(b) Day-ahead residuals

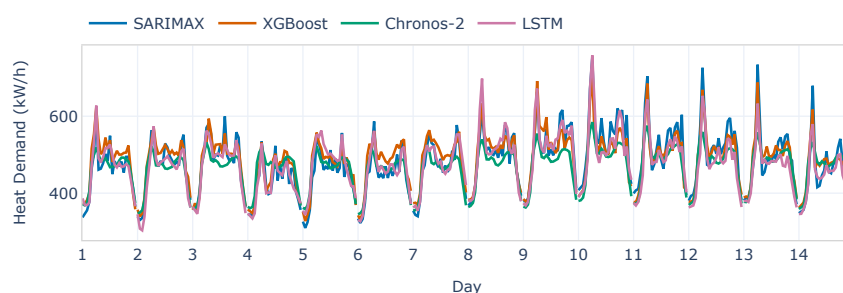


(c) Week-ahead forecasts

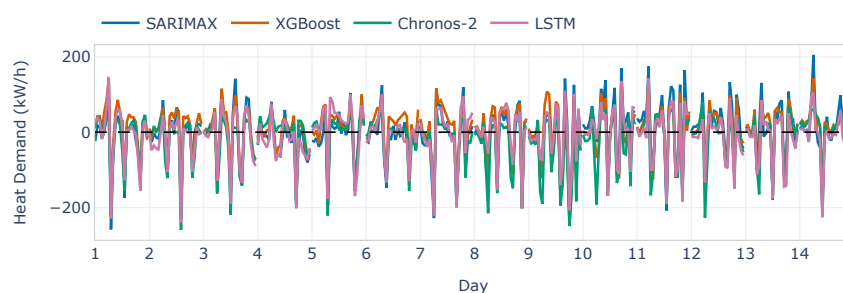


(d) Week-ahead residuals

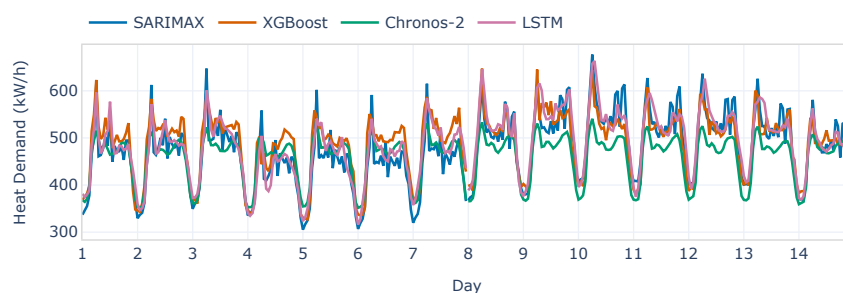
Figure C.3: Two-week winter excerpt for series F3: day-ahead forecasts (a) and residuals (b); week-ahead forecasts (c) and residuals (d).



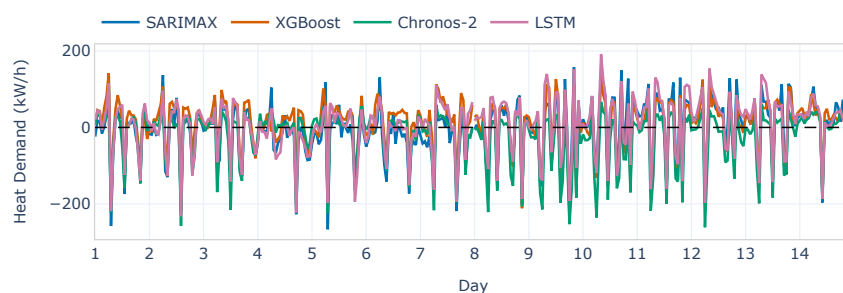
(a) Day-ahead forecasts



(b) Day-ahead residuals

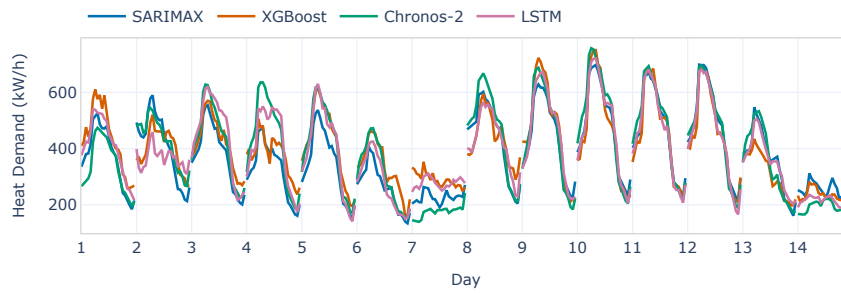


(c) Week-ahead forecasts

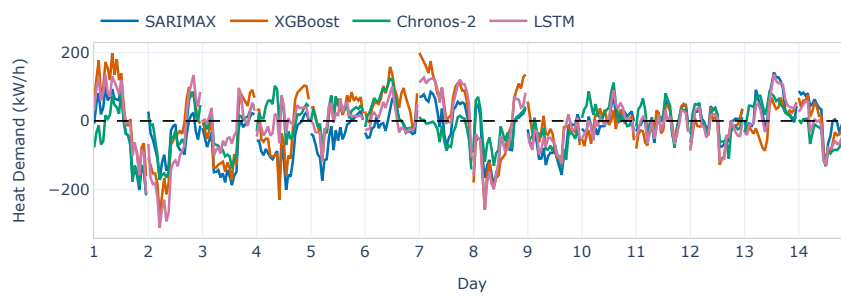


(d) Week-ahead residuals

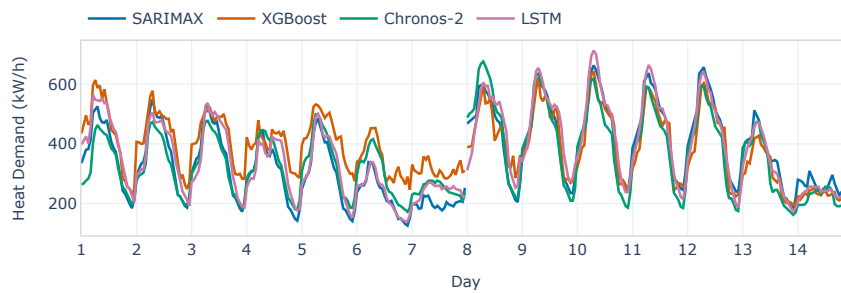
Figure C.4: Two-week winter excerpt for series F4: day-ahead forecasts (a) and residuals (b); week-ahead forecasts (c) and residuals (d).



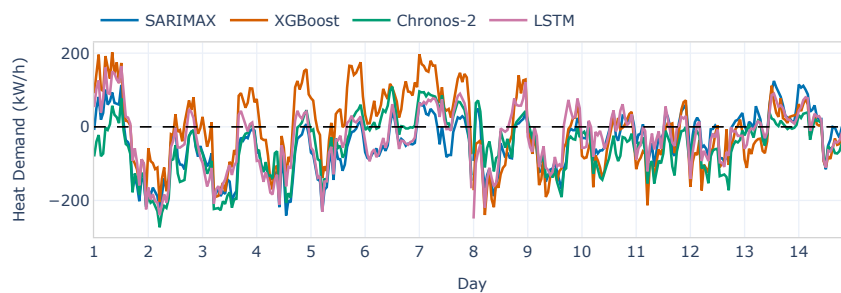
(a) Day-ahead forecasts



(b) Day-ahead residuals



(c) Week-ahead forecasts



(d) Week-ahead residuals

Figure C.5: Two-week winter excerpt for series F5: day-ahead forecasts (a) and residuals (b); week-ahead forecasts (c) and residuals (d).

Appendix D

Failure of the iid Bootstrap under Temporal Dependence

In this appendix, we discuss what happens when the iid bootstrap is applied to data exhibiting temporal dependence. We mainly follow Lahiri [37] (Sections 2.2–2.3) and Singh [54].

Consider a sequence of random variables $\{X_k\}_{k \in \mathbb{Z}}$ defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Suppose first that the variables are iid with common distribution F , and let $\mathcal{X}_n = \{X_1, \dots, X_n\}$ denote the observed sample. Let

$$T_n = t_n(\mathcal{X}_n; F)$$

be a statistic of interest, and let G_n denote its distribution.

For iid data, the bootstrap method introduced by Efron [20] provides a way of approximating G_n . To this end, consider a bootstrap sample $\mathcal{X}_m^* = \{X_1^*, \dots, X_m^*\}$ obtained by sampling with replacement from $\mathcal{X}_n$. More precisely, let $I_1, \dots, I_m$ be iid random variables with uniform distribution on $\{1, \dots, n\}$, and define

$$X_k^* = X_{I_k} = \sum_{i=1}^n X_i \mathbb{1}(I_k = i), \quad k = 1, \dots, m,$$

where $\mathbb{1}(\cdot)$ denotes the indicator function, equal to one when its argument is true and zero otherwise. Conditionally on the observed data, the variables $X_1^*, \dots, X_m^*$ are

iid with common distribution

$$F_n = \frac{1}{n} \sum_{i=1}^n \delta_{X_i}, \quad (\text{D.1})$$

where F_n is the empirical distribution and δ_c denotes the Dirac measure at c . The bootstrap principle is then to approximate G_n by the conditional distribution of

$$T_{m,n}^* = t_m(\mathcal{X}_m^*, F_n).$$

We now focus on the centred and scaled sample mean,

$$T_n = \sqrt{n}(\bar{X}_n - \mu),$$

where

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i, \quad \mu = \mathbb{E}[X_1].$$

The bootstrap analogue of this statistic is

$$T_{m,n}^* = \sqrt{m}(\bar{X}_m^* - \mathbb{E}_*[X_1^*]), \quad (\text{D.2})$$

where $\mathbb{E}_*$ denotes conditional expectation given $\mathcal{X}_n$, that is, $\mathbb{E}_*[\cdot] = \mathbb{E}[\cdot \mid \mathcal{X}_n]$. From (D.1) it follows that

$$\mathbb{E}_*[X_1^*] = \frac{1}{n} \sum_{i=1}^n X_i = \bar{X}_n,$$

so that (D.2) may be written as

$$T_{m,n}^* = \sqrt{m}(\bar{X}_m^* - \bar{X}_n).$$

For simplicity, we adopt the common choice $m = n$, yielding

$$T_{n,n}^* = \sqrt{n}(\bar{X}_n^* - \bar{X}_n).$$

If $\mathbb{E}[X_1^2] < \infty$, it can be shown [54] that

$$\sup_{x \in \mathbb{R}} \left| \mathbb{P}(T_n \leq x) - \mathbb{P}_*(T_{n,n}^* \leq x) \right| \rightarrow 0 \quad \text{almost surely as } n \rightarrow \infty,$$

where $\mathbb{P}_*$ denotes conditional probability given $\mathcal{X}_n$. Thus, in the iid case, the bootstrap distribution consistently approximates the sampling distribution of the statistic. A standard proof [54] uses the Berry–Esseen theorem together with the Marcinkiewicz–Zygmund strong law of large numbers (see, for example, Appendix A of Lahiri [37] for statements of these results) to establish first that

$$\sup_{x \in \mathbb{R}} |\mathbb{P}_*(T_{n,n}^* \leq x) - \Phi(x/\sigma)| \rightarrow 0 \quad \text{almost surely,}$$

where Φ denotes the standard normal cumulative distribution function and $\sigma^2 = \text{Var}(X_1)$. On the other hand, the classical central limit theorem yields

$$\sqrt{n}(\bar{X}_n - \mu) \Rightarrow \mathcal{N}(0, \sigma^2),$$

where $\Rightarrow$ denotes convergence in distribution. Since the limiting distribution is continuous, this implies uniform convergence of the corresponding distribution functions [51], namely

$$\sup_{x \in \mathbb{R}} |\mathbb{P}(T_n \leq x) - \Phi(x/\sigma)| \rightarrow 0.$$

The result then follows by the triangle inequality.

Suppose now instead that $\{X_k\}_{k \in \mathbb{Z}}$ is strictly stationary and M -dependent for some $M \geq 1$, meaning that for every integer j , the two sets of random variables $\{X_k\}_{k \leq j}$ and $\{X_k\}_{k \geq j+M+1}$ are independent. Assume again that $\mathbb{E}[X_1^2] < \infty$. In this case, the central limit theorem for M -dependent sequences (see again Appendix A of Lahiri [37]) states that

$$\sqrt{n}(\bar{X}_n - \mu) \Rightarrow \mathcal{N}(0, \sigma_\infty^2), \tag{D.3}$$

where the long-run variance is

$$\sigma_\infty^2 = \text{Var}(X_1) + 2 \sum_{k=1}^M \text{Cov}(X_1, X_{1+k}).$$

Since the limiting distribution is again continuous, (D.3) implies

$$\sup_{x \in \mathbb{R}} |\mathbb{P}(T_n \leq x) - \Phi(x/\sigma_\infty)| \rightarrow 0.$$

By contrast, conditionally on the observed data, the bootstrap resample consists of iid draws from F_n , even though the original observations are not independent. Hence, by an argument similar in spirit to the one above, although not identical (see Lahiri [37], Section 2.3), one can show that

$$\sup_{x \in \mathbb{R}} |\mathbb{P}_*(T_{n,n}^* \leq x) - \Phi(x/\sigma)| \rightarrow 0 \quad \text{almost surely.} \quad (\text{D.4})$$

That is, the iid bootstrap continues to approximate the distribution that would arise under independence, with variance $\sigma^2 = \text{Var}(X_1)$, rather than the correct long-run variance σ_∞^2 .

Combining the two uniform convergence results, it follows from the reverse triangle inequality for the sup norm that

$$\begin{aligned} & \left| \sup_{x \in \mathbb{R}} |\mathbb{P}(T_n \leq x) - \mathbb{P}_*(T_{n,n}^* \leq x)| - \sup_{x \in \mathbb{R}} |\Phi(x/\sigma_\infty) - \Phi(x/\sigma)| \right| \\ & \leq \sup_{x \in \mathbb{R}} |\mathbb{P}(T_n \leq x) - \Phi(x/\sigma_\infty)| + \sup_{x \in \mathbb{R}} |\mathbb{P}_*(T_{n,n}^* \leq x) - \Phi(x/\sigma)|, \end{aligned}$$

and therefore

$$\sup_{x \in \mathbb{R}} |\mathbb{P}(T_n \leq x) - \mathbb{P}_*(T_{n,n}^* \leq x)| \rightarrow \sup_{x \in \mathbb{R}} |\Phi(x/\sigma_\infty) - \Phi(x/\sigma)| \quad \text{almost surely.}$$

Therefore, whenever

$$\sum_{k=1}^M \text{Cov}(X_1, X_{1+k}) \neq 0,$$

we have $\sigma_\infty^2 \neq \sigma^2$, and hence

$$\sup_{x \in \mathbb{R}} |\Phi(x/\sigma_\infty) - \Phi(x/\sigma)| > 0.$$

This shows that, in general, applying the iid bootstrap to stationary M -dependent data does not yield a consistent approximation to the sampling distribution of the sample mean, and may therefore lead to incorrect statistical inference. In particular, (D.3) and (D.4) show that the iid bootstrap mis-estimates the asymptotic variability of the sample mean, so bootstrap procedures based on it may produce confidence intervals that are too narrow or too wide. If the serial covariances are predominantly positive, the iid bootstrap will tend to underestimate the variability of the sample

mean and therefore produce confidence intervals that are too narrow. Conversely, if the serial covariances are predominantly negative, it will tend to overestimate that variability and thus produce confidence intervals that are too wide.

Example (a simple 1-dependent process). Let $\{\varepsilon_t\}_{t \in \mathbb{Z}}$ be iid with

$$\mathbb{E}[\varepsilon_t] = 0, \quad \text{Var}(\varepsilon_t) = \sigma_\varepsilon^2 < \infty,$$

and define

$$X_t = \mu + \varepsilon_t + \frac{\eta}{2}\varepsilon_{t-1}, \quad \mu \in \mathbb{R}, \quad \eta \in \{\pm 1\}.$$

Then $(X_t - \mu)$ is a moving average process with iid innovations and is therefore strictly stationary; see, for example, Brockwell and Davis [11, Ch. 2]. Moreover, $\{X_t\}$ is 1-dependent, since each X_t depends only on ε_t and ε_{t-1} .

The marginal variance is

$$\sigma^2 = \text{Var}(X_t) = \text{Var}\left(\varepsilon_t + \frac{\eta}{2}\varepsilon_{t-1}\right) = \frac{5}{4}\sigma_\varepsilon^2.$$

At lag 1, the autocovariance is

$$\text{Cov}(X_t, X_{t+1}) = \text{Cov}\left(\varepsilon_t + \frac{\eta}{2}\varepsilon_{t-1}, \varepsilon_{t+1} + \frac{\eta}{2}\varepsilon_t\right) = \frac{\eta}{2}\sigma_\varepsilon^2.$$

Thus, if $\eta = 1$, consecutive observations are positively correlated, whereas if $\eta = -1$, they are negatively correlated. Intuitively, if $\eta = 1$, then ε_t contributes with the same sign to both X_t and X_{t+1} , so the two are more likely to lie on the same side of μ ; the opposite holds if $\eta = -1$. Since the process is 1-dependent, all autocovariances at lags $h \geq 2$ vanish, and hence the long-run variance is

$$\sigma_\infty^2 = \text{Var}(X_t) + 2 \text{Cov}(X_t, X_{t+1}) = \frac{5}{4}\sigma_\varepsilon^2 + \eta\sigma_\varepsilon^2 = \left(\frac{5}{4} + \eta\right)\sigma_\varepsilon^2.$$

Therefore,

$$\sqrt{n}(\bar{X}_n - \mu) \Rightarrow \mathcal{N}(0, \sigma_\infty^2),$$

with

$$\sigma_\infty^2 = \begin{cases} 9\sigma_\varepsilon^2/4, & \text{if } \eta = 1, \\ \sigma_\varepsilon^2/4, & \text{if } \eta = -1. \end{cases}$$

By contrast, the iid bootstrap treats the observations as independent, and is therefore calibrated to the marginal variance

$$\sigma^2 = \text{Var}(X_t) = \frac{5}{4}\sigma_\varepsilon^2,$$

rather than the long-run variance σ_∞^2 . Consequently, it underestimates the asymptotic variance by a factor of 5/9 when $\eta = 1$, and overestimates it by a factor of 5 when $\eta = -1$.

Bibliography

- [1] Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, et al. Chronos-2: From univariate to universal forecasting. *arXiv preprint arXiv:2510.15821*, 2025. URL <https://arxiv.org/abs/2510.15821>.
- [2] Craig F. Ansley. An algorithm for the exact likelihood of a mixed autoregressive-moving average process. *Biometrika*, 66(1):59–65, April 1979. doi: 10.1093/biomet/66.1.59. URL <https://doi.org/10.1093/biomet/66.1.59>. JSTOR stable identifier: 2335242 (often cited as doi:10.2307/2335242).
- [3] George De Ath, Richard M. Everson, Alma A. M. Rahat, and Jonathan E. Fieldsend. Greed is good: Exploration and exploitation trade-offs in bayesian optimisation. *ACM Transactions on Evolutionary Learning and Optimization*, 1(1): 122, 2021. doi: 10.1145/3425501. URL <https://dl.acm.org/doi/10.1145/3425501>.
- [4] Kasun Bandara, Rob J. Hyndman, and Christoph Bergmeir. Mstl: a seasonal-trend decomposition algorithm for time series with multiple seasonal patterns. *International Journal of Operational Research*, 52(1):79–98, 2025. doi: 10.1504/IJOR.2025.143957. URL <https://www.inderscience.com/info/inarticle.php?artid=143957>.
- [5] Souhaib Ben Taieb, Gianluca Bontempi, Amir F. Atiya, and Antti Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition. *Expert Systems with Applications*, 39(8):7067–7083, 2012. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2012.01.039>. URL <https://www.sciencedirect.com/science/article/pii/S0957417412000528>.

- [6] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*, NIPS'15, pages 1171–1179, Cambridge, MA, USA, 2015. MIT Press. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/e995f98d56967d946471af29d7bf99f1-Paper.pdf.
- [7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.
- [8] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, USA, 2011. Curran Associates Inc. ISBN 978-1-61839-599-3. URL <http://dl.acm.org/citation.cfm?id=2986459.2986743>.
- [9] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–252, 1964. doi: 10.1111/j.2517-6161.1964.tb00553.x. URL <https://rss.onlinelibrary.wiley.com/doi/10.1111/j.2517-6161.1964.tb00553.x>.
- [10] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *ArXiv*, abs/1012.2599, 2010. URL <https://api.semanticscholar.org/CorpusID:1640103>.
- [11] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer International Publishing, Cham, 3rd edition, 2016. ISBN 978-3-319-29852-8. doi: 10.1007/978-3-319-29854-2.
- [12] Joseph E. Cavanaugh and Andrew A. Neath. The akaike information criterion: Background, derivation, properties, application, interpretation, and refinements. *WIREs Computational Statistics*, 11:e1460, 2019. doi: 10.1002/wics.1460. URL <https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.1460>.
- [13] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge*

- Discovery and Data Mining*, KDD '16, page 785794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. URL <https://doi.org/10.1145/2939672.2939785>.
- [14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL <https://aclanthology.org/D14-1179/>.
- [15] Oscar Claveria, Enric Monte, and Salvador Torra. Data pre-processing for neural network-based forecasting: does it really matter? *Technological and Economic Development of Economy*, 23(5):709–725, 2017. doi: 10.3846/20294913.2015.1070772. URL <https://doi.org/10.3846/20294913.2015.1070772>.
- [16] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–33, 1990. URL <https://www.scb.se/contentassets/ca21efb41fee47d293bbee5bf7be7fb3/stl-a-seasonal-trend-decomposition-procedure-based-on-loess.pdf>.
- [17] William S. Cleveland. Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, December 1979. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.1979.10481038. URL <http://www.tandfonline.com/doi/abs/10.1080/01621459.1979.10481038>.
- [18] Francis X. Diebold. Exact maximum-likelihood estimation of autoregressive models via the Kalman filter. *Economics Letters*, 22(2):197–201, 1986. ISSN 0165-1765. doi: 10.1016/0165-1765(86)90231-4. URL <https://www.sciencedirect.com/science/article/pii/0165176586902314>.
- [19] Grzegorz Dudek, Pawe Piotrowski, and Dariusz Baczyski. Forecasting in modern power systems: Challenges, techniques, and emerging trends. *En-*

- ergies, 18(14), 2025. ISSN 1996-1073. doi: 10.3390/en18143589. URL <https://www.mdpi.com/1996-1073/18/14/3589>.
- [20] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. ISSN 00905364, 21688966. URL <http://www.jstor.org/stable/2958830>.
- [21] Graham Elliott, Thomas J. Rothenberg, and James H. Stock. Efficient tests for an autoregressive unit root. *Econometrica*, 64(4):813–836, 1996. doi: 10.2307/2171846. URL <https://www.jstor.org/stable/2171846>.
- [22] Patrick Emami, Abhijeet Sahu, and Peter Graf. Buildingsbench: a large-scale dataset of 900k buildings and benchmark for short-term load forecasting. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS ’23, Red Hook, NY, USA, 2023. Curran Associates Inc.
- [23] Kun Fan, Chungin Joung, and Seungjun Baek. Sequence-to-sequence video prediction by learning hierarchical representations. *Applied Sciences*, 10(22):8288, 2020. doi: 10.3390/app10228288. URL <https://www.mdpi.com/2076-3417/10/22/8288>.
- [24] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 10 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL <https://doi.org/10.1162/089976600300015015>.
- [25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. ISBN 978-0-262-03561-3. URL <https://www.deeplearningbook.org/>. Chapter 10 provides a comprehensive overview of recurrent neural networks, including training methods and theoretical foundations.
- [26] Paul Goodwin and Richard Lawton. On the asymmetry of the symmetric mape. *International Journal of Forecasting*, 15(4):405–408, 1999. ISSN 0169-2070. doi: [https://doi.org/10.1016/S0169-2070\(99\)00007-2](https://doi.org/10.1016/S0169-2070(99)00007-2). URL <https://www.sciencedirect.com/science/article/pii/S0169207099000072>.
- [27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>.

- [28] Giles Hooker. Generalized functional anova diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3):709–732, 2007. ISSN 10618600. URL <http://www.jstor.org/stable/27594267>.
- [29] Clifford M. Hurvich and Chih-Ling Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989. doi: 10.1093/biomet/76.2.297. URL <https://doi.org/10.1093/biomet/76.2.297>.
- [30] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Beijing, China, 22–24 Jun 2014. PMLR. URL <https://proceedings.mlr.press/v32/hutter14.html>.
- [31] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2nd edition, 2018. URL <https://otexts.com/fpp2/>. Online textbook.
- [32] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 27(3):1–22, 2008. doi: 10.18637/jss.v027.i03. URL <https://www.jstatsoft.org/v27/i03>.
- [33] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. ISSN 0169-2070. doi: <https://doi.org/10.1016/j.ijforecast.2006.03.001>. URL <https://www.sciencedirect.com/science/article/pii/S0169207006000239>.
- [34] Herbert Jaeger. Tutorial on training recurrent neural networks, covering bptt, rtrl, ekf and the "echo state network" approach. Technical Report GMD Report 159, German National Research Center for Information Technology (GMD), Sankt Augustin, Germany, 2002. URL <https://www.ai.rug.nl/minds/uploads/ESNTutorialRev.pdf>.
- [35] Wenfu Ku, Robert H. Storer, and Christos Georgakis. Disturbance detection and isolation by dynamic principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 30(1):179–196, 1995. ISSN 0169-7439. doi:

- [https://doi.org/10.1016/0169-7439\(95\)00076-3](https://doi.org/10.1016/0169-7439(95)00076-3). URL <https://www.sciencedirect.com/science/article/pii/S0169743995000763>. In CINC '94 Selected papers from the First International Chemometrics Internet Conference.
- [36] Denis Kwiatkowski, Peter C. B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1–3):159–178, 1992. doi: 10.1016/0304-4076(92)90104-Y. URL <https://www.sciencedirect.com/science/article/pii/S030440769290104Y>.
- [37] S. N. Lahiri. *Resampling Methods for Dependent Data*. Springer Series in Statistics. Springer, New York, NY, 2003. ISBN 978-0-387-00928-5. doi: 10.1007/978-1-4757-3803-2. URL <https://link.springer.com/book/10.1007/978-1-4757-3803-2>.
- [38] Pedro Lara-Benítez, Manuel Carranza-García, and José C. Riquelme. An experimental review on deep learning architectures for time series forecasting. *Applied Sciences*, 11(16):7831, 2021. doi: 10.3390/app11167831. URL <https://doi.org/10.3390/app11167831>.
- [39] Bryan Lim, Sercan Ö. Ark, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021. ISSN 0169-2070. doi: <https://doi.org/10.1016/j.ijforecast.2021.03.012>. URL <https://www.sciencedirect.com/science/article/pii/S0169207021000637>.
- [40] Bryan Lim, Sercan Ö. Ark, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021. ISSN 0169-2070. doi: <https://doi.org/10.1016/j.ijforecast.2021.03.012>. URL <https://www.sciencedirect.com/science/article/pii/S0169207021000637>.
- [41] Guan-Yu Lin and Pu-Jen Cheng. R-teafor: Regularized teacher-forcing for abstractive summarization. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6303–6311, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi:

- 10.18653/v1/2022.emnlp-main.423. URL <https://aclanthology.org/2022.emnlp-main.423/>.
- [42] Gaoyong Lu, Yang Ou, Zhihong Wang, Yingnan Qu, Yingsheng Xia, Dibin Tang, Igor Kotenko, and Wei Li. A survey of deep learning for time series forecasting: Theories, datasets, and state-of-the-art techniques. *Computers, Materials and Continua*, 85(2):2403–2441, 2025. ISSN 1546-2218. doi: <https://doi.org/10.32604/cmc.2025.068024>. URL <https://www.sciencedirect.com/science/article/pii/S1546221825008872>.
- [43] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLoS ONE*, 13, 2018. doi: <https://doi.org/10.1371/journal.pone.0194889>. URL <https://api.semanticscholar.org/CorpusID:4665547>.
- [44] Daniel L. Marino, Kasun Amarasinghe, and Milos Manic. Building energy load forecasting using deep neural networks. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, page 70467051. IEEE Press, 2016. doi: 10.1109/IECON.2016.7793413. URL <https://doi.org/10.1109/IECON.2016.7793413>.
- [45] Lorenzo Nespoli and Vasco Medici. Multivariate boosted trees and applications to forecasting and control. *Journal of Machine Learning Research*, 23(246):1–47, 2022. URL <https://jmlr.org/papers/v23/21-0247.html>.
- [46] Rui Nian, Yu Cai, Zhengguang Zhang, Hui He, Jingyu Wu, Qiang Yuan, Xue Geng, Yuqi Qian, Hua Yang, and Bo He. The identification and prediction of mesoscale eddy variation via memory in memory with scheduled sampling for sea level anomaly. *Frontiers in Marine Science*, 8:753942, 2021. ISSN 2296-7745. doi: 10.3389/fmars.2021.753942. URL <https://www.frontiersin.org/articles/10.3389/fmars.2021.753942>.
- [47] Vasilis Papastefanopoulos, Pantelis Linardatos, Theodor Panagiotakopoulos, and Sotiris Kotsiantis. Multivariate time-series forecasting: A review of deep learning methods in internet of things applications to smart cities. *Smart Cities*, 6(5):2519–2552, 2023. ISSN 2624-6511. doi: 10.3390/smartcities6050114. URL <https://www.mdpi.com/2624-6511/6/5/114>.

- [48] Andrew Patton, Dimitris N. Politis, and Halbert White. Correction to automatic block-length selection for the dependent bootstrap by d. politis and h. white. *Econometric Reviews*, 28(4):372–375, 2009. doi: 10.1080/07474930802459016. URL <https://doi.org/10.1080/07474930802459016>.
- [49] D. N. Politis and J. P. Romano. A circular block-resampling procedure for stationary data. In R. LePage and L. Billard, editors, *Exploring the Limits of Bootstrap*, pages 263–270. Wiley, New York, 1992. Also Stanford Univ. Tech. Rep. EFS NSF 370, March 1991.
- [50] Dimitris N. Politis and Halbert White. Automatic block-length selection for the dependent bootstrap. *Econometric Reviews*, 23(1):53–70, 2004. doi: 10.1081/ETC-120028836. URL <https://doi.org/10.1081/ETC-120028836>.
- [51] Georg Pólya. Über den zentralen grenzwertsatz der wahrscheinlichkeitsrechnung und das momentenproblem. *Mathematische Zeitschrift*, 8(3):171–181, 1920. ISSN 1432-1823. doi: 10.1007/BF01206525. URL <https://doi.org/10.1007/BF01206525>.
- [52] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2006. ISBN 026218253X. URL <https://gaussianprocess.org/gpml/>.
- [53] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 1986. ISBN 0-412-24620-1. URL <https://ned.ipac.caltech.edu/level5/March02/Silverman/paper.pdf>.
- [54] Kesar Singh. On the asymptotic accuracy of efron’s bootstrap. *The Annals of Statistics*, 9(6):1187–1195, 1981. ISSN 00905364, 21688966. URL <http://www.jstor.org/stable/2240409>.
- [55] Jiaming Song, Lantao Yu, Willie Neiswanger, and Stefano Ermon. A general recipe for likelihood-free Bayesian optimization. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 20384–20404. PMLR, 17–23 Jul 2022. URL <https://proceedings.mlr.press/v162/song22b.html>.

- [56] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 31043112, Cambridge, MA, USA, 2014. MIT Press.
- [57] Philipp Teutsch and Patrick Mäder. Flipped classroom: Effective teaching for time series forecasting. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL <https://openreview.net/forum?id=w3x20YEcQK>.
- [58] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer New York, NY, 2 edition, 2000. ISBN 9780387987804. doi: 10.1007/9781475732641. URL <https://doi.org/10.1007/9781475732641>. Originally published 1995.
- [59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, 2017.
- [60] Shuheì Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *ArXiv*, abs/2304.11127, 2023. URL <https://api.semanticscholar.org/CorpusID:258291728>.
- [61] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv: Machine Learning*, 2017. URL <https://api.semanticscholar.org/CorpusID:52832390>.
- [62] Jesse Wheeler and Edward L. Ionides. Revisiting inference for ARMA models: Improved fits and superior confidence intervals. *PLOS ONE*, 20(10):e0333993, October 2025. doi: 10.1371/journal.pone.0333993. URL <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0333993>.
- [63] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270. URL <https://doi.org/10.1162/neco.1989.1.2.270>.

- [64] Henning Wilms, Marco Cupelli, and Antonello Monti. Combining auto-regression with exogenous variables in sequence-to-sequence recurrent neural networks for short-term load forecasting. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pages 673–679, 2018. doi: 10.1109/INDIN.2018.8471953.
- [65] G.Peter Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514, 2005. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2003.08.037>. URL <https://www.sciencedirect.com/science/article/pii/S0377221703005484>. Decision Support Systems in the Internet Age.
- [66] Zhendong Zhang and Cheolkon Jung. GBDT-MO: Gradient-Boosted Decision Trees for Multiple Outputs. *IEEE Transactions on Neural Networks and Learning Systems*, 32(7):3156–3167, 2021. doi: 10.1109/TNNLS.2020.3009776. URL <https://ieeexplore.ieee.org/document/9158536>.