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**

**Academic Year 2024/2025**

# Contents

# Chapter 1

# Introduction

## 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 work forms part of an industrial collaboration carried out during an internship at Optit[1], where enhanced thermal-load forecasts are required to support operational planning within the company's OptiEPTM platform.

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 [13]. 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

[1] https://optit.net/

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 2030[2]. 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 [13]. 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.

In parallel, recent advances in digitalisation have accelerated the adoption of modelling and optimisation tools within energy-production systems. OptiEPTM[3], 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-

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

[3] https://optit.net/soluzioni/

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 [27, 28]. This has contributed to what authors describe as a “renaissance” of architectures, in which both new and traditional modelling strategies are being actively 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 [27]:

- *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 statistical baselines and recent advances in deep-learning architectures, a balance reflected in the models examined in this study.

## 1.3 Objectives of the Study

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 company's existing methods. To this end, we investigate a deliberately diverse set of modelling paradigms[4], reflecting both historically established techniques and more recent advances in deep learning. Specifically, we extend the modelling landscape in two main directions:

- *Statistical baselines*, represented by MSTL-ETS and SARIMAX, which provide a principled and interpretable benchmark grounded in well-understood time-series dynamics.

- *Modern deep-learning architectures*, including Long Short-Term Memory networks (LSTMs), a fundamental class of sequence models, together with the Temporal Fusion Transformer (TFT), which augments LSTMs with attention and variable-selection mechanisms. A comparison with the recent zero-shot foundation-style model Chronos-2 is planned for the final version of the thesis.

[4]All code used in this thesis is available at `https://github.com/GioLombardiDev/Internship`

By evaluating these models on the same forecasting tasks and under the same operational constraints as the existing platform, the goal is to determine whether improvements in predictive accuracy and robustness to atypical or rapidly changing operating conditions can be achieved, while containing computational overhead to the lowest practical level despite the increased complexity of deep-learning architectures.

## 1.4 Scope of the Forecasting Tasks

To assess the performance of the selected models under realistic operating conditions, we consider the following forecasting tasks and datasets.

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 that would usually be 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 serve for training and validation.

The analysis focuses on short-term forecasting horizons, specifically one day ahead and one week ahead. As is common in heat-load forecasting, accurate forecasts of the exogenous meteorological variables are assumed to be available. In this synthetic setting, they coincide with the generated historical values, an assumption that is generally reasonable for day-ahead forecasting but less reliable for longer horizons, which poses a limitation when assessing weekly forecasts.

Before modelling, a diagnostic analysis using Dynamic PCA (DPCA) [22] 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 target and exogenous variables is provided in the next section.

## 1.5 Exploratory Data Analysis

An exploratory analysis was carried out on the full dataset, in which we examined the correlations between the target and the exogenous variables and identified the features most suitable as primary predictors.

The target series exhibit marked annual seasonality, with demand remaining largely flat throughout the summer, rising in mid-autumn, and returning to a flatter pattern in spring.

Figure 1.1 presents a four-week segment of the series during a peak-demand period. This visualisation highlights the temporal dynamics observed in high-demand phases, revealing clear daily seasonality. Series F1 and F2 appear relatively regular and predictable, whereas F3 to F5 show more irregular patterns. In particular, F3 and F5 display pronounced weekly seasonality.

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

| | |
|---|---|
| **Autumn** | September 20 to December 21 |
| **Winter** | December 21 to March 21 |
| **Spring** | March 21 to June 21 |

Shaded regions denote the 10th to 90th percentile variability bands. As expected, winter shows the highest demand. Autumn exhibits the greatest intra-seasonal variability: demand typically starts at low levels and increases steadily until late November. By contrast, spring begins after demand has already passed its peak and is in decline, which results in narrower variability bands and lower mean values compared with autumn.

**Variance Behaviour.** Daily fluctuations increase with the overall level of the series, indicating heteroscedasticity, which can impair model performance, especially for statistical forecasting methods. To address this, we apply a variance-stabilising transformation to the target variable before it is provided to the statistical models, as described in later sections. Because many such transformations can overly amplify values near zero, we introduce a lower clipping threshold of 5 kWh. Fewer than 5% of observations fall below this level, even in summer. This threshold ensures numerical stability and limits distortions of small 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 [12] to highlight linear and potential non-linear trends;
- the corresponding Spearman ($\rho_s$) and Pearson ($\rho_p$) correlation coefficients.

Density plots for each variable appear along the diagonal. The figure was produced using the full dataset.

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: pairwise 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 |
| $\rho_s$ | -0.53 | -0.20 | -0.56 | -0.59 | -0.34 | -0.91 | -0.90 | -0.81 | -0.87 | -0.64 |
| $\rho_p$ | -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 un-normalised 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

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

where $y_t$ and $\hat{y}_t$ denote the observed and predicted heat demand at timestamp $t$, and $T$ is the *cutoff* 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) [21], 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 [16], this adjustment introduces 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 frequently occur.

Finally, we considered using the Mean Absolute Scaled Error (MASE) [21], 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 corresponds to the average $s$-steps-ahead forecast error of the seasonal naïve model, computed over the first $T$ observations (in our case, the natural choice is $s = 24$).

In our setting, however, the denominator becomes very small because the seasonal naïve model makes almost no error during the extremely stable summer months, when the series is nearly flat. As a result, reasonable winter forecasting errors appear disproportionately large when scaled by this quantity, making MASE values difficult to interpret. The same effect also distorts comparisons across sites: series having a longer flat warm period naturally yield a smaller denominator and therefore higher MASE values, even when the underlying forecasting performance is similar. For these reasons, MASE was not adopted in this work.

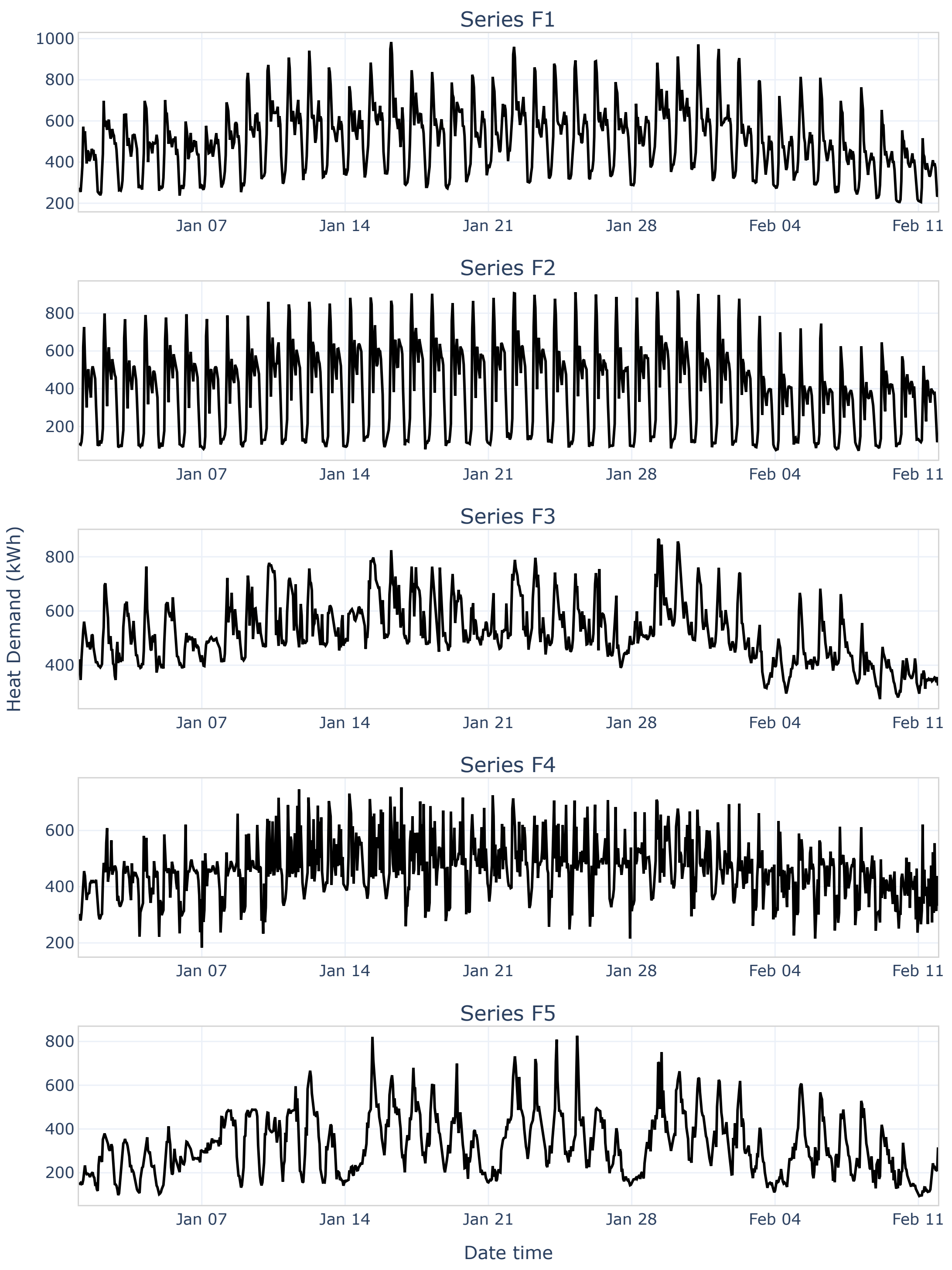


Figure 1.1: Target series for each site over a four-week peak-demand period.

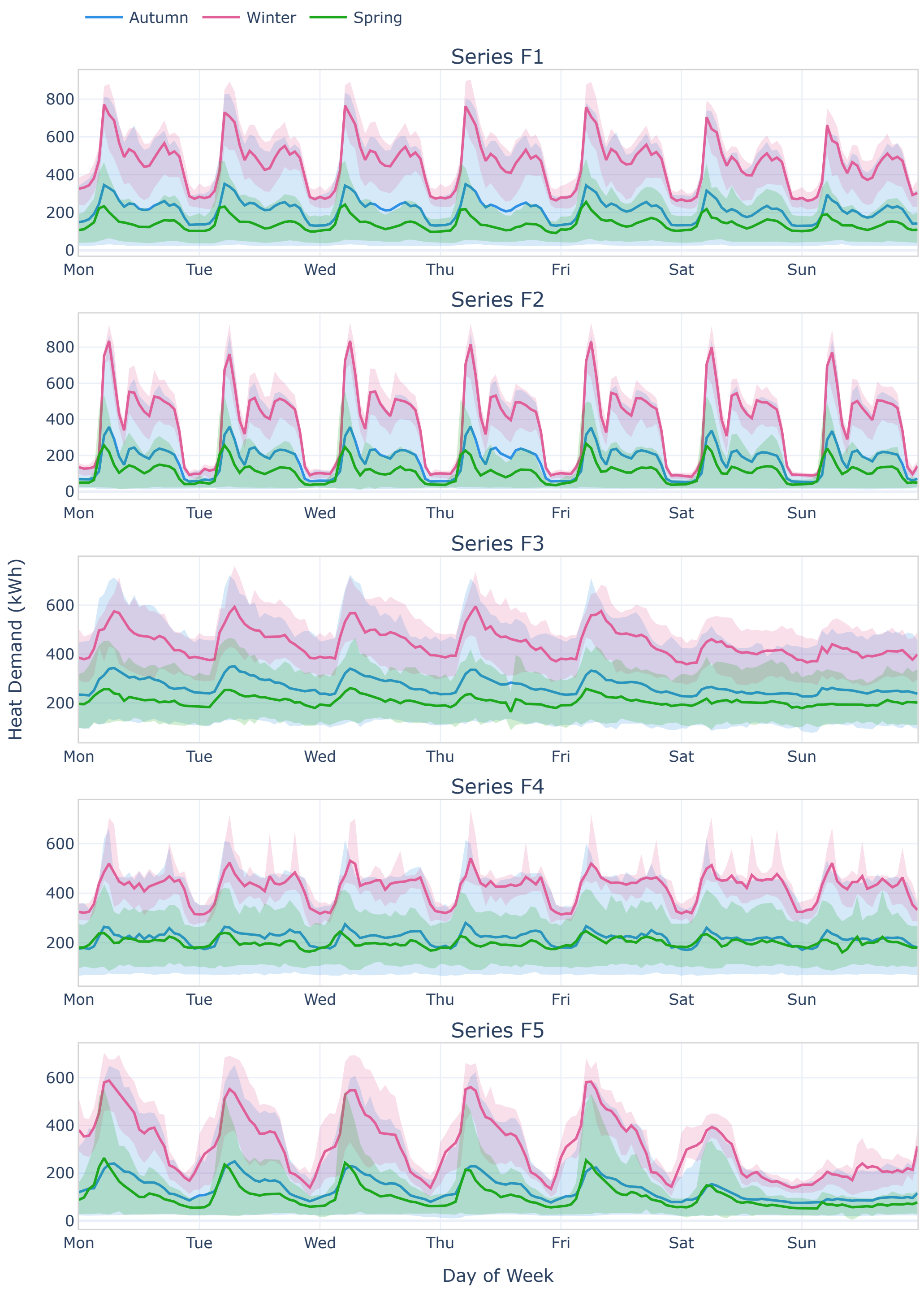


Figure 1.2: Weekly heat-demand profiles for each facility, aggregated by season (autumn, winter, spring). Solid lines represent the mean heat load per hour of week, while shaded ribbons denote the 80% variability range (10th–90th percentiles).

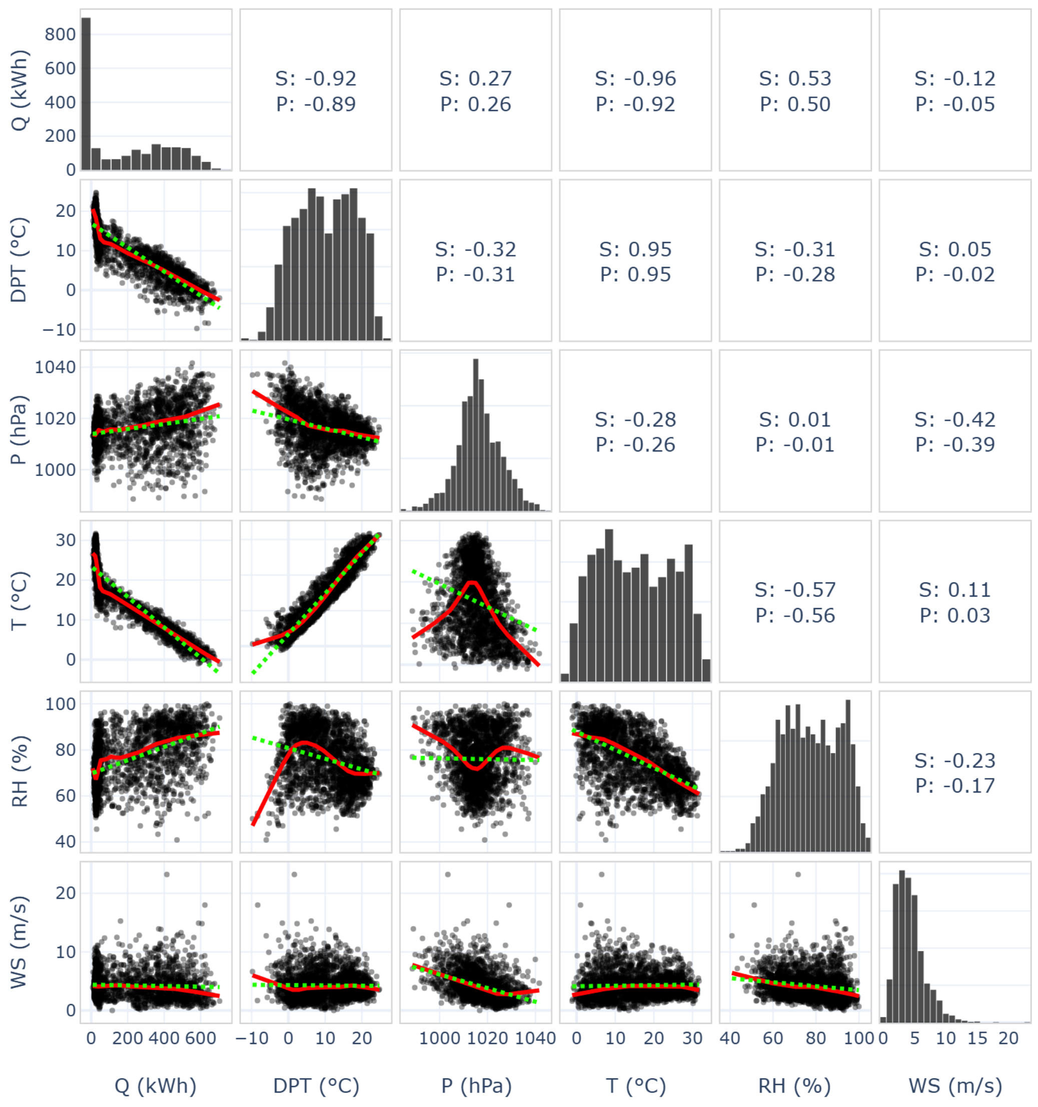


Figure 1.3: Pairwise relationships between heat demand and meteorological variables for Facility F1 (daily data). The lower triangle shows scatterplots with linear regression (green, dotted) and LOWESS smoothing (red). The diagonal panels display the marginal distributions, and the upper triangle reports 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

## 2.1 MSTL-ETS Pipeline

This section introduces the first forecasting pipeline developed in this work, denoted *MSTL-ETS*. The method combines Multiple Seasonal–Trend decomposition using Loess (MSTL) [2] with an Exponential Smoothing State Space (ETS) model [20]. MSTL isolates the high-frequency seasonal patterns, namely weekly and daily cycles, while ETS forecasts the resulting deseasonalised trend-remainder component. This decomposition-based framework is applied to the five time series presented in Chapter 1 (F1 to F5) for both daily and weekly forecast horizons.

The underlying idea is to represent each series through interpretable components: annual, weekly and daily seasonalities, a slowly evolving trend and a residual term. These components are forecast separately and then recombined to produce the final predictions. The approach uses no exogenous regressors, relying exclusively on the historical structure of each series. Although this may limit performance relative to models that incorporate external drivers, it avoids dependence on auxiliary information that may be unavailable, unreliable, or difficult to forecast.

### 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 (extended winter), where heat demand is most variable and most 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, 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) [11] 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 $Y_t$, $t = 1, \ldots, n$, into trend, seasonal and remainder components using locally weighted regressions,

$$Y_t = T_t + S_t + R_t\,, \qquad t = 1, \ldots, n.$$

It was designed to be simple, robust to outliers and applicable to any periodicity. Unlike simpler decomposition methods, it allows the seasonal pattern to evolve gradually over time and accommodates missing values without requiring special preprocessing.

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* (with $T_t^{(0)} = 0$). 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. 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 to the seasonally adjusted series.

In the *outer loop*, robustness weights are computed from the remainder component and used to down-weight outliers and localised anomalies, producing a robust version of the algorithm that is less sensitive to irregular spikes.

Although conceptually simple, STL exposes several parameters that control the smoothness of the seasonal and trend components and the strength of the robustness iteration. Cleveland et al. [11] 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, there is no risk of numerical issues due to near-zero observations under the log.

As an alternative, we consider the Box–Cox transformation [5]:

$$\mathrm{BoxCox}_\lambda(y) = \begin{cases} \dfrac{y^\lambda - 1}{\lambda}, & \lambda \neq 0, \\ \ln(y), & \lambda = 0, \end{cases} \tag{2.1}$$

where $\lambda$ controls the strength of the transformation. Values of $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$. 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 $\lambda$ 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 $\lambda$ 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 level. 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.

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 substantial 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 seasonalities. 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.

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, 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 places particular emphasis on the choice of this parameter [11, 2].

### 2.1.5 Component Forecasting

After decomposition, each time series can be expressed 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 naive seasonal strategy: the last observed seasonal cycle is simply repeated over the forecast horizon. This choice reflects the assumption of stable or slowly evolving seasonal patterns over the forecast periods considered.

- *Trend plus remainder.* The deseasonalised series (original series minus all seasonal components) is modelled using an Exponential Smoothing State Space (ETS) model, estimated via Nixtla's `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) [18]. AICc is a refinement of the Akaike Information Criterion (AIC) [7] that incorporates a finite-sample correction to mitigate AIC's tendency to favour overly complex models when the data are limited. AIC itself is an information-theoretic criterion that balances goodness of fit against model complexity by estimating the expected Kullback–Leibler discrepancy between the fitted model and the true data-generating process. It therefore penalises models with more parameters, and lower AIC values indicate models expected to generalise more effectively.

Once the best ETS model is selected, point forecasts and prediction intervals 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 different 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 (daily and weekly). 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 forecast horizon, we define a sequence of evenly spaced rolling evaluation windows within this period:

- 30 evaluation windows for the daily forecast horizon;
- 20 evaluation windows for the weekly forecast horizon.

In each cross-validation 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. This procedure is repeated across all rolling windows and all hyperparameter combinations. For each configuration, we then compute the mean and standard deviation of the metrics across folds and use these summary statistics to identify the most 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 $\lambda$ 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.

### 2.1.7 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

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

| Hyperparameter | Values / Range |
|---|---|
| `transform` | {"log", "boxcox"} |
| `train_horizon` | {7, 11, 15, 19, 23} *(weeks of hourly data)* |
| `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.

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

In this section we offer a brief overview of our SARIMAX pipeline. A more detailed explanation, together with our tuning strategy, will be provided in a more advanced version of the thesis.

### 2.2.1 Pipeline Overview and Rationale

The SARIMAX pipeline provides a classical statistical benchmark for the forecasting tasks considered in this work. Unlike the MSTL-ETS model, it is specifically designed to incorporate the available climate-based exogenous features.

It is implemented using Nixtla's `statsforecast` library and combines two complementary components: a linear regression on engineered exogenous features and a Seasonal ARIMA (SARIMA) model applied to the residuals. This structure allows the model to incorporate external drivers, such as temperature and calendar information, while still capturing autocorrelation patterns and recurring seasonal effects present in the target series.

Given a target series $y_t$ and a set of exogenous regressors $\boldsymbol{x}_t$, the model assumes

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

so that the linear component accounts for the influence of explanatory variables, and the SARIMA component models the remaining short-range dynamics and seasonal behaviours. Nixtla's implementation estimates all parameters via exact maximum likelihood and provides an `AutoARIMA` routine that selects $(p,d,q,P,D,Q)$ by minimising the corrected AIC.

This approach offers a balance between interpretability and modelling flexibility. However, the method relies on carefully engineered features for seasonality and weather effects, and does not automatically discover non-linear patterns in the way deep-learning models can.

The key aspects of our modelling strategies are the following:

- *Daily seasonality* is handled by the SARIMA residual component, supported by temperature-based exogenous inputs, which together provide a flexible and adaptive representation of evolving intraday dynamics.

- *Weekly seasonality* is modelled explicitly through exogenous Fourier terms.

- *Long-term (e.g., annual) seasonality* is captured through temperature-derived exogenous variables, reflecting broader climate-driven trends present in the data.

### 2.2.2 Feature Engineering

The feature engineering strategy for the SARIMAX pipeline was designed to supply the model with structured climate-aware information. Although the implementation supports a broad range of configurable transformations, the final tuned configurations relied on a focused subset of these components, selected for their consistent contribution to forecast accuracy across the five series.

A first key element is the use of temperature as the primary exogenous driver. Besides being included in raw form, temperature was smoothed through a four-day moving average, which proved effective in reducing short-term noise while preserving the underlying thermal signal. This moving-average series was also used to construct a data-driven cold–warm regime indicator based on series-specific thresholds. The resulting binary label allowed the model to activate different sets of seasonal features depending on the prevailing climatic conditions. In practice, all other exogenous variables were split according to this indicator. For example, temperature itself was decomposed into a cold-regime predictor, which is zero outside the cold period, and a warm-regime predictor, which is zero during the cold period. This structure enabled the model to learn season-specific effects, which was essential given the drastic change in the relationship between demand and exogenous variables from the flat-demand summer period to the peak-demand winter one.

Weekly seasonality was modelled through Fourier terms of order $k_{\text{week}} \in \{5, 7, 9\}$, depending on the series. Daily seasonality, by contrast, was delegated to the SARIMA residual component, which offered a more adaptive and data-driven treatment of intraday patterns than fixed Fourier expansions. To ensure that the fixed-amplitude Fourier exogenous terms matched the true seasonal structure of the data, the target series was stabilised using a winter-focused Box–Cox transformation, as previously introduced for the MSTL–ETS model. This transformation mitigates the strong temperature-driven variability in weekly fluctuations, making the seasonality more consistent and therefore more compatible with Fourier-based modelling.

Additional short-range climatic effects were introduced through low-order lag features (when selected by tuning), typically involving lags of 1–3 hours. Multi-day averages of temperature, computed over windows of one and four days, were also included for all series, providing the model with slower-moving thermal indicators relevant for heating-demand regimes.

For some series, peak-hour splits were activated, allowing both the temperature-derived features and the shorter-term aggregates to be duplicated into `*_peak` and `*_offpeak` variants. This mechanism gave the model the ability to distinguish between high-demand and low-demand periods within the day. The rationale is that critical peak-demand intervals are characterised by behaviourally driven dynamics, in which the relationship between temperature and the target may differ from the rest of the day. By splitting features in this way, the model can learn these specific interactions more effectively, with the aim of capturing the operationally important morning peak in heat demand more accurately.

Overall, the engineered feature set brings together short-term climatic signals, smoothed multi-day trends, regime information, and compact seasonal encodings. Together, these elements proved sufficient to support a competitive and interpretable statistical benchmark, as discussed in Chapters 5 and 6.

# Chapter 3

# LSTM Pipeline

To improve forecasting accuracy beyond classical baselines, we adopt an Encoder–Decoder architecture based on the Long Short-Term Memory (LSTM) network. This chapter begins with a brief overview of LSTMs, proceeds to the specific Encoder–Decoder configuration used, and concludes with our tuning methodology.

## 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) [15] 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. [3] 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) [17] 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 [14]) 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 [24], 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 $x = (x_1, \dots, x_T)$, an LSTM iterates the following updates for $t = 1, \dots, T$ (although variants exist):

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + b_{ix} + W_{ih}h_{t-1} + b_{ih}) && (3.1)\\
f_t &= \sigma(W_{fx}x_t + b_{fx} + W_{fh}h_{t-1} + b_{fh}) && (3.2)\\
g_t &= \tanh(W_{cx}x_t + b_{cx} + W_{ch}h_{t-1} + b_{ch}) && (3.3)\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t && (3.4)\\
o_t &= \sigma(W_{ox}x_t + b_{ox} + W_{oh}h_{t-1} + b_{oh}) && (3.5)\\
h_t &= o_t \odot \tanh(c_t)\,. && (3.6)
\end{aligned}
$$

Here, the $W$ terms are weight matrices shared through time and the $b$ terms are bias vectors; $\sigma$ is the logistic sigmoid; $i_t, f_t, o_t$ are the input, forget, and output gate activations; $c_t$ is the cell (or memory) state; and $h_t$ 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 $h_t$ and the cell state $c_t$; 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

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

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

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

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

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

**Output gate $\boldsymbol{o_t}$.** Regulates how much of the memory in $c_t$ is exposed to the rest of the network via $h_t = o_t \odot \tanh(c_t)$, allowing information to be stored without being revealed at every step.

Functionally, $h_t$ is a gated, squashed view of the memory cell $c_t$, whereas $c_t$ itself is the durable substrate on which information is accumulated, preserved, or decayed over time. In practice, $h_t$ is the working output used to make predictions: in sequence tagging it is mapped to the current label $y_t$; in one-step forecasting $h_t$ serves as features to predict the next value $y_{t+1}$; and in Encoder–Decoder models (e.g., for translation) one passes $h_T$ (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 produce multi-step-ahead forecasts from time $T$ to $T + h$, given known future covariates, we employ an LSTM-based encoder–decoder (seq2seq) architecture. Both encoder and decoder are LSTMs. The encoder ingests 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 $(h_T, c_T)$. The decoder is initialised with these states and, for each step $t = T + 1, \dots, T + h$, conditions on the corresponding future covariates $x_t$ (e.g., calendar or weather features) to produce a hidden state $h_t$. A linear (or shallow MLP) projection maps $h_t$ to the forecast $\hat{y}_t$. Optionally, the decoder operates autoregressively by feeding its previous prediction $\hat{y}_{t-1}$ (together with $x_t$) back as input at time $t$. We also allow stacking multiple LSTM layers in both encoder and decoder (typically with a shared hidden size) to obtain a deeper model. Figure 3.1 depicts the single-layer case.

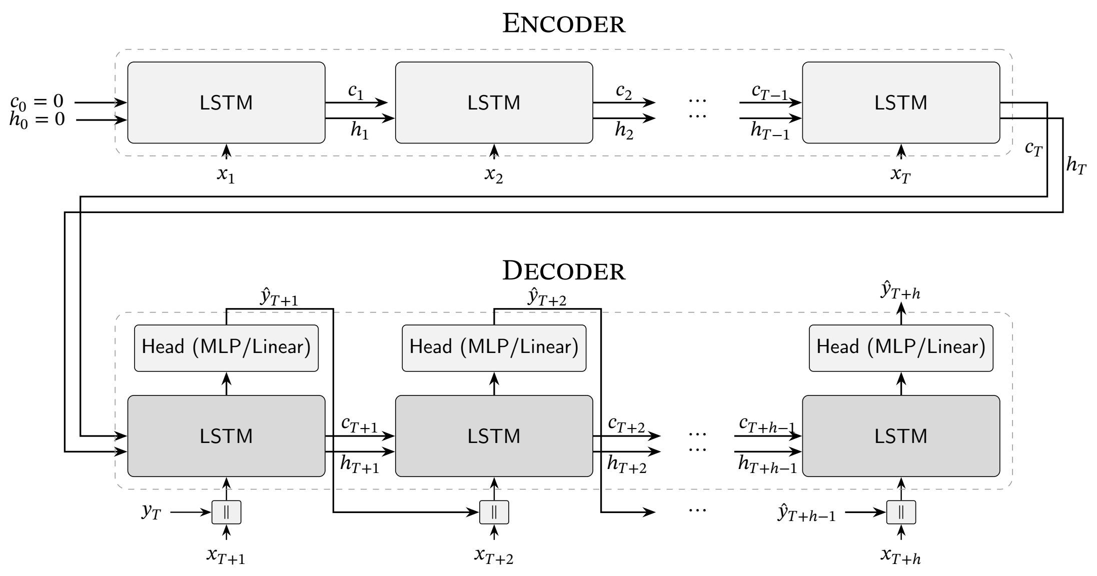


Figure 3.1: Encoder–decoder LSTM architecture used in this study. For simplicity of notation, the encoder length is shown as $k = T$.

### 3.2.2 Design Rationale and Alternatives

The encoder–decoder architecture was originally proposed in the context of machine translation in 2014, as a way to handle input and output sequences that may differ in length [36, 9]. 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-length 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.

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 [40, 29]. 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. [39] 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. As a result, it is less prone to error accumulation over long horizons. However, this does not guarantee improved accuracy in all settings: for instance, when the target series is highly autocorrelated at small lags, autoregressive models may offer an advantage. Nonetheless, MLP-based decoders offer a compelling alternative, especially when future covariates are available.

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 conditions on fixed-length blocks of past target values (such as a 24-hour window) that are updated progressively as predictions are generated. Specifically, 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 `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 hyperparameters

| 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: Main data loading and preprocessing hyperparameters

| 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 only training strategy particularly tailored to the considered architecture (and, more generally, to recurrent models) is *teacher forcing* (TF). 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 a block size $b$, this generalises to replacing $\hat{y}_{t-b}$ with the corresponding true value $y_{t-b}$.

Teacher forcing is beneficial because it stabilises training in the early stages, allowing the model to learn from accurate context rather than being exposed to its own possibly erroneous predictions. This helps prevent error accumulation over time, which can otherwise derail the learning process, particularly when the forecast horizon is long or the target dynamics are complex.

However, because TF relies on ground-truth values that are unavailable at inference time, it creates a mismatch between training and deployment conditions. To address this, it is common to gradually reduce the use of TF during training, typically through probabilistic scheduling strategies that decrease the forcing probability over time.

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 design keeps computation efficient, as batches that use teacher forcing can be processed in a single parallel forward pass, which is advantageous for GPU workloads. As training progresses, we gradually reduce the probability of applying teacher forcing, allowing the model to rely increasingly on its own predictions. This schedule narrows the gap between training and inference behaviour, while preserving stability in the early stages.

We employ a linear schedule to gradually 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 first 20% of this interval acts as a warm-up stage, during which teacher forcing remains fully enabled to support stable early optimisation. 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 (i.e., probability zero), except for periodic "spikes" in which a TF probability of 0.3 is temporarily reintroduced every three epochs. This heuristic serves as a safeguard against training instability that may arise due to error accumulation in fully autoregressive decoding.

Table 3.3: Main training-related hyperparameters

| 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` | `int \| None` | 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). |

### 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.** A set of 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, such as 24-hour and 168-hour moving averages, smooth short term fluctuations and highlight longer term behaviour, which helps the model learn trend components. Each rolling average is computed over a window of length $w$ as

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

where $x_t$ denotes the raw input value at time $t$.

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 [10, 41], 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 deliberately avoided incorporating explicit signal decompositions for practical reasons. First, to prevent data leakage, such decompositions would have to be recomputed dynamically for each encoder window rather than applied once to the entire dataset. More critically, in the presence of autoregression in the decoder, the decomposition would need to be re-applied at every decoding step on the models own predictions. This repeated computation becomes prohibitive in terms of computation time, particularly given that our target series often exhibits multiple seasonalities (daily and weekly). Capturing these would require more sophisticated decomposition methods than standard STL or CD, significantly increasing the computational burden.

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 (cf. 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 (HOD), 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, the HOD is encoded as

$$\mathrm{hod}_{\sin}(t) = \sin\left(2\pi \cdot \frac{\mathrm{hour}(t)}{24}\right), \qquad \mathrm{hod}_{\cos}(t) = \cos\left(2\pi \cdot \frac{\mathrm{hour}(t)}{24}\right),$$

where $\mathrm{hour}(t) \in \{0, 1, \dots, 23\}$ denotes the hour component of timestamp $t$. 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 [10]. 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. [6], BO provides a probabilistic framework for efficiently locating the extremum of a black-box function $f : \mathcal{X} \to \mathbb{R}$, 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),

$$f \sim \mathcal{GP}\big(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\big)\,.$$

This prior encodes initial beliefs about the smoothness and general behaviour of

Table 3.4: Main feature engineering hyperparameters

| Name | Type | Description |
|---|---|---|
| `exog_vars` | `Tuple[str,...]` | Names of whether exogenous variables, selected from {`temperature`, `dew_point`, `wind_speed`, `pressure`, `humidity`}. |
| `hour_averages` | `Tuple[int,...]` | Rolling means (in hours) computed for both endogenous and exogenous features. For example, (24, 168) adds daily and weekly averages. |
| `endog_hour_lags` | `Tuple[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` | `Tuple[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. |

$f$. Each evaluation of the true function, composing a dataset of noisy observations $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, is used to update the posterior belief over $f$:

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

where $m'(\boldsymbol{x})$ and $k'(\boldsymbol{x}, \boldsymbol{x}')$ denote the posterior mean and covariance functions, respectively, derived from the prior and the observed data. These functions refine the surrogate model's representation of the underlying landscape.

The optimisation process proceeds sequentially: at iteration $n$, the current probabilistic model guides the selection of the next evaluation point $\boldsymbol{x}_{n+1} \in \mathcal{X}$ 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 $m'(\boldsymbol{x})$

being low in a minimisation setting.

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

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

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

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

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

thereby updating the surrogate model.

Common choices for the acquisition function include the Expected Improvement (EI) and the Probability of Improvement (PI) criteria [1].

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 $y^\star$:

$$\mathrm{PI}_{y^\star}(\boldsymbol{x}) = \mathbb{P}(Y \leq y^\star \mid \boldsymbol{x}, \mathcal{D}_n) = \int_{-\infty}^{y^\star} p(y \mid \boldsymbol{x}, \mathcal{D}_n)\, \mathrm{d}y, \tag{3.7}$$

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

$$p(y \mid \boldsymbol{x}, \mathcal{D}_n) = \int p(y \mid f(\boldsymbol{x}))\, p(f(\boldsymbol{x}) \mid \boldsymbol{x}, \mathcal{D}_n)\, \mathrm{d}f(\boldsymbol{x}),$$

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

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

$$\mathrm{EI}_{y^\star}(\boldsymbol{x}) = \mathbb{E}[(y^\star - Y)_+ \mid \boldsymbol{x}, \mathcal{D}_n] = \int_{-\infty}^{y^\star} (y^\star - y)\, p(y \mid \boldsymbol{x}, \mathcal{D}_n)\, \mathrm{d}y, \tag{3.8}$$

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

In practice, it is common to introduce a small positive margin $\xi$ and compute the probability (or expectation) of achieving an improvement over $y^\star + \xi$ rather than $y^\star$, that is, using $\mathrm{PI}_{y^\star+\xi}$ and $\mathrm{EI}_{y^\star+\xi}$. 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 $y^\star$ 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 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. [4], 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 [38] 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) [34], 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 [38]. 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 $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ is already sorted so that $y_1 \leq \cdots \leq y_n$.

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, $p(\boldsymbol{x}, y | \mathcal{D}_n)$. The joint distribution can be expressed through the following factorisation:

$$p(\boldsymbol{x}, y \mid \mathcal{D}_n) = p(\boldsymbol{x} \mid y, \mathcal{D}_n)\, p(y \mid \mathcal{D}_n).$$

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:

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

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);

- $y^\gamma$ denotes the $\gamma$-quantile of the observed objective values, and $\mathcal{D}_n^{(\ell)}$ is the corresponding top $\gamma$ fraction of the data,

$$\mathcal{D}_n^{(\ell)} = \{(\boldsymbol{x}, y) \in \mathcal{D}_n \mid y \leq y^\gamma\};$$

- $\mathcal{D}_n^{(g)}$ is the complementary set,

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

Since we assume that the set $\mathcal{D}_n$ is sorted in ascending order of the objective values, we can write

$$\mathcal{D}_n^{(\ell)} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n^{(\ell)}}, \qquad \mathcal{D}_n^{(g)} = \{(\boldsymbol{x}_i, y_i)\}_{i=n^{(\ell)}+1}^{n},$$

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

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,

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

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

We also introduce the shorthand notation

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

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 $y^\gamma$ (from equations (3.8) and (3.7), respectively). Following the derivations in Bergstra et al. [4] and Watanabe [38], 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*

$$\mathrm{EI}_{y^\gamma}(\boldsymbol{x}) \;\propto\; \mathrm{PI}_{y^\gamma}(\boldsymbol{x}) \;\propto\; \frac{\ell(\boldsymbol{x})}{p(\boldsymbol{x} \mid \mathcal{D}_n)}, \tag{3.11}$$

*for all $\boldsymbol{x} \in \mathcal{X}$ such that $p(\boldsymbol{x} \mid \mathcal{D}_n) > 0$, with proportionality constants in each expression that are strictly positive. Consequently, if g is positive, for a fixed threshold $y^\gamma$, the maximisers of both the EI and the PI coincide and are given by*

$$\operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \frac{\ell(\boldsymbol{x})}{g(\boldsymbol{x})}.$$

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

$$\begin{aligned}
\mathrm{EI}_{y^\gamma}(\boldsymbol{x}) &= \int_{-\infty}^{y^\gamma} (y^\gamma - y) p(y \mid \boldsymbol{x})\, \mathrm{d}y \\
&= \int_{-\infty}^{y^\gamma} (y^\gamma - y) \frac{p(\boldsymbol{x} \mid y) p(y)}{p(\boldsymbol{x})}\, \mathrm{d}y \\
&= \frac{\ell(\boldsymbol{x})}{p(\boldsymbol{x})} \int_{-\infty}^{y^\gamma} (y^\gamma - y) p(y)\, \mathrm{d}y\,.
\end{aligned}$$

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

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

For the PI criterion,

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

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

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

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

$$\begin{aligned} p(\boldsymbol{x}) &= \int_{\mathbb{R}} p(\boldsymbol{x} \mid y)\, p(y)\, \mathrm{d}y \\ &= \ell(\boldsymbol{x}) \int_{-\infty}^{y^\gamma} p(y)\, \mathrm{d}y + g(\boldsymbol{x}) \int_{y^\gamma}^{\infty} p(y)\, \mathrm{d}y \\ &= \gamma\, \ell(\boldsymbol{x}) + (1-\gamma)\, g(\boldsymbol{x}), \end{aligned}$$

using equation (3.10). Substituting this expression into the ratio in (3.11) yields

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

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

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

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

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

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

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

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

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 $\boldsymbol{x} \in \mathcal{X}$ can be represented as $\boldsymbol{x} = (x_1, \dots, x_d)^\top$, where $x_j$ denotes the value of the $j$th hyperparameter. Likewise, an observed configuration $\boldsymbol{x}_i$ is written as $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id})^\top$.

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

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

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

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 $\ell(\boldsymbol{x})$ 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 $r(\boldsymbol{x})$ is equivalent to maximising it in each coordinate separately, since

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

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`) models the joint density of all parameters directly, rather than assuming independence across dimensions. In this case, the Parzen estimators are defined as

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

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

where each mixture component corresponds to a joint sample $\boldsymbol{x}_i$, and its contribution is the product of one-dimensional kernels centered at that point. 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 $\ell(\boldsymbol{x})$ and $g(\boldsymbol{x})$ to assign higher probability to regions where promising parameter combinations tend to co-occur.

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 $\ell$ 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 $(x_i, y_i)$, belongs to the good set $\mathcal{D}_n^{(\ell)}$ or the bad set $\mathcal{D}_n^{(g)}$.
- Let $m_i = \left|\mathcal{D}_n^{(h_i)}\right|$ 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 $T_{\text{new}}$ 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

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

A more recent weighting scheme proposed by Song et al. [35] suggests using weights $w_n$ in $\ell(x)$ proportional to $y^\gamma - y_n$, 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:

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

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:

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

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

For an integer-valued hyperparameter with domain $\mathcal{X}_j = \{L_j, L_j + q, \dots, R_j\} \subset \mathbb{Z}$, the kernel $k_j$ is defined as a discretised Gaussian kernel:

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

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

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

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

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 $g(x) > 0$ for all configurations, thereby allowing $r(x)$ 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 promising regions later on.

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

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

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 evaluates multiple hyperparameter configurations, each referred to as a *trial*. Every trial involves three key components:

- a *suggester*, which proposes a configuration to be tested;
- an *objective function*, which evaluates the configuration and returns an objective value (e.g., validation loss); and
- a *pruner*, which monitors intermediate results and terminates unpromising trials early.

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 time-series cross-validation would have provided a more statistically robust estimate of model performance, it would also have substantially increased computational cost given the number of models and series to be tuned. In our setting, the validation and training periods exhibited similar seasonal patterns and comparable overall levels, indicating that no major regime shifts occurred during the evaluation horizon. This mitigates the main drawback of omitting cross-validation, namely the risk that the performance estimate may be biased by structural changes in the target series. Under these conditions, the relatively long validation window offers a reasonable and computationally efficient measure of the model's 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 [38]. 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 weekly 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 $\mu$ and $\sigma$ 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

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

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

$$\mathrm{CI}^{*}_{95\%} = \dots,\ (n = \dots).$$

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.

### 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.

- 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.

**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 in those regions of the search space. 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 $\sim 20$ MAE but did not improve beyond that threshold.

Table 3.5: Search space for Step 1. The column Type indicates the parameter class, which determines the prior and density model used by TPE for that hyperparameter.

| Hyperparameter | Type | Values / Range |
|---|---|---|
| `head` | Categorical | {“`linear`”, “`mlp`”} |
| `hidden_size` | Categorical | {32, 64, 128} |
| `num_layers` | Categorical | {1, 2} |
| `dropout` | Categorical | {0.0, 0.15, 0.3} |
| `input_len` | Categorical | {72, 168} |
| `batch_size` | Categorical | {64, 128} |
| `learning_rate` | Float (log) | [0.0001, 0.005] |

*Fixed:* `n_epochs=25` (with early-stopping), `use_ar=“prev”`, `norm_mode=“global”`, `tf_drop_epochs=15`, `lr_drop_epoch=15`.
*Constraint:* exclude computationally heavy configurations in which the choices `hidden_size=128`, `num_layers=2`, and `input_len=168` occur together.

**Loss dynamics.** Validation trajectories were typically noisy during the first 10–15 epochs and then stabilised for runs exceeding 15 epochs (~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* (<40k parameters): mean objective 22.08 ± 1.64 ($n = 14$); the smallest configuration (1 layer, 32 hidden units; ~10k parameters) showed mild underfitting.
- *Heavy models* (>160k parameters): mean objective 21.12 ± 0.72 ($n = 60$).
- *Intermediate sweet spot* (~40k–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 $23.15 \pm 2.25$) involved high dropout (0.3) together with a medium or small parameter count ($< 110$k). Learning rates around $3 \times 10^{-3}$ 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 [19], a variance based method for estimating the marginal influence of each hyperparameter, 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), 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 (∼2% on ∼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 (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 $\sim$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 $\sim$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). 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 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 [30]. 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.

| Hyperparameter | Type | Values / Range |
|---|---|---|
| `input_len` | Categorical | {24, 72, 168, 504} |
| `endog_hour_lags` | Categorical | {(), (168,), (168, 24)} |
| `hidden_size` | Categorical | {64, 128} |

*Fixed:* `num_layers=2`, `head="linear"`, `dropout=0.1`, `lr=7e-4`, `batch_size=64`, `norm.mode="global"`, `n_epochs=25` (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 $\mathrm{CI}^{*}_{95\%}$=[20.47, 21.10], $n = 24$), 22.30 for 7D ($\mathrm{CI}^{*}_{95\%}$=[22.04, 22.61], $n = 24$), 24.25 for N ($\mathrm{CI}^{*}_{95\%}$=[23.81, 24.72], $n = 24$). fANOVA confirms this effect, attributing ∼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 ∼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 evaluated our best-performing configuration (previously tuned mainly on F1) in both the autoregressive (AR) and non-autoregressive (NAR) settings, as described in Section section 3.2.3 on page 34. In the AR mode, the decoder receives the predicted target from the previous timestep, $\hat{y}_{t-1}$, 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 with forecast horizon.* This overhead grows rapidly as the horizon increases. For example, extending the output window to a week (using similar configurations as those used until now) a single training epoch with AR decoding took more than 20 minutes on GPU, compared to only $\sim$10 seconds without AR feedback. This scaling makes AR decoding unusable for longer horizons.

- *Extension to longer horizons.* The non-AR variant offers a direct and scalable route to weekly forecasts. Rather than decoding hour by hour as in the AR setting (which, as discussed, would be computationally impractical) we predict in 24-hour blocks and feed each blocks output back as a lagged input to the subsequent decoder. This *block-autoregressive* mode (see Section 3.2.3) substitutes the unavailable true 24-hour lag over multi-day horizons with a model-generated proxy, thereby retaining the empirical advantage of that feature established in the previous step. Exogenous lags remain applicable, as forecasts for 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 ∼75% of the variance to the repeat index and only ∼25% to the AR factor. In contrast, runtime was nearly four times faster without AR (averaging ∼1′30″ per trial versus ∼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 ∼1′20″ per trial without AR versus ∼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 but not excessive 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.

Table 3.7: Main tuning search space per series.

| Hyperparameter | Type | Values / Range (typical) | Series / Applicability |
|---|---|---|---|
| `use_ar` | Categorical | {“`none`”, “`24h`”} | F1–F3: both explored<br>F4–F5: fixed to “`24h`” |
| `num_layers` | Categorical | {1, 2} | F1–F4: searched<br>F5: fixed to 1 |
| `hidden_size` | Int | Maximum span across series: [8, 128] | F1: [32,128], step 16<br>F2: [16,112], step 16<br>F3: [8,32], step 8<br>F4: [8,24], step 4<br>F5: [8,32], step 4 |
| `dropout` | Float | [0.0, 0.3] | All series |
| `learning_rate` | Float (log) | $[1\times10^{-4}, 2\times10^{-3}]$ | All series |
| `lr_drop_epoch` | Categorical | {4, 7} | All series |
| `use_weight_decay` | Categorical | {False, True} | F3 only |
| `weight_decay` | Float (log) | $[1\times10^{-6}, 1\times10^{-3}]$ | F3 only |
| `tf_drop_epochs` | Derived | Equal to `lr_drop_epoch` | Active whenever `use_ar=“24h”` |

*Fixed across studies:* `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`.

**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 (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, 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) over approximately 200 configurations per study, with a forecast horizon of one week. When both `use_ar="none"` and `use_ar="24h"` where tested, two different studies where made (hence 400 trials total). All Optuna studies were conducted as described in Section 3.4.3. For each series, after tuning the model on the 7-day-ahead forecasting task, we retained one or two promising configurations and selected a final one, along with the corresponding number of epochs, performing roughly ten repeated runs for each. Once the final model for the weekly forecast task 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 new 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.

### 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 $1 \times 10^{-3}$) 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, arguably the noisiest, 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

## 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) [25] model, provided by the Python library Darts[1]. 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. [25], where TFT was first introduced. Here, we summarise its principal 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

[1]https://unit8co.github.io/darts/

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.

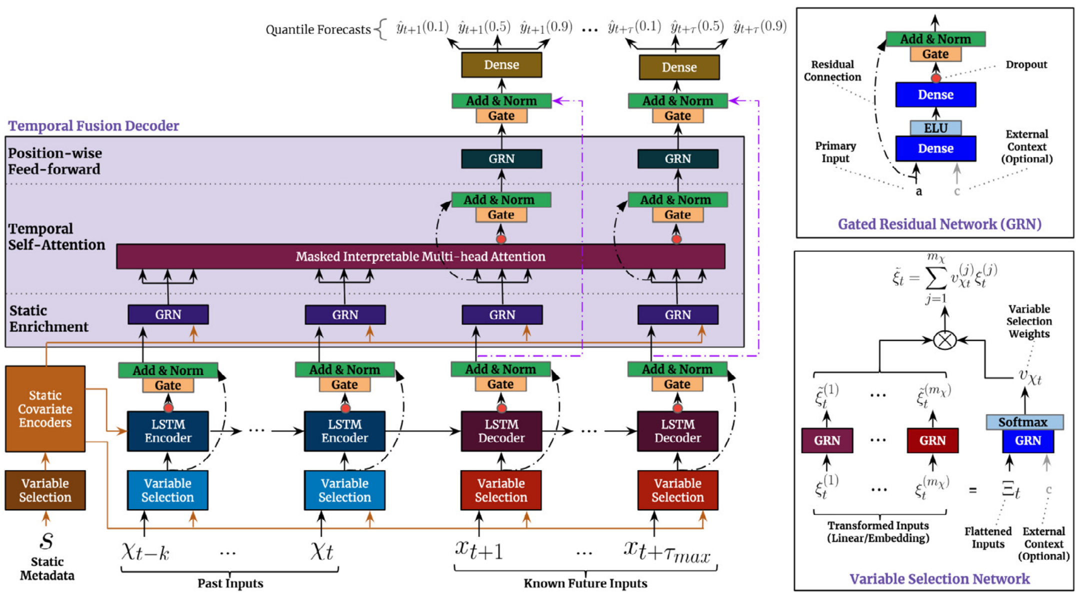


Figure 4.1: TFT architecture, reproduced from [25] (Fig. 2).

## 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. [25], 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 attention mechanism directly connects distant time points, allowing the model to capture complex lag structures and periodicities that would be difficult for the LSTM to preserve through recurrent state alone, as also observed in Section 3.5.1. TFT modifies standard attention by sharing value projections across heads and averaging head outputs. This ensures that the resulting attention weights can be read as meaningful indicators of temporal relevance rather than opaque mixtures of head-specific transformations. Decoder masking is applied so that each forecast step attends only to permissible past inputs.

Following the 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 Temporal Fusion Transformer 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 around the best training epoch.

### 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%

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) | $[1 \times 10^{-4}, 1 \times 10^{-2}]$ |

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 $3 \times 10^{-4}$). 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 was used in the testing phase described in Chapter 5.

# Chapter 5

# Comparative Assessment of Forecasting Models

This chapter provides a comparative evaluation of the forecasting models developed in the previous sections, tuned jointly per horizon and per series, for the day-ahead and week-ahead tasks on the five heat-demand series F1 to F5 described in Sections 1.4 and 1.5.

The models are evaluated in a realistic operational setting to generate both daily and weekly forecasts. The analysis is conducted across three seasons, namely autumn, winter and spring, which capture the main phases of the heat-demand cycle: the rise in mid-autumn, the winter peak and the decline in spring. Comparisons focus on three key aspects: forecast accuracy, computational cost and the structure of the residuals.

**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 [8], complemented by a second model based on Support Vector Machine Regression (SVMR) [37].

**XGBoost** is an optimised gradient boosting framework that constructs an 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. In this evaluation, the future temperature values correspond to historical measurements recorded later in time near each facility. In operational settings, these values would instead come from temperature forecasts.

This distinction is important when interpreting the results, since real operational errors, particularly for week-ahead forecasts, will generally be higher than those reported here. The comparison across models nevertheless remains fair, as all top-performing approaches similarly depend on temperature forecasts in practice. The impact of temperature forecast errors on model accuracy has not been examined in this project.

Finally, the company models are trained end-to-end for both the day-ahead and week-ahead tasks, separately for each series. This ensures consistency with the evaluation protocol adopted for the models developed in this work.

## 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. In practical terms, the step between successive cutoffs (that is, the datetimes marking the inclusive end of the available contextual data) is equal to the forecast horizon.

To control computational cost, each model is retrained only once every seven days. For the daily horizon, the most recently trained model is reused during the intermediate days within the same week. This structure maintains a balance between computational efficiency and adaptability to evolving data.

**Implementation details.** The first cutoff is set at 23:00 on Sunday, 2024-05-19. Models are trained on all data up to and including that cutoff (or on a model-specific training window). The subsequent operations depend on the forecast horizon:

- *Daily horizon.* The model generates forecasts for the next day. The following cutoff 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. Cutoffs then advance every 24 hours, with retraining occurring each Sunday at 23:00.

Table 5.1: Training and inference times for all models. 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) |

- *Weekly horizon.* The model generates forecasts for the subsequent week. The next cutoff is at 23:00 on Sunday, 2024-05-26, when the model is retrained and used to forecast the following week. Cutoffs then advance every seven days, with retraining each Sunday at 23:00.

## 5.3 Series F1: Day-Ahead Forecast Evaluation

We begin by analysing the results for the day-ahead forecasts for series F1, which comprises 90 validation windows.

### 5.3.1 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 train extremely quickly, even relative to the statistical approaches. 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.

It is worth noting that, in many operational environments, model fitting is performed only once per week for each series, so the additional cost of training deep learning models is incurred relatively infrequently. In our setting, however, training is executed independently for each series and each forecast horizon. At larger scales, this can accumulate to several hours of additional computation per retraining cycle.

The table also reports the inference times. SARIMAX displays by far the highest inference cost, while the LSTM achieves unexpectedly low inference times. The slightly higher latency of Naive24h compared with the LSTM is attributable to the Statsforecast `SeasonalNaive` implementation, which simply copies the previous day's observations but still introduces some overhead during execution.

For brevity, computational times for the additional forecasting tasks (that is, the remaining horizons and series) are not reported in the main text; the complete results are provided in Appendix **??**. Although the absolute values vary slightly across tasks, particularly for the LSTM and TFT whose training durations depend on the number of epochs selected during tuning, 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.

### 5.3.2 Per-Window Errors

This section examines how forecasting accuracy varies over individual day-ahead windows. Table 5.2 reports summary statistics for each season, presenting mean and standard deviation for the MAE, RMSE and sMAPE metrics, together with the win rate (denoted "wrate"), which measures how often each model attains the lowest error across all models. The tables are organised by ascending mean MAE to facilitate comparison. Figure 5.1 complements these results by displaying, for each cutoff, how much the MAE of the competing models differs from that of the LSTM baseline. 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) | 30.0 | 20.54 (11.78) | 36.0 | 8.55 ( 6.78) | 33.0 |
| XGBoost | 18.55 ( 9.58) | 12.0 | 23.66 (11.98) | 12.0 | 9.98 ( 6.54) | 12.0 |
| SARIMAX | 20.11 (11.81) | 14.0 | 25.19 (14.36) | 12.0 | 10.07 ( 7.32) | 16.0 |
| TFT | 22.36 (12.73) | 22.0 | 27.83 (15.63) | 18.0 | 11.76 ( 8.90) | 20.0 |
| SVMR | 23.75 (13.67) | 5.0 | 29.52 (16.81) | 4.0 | 14.43 (13.17) | 3.0 |
| Naive24h | 26.19 (21.54) | 8.0 | 33.16 (26.20) | 9.0 | 11.25 ( 7.22) | 8.0 |
| MSTL-ETS | 29.87 (20.22) | 9.0 | 37.63 (24.27) | 9.0 | 13.24 ( 8.14) | 9.0 |

(b) Winter (90 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 20.22 ( 7.35) | 49.0 | 25.59 ( 9.66) | 53.0 | 4.57 ( 1.44) | 51.0 |
| SARIMAX | 24.82 (10.26) | 17.0 | 31.09 (12.84) | 13.0 | 5.55 ( 1.97) | 17.0 |
| XGBoost | 25.65 ( 9.60) | 12.0 | 32.19 (11.81) | 13.0 | 5.89 ( 2.20) | 13.0 |
| TFT | 26.44 (10.04) | 9.0 | 32.21 (11.34) | 11.0 | 5.97 ( 1.88) | 7.0 |
| SVMR | 28.37 ( 8.73) | 6.0 | 35.69 (11.37) | 4.0 | 6.54 ( 2.02) | 7.0 |
| Naive24h | 40.87 (22.20) | 8.0 | 49.94 (24.89) | 3.0 | 9.29 ( 4.73) | 6.0 |
| MSTL-ETS | 45.29 (18.76) | 0.0 | 58.43 (23.60) | 1.0 | 9.93 ( 4.06) | 0.0 |

(c) Spring (59 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 10.81 ( 7.12) | 38.0 | 14.58 (10.06) | 50.0 | 8.17 ( 1.88) | 50.0 |
| TFT | 11.86 ( 7.71) | 50.0 | 16.26 (11.12) | 38.0 | 8.51 ( 1.05) | 38.0 |
| XGBoost | 15.26 ( 9.77) | 0.0 | 20.45 (13.40) | 0.0 | 10.76 ( 2.95) | 0.0 |
| Naive24h | 25.37 (17.82) | 0.0 | 31.36 (21.99) | 0.0 | 18.12 (10.39) | 0.0 |
| MSTL-ETS | 25.70 (16.23) | 0.0 | 33.02 (21.19) | 0.0 | 18.24 ( 6.32) | 0.0 |
| SARIMAX | 25.72 (15.34) | 12.0 | 30.92 (18.44) | 12.0 | 19.58 (11.57) | 12.0 |
| SVMR | 28.24 ( 7.86) | 0.0 | 33.95 ( 8.11) | 0.0 | 27.02 (14.88) | 0.0 |

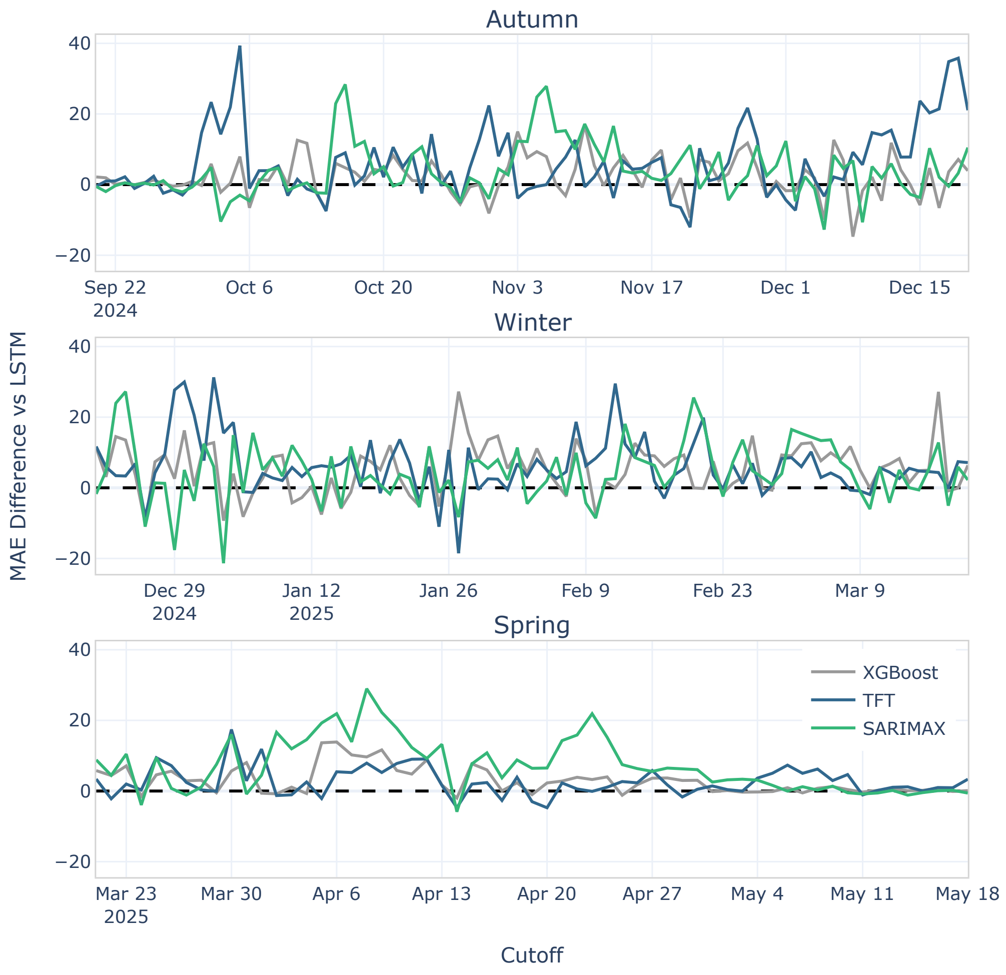


Figure 5.1: MAE differences per cutoff relative to LSTM baseline across autumn, winter and spring. Positive values indicate worse performance than LSTM.

## Performance Evaluation: LSTM

The LSTM model is consistently more accurate than the others across all seasons and across all evaluated metrics, achieving both lower mean errors and higher win rates. This advantage is particularly evident in winter, the most operationally critical period, where LSTM attains the lowest error in roughly 50% of the 90 forecast windows, while all competing models remain below 18% (see Table 5.2). The gap becomes even clearer in pairwise comparisons: LSTM outperforms SARIMAX on 74.4% of windows in terms of MAE and on 72.2% in terms of sMAPE, with similar figures observed against XGBoost. Against TFT, the LSTM achieves win rates of 82%

for MAE and 87% for sMAPE. The reported standard deviations also tend to be lower for LSTM, which indicates more stable performance across 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 5.3.3.

Although the relative improvement of LSTM over the competing models may appear substantial, the effect becomes more modest when viewed in relation to the scale of the series. For instance, the mean MAE of LSTM during winter is roughly 20% lower than that of XGBoost. However, once absolute errors are normalised by the magnitude of the series, as is approximately achieved by the sMAPE metric, the resulting 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. The usefulness of this improvement, especially given the longer training time required by the LSTM, ultimately depends on the accuracy requirements of the specific application.

#### Performance Evaluation: SARIMAX

Among the remaining models, SARIMAX, XGBoost and TFT are competitive and consistently occupy ranks 2 to 4. Within this group, SARIMAX performs best during winter, where it ranks second only to LSTM, while its accuracy deteriorates markedly 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 also highlights an intrinsic limitation of SARIMAX, and of classical statistical models more generally, when applied to non-stationary time series. The use of exogenous temperature variables and of a Box–Cox transformation mitigates this issue only to a limited extent. In contrast, machine learning models are not constrained by a linear specification of exogenous features and therefore have the capacity to learn more complex dependencies between calendar features, temperature and the target variable. This flexibility typically allows them to perform better during the transition periods.

A possible explanation for why SARIMAX performs particularly poorly in spring, compared with autumn, is that the temperature threshold used to define the warm and cold regimes is more closely aligned with the autumn transition. In spring, strong changes in heat demand may occur later than the corresponding temperature changes, or may unfold more gradually. This increases the likelihood that the model's regime switching becomes misaligned with the true dynamics, and the resulting misalignment tends to persist for longer. This would naturally amplify the performance deterioration observed in spring.

#### 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. Estimating stable weekly and daily seasonal components is inherently difficult, because these patterns are subject to continuous small shifts. As a result, the MSTL-ETS algorithm can occasionally extract seasonal components that are less appropriate than a simple repetition of the previous day's pattern, which may amplify noise or capture spurious fluctuations. In such situations, simply repeating the previous day's profile, as the Naive24h baseline does, produces more accurate forecasts.

Figure 5.2 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.

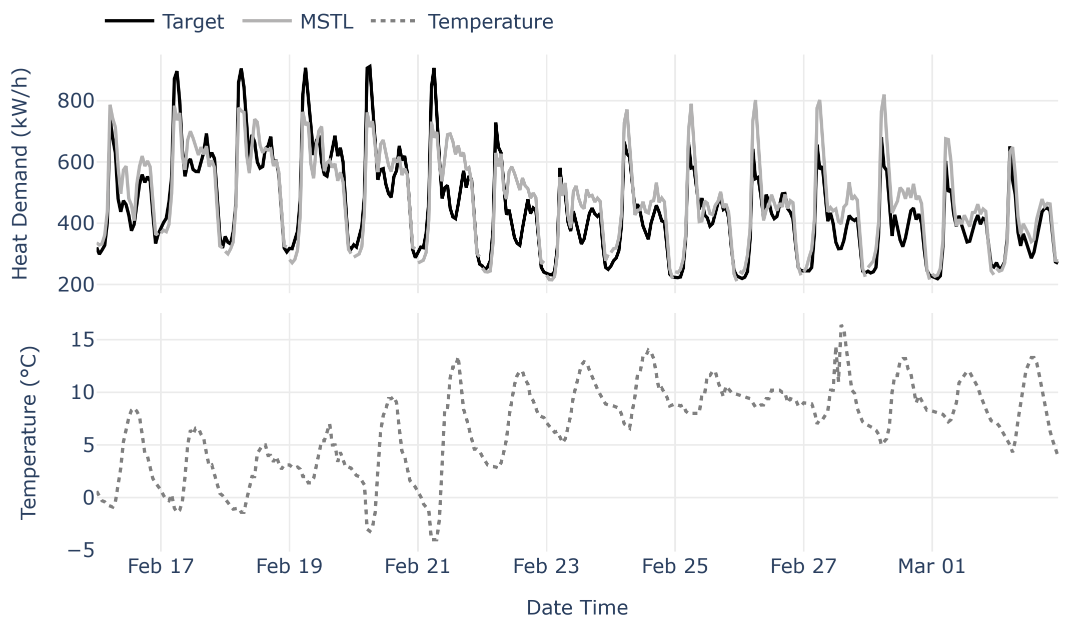


Figure 5.2: Forecasts produced by the MSTL-ETS model for F1 during selected winter days. The figure illustrates a delayed response of the model to target fluctuations associated with temperature variations.

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

TFT shows marked variability in predictive accuracy across seasons. It performs comparably to LSTM in spring, yet it falls to fourth place in the other periods. This variability is consistent with the greater training difficulty of TFT, which contains many more parameters and architectural components than a standard LSTM. Under limited data, this additional complexity can lead to less stable optimisation and increased sensitivity to hyperparameter choices and training schedules.

The fact that LSTM consistently outperforms TFT, even though TFT builds on the same recurrent foundation by adding attention layers, gating mechanisms and variable selection networks, indicates that the combination of a high capacity architecture and restricted data availability can outweigh the theoretical benefits of the more expressive design.

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 [26]. 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. Its large number of parameters therefore becomes substantially harder to estimate in a reliable way, which further contributes to the variability observed across seasons.

### 5.3.3 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 will often do so on the next one also, and because the models are re-estimated only once per week. In addition, the distribution of the per window errors may vary 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 [32]. Let $\{X_t\}_{t\in\mathbb{Z}}$ denote a strictly stationary process with observed 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 LSTM for a specified metric. For $i = 1, \dots, n$, define the circular block of length $\ell$ by

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

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 boostrap 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

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

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

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

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

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

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 [23, 33]. To align with these assumptions, we restrict attention to a central cold regime period in which the loss differences appear approximately stationary (see Figure 5.1) and do not exhibit structural changes. The selected period spans 2024-11-10 to 2025-03-17.

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 [33] with the correction of Patton et al. [31]. 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 5.3 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 equal to zero, and therefore remain zero after multiple testing adjustments such as the Bonferroni correction. 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 equal to zero.

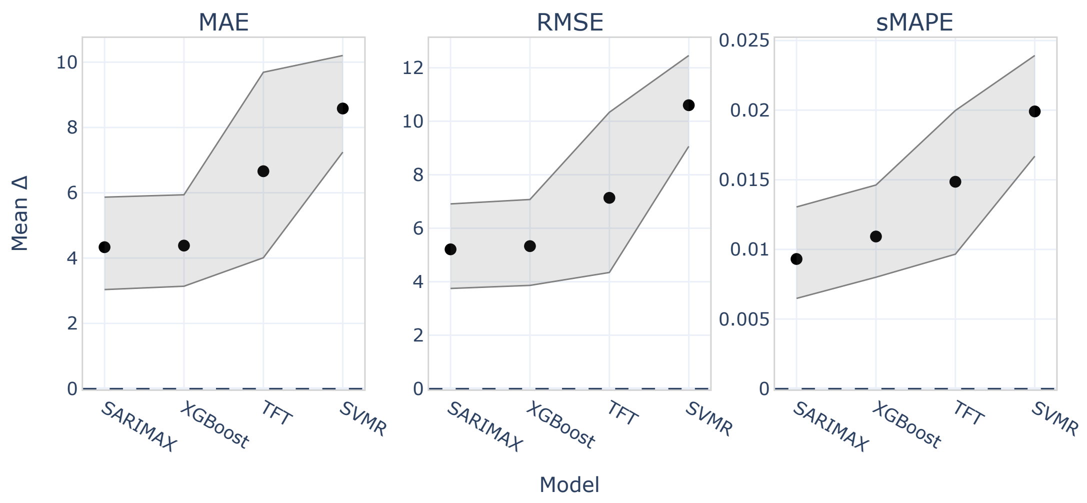


Figure 5.3: Mean performance differences ($model - LSTM$) for MAE, RMSE and sMAPE across all models. Point estimates are shown with 95% bootstrap confidence intervals. Positive values indicate that the LSTM achieves lower error.

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.

### 5.3.4 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 cutoff. 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 5.4 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 5.3.3. 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 error profiles become relatively stable across hours. For the most accurate models (LSTM, SARIMAX, XGBoost, TFT), 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 $h = 1$. The only clear exception is SVMR, for which there are signs of diverging accuracy at $h = 23$ and $h = 24$.

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

### 5.3.5 Residual Analysis

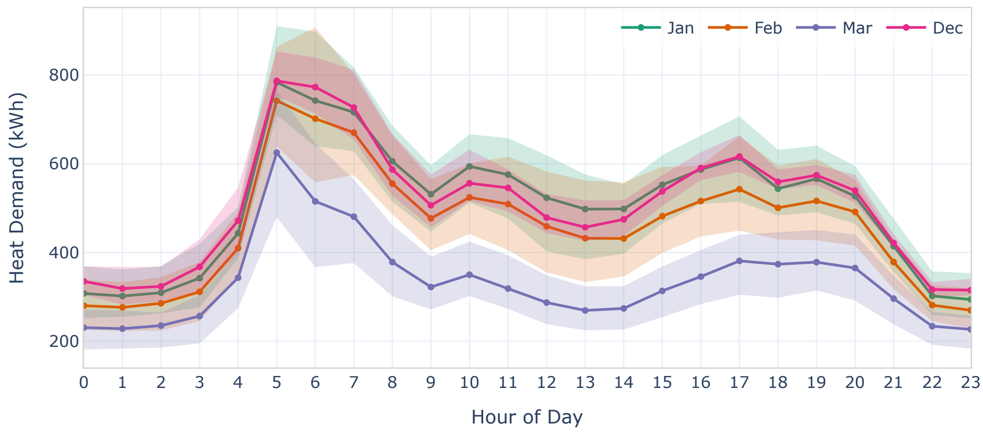
Because heat demand behaves relatively consistently during the winter months, without the regime shifts typically seen in transitional seasons, Auto-Correlation Function (ACF) plots provide a useful way to assess how model errors relate across different lags. Lower autocorrelation values indicate that fewer systematic structures remain in the residuals. An ideal model, one that captures the relevant dynamics and incorporates all necessary explanatory information, would display statistically negligible autocorrelations at all lags.

Figure 5.5 presents the ACFs for a selection of models. The LSTM shows the weakest overall temporal dependence, with autocorrelations falling near the 95% confidence interval for the ACF under white noise assumptions by around lag 12, though 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 within those lags is expected. The TFT, by contrast, exhibits the strongest autocorrelation patterns, particularly 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.

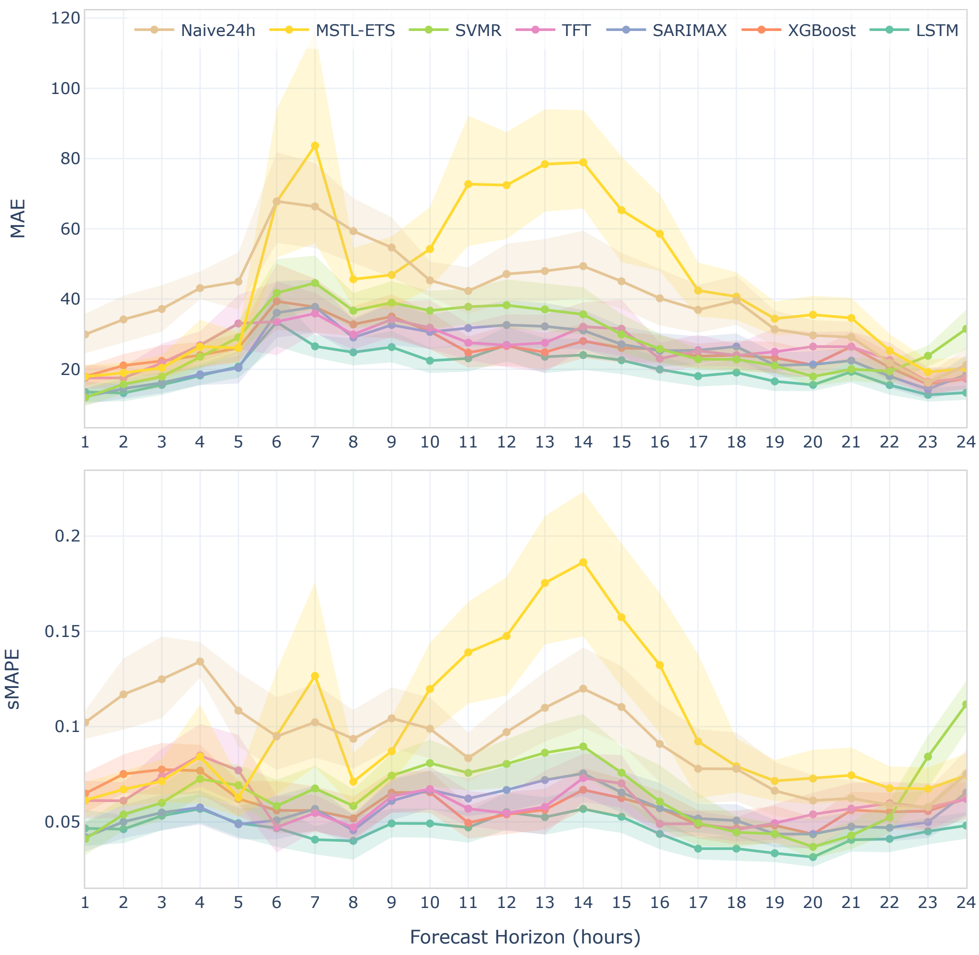
Histogram plots of the residuals show broadly symmetric distributions across models, although small systematic biases remain. For SARIMAX and LSTM the mean residuals are around $-1.5$, while XGBoost and TFT show larger negative biases of approximately $-5$ and $-7$ respectively. These values are non-negligible when compared with the mean MAE errors, which are in the range of 25 to 30, but they remain small relative to the overall level of the series, which is on the order of a few hundred.

## 5.4 Series F1: Week Ahead Forecast Evaluation

We next examine the performance of all models on the week ahead forecasting task for series F1. Each forecast window spans a full week, and because the windows are non-overlapping the number of windows per season is limited (for example, only



(a)



(b)

Figure 5.4: Hourly heat demand and forecasting performance by horizon. (a) Average hourly heat demand profiles by month, with 80% variability bands. (b) MAE and sMAPE by forecast horizon, with 95% bootstrap percentile intervals.

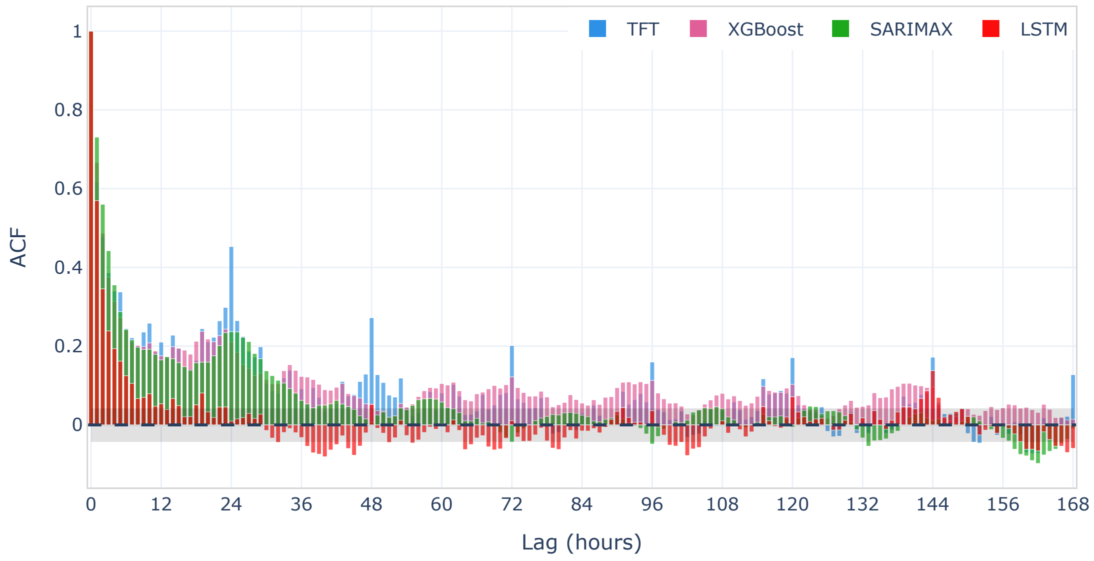


Figure 5.5: ACF of residuals for the TFT, XGBoost, SARIMAX, and LSTM models over lags up to 168 hours. The shaded grey band denotes the 95% confidence interval for the ACF under white-noise assumptions, providing a reference for identifying statistically significant autocorrelations.

13 in winter). This inevitably increases the uncertainty of the per season summaries, yet the patterns visible in Table 5.3 remain coherent across metrics and seasons.

Across nearly all seasons and error measures, the LSTM model displays a marked advantage. Its win rates reach close to 70% for the absolute metrics and around 60% for the scaled sMAPE in the winter period, and similar tendencies are visible in the other seasons, although less pronounced. Compared with the day-ahead horizon, the superiority of LSTM becomes more pronounced, suggesting that it maintains accuracy more effectively as the forecasting horizon increases.

The TFT also benefits from the longer horizon. While it ranked below several competitors at the day-ahead horizon in autumn and winter, it now consistently places second across most metrics. This suggests that both deep learning models capture medium-range temporal dependencies more effectively than the classical or machine-learning baselines. Their long look-back windows (up to 240 hours, i.e., 10 days), their ability to represent non-linear interactions, and their explicit modelling of sequential structure all appear particularly valuable at longer horizons, where errors accumulate over time and relevant dependencies often span several days.

The statistical significance of these improvements is confirmed using the same

Circular Block Bootstrap procedure described in Section 5.3. Applied to an evaluation period of 19 weekly windows covering the coldest months (the same used in Section 5.3), the resulting confidence intervals for the difference in mean loss between LSTM and each alternative model lie well above zero for all metrics. This provides strong evidence that the observed gains are not due to sampling variability. At the same time, the relatively high standard deviations in Table 5.3 show that the week-to-week errors fluctuate considerably. Consequently, when only a small number of windows is examined, the models can appear practically similar in performance, as already discussed in Section 5.3 for the day-ahead-forecasting task.

## 5.5 Series F2

For series F2 and onwards, the TFT model was omitted due to its prohibitively high computational cost relative to the modest accuracy gains observed on F1. The results for F2 closely mirror those for F1: the LSTM again attains the highest accuracy, with improvements that are modest relative to the variability and scale of the series but consistent across seasons and metrics. These gains become more pronounced at longer horizons.

### 5.5.1 Day-Ahead Forecasts

Table table 5.4 on page 104 reports per-window errors by season.

**LSTM remains the most accurate model.** The LSTM is the leading method across all seasons and metrics. Its mean MAE is on average 10–20% lower than that of the strongest competing model, and its win rates reach around 50% or more in winter and spring, with only slightly lower values in autumn. Improvements in sMAPE are small, typically in the range of one to three percentage points. Moreover, as in F1, the standard deviations of the per-window errors exceed the mean differences, indicating that day-to-day variability is larger than the average performance gap.

**MSTL and SVMR underperform.** MSTL is consistently weaker than the naïve baseline in all seasons, reflecting the same adaptability limitations observed for F1. In fact, the Naive24h baseline performs comparatively better on F2 because the series exhibits a more stable daily structure, as indicated by the narrower morning variability bands in Figure fig. 1.2 on page 12. SVMR also lags behind the stronger models and does not provide a competitive alternative.

**Bootstrap assessment of significance.** The day-ahead bootstrap analysis in Figure fig. 5.6 on the next page follows the same Circular Block Bootstrap protocol used for F1. Confidence intervals for the mean loss differences ($model-$ LSTM) are strictly positive across all competing models and metrics, with all $p$-values equal to zero, indicating that the LSTM advantage on F2 cannot be attributed to sampling variability. As in F1, the conclusions are robust to different choices of block length, with all reasonable specifications yielding the same qualitative outcome.

### 5.5.2 Week-Ahead Forecasts

The week-ahead results, summarised in Table table 5.5 on page 105, reinforce the conclusions drawn from the day-ahead analysis. As in F1, the LSTM maintains a clear lead across all seasons and metrics. Its advantage becomes even more pronounced at this longer horizon: win rates exceed those seen in the day-ahead task, particularly in winter, where the LSTM attains a sMAPE win rate above 90%. This mirrors the behaviour observed for F1, where end-to-end sequence models benefit from horizon-level optimisation and better preserve accuracy as forecast length increases.

The remaining models form the same second tier as in the day-ahead setting. XGBoost and SARIMAX emerge as the strongest baselines, with their relative ranking varying by season, but neither approaches the consistency of the LSTM. SARIMAX again deteriorates markedly during the spring transition period, performing worse than all other models, consistent with its sensitivity to regime changes already noted for F1.

Overall, the week-ahead results for F2 closely parallel those of F1 and underscore the same structural differences among the modelling approaches: the LSTM scales more effectively to longer horizons, while classical and single-step machine learning baselines tend to lose accuracy more rapidly as errors accumulate over multiple recursive steps.

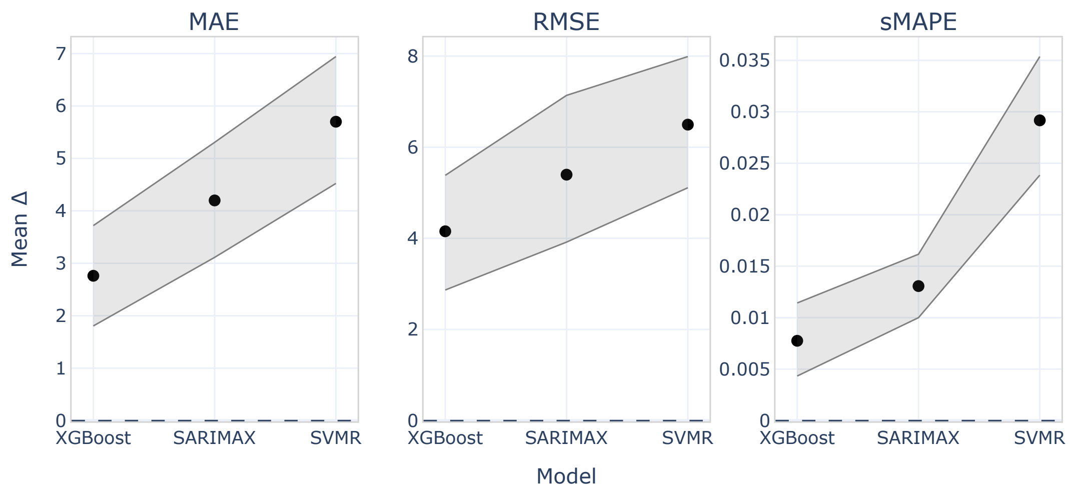


Figure 5.6: Mean performance differences ($model - LSTM$) for selected metrics and models on series F2 and daily-ahead forecasts. Point estimates are shown with 95% bootstrap confidence intervals. Positive values indicate that the LSTM achieves lower error.

## 5.6 Series F3–F5

For the F3 to F5 series, the scope for improvement is rather limited. While the LSTM generally remains among the best performing approaches, its advantages over the competing models are often modest.

### 5.6.1 Day-Ahead Forecasts

Tables 5.6 to 5.8 (from page 106 to 108) report per-window statistics for day-ahead forecasts grouped by season. Across these series, the LSTM often ranks first, but the magnitude of these improvements is usually small. The clearest gains appear in spring, suggesting that the LSTM may still provide marginal benefits during transitional phases.

This impression of having non-significant improvements during the peak demand period is supported by the bootstrap analysis. The confidence intervals constructed using the procedure described in Section 5.3.3, and presented in Figure 5.8, show that the average errors of XGBoost and SARIMAX frequently overlap with those of the LSTM. Consequently, many of the apparent advantages visible in the point estimates are not statistically meaningful.

SARIMAX continues to underperform during the mid-seasons compared with LSTM and XGBoost, a pattern particularly evident for F3, likely due to its rigid treatment of weekly seasonality through Fourier terms in the Box–Cox-transformed space.

### 5.6.2 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 5.9 to 5.11 (from page 109 to 111), LSTM and XGBoost generally remain the leading models, but their relative ordering varies more noticeably across seasons. Their win-rate values also diverge from their mean errors more frequently than in the day-ahead results, indicating more uneven behaviour across windows.

Most series display the same broad pattern as in the day-ahead horizon, with only one clear departure: in winter for F4, the LSTM performs slightly worse than most other models. Aside from this exception, the overall hierarchy of methods largely mirrors that observed in the day-ahead forecasts, albeit with increased variability.

### 5.6.3 Interpretation and Error Structure

The limited improvements offered by the LSTM on these series suggest that the available input data may not support large accuracy differences between models. As the noise level increases, the benefits of refining the same information tend to diminish, and further gains might require additional explanatory variables. For example, occupancy-related information could plausibly introduce predictive structure that is not captured by the existing inputs.

Other explanations remain possible. Different models may still be missing relevant patterns, or additional modelling choices could yield improvements. Nonetheless, the observation that methods of diverse nature achieve broadly similar accuracies on these series (especially during winter) indicates that the current dataset does not strongly favour one approach over another.

Further insight comes from an examination of the residual correlations. Figure 5.7 reports the correlations between the winter residuals of the LSTM and those of the other models. These correlations strengthen progressively from F3 to F5, indicating that an increasingly large fraction of the error is shared across modelling approaches. The pattern is most pronounced for XGBoost, whereas SARIMAX shows slightly lower correlation on F3 and F5, which may reflect its more rigid handling of weekly seasonality. These residual similarities support the view that the remaining errors are not model-specific, but rather that the models tend to fail in similar ways (to a greater extent than in F1 and F2).

In summary, for series F3–F5 the LSTM does not provide clear improvements over simpler alternatives, although it consistently remains among the models with the lowest errors across both forecasting horizons. XGBoost typically ranks immediately behind it, whereas SARIMAX tends to underperform except during peak demand periods. The combination of overlapping confidence intervals, shared residual structure, and increased noise suggests that these series are affected by a practical accuracy ceiling under the current modelling and data constraints.

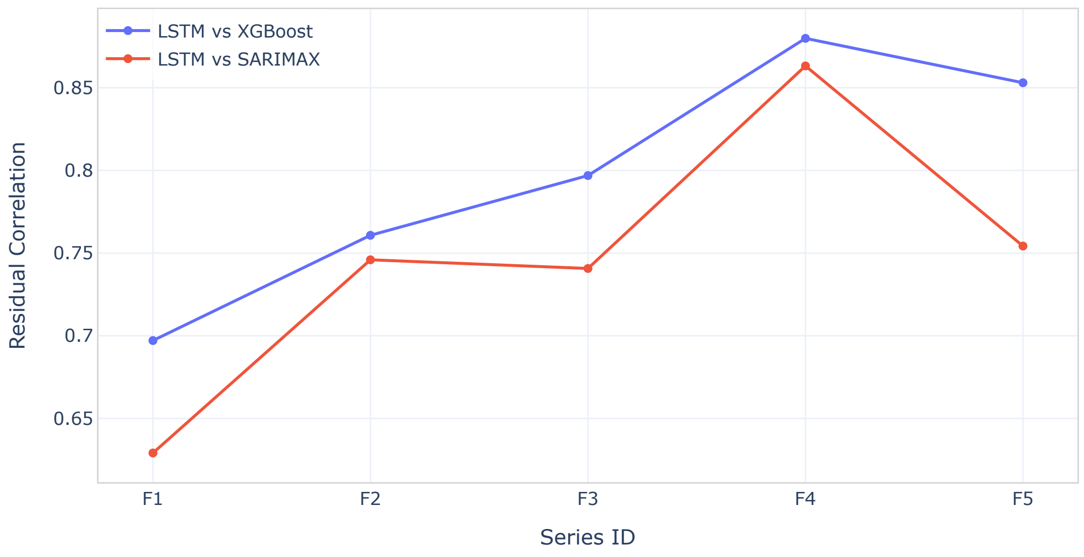
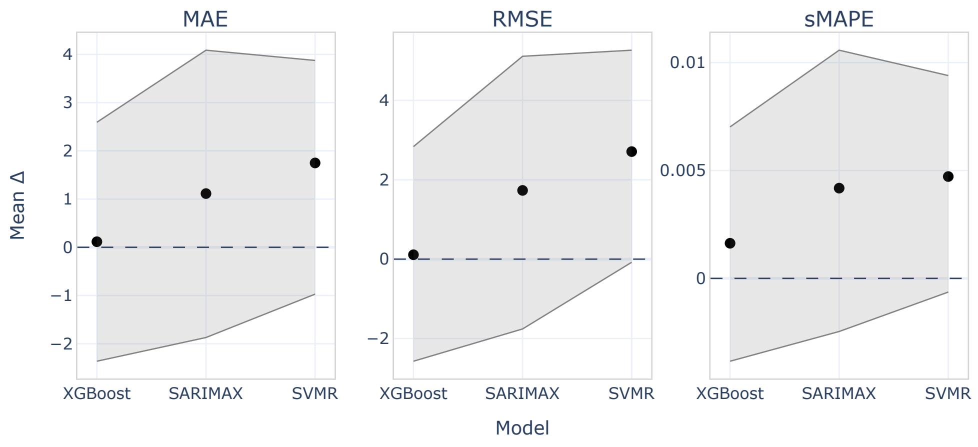
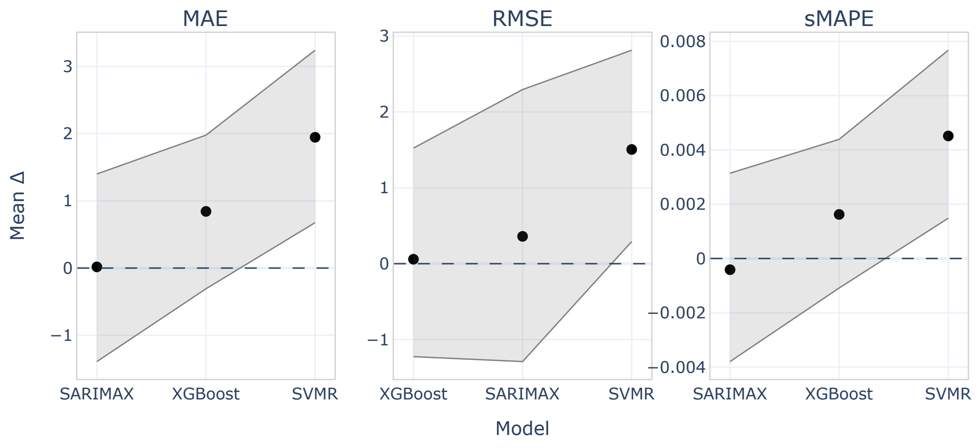


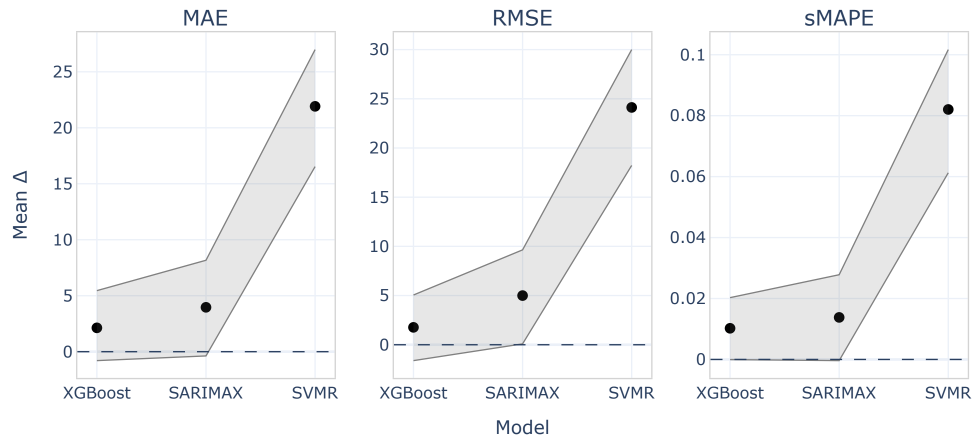
Figure 5.7: Correlations between the winter residuals of the LSTM and those of XGBoost and SARIMAX across series F1 to F5 for day-ahead forecasts.



(a)



(b)



(c)

Figure 5.8: Mean performance differences ($model - LSTM$) for selected metrics and models on series F3–F5 and daily-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) | 46.0 | 28.49 (12.96) | 46.0 | 13.70 (14.88) | 54.0 |
| TFT | 25.14 (13.48) | 31.0 | 31.67 (14.59) | 31.0 | 14.64 (14.36) | 23.0 |
| XGBoost | 26.89 (13.00) | 15.0 | 34.51 (14.38) | 8.0 | 15.38 (14.29) | 15.0 |
| SARIMAX | 30.07 (17.21) | 0.0 | 37.96 (19.47) | 0.0 | 16.79 (14.56) | 0.0 |
| MSTL | 36.39 (23.05) | 8.0 | 46.71 (26.25) | 15.0 | 16.70 (10.23) | 8.0 |
| SVMR | 42.05 (16.37) | 0.0 | 51.12 (18.25) | 0.0 | 31.51 (34.38) | 0.0 |
| Naive24h | 49.00 (35.78) | 0.0 | 60.16 (40.39) | 0.0 | 22.53 (15.87) | 0.0 |

(b) Winter (13 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 24.70 ( 6.89) | 69.2 | 31.99 (10.20) | 69.2 | 5.74 ( 5.38) | 61.5 |
| TFT | 28.11 ( 7.75) | 7.7 | 36.16 (10.74) | 7.7 | 6.62 ( 5.67) | 15.4 |
| SARIMAX | 32.01 (12.72) | 23.1 | 40.40 (14.78) | 23.1 | 7.38 ( 6.56) | 23.1 |
| XGBoost | 32.43 ( 9.65) | 0.0 | 41.80 (13.17) | 0.0 | 7.52 ( 6.85) | 0.0 |
| SVMR | 38.58 (10.79) | 0.0 | 48.34 (13.56) | 0.0 | 9.26 ( 8.15) | 0.0 |
| MSTL | 53.75 (12.31) | 0.0 | 69.52 (14.14) | 0.0 | 12.36 (12.22) | 0.0 |
| Naive24h | 57.57 (24.06) | 0.0 | 71.25 (27.89) | 0.0 | 13.28 (10.94) | 0.0 |

(c) Spring (8 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 10.81 ( 7.12) | 38.0 | 14.58 (10.06) | 50.0 | 8.17 ( 1.88) | 50.0 |
| TFT | 11.86 ( 7.71) | 50.0 | 16.26 (11.12) | 38.0 | 8.51 ( 1.05) | 38.0 |
| XGBoost | 15.26 ( 9.77) | 0.0 | 20.45 (13.40) | 0.0 | 10.76 ( 2.95) | 0.0 |
| Naive24h | 25.37 (17.82) | 0.0 | 31.36 (21.99) | 0.0 | 18.12 (10.39) | 0.0 |
| MSTL | 25.70 (16.23) | 0.0 | 33.02 (21.19) | 0.0 | 18.24 ( 6.32) | 0.0 |
| SARIMAX | 25.72 (15.34) | 12.0 | 30.92 (18.44) | 12.0 | 19.58 (11.57) | 12.0 |
| SVMR | 28.24 ( 7.86) | 0.0 | 33.95 ( 8.11) | 0.0 | 27.02 (14.88) | 0.0 |

Table 5.4: Per-window error statistics for the day-ahead forecast horizon on F2. 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 | 17.25 ( 8.49) | 48.0 | 23.07 (11.94) | 39.0 | 13.86 ( 9.04) | 36.0 |
| XGBoost | 20.31 (10.78) | 11.0 | 26.43 (14.44) | 16.0 | 16.14 (12.38) | 14.0 |
| SARIMAX | 20.99 (14.43) | 21.0 | 27.51 (18.68) | 21.0 | 15.54 (12.50) | 28.0 |
| SVMR | 23.08 (11.11) | 8.0 | 29.27 (13.95) | 11.0 | 22.45 (18.95) | 4.0 |
| Naive24h | 24.51 (15.17) | 4.0 | 31.51 (19.48) | 4.0 | 17.69 (11.20) | 5.0 |
| MSTL | 40.75 (33.27) | 9.0 | 53.19 (41.64) | 9.0 | 25.98 (21.10) | 12.0 |

(b) Winter (90 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 18.99 ( 7.18) | 54.0 | 26.16 (13.51) | 47.0 | 6.38 ( 2.05) | 47.0 |
| XGBoost | 22.10 ( 9.69) | 21.0 | 30.93 (16.92) | 23.0 | 7.28 ( 2.68) | 26.0 |
| SARIMAX | 23.86 ( 8.68) | 9.0 | 32.25 (14.75) | 15.0 | 7.88 ( 2.56) | 9.0 |
| SVMR | 24.40 ( 8.75) | 7.0 | 32.57 (15.04) | 6.0 | 9.02 ( 3.20) | 5.0 |
| Naive24h | 30.21 (12.07) | 7.0 | 40.23 (19.39) | 8.0 | 10.09 ( 4.00) | 5.0 |
| MSTL | 37.51 (15.85) | 2.0 | 50.65 (21.52) | 1.0 | 10.60 ( 4.41) | 8.0 |

(c) Spring (59 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 12.56 (10.15) | 58.0 | 17.00 (15.02) | 58.0 | 15.14 ( 7.69) | 59.0 |
| XGBoost | 15.53 (12.37) | 15.0 | 21.20 (17.72) | 19.0 | 18.78 ( 9.17) | 12.0 |
| SVMR | 18.11 (11.21) | 5.0 | 23.62 (15.83) | 8.0 | 26.06 (14.36) | 5.0 |
| Naive24h | 18.81 (15.01) | 3.0 | 25.00 (20.57) | 3.0 | 20.67 ( 9.07) | 5.0 |
| SARIMAX | 19.45 (14.64) | 17.0 | 26.13 (20.37) | 10.0 | 19.87 ( 9.38) | 15.0 |
| MSTL | 37.43 (29.70) | 2.0 | 51.35 (39.46) | 2.0 | 34.12 (17.11) | 3.0 |

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) | 23 |
| SARIMAX | 33.38 (29.71) | 23 | 43.32 (36.10) | 8 | 25.98 (29.15) | 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 |
| 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) | 8 | 15.13 ( 6.12) | 0 |

(c) Spring (8 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 17.44 (11.10) | 41 | 22.09 (14.31) | 39 | 8.97 ( 5.09) | 37 |
| SVMR | 21.17 (12.08) | 22 | 26.38 (15.38) | 15 | 11.05 ( 6.28) | 22 |
| XGBoost | 22.30 (15.58) | 12 | 27.19 (18.05) | 17 | 11.15 ( 6.21) | 14 |
| MSTL | 23.41 (13.22) | 10 | 28.03 (15.80) | 17 | 11.84 ( 4.82) | 10 |
| Naive24h | 25.95 (18.13) | 8 | 32.60 (21.93) | 7 | 12.47 ( 6.40) | 10 |
| SARIMAX | 30.01 (18.82) | 7 | 34.75 (20.42) | 5 | 15.34 ( 8.81) | 7 |

Table 5.6: 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 |
| LSTM | 32.96 (22.62) | 25 | 41.24 (28.43) | 28 | 13.11 (11.15) | 24 |
| SVMR | 33.85 (20.99) | 22 | 42.92 (26.81) | 22 | 14.13 (11.53) | 24 |
| XGBoost | 36.37 (18.50) | 11 | 44.57 (23.19) | 14 | 17.75 (16.61) | 11 |
| MSTL | 38.70 (33.72) | 14 | 48.77 (37.28) | 13 | 14.63 ( 8.16) | 14 |
| SARIMAX | 38.84 (21.37) | 16 | 47.54 (26.01) | 13 | 17.80 (14.91) | 13 |
| Naive24h | 39.29 (25.84) | 12 | 51.27 (32.90) | 10 | 15.64 (12.12) | 14 |

(b) Winter (90 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 33.20 (18.73) | 28 | 41.47 (22.73) | 29 | 7.27 ( 3.76) | 26 |
| XGBoost | 35.29 (17.44) | 17 | 43.42 (19.84) | 18 | 7.87 ( 3.71) | 16 |
| SVMR | 35.55 (16.21) | 13 | 44.49 (19.88) | 11 | 7.92 ( 3.45) | 13 |
| SARIMAX | 36.36 (16.18) | 22 | 45.22 (19.83) | 26 | 8.12 ( 3.54) | 24 |
| MSTL | 42.78 (22.54) | 14 | 52.81 (25.57) | 10 | 9.42 ( 4.45) | 14 |
| Naive24h | 47.71 (24.71) | 6 | 60.61 (30.23) | 7 | 10.54 ( 5.17) | 7 |

(c) Spring (59 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 17.44 (11.10) | 41 | 22.09 (14.31) | 39 | 8.97 ( 5.09) | 37 |
| SVMR | 21.17 (12.08) | 22 | 26.38 (15.38) | 15 | 11.05 ( 6.28) | 22 |
| XGBoost | 22.30 (15.58) | 12 | 27.19 (18.05) | 17 | 11.15 ( 6.21) | 14 |
| MSTL | 23.41 (13.22) | 10 | 28.03 (15.80) | 17 | 11.84 ( 4.82) | 10 |
| Naive24h | 25.95 (18.13) | 8 | 32.60 (21.93) | 7 | 12.47 ( 6.40) | 10 |
| SARIMAX | 30.01 (18.82) | 7 | 34.75 (20.42) | 5 | 15.34 ( 8.81) | 7 |

Table 5.7: 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 |
| LSTM | 26.09 (19.51) | 37 | 35.55 (27.05) | 32 | 9.76 ( 4.51) | 32 |
| XGBoost | 26.91 (20.85) | 22 | 36.59 (28.12) | 26 | 10.29 ( 5.29) | 23 |
| SARIMAX | 28.76 (23.82) | 14 | 38.56 (30.77) | 15 | 11.08 ( 6.61) | 15 |
| SVMR | 29.20 (19.05) | 10 | 38.69 (26.02) | 13 | 12.20 ( 6.81) | 10 |
| MSTL | 29.68 (23.13) | 10 | 40.01 (30.70) | 10 | 11.19 ( 5.91) | 11 |
| Naive24h | 31.62 (26.34) | 8 | 44.65 (35.87) | 4 | 12.14 ( 7.41) | 10 |

(b) Winter (90 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| SARIMAX | 30.30 (16.09) | 28 | 41.13 (22.85) | 27 | 6.80 ( 2.92) | 29 |
| LSTM | 30.70 (13.82) | 21 | 41.51 (20.04) | 21 | 6.98 ( 2.45) | 22 |
| XGBoost | 31.71 (15.57) | 10 | 41.41 (20.33) | 21 | 7.16 ( 2.79) | 9 |
| SVMR | 32.87 (14.84) | 7 | 43.33 (20.07) | 11 | 7.50 ( 2.74) | 9 |
| Naive24h | 34.18 (19.11) | 26 | 50.56 (29.48) | 13 | 7.62 ( 3.52) | 24 |
| MSTL | 39.21 (17.83) | 9 | 54.87 (25.80) | 7 | 8.80 ( 3.25) | 7 |

(c) Spring (59 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 16.76 (12.62) | 34 | 21.27 (16.07) | 36 | 8.70 ( 5.34) | 34 |
| SVMR | 18.65 (11.76) | 17 | 23.57 (14.58) | 10 | 10.37 ( 5.97) | 17 |
| XGBoost | 19.10 (17.44) | 10 | 24.14 (20.56) | 10 | 9.91 ( 8.09) | 8 |
| Naive24h | 20.93 (17.34) | 8 | 26.96 (22.61) | 8 | 10.61 ( 6.77) | 10 |
| MSTL | 21.43 (14.83) | 10 | 26.39 (17.37) | 7 | 10.91 ( 5.50) | 10 |
| SARIMAX | 23.10 (18.76) | 20 | 27.28 (21.15) | 29 | 12.15 ( 9.63) | 20 |

Table 5.8: 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 |
| LSTM | 26.27 (23.22) | 21 | 35.99 (32.68) | 14 | 18.06 (14.34) | 21 |
| XGBoost | 28.61 (25.97) | 18 | 38.21 (35.16) | 25 | 17.88 (14.16) | 20 |
| SARIMAX | 30.81 (27.57) | 20 | 41.00 (37.62) | 16 | 18.93 (14.83) | 20 |
| MSTL | 39.22 (37.60) | 9 | 50.30 (47.82) | 11 | 23.46 (19.10) | 10 |
| SVMR | 42.59 (32.36) | 9 | 54.01 (40.55) | 9 | 33.54 (24.03) | 8 |
| Naive24h | 47.21 (59.60) | 24 | 63.32 (73.40) | 25 | 24.22 (24.19) | 23 |

(b) Winter (90 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 43.78 (32.64) | 34 | 54.21 (36.64) | 34 | 14.45 ( 9.94) | 32 |
| XGBoost | 45.10 (28.80) | 16 | 55.30 (33.21) | 16 | 14.99 ( 8.49) | 16 |
| SARIMAX | 46.86 (26.79) | 18 | 58.19 (31.99) | 18 | 15.65 ( 8.48) | 21 |
| MSTL | 57.72 (34.62) | 16 | 70.88 (40.44) | 16 | 19.37 (11.81) | 14 |
| SVMR | 63.36 (28.94) | 3 | 75.53 (31.60) | 6 | 22.52 (12.62) | 4 |
| Naive24h | 89.34 (64.67) | 13 | 107.97 (73.41) | 11 | 29.79 (21.33) | 12 |

(c) Spring (59 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 17.14 (12.66) | 34 | 23.39 (17.39) | 31 | 22.44 (18.20) | 32 |
| XGBoost | 17.25 (11.47) | 31 | 23.43 (15.61) | 29 | 22.43 (18.80) | 24 |
| SARIMAX | 23.40 (15.53) | 10 | 31.17 (21.25) | 15 | 27.47 (18.74) | 17 |
| SVMR | 28.28 (19.68) | 7 | 37.08 (26.17) | 7 | 37.20 (24.73) | 8 |
| MSTL | 32.29 (26.23) | 2 | 45.04 (35.43) | 5 | 36.64 (20.05) | 3 |
| Naive24h | 33.25 (35.54) | 17 | 44.99 (46.22) | 14 | 32.40 (22.28) | 15 |

Table 5.9: 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) | 8 | 57.55 (19.57) | 8 | 24.11 (20.95) | 8 |
| LSTM | 47.45 (21.61) | 15 | 58.39 (26.83) | 23 | 22.17 (19.00) | 15 |
| SVMR | 48.73 (20.11) | 31 | 60.88 (25.87) | 31 | 21.67 (15.96) | 31 |
| SARIMAX | 51.93 (23.53) | 23 | 61.87 (25.69) | 23 | 26.02 (21.81) | 23 |
| MSTL | 52.60 (26.10) | 0 | 65.12 (29.31) | 15 | 22.49 (17.23) | 0 |
| Naive24h | 57.12 (37.55) | 23 | 69.86 (39.92) | 0 | 25.36 (21.56) | 23 |

(b) Winter (13 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 37.97 (13.31) | 54 | 48.05 (15.55) | 54 | 8.58 ( 2.46) | 54 |
| XGBoost | 43.24 (12.60) | 23 | 53.32 (14.32) | 23 | 9.97 ( 2.97) | 23 |
| SARIMAX | 44.88 (16.46) | 23 | 55.77 (18.26) | 23 | 10.25 ( 3.34) | 23 |
| SVMR | 49.45 (14.13) | 0 | 60.89 (15.94) | 0 | 11.24 ( 3.20) | 0 |
| MSTL | 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 | 30.83 (19.10) | 12 | 37.24 (21.55) | 25 | 16.16 ( 9.53) | 25 |
| 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) | 25 | 44.78 (11.64) | 12 | 20.96 ( 9.01) | 12 |
| SARIMAX | 42.23 (24.63) | 25 | 48.60 (26.17) | 25 | 21.60 (10.52) | 25 |
| Naive24h | 42.40 (27.35) | 0 | 51.04 (31.28) | 0 | 21.56 (11.66) | 0 |

Table 5.10: 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) | 46 | 42.92 (30.11) | 38 | 12.01 ( 6.17) | 46 |
| LSTM | 32.48 (24.23) | 15 | 43.34 (30.62) | 23 | 12.11 ( 5.85) | 15 |
| SARIMAX | 36.27 (31.09) | 31 | 48.18 (36.68) | 31 | 14.25 ( 9.87) | 31 |
| SVMR | 37.42 (19.07) | 8 | 49.10 (25.25) | 8 | 16.93 ( 9.05) | 8 |
| MSTL | 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 |
| XGBoost | 36.20 (15.09) | 23 | 46.98 (17.31) | 46 | 8.36 ( 2.86) | 23 |
| SARIMAX | 36.51 (15.67) | 31 | 48.61 (19.93) | 23 | 8.35 ( 2.67) | 23 |
| Naive24h | 38.80 (14.44) | 31 | 58.49 (22.90) | 8 | 9.03 ( 3.02) | 31 |
| SVMR | 38.89 (13.82) | 0 | 50.94 (17.25) | 8 | 9.11 ( 2.80) | 8 |
| LSTM | 40.08 (13.42) | 15 | 51.48 (15.96) | 15 | 9.59 ( 3.19) | 15 |
| MSTL | 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) | 12 | 33.40 (28.23) | 12 | 14.35 (12.37) | 12 |
| SVMR | 30.65 (19.45) | 0 | 36.46 (20.67) | 0 | 16.71 ( 9.30) | 0 |
| SARIMAX | 31.26 (28.50) | 38 | 36.23 (30.23) | 38 | 16.34 (14.70) | 38 |
| MSTL | 34.31 (26.93) | 0 | 40.37 (30.17) | 0 | 17.24 (11.31) | 0 |
| Naive24h | 35.31 (36.17) | 25 | 43.74 (40.25) | 25 | 16.84 (15.28) | 25 |

Table 5.11: 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) | 31 | 49.14 ( 39.67) | 38 | 22.73 (11.99) | 31 |
| SARIMAX | 37.24 (32.78) | 15 | 50.85 ( 41.36) | 15 | 24.01 (15.08) | 15 |
| MSTL | 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) | 23 | 114.48 (103.36) | 15 | 40.69 (27.11) | 23 |

(b) Winter (13 forecast windows)

| Model | MAE | | RMSE | | sMAPE (%) | |
|---|---|---|---|---|---|---|
| | mean (std) | wrate | mean (std) | wrate | mean (std) | wrate |
| LSTM | 50.46 (23.45) | 54 | 67.50 (31.85) | 38 | 16.17 ( 6.35) | 46 |
| XGBoost | 50.86 (21.31) | 31 | 66.53 (27.04) | 38 | 16.75 ( 5.60) | 31 |
| SARIMAX | 56.57 (21.32) | 0 | 73.46 (30.81) | 8 | 18.87 ( 5.68) | 8 |
| SVMR | 72.37 (19.33) | 8 | 88.84 (20.94) | 8 | 25.79 ( 5.48) | 0 |
| MSTL | 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) | 50 | 29.99 (18.49) | 25 | 25.75 (13.53) | 50 |
| XGBoost | 23.90 (14.25) | 0 | 33.37 (19.64) | 25 | 28.81 (15.99) | 12 |
| SARIMAX | 28.81 (13.79) | 0 | 40.14 (20.47) | 12 | 34.78 (18.34) | 0 |
| MSTL | 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 real-world 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. The evaluation protocol incorporated rolling cross-validation, periodic retraining, season-specific analyses, per-window and per-hour investigations, residual diagnostics and a bootstrap-based significance assessment that accounted for temporal dependence.

## 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 deep-learning models.* Both LSTM and TFT architectures were implemented and adapted 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.

3. *A detailed comparison with company models.* The thesis provided a rigorous and quantitative comparison between the currently deployed XGBoost and SVMR models and more advanced sequential architectures. This clarified where the existing methods perform well, where they fall short and the extent to which modern recurrent models can yield robust improvements under realistic operating conditions.

## Main Empirical Findings

The empirical results reveal clear differences in model behaviour across the five series. For the two most structured series, F1 and F2, the LSTM achieved the highest accuracy across all seasons and both horizons. These improvements remained statistically significant under the Circular Block Bootstrap, which confirms that the gains cannot be attributed to sampling variability.

Forecast horizon plays an important role. While several models perform competitively at the day-ahead horizon, the performance gaps widen substantially at the week-ahead horizon. The LSTM maintains accuracy far more effectively as the horizon increases. The TFT also benefits from the longer horizon, but its behaviour remains less stable, and the single-series setting restricts the effectiveness of its high-capacity attention mechanisms.

Classical and machine-learning baselines remain strong competitors. XGBoost performs particularly well across many settings and often ranks immediately behind the LSTM. SARIMAX excels during winter but underperforms during transitional periods, reflecting the limitations of its regime-switching structure. These patterns were consistent across both day-ahead and week-ahead evaluations.

For the remaining series, F3 to F5, LSTM and XGBoost obtained broadly similar accuracies, whereas SARIMAX continued to struggle during transitional periods. The residual correlation analyses showed that a larger fraction of the error is shared across models for these series, indicating that the available data may not contain enough explanatory information to support substantial performance differences. In these noisier settings, additional exogenous or behavioural variables are likely required before increased architectural complexity can translate into meaningful gains.

Although several meteorological variables were available, the empirical analysis showed that only temperature (and, redundantly, dew point) provided substantial predictive value. Other exogenous inputs contributed no measurable improvement. This suggests that further accuracy gains on the more irregular series will require information related to usage patterns, occupancy or operational scheduling rather than additional weather features.

Finally, MSTL-ETS proved ill suited to short-term heat-demand forecasting in most cases. Its decomposition-based structure responds slowly to rapid fluctuations, and its performance degrades under infrequent retraining. Nevertheless, on series with clear weekly seasonality it outperformed the naïve baseline, making it a potentially reasonable reference model in contexts where simplicity, interpretability and the absence of temperature forecasts are important operational requirements.

## Practical Implications

Taken together, these findings offer several implications for operational forecasting practice. For the structured series F1 and F2, the LSTM delivers clear accuracy gains over the existing company models. Whether these gains justify replacing XGBoost in practice depends on the relative weight assigned to predictive accuracy versus computational efficiency and scalability. The LSTM remains computationally feasible under weekly retraining and is fully deployable for a small number of series, but XGBoost retains a substantial advantage in speed and ease of scaling to many sites.

On the noisier series, the benefits of deep learning diminish markedly. Without additional exogenous or behavioural variables, the extra modelling complexity does not translate into meaningful improvements, suggesting that XGBoost remains a competitive and operationally attractive option for these cases. Nonetheless, LSTM may still offer marginal improvements during transitional phases, although these are more difficult to assess in a statistically rigorous way.

## Limitations

The two main limitations of this work are summarised below.

First, the evaluation was based on historical temperature values rather than on real forecasting inputs. Actual operational accuracy will therefore be lower, especially for week-ahead predictions, and the relative robustness of the various models to temperature-forecast errors remains unknown. This aspect, not examined in the present study, may influence the relative attractiveness of different modelling approaches in deployment.

Second, the company models and those developed in this project were tuned with slightly different objectives. The models developed here were tuned primarily for the more challenging week-ahead forecasting task, whereas the company models were tuned primarily for the day-ahead task, as short-term predictions are operationally more important. This mismatch has two main consequences:

- The LSTM, if tuned with a primary focus on the day-ahead task, may further improve on the accuracy of the company's current XGBoost model.
- The larger accuracy gap between LSTM and XGBoost in the week-ahead setting may be partly attributable to this tuning misalignment, although given the LSTM's improvement even in the less favourable day-ahead case, it is likely to still offer a substantial performance gain.

## 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. The inclusion of additional exogenous information, such as occupancy patterns, building metadata or holiday calendars, appears especially important for the series with weak temperature dependence.

Another important direction concerns predictive uncertainty quantification. The current analysis focuses solely 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.

A further direction concerns model design. Global training across multiple related series and transfer-learning approaches could help recurrent models exploit shared structure and mitigate the limited data available for individual sites.

Finally, deployment-focused improvements could be explored. These include assessing whether training on the full historical record is necessary or whether models can achieve comparable performance 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 the ability to capture sequential structure.

## Final Remarks

In conclusion, this thesis has shown that a carefully tuned yet relatively simple recurrent neural network can achieve clear and statistically robust accuracy gains for short-term heat-demand forecasting in settings where only one or a few series are available. In this context, the model outperforms both classical statistical and machine-learning baselines as well as the more complex TFT architecture. The results also highlight the constraints posed by limited data, the importance of exogenous information, and the practical considerations associated with computation time and hardware resources. Overall, the methodology and findings offer a basis for enhancing operational forecasting systems and for supporting future developments in data-driven heat-demand modelling in scenarios where efficiency, robustness, and the ability to work with scarce data are essential.

# Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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/.

[10] 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.

[11] 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.wessa.net/download/stl.pdf.

[12] 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.

[13] Grzegorz Dudek, Pawe Piotrowski, and Dariusz Baczyski. Forecasting in modern power systems: Challenges, techniques, and emerging trends. *Energies*, 18(14), 2025. ISSN 1996-1073. doi: 10.3390/en18143589. URL https://www.mdpi.com/1996-1073/18/14/3589.

[14] 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.

[15] 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.

[16] 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. URL https://www.sciencedirect.com/science/article/pii/S0169207099000072.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):17351780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[18] 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.

[19] 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, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/hutter14.html.

[20] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2nd edition, 2018. URL https://otexts.com/fpp2/. Online textbook.

[21] 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.

[22] 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. URL https://www.sciencedirect.com/science/article/pii/0169743995000763. InCINC '94 Selected papers from the First International Chemometrics Internet Conference.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Shuhei 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.

[39] 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.

[40] Henning Wilms, Marco Cupelli, and Antonello Monti. Combining autoregression 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.

[41] 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.