Deliverable 4.1b:

Residential Load Forecasting

Jean-Luc Timmermans
Pierre Henneaux

Brussels, September 2025

Contents

1 Introduction

Electric load forecasting is essential for ensuring the operational security and stability of electrical power systems. To prevent partial or total blackouts, several constraints must be met at all times: the balance between consumption and production must be maintained, voltage levels must remain within acceptable bounds, and transmission and distribution infrastructure, such as lines and transformers, must not exceed their rated capacities. Historically, system operators have relied on load forecasting at high levels of aggregation (national or regional) to inform unit commitment, dispatch, and market clearing, thereby reducing costs and improving operational planning. However, in recent years, the transition towards low-carbon energy systems, characterized by a growing share of Distributed Energy Resources (DER) and increased electrification of end-use sectors, has introduced new challenges. These developments have made forecasting at the local level critical to ensure a safe and reliable operation of power systems. In Europe, the transition to a low-carbon energy system is guided by the legally binding objective of net-zero greenhouse-gas emissions by 2050 [1]. Progress toward this goal relies on several key measures: expanding renewable energy (especially wind [2] and photovoltaics [3]), promoting the adoption of electric vehicles [4], and increasing heat pump deployment [5].

At the local level, changes in both electricity production and consumption are creating additional complexities for distribution networks. The proliferation of distributed energy resources (DERs), such as household photovoltaic (PV) systems, is leading to new operational challenges, particularly in voltage regulation. For instance, when local PV generation exceeds demand, excessive grid injection can cause overvoltage, prompting PV inverters to disconnect automatically, leading to a loss of energy [6]. Additionally, evolving consumption patterns, driven by the widespread adoption of electric vehicles (EVs) and electric heating systems (e.g., heat pumps), are increasing peak loads at the local level. Simultaneous EV charging within residential areas can exceed the capacity of distribution lines and transformers, causing potential overloads and voltage drops [7], [8]. Reinforcing Europe's distribution infrastructure to accommodate these changes will take years and cost billions [9]. In the meantime, a faster and more cost-effective alternative lies in optimal EV charging and optimal scheduling of residential loads, which can enhance self-consumption, reduce peak demand, and improve grid efficiency. Such scheduling depends on highly accurate local load forecasts, a topic that remains relatively underexplored in the literature[1].

This work focuses on day-ahead load forecasting at the residential level, where accurate predictions are crucial for optimizing demand-side management strategies. Specifically, this report presents a comparative analysis of state-of-the-art forecasting models for residential day-ahead load prediction with a low level of aggregation. By evaluating multiple approaches, we aim to identify the most effective forecasting techniques for enhancing local grid reliability and supporting optimal residential load scheduling.

The remainder of this document is organized as follows. Section 2 highlights the contributions of this work to the existing literature. Section 3 introduces the fundamental concepts and main categories of forecasting methodologies, providing the theoretical background relevant to electric load forecasting. Section 4 describes the forecasting models selected for comparison and explains their core mechanisms. Section 5 details the methodology used to ensure a fair and robust evaluation across algorithms. Section 6 presents the experimental results. Finally, Section 7 summarizes the key findings of this comparative study.

[1]As highlighted in Section 2

2 Contributions

In the literature, the majority of publications have traditionally focused on load forecasting at high levels of aggregation, such as national or regional scales, driven by the needs of system operators. In recent years, the widespread deployment of smart meters has enabled access to fine-grained consumption data, leading to a growing number of studies targeting load forecasting at the household level. Despite this progress, forecasting performance at such a granular scale remains limited due to high variability and uncertainty inherent in individual consumption patterns. In contrast, the number of studies addressing load forecasting at low levels of aggregation, ranging from a few households to approximately one hundred, remains relatively limited, particularly for day-ahead forecasting. Yet, given the rise of decentralized energy resources and the evolving nature of electricity consumption, local energy optimization is becoming increasingly crucial. It is therefore important to assess the performance of state-of-the-art algorithms at these lower aggregation levels, in order to determine which methods perform best and to identify the minimum group size for which day-ahead forecasting reliably supports optimal load scheduling. Recent surveys compare STLF methods at the national scale [10] and at the single-building scale [11], but very few papers examine day-ahead forecasting for small groups of households. One review devoted to this niche [12] is limited: it tests only basic models, omits state-of-the-art methods such as XGBoost, TFT, and N-BEATS, ignores exogenous inputs that influence demand, and relies on a single, non-public dataset. Hence, despite a rich STLF literature, the problem of day-ahead load forecasting at low aggregation remains largely unexplored. In this context, the goal of this work is to evaluate and compare state-of-the-art methods for day-ahead electric load forecasting for residential consumption at low levels of aggregation. These methods could be particularly beneficial to optimize self-consumption in local energy communities with local production of energy. In particular, the contributions of this work are the following:

1. We provide a comparative study of state-of-the-art forecasting methods for day-ahead residential electricity consumption at low aggregation levels, ranging from 10 to 100 households. Forecasting studies at this level of aggregation and for this horizon are largely absent from the literature, yet they are of paramount importance for load scheduling at the energy community and local levels.

2. The selected algorithms include traditional statistical models (SARIMAX, Holt–Winters), state-of-the-art machine learning models (XGBoost), industrial modular regression models (Prophet), and advanced deep learning models (N-BEATS, Temporal Fusion Transformer). The comparison is therefore not restricted to machine or deep learning methods, nor to simple benchmarks, unlike many existing studies [13].

3. The comparison is performed on two open-access datasets composed of real smart-meter data, which is uncommon in the literature [13] and supports the generalization of the findings. Performance is evaluated using widely adopted metrics, facilitating the use of our results as a reference in future publications.

4. The comparison is rigorous, thanks to the use of rolling origin and expanding origin cross-validation, and consistent tuning of all the models using Bayesian optimization or complete grid search.

A summary of the work presented in this document has been accepted for presentation at the IEEE Innovative Smart Grid Technologies (ISGT) 2025 conference and will be published in the conference proceedings.

3 General Concept

Forecasting methods can be categorized based on several criteria, including forecasting time horizon, forecast step structure, prediction type, and load level. This section provides an overview of these classifications, emphasizing their significance in different applications.

3.1 Forecasting Time Horizon

The time horizon of a forecast refers to the period between the forecast issuance and the actual realization of the predicted values. Depending on the application, electric load forecasting is typically divided into four main categories [14], [15], [16]:

- Very Short-Term Load Forecasting (VSTLF): Predictions made for a few minutes to one hour ahead. These forecasts are used for real-time grid balancing [17].
- Short-Term Load Forecasting (STLF): Forecasts with horizons typically ranging from one hour to one week. These are widely used for unit commitment, economic dispatch, and day-ahead market trading [18].
- Medium-Term Load Forecasting (MTLF): Forecasts covering a period of one week to a few months ahead. These forecasts are used for maintenance scheduling, fuel procurement, and mid-term energy trading [19].
- Long-Term Load Forecasting (LTLF): Forecasts extending from several months to multiple years ahead. These forecasts are used for grid development planning, infrastructure investments, and policy-making [20].

3.2 Forecasting Step Structure

Another important aspect of forecasting models is the number of steps they predict into the future:

- Single-step forecasting: These methods predict one time step ahead (e.g., the next hour or the next day). By iteratively using predicted values as inputs, they can also be applied to multi-step forecasting. Single-step forecasts are commonly used in real-time operations where immediate decisions are required [21].
- Multi-step forecasting, also called multi-horizon forecasting: These methods generate forecasts for multiple time steps ahead (e.g., hourly load for the next 24 hours). This approach is particularly useful for applications such as day-ahead scheduling [22].

3.3 Point and probabilistic forecasting

Forecasts can also be categorized based on whether they provide a single value prediction or account for uncertainty:

- Point forecasting. Point forecasting generates a single-value estimate of future electricity demand. It has traditionally been the dominant method in load forecasting and remains widely studied in the literature.
- Probabilistic forecasting. Probabilistic forecasting provides a distribution of possible outcomes with associated probabilities, often in the form of prediction intervals or probability density functions. As uncertainties in both energy supply and demand continue to rise, probabilistic forecasting has gained increasing prominence. It offers a more comprehensive representation of future load, enabling grid operators and market participants to make more informed decisions under uncertainty [23], [24].

The methodological frameworks of point and probabilistic forecasting differ substantially. Point forecasting methods are trained to minimize deterministic error functions such as the Mean Squared Error (MSE), and their evaluation relies on accuracy metrics that compare predicted values to realized observations, such as the Mean Absolute Percentage Error (MAPE) (see Section 5.4). Probabilistic forecasting, by contrast, requires models that can capture and represent uncertainty. These models are optimized with distribution-aware loss functions, such as quantile loss [25], the Continuous Ranked Probability Score (CRPS) [26], or the negative log-likelihood [27], and are assessed using proper scoring rules. Such rules evaluate both *reliability* (also referred to as calibration or coverage, i.e., the alignment between predicted probabilities and observed frequencies) and *sharpness* (the concentration of the predictive distribution) [28]. Thus, the difference in predictive output, a single estimate versus a full distribution, directly leads to differences in algorithms, training objectives, and evaluation criteria [29].

3.4 Load Level

In electric load forecasting, the load level refers to the degree of aggregation at which electricity demand is predicted. Figure 1, adapted from the review "Review of Low Voltage Forecasting: Methods, Applications, and Recommendations", by Haben et al [13], provides a simplified representation of an electrical grid layout.

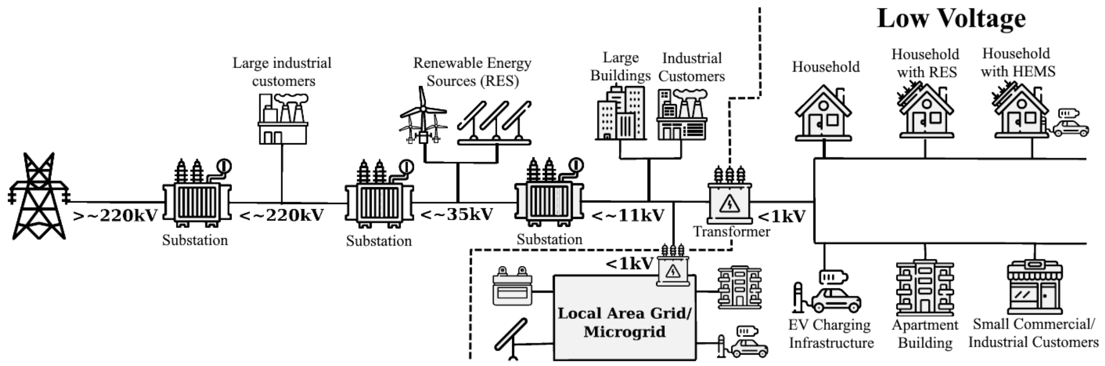


Figure 1: Representation of load level of a simplified electrical grid from [13].

At the transmission, system operators forecast electricity demand for large regions or entire countries to maintain the balance between supply and demand. In Belgium, the transmission system operator (TSO), *Elia*, is responsible for ensuring this balance and forecasting the national electricity demand.

At the opposite end of the spectrum, the Low Voltage (LV) level encompasses electricity consumption by households, buildings, small businesses, and small industries[2]. In the literature, studies focus on different levels of aggregation within the LV network, for instance, individual household load forecasting [30] [31] [32], building load forecasting [33] [34], and transformer load forecasting [35].

[2]In Belgium, the low-voltage level corresponds to voltages below 1 kV, typically 230 V (single-phase) or 400 V (three-phase).

4 Implemented models

In this section, the models selected for our comparative study are described in detail, and their core mechanisms are explained. The forecasting models span several categories: a naive baseline, traditional statistical models (SARIMA, Holt–Winters), a machine learning model (XGBoost), deep learning models (N-BEATS and TFT), and a modular regression model (Prophet). These models were chosen based on their strong performance in short-term load forecasting and their widespread use in recent literature.

4.1 Naive Forecast model

To compare our models against a simple benchmark, a naive forecast model is defined. It predicts the next 24 hours (48 half-hourly steps) by repeating the load observed during the preceding 24 hours. This benchmark is justified by the strong 24-hour periodicity commonly observed in electric load profiles. This daily pattern is visible in Figures 5 and 6 in Section 5.1. An alternative naive forecast could use the load observed one week earlier, leveraging weekly periodicity. However, in our experiments, the naive forecast based on daily seasonality consistently produced better results for both datasets.

4.2 Holt-Winters models

The Holt-Winters model, also known as triple exponential smoothing, was first introduced in 1960 [36] and remains widely used in many forecasting applications today.

Exponential smoothing is a technique where the next value of a time series is predicted as a weighted sum of past observations, with exponentially decreasing weights applied to older values. Mathematically, given a time series $y = \{y_t, y_{t-1}, y_{t-2}, ..\}$ The next value is predicted by $F_t + 1$ in equation (1).

$$F_{t+1} = \alpha y_t + (1 - \alpha)F_t \tag{1}$$

With

$$F_0 = y_0 \tag{2}$$

Where α is the smoothing factor ($0 \leq \alpha \leq 1$). Expanding this equation iteratively by substituting F_t into F_{t+1}, it becomes evident that the forecasted value is a weighted sum of all past observations, with weights decreasing exponentially as observations become older. When α is high, the forecasted value F_{t+1} depends mostly on the most recent observation y_t, while earlier values contribute much less. When α is close to zero, older values still have a significant influence on the forecast, making the predictions smoother and less sensitive to recent fluctuations. This basic form of exponential smoothing is called simple exponential smoothing and works well for data without a trend or seasonality. In 1957, Holt introduced the double exponential smoothing method [37], which extends simple exponential smoothing by adding a trend component, and in 1960, Winters introduced the triple exponential smoothing method by adding a seasonal component [36], leading to what is now known as the Holt-Winters exponential smoothing model. As shown in Figure 2, seasonality can be either additive or multiplicative. Accordingly, the Holt–Winters model is available in both additive and multiplicative forms.

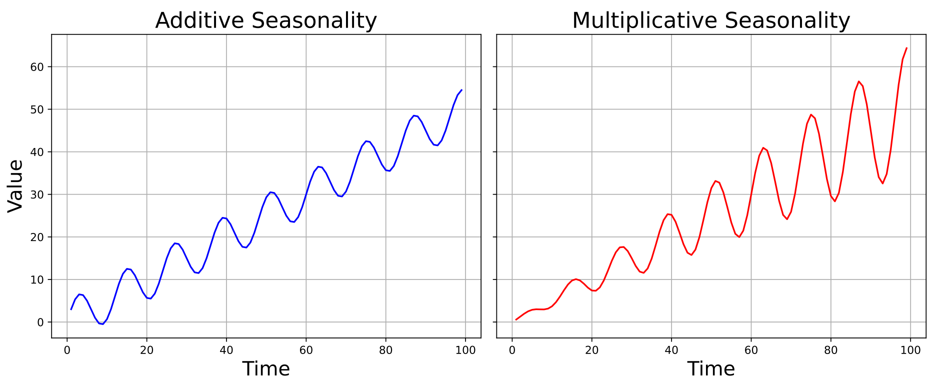


Figure 2: Additive and multiplicative time series.

Equations (3) to (6) provide the multiplicative form of the HW model, while equations (7) to (10) give the additive form (from [38]). The forecasted value for h time steps into the future is denoted as F_{t+h}. The components L_t, T_t, and S_t represent the level, trend, and seasonal components, respectively. The parameters α, β, and γ correspond to the smoothing factor, trend smoothing factor, and seasonal change smoothing factor, respectively, all constrained within the range $[0, 1]$. The variable s denotes the number of periods in a season; for instance, in the case of weekly seasonality with an hourly time step, $s = 168$.

$$L_t = \alpha \frac{y_t}{S_{t-s}} + (1-\alpha)(L_{t-1} + T_{t-1}) \tag{3}$$
$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \tag{4}$$
$$S_t = \gamma \frac{y_t}{L_t} + (1-\gamma)S_{t-s} \tag{5}$$
$$F_{t+h} = (L_t + hT_t)S_{t-s+h} \tag{6}$$

$$L_t = \alpha(y_t - S_{t-s}) + (1-\alpha)(L_{t-1} + T_{t-1}) \tag{7}$$
$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \tag{8}$$
$$S_t = \gamma(y_t - L_t) + (1-\gamma)S_{t-s} \tag{9}$$
$$F_{t+h} = L_t + hT_t + S_{t-s+h} \tag{10}$$

Despite its age, the Holt-Winters model is still used in forecasting applications, including electric load forecasting, in the literature [39].

In our comparative study, the Holt–Winters model is implemented in *Python* using the *statsmodels* library[3].

4.3 ARMA models

ARMA stands for auto-regressive moving average. It is a statistical model used to analyze and forecast stationary time series, and was popularized and formally introduced by Box and Jenkins in 1970 [40]. An ARMA model consists of two components: an autoregressive (AR) part and a moving average

[3]https://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html

(MA) part. In an AR process, the current value of a time series is expressed as a linear combination of its past values, along with a stochastic error term. Mathematically[4], given a stationary time series $y = \{y_{t-1}, y_{t-2}, y_{t-3}, \ldots\}$, the present value, y_t, in an AR process is defined by equation (11).

$$y_t = c + \sum_{i=1}^{p} \beta_i y_{t-i} + \epsilon_t \tag{11}$$

In (11), p is the AR order of the model, β_i are the coefficients, c is a constant, and ϵ_t is the error term. In a moving average (MA) process, the current value of a time series is expressed as a linear combination of past error terms, the mean of the series, and an error term. Mathematically, given a stationary time series $y = \{y_{t-1}, y_{t-2}, y_{t-3}, \ldots\}$, the present value, y_t, in a MA process is defined by equation (12).

$$y_t = \mu + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t \tag{12}$$

In (12), q is the MA order of the model, θ_i are the coefficients, μ is the mean, and ϵ_t is the error term. The ARMA model of order (p, q) is given by equation (13).

$$y_t = c + \sum_{i=1}^{p} \beta_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t \tag{13}$$

The parameters of an ARMA model, β, θ, and c, are typically estimated using maximum likelihood methods. The residual at time step t is defined as the difference between the predicted and actual values of the time series: $\epsilon_t = F_t - y_t$. The predicted value at time t is given by equation (14).

$$F_t = c + \sum_{i=1}^{p} \beta_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{14}$$

If a time series is not stationary[5], ARMA models can be extended to ARIMA models, where the "I" stands for Integrated. In an ARIMA model, the time series is differenced to eliminate non-stationarity before fitting the ARMA structure. An ARIMA(p, d, q) model applies d levels of differencing to achieve stationarity, after which an ARMA(p, q) model is used for forecasting.

While ARIMA models are effective for forecasting non-stationary time series, they do not explicitly account for seasonal patterns. To handle seasonality, ARIMA is extended to the Seasonal ARIMA (SARIMA) model, which incorporates an additional seasonal component. A $SARIMA(p, d, q) \times (P, D, Q)_s$ model includes seasonal auto-regressive, differencing, and moving average terms of order (P, D, Q), with s representing the number of periods in a season. By applying seasonal differencing and seasonal ARMA terms, SARIMA effectively models time series with recurring patterns. The SARIMA model is descibed in equation (15) where B is the back-shift operator, $\phi_p(B)$ and $\theta_q(B)$ are non-seasonal AR and MA polynomials of orders p and q, and $\Phi_P(B^s)$ and $\Theta_Q(B^s)$ are their seasonal counterparts of orders P and Q.

$$\Phi_P\left(B^s\right)\phi_p(B)(1 - B^s)^D(1 - B)^d y_t = \Theta_Q\left(B^s\right)\theta_q(B)\varepsilon_t \tag{15}$$

To incorporate exogenous variables, the model is extended to SARIMAX (SARIMA with exogenous regressors). A $SARIMAX(p, d, q) \times (P, D, Q)_s$ model includes all the components of SARIMA while also allowing for the inclusion of external predictor variables that may influence the time series. The

[4]The mathematical developments in this section are mostly inspired by [41].

[5]A time series is considered stationary if its statistical properties do not change over time. In practice, weak stationarity is sufficient for ARMA models, meaning that the mean and variance remain constant, and the autocovariance depends only on the lag, not on the specific time.

SARIMAX model extends SARIMA by incorporating a linear combination of exogenous variables, allowing external factors to influence the predictions as described in equation (16). Where X_t^i represents the exogenous variable at time t, and γ_i are the corresponding coefficients.

$$SARIMAX(p,d,q) \times (P,D,Q)_s = SARIMA(p,d,q) \times (P,D,Q)_s + \sum_{i=1}^{n} \gamma_i X_t^i \tag{16}$$

To select the orders (p, q, P, Q) of a SARIMA model, information criteria such as the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) are commonly used. The AIC provides an estimate of the relative quality of statistical models fitted to the same dataset. It balances model fit (how well the model explains the data) against model complexity (the number of estimated parameters). The AIC is defined with equation (17).

$$\text{AIC} = 2k - 2\log(\hat{L}), \tag{17}$$

where k is the number of estimated parameters in the model and $\log(\hat{L})$ is the maximized log-likelihood. The log-likelihood measures how likely it is to observe the data under the assumed model. The term $2k$ penalizes more complex models and is defined by equation (18).

$$2k = 2\big(p + q + P + Q + 1 \text{ (if an intercept is included)} + 1 \text{ (for the error variance)}\big). \tag{18}$$

In practice, AIC values are computed for a set of candidate SARIMA models with different orders (p, q, P, Q). The model with the lowest AIC is typically selected. The BIC is another widely used criterion, defined similarly to AIC but with a stronger penalty for the number of parameters. In this study, we use the AIC to select the best model because it tends to allow for slightly more complex models, which is desirable in forecasting applications.

ARIMA is still used for forecasting of electric load in recent literature, often as a benchmark [42], [43], [44].

In our comparative study, the SARIMAX model is implemented in *Python* using the *statsmodels* library[6].

4.4 Prophet

Prophet is an open-source forecasting model developed by *Meta* and released in 2017 [45]. The objective of *Meta* was to provide a forecasting tool tailored for business applications with two main characteristics: (i) accessibility and configurability, enabling a wide range of users (including non-experts in forecasting) to incorporate their domain knowledge into the analysis, and (ii) strong performance across a diverse set of forecasting problems. Prophet offers several other advantages, including fast model fitting, robustness to outliers and missing values, and the ability to handle time series with irregularly spaced observations. Prophet is a modular additive regression model formulated in equation (19), comprising three main components: trend g_t, seasonality s_t, and holiday effects h_t. ϵ_t is the error at time step t.[7]

$$y_t = g_t + s_t + h_t + \epsilon_t \tag{19}$$

In equation (19), the seasonality is modeled as an additive component, but multiplicative seasonality is also possible with this model.

The trend component or growth component g_t is either modeled by a saturating growth model or a piecewise linear model. A linear and a basic logistic growth model are respectively described

[6]https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html

[7]Mathematical development in this section are from the original paper [45].

by equations (20) and (21), where C is the carrying capacity, k the growth rate, and m the offset. Nevertheless, in Prophet, the carrying capacity C and the growth rate k are not constant; the capacity is replaced by a time-varying capacity C_t and the growth rate is allowed to change by explicitly defining changepoints in the time series.

$$g_t = kt + m \tag{20}$$

$$g_t = \frac{C}{1 + exp(-k(t-m))} \tag{21}$$

Prophet allows for the modeling of multiple seasonalities, such as weekly and yearly patterns. Seasonality is represented using a standard Fourier series, equation (22), where P denotes the periodicity (e.g., $P = 7$ for daily data with weekly seasonality), and a_n and b_n are the Fourier coefficients.

$$s_t = \sum_{n=1}^{N} \left(a_n cos\left(\frac{2\pi nt}{P}\right) + b_n sin\left(\frac{2\pi nt}{P}\right) \right) \tag{22}$$

Prophet allows users to add a list of past and future holidays. The effect of each holiday is considered independent and is modeled as a regressor.

Finally, in Prophet, fitting is done with L-BFGS, a gradient-based optimization algorithm used to efficiently find the maximum a posteriori estimate of model parameters by balancing data likelihood and prior information. Prophet is used for electric load forecasting in recent literature as illustrated in [46], [47].

In our comparative study, the prophet model is implemented in *Python* using the *Prophet* library[8].

4.5 XGBoost

Extreme Gradient Boosting (XGBoost) is an efficient and scalable implementation of the gradient boosting algorithm, introduced by Chen and Guestrin in 2016 [48]. XGBoost enhances the classical gradient boosting framework through several key improvements, including: a second-order Taylor expansion to approximate the loss function for faster and more accurate optimization; a regularized objective function to reduce overfitting; a sparsity-aware learning algorithm to effectively handle sparse inputs and missing values; and parallelized tree construction to accelerate model training. Gradient boosting is an ensemble learning technique that builds a strong predictive model by sequentially adding weak learners. In the case of XGBoost, the weak learner is a Classification And Regression Tree (CART). At each iteration, a new tree is trained to minimize the residual errors of the ensemble of previously built trees. Mathematically, given a data set $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ with m features, the prediction of the XGBoost model is defined with equation (23), where $\mathcal{F}$ is the space of regression trees (CART), and K is the number of trees.

$$\hat{y_i} = \sum_{k=1}^{K} f_k(\boldsymbol{x_i}), f_k \in \mathcal{F} \tag{23}$$

The model is trained by minimizing a regularized objective function, given by equation (24). In this equation (24), $l(y_i, \hat{y_i})$ is a convex loss function (e.g., mean squared error) and $\Omega(f)$ is a regularization term that penalizes model complexity. This regularization term is given by equation (25), where T is the number of leaves in the tree, w represents the leaf weights, and γ and λ are regularization parameters.

$$Obj = \sum_{i}^{n} l(y_i, \hat{y_i}) + \sum_{k=1}^{K} \Omega(f_k) \tag{24}$$

[8]https://facebook.github.io/prophet/

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2 \tag{25}$$

At each iteration, the structure and parameters of the $t-th$ tree are learned by minimizing the objective function at step t, where a second-order Taylor expansion of the loss function is employed to accelerate the optimization process, as shown in equation (26).

$$Obj^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(\boldsymbol{x_i}) + \frac{1}{2}h_i f_t^2(\boldsymbol{x_i})] + \Omega(f_t) \tag{26}$$

In equation (26), g_i and h_i are the first and second derivatives, respectively, given by equations (27) and (28).

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}} \tag{27}$$

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial^2 \hat{y}_i^{(t-1)}} \tag{28}$$

The complete mathematical derivation and algorithmic details are provided in the original work by Chen and Guestrin [48]. In recent years, XGBoost has been successfully applied across various electric load forecasting problems [49], [50], [51], [52].

In our comparative study, the XGBoost model is implemented in *Python* using the *XGBoost* library[9].

4.6 N-BEATS

N-BEATS (Neural Basis Expansion Analysis for Time Series Forecasting), introduced by Oreshkin et al. in 2020 [53], is a deep learning architecture specifically developed for univariate time series forecasting. The N-BEATS architecture is characterized by a hierarchical, doubly residual topology illustrated in Figure 3 from [53]. The global forecast is the aggregation of the partial forecasts of each stack. The partial forecast of each stack is itself the aggregation of the partial forecasts of each block. The basic building blocks are composed of fully connected layers with *Relu* activation function and fully connected layers with linear activation function. This architecture allows a hierarchical decomposition of the forecast and a better gradient backpropagation during training, according to the authors. The N-BEATS architecture can be either generic or interpretable. The generic N-BEATS achieved state-of-the-art performance on multiple benchmark datasets, highlighting the capability of deep learning models to operate in a fully data-driven manner without relying on traditional statistical approaches, handcrafted features, or domain expertise. The interpretable architecture allows interpretability of the deep learning models, a quality missing in classical DL implementation.

[9]https://xgboost.readthedocs.io/en/stable/index.html

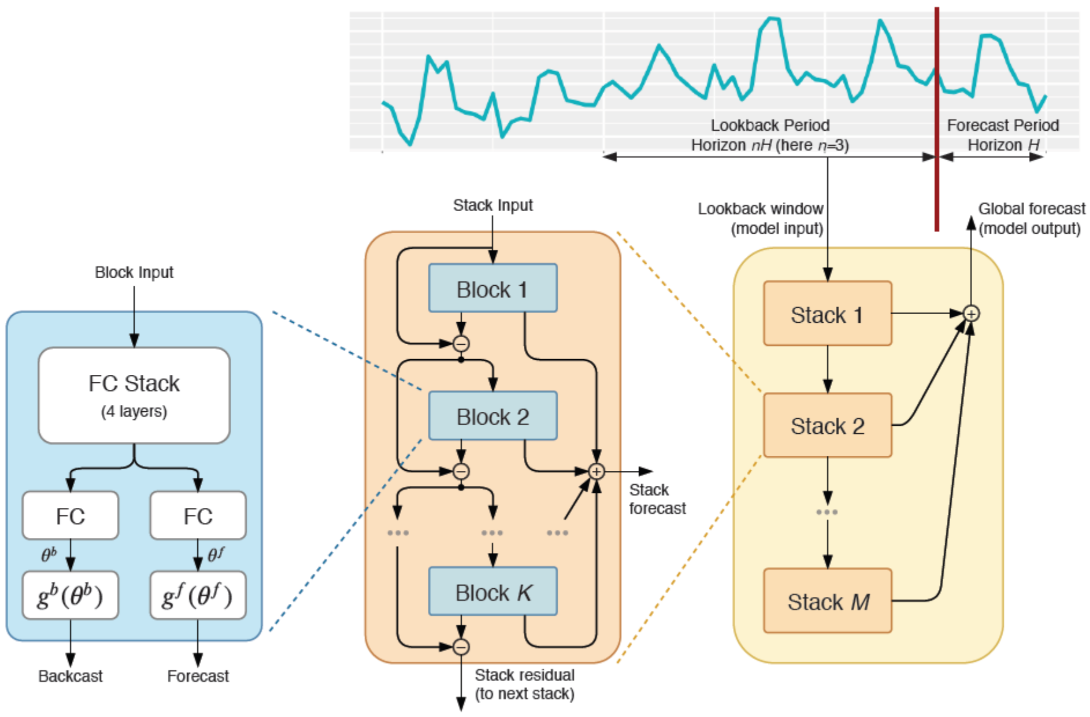


Figure 3: N-BEATS architecture from [53].

Considering the $l-th$ block within the $s-th$ stack of the N-BEATS architecture, the block receives a single vector as input $\boldsymbol{x}_l$ and produces two outputs: the backcast $\hat{\boldsymbol{x}}_l$ and the forecast $\hat{\boldsymbol{y}}_l$. The input $\boldsymbol{x}_l$ is computed as the residual of the previous block's input and its backcast, given by $\boldsymbol{x}_l = \boldsymbol{x}_{l-1} - \hat{\boldsymbol{x}}_{l-1}$. The output $\hat{\boldsymbol{y}}_l$ represents the block's forecast contribution to the final prediction, while $\hat{\boldsymbol{x}}_l$ is the block's best approximation, or backcast, of the input signal $\boldsymbol{x}_l$. The internal operation of each block is governed by a series of fully connected layers, as defined in equation (29). First, the input $\boldsymbol{x}_l$ passes through a sequence of n ReLU-activated layers to generate a latent representation $\boldsymbol{h}_{l,n}$. From this latent vector, two separate parameter vectors, θ_l^b and θ_l^f, are produced via linear projections. These parameters are then used to compute the backcast and forecast by applying them to predefined basis functions.

$$\boldsymbol{h}_{l,1} = ReLU(\boldsymbol{W}_{l,1}\boldsymbol{x}_l + \boldsymbol{b}_{l,1}) \tag{29a}$$
$$\boldsymbol{h}_{l,2} = ReLU(\boldsymbol{W}_{l,2}\boldsymbol{h}_{l,1} + \boldsymbol{b}_{l,2}) \tag{29b}$$
$$\boldsymbol{h}_{l,n} = ReLU(\boldsymbol{W}_{l,n}\boldsymbol{h}_{l,n-1} + \boldsymbol{b}_{l,n}) \tag{29c}$$
$$\boldsymbol{\theta}_l^b = LINEAR(\boldsymbol{W}_{l,b}\boldsymbol{h}_{l,n}) \tag{29d}$$
$$\boldsymbol{\theta}_l^f = LINEAR(\boldsymbol{W}_{l,f}\boldsymbol{h}_{l,n}) \tag{29e}$$

In the generic version of the N-BEATS architecture, the basis functions $\hat{\boldsymbol{y}}_l = g_l^f(\boldsymbol{\theta}_l^f)$ and $\hat{\boldsymbol{x}}_l = g_l^b(\boldsymbol{\theta}_l^b)$ are implemented as fully connected (dense) layers with bias terms and a linear activation function. This configuration allows the model to learn arbitrary mappings from the latent parameters to the forecast and backcast outputs, offering maximum flexibility but limited interpretability. In contrast, the interpretable variant of N-BEATS is structured into two stacks: a trend stack followed by a seasonality stack. In the trend stack, the functions g_l^f and g_l^b are constrained to be a polynomial of small degree, in order to force $\hat{\boldsymbol{y}}_l^{tr}$ to capture the trend. In the seasonality stack g_l^f and g_l^b are constrained to be a periodic function to force $\hat{\boldsymbol{y}}_l^{seas}$ to capture seasonal patterns.

The N-BEATS algorithm has been applied in recent literature, demonstrating state-of-the-art performance in electric load forecasting [54], [55], [10], [56].

In our comparative study, the N-BEATS model is implemented in *Python* using an open-source implementation[10] [57], which we slightly modified to have more freedom on the number of layers per stack.

4.7 Temporal Fusion Transformer

Temporal Fusion Transformer (TFT), introduced by Bryan Lim et al. in 2021 [58], is an attention-based deep learning architecture designed for interpretable multi-horizon time series forecasting. TFT is capable of handling heterogeneous input types, including static covariates, known future inputs, and observed historical variables. The overall architecture of TFT is illustrated in Figure 4, and comprises five core components:

- Variable Selection Mechanism: variable selection networks allow the model to select relevant inputs at each time step and remove unnecessary inputs that can negatively impact performance.
- Gating Mechanism: gated residual networks allow the model to skip unnecessary parts of the network, improving performance by applying non-linear processing only when needed.
- Static Covariate Encoder: static features are used to condition and modulate the temporal dynamics throughout the model.
- Sequence Encoding: a sequence-to-sequence layer based on LSTM encoders and decoders allows the model to learn local dependencies from past observed inputs and known future inputs.
- Temporal Self-Attention: a multi-head attention block allows the model to learn long-term relationships. This multi-head attention block was modified to be interpretable.

[10] `https://github.com/philipperemy/n-beats`

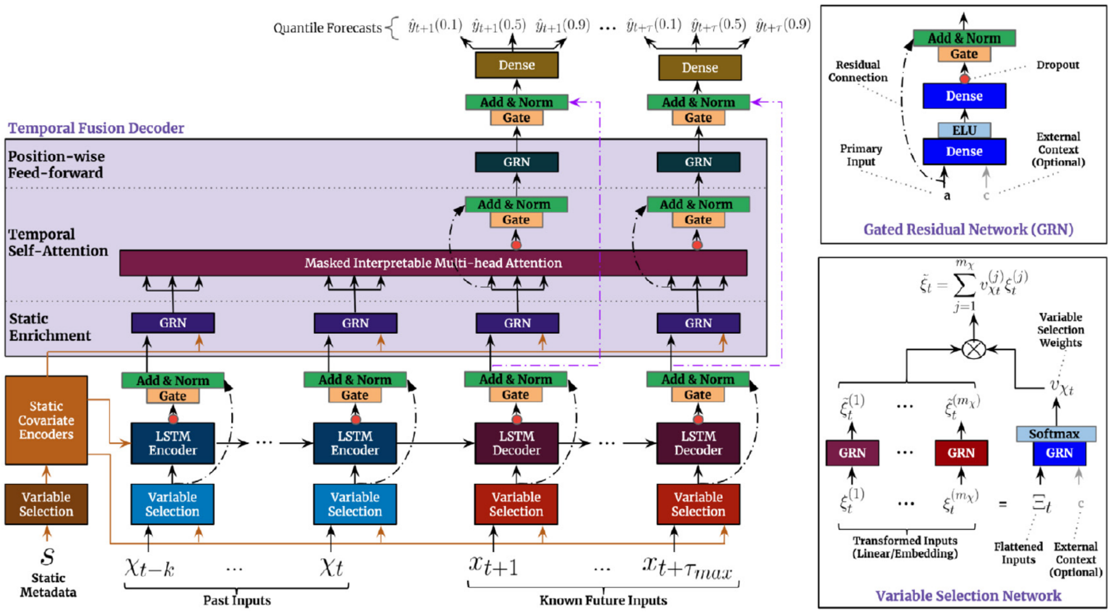


Figure 4: TFT architecture from [58].

The TFT prediction of the target value at time $t+\tau$, denoted $\hat{y}_{t+\tau}$, can be represented by equation (30).

$$\hat{y}_{t+\tau} = f(\tau, y_{t-k:t}, z_{t-k:t}, x_{t-k:t+\tau}, s) \tag{30}$$

In this equation (26), τ is the forecast horizon, with $\tau \in \{0, \ldots, \tau_{\max}\}$, with $\tau_{\max}$ being the maximum forecasting horizon, y_t denotes past values of the target variable, and z_t represents observed inputs that are only available up to the current time step. For example, z_t could correspond to the measured temperature or the actual load in another region in the context of electric load forecasting. The parameter k refers to the size of the finite look-back window. x_t includes known future inputs, such as the day of the week or holiday indicators. The variable s represents static covariates, which may include consumer type, geographical location, or other time-invariant characteristics. The Temporal Fusion Transformer can generate both point forecasts and prediction intervals. Before training, the user can configure the model to predict multiple quantiles (e.g., 10th, 50th, and 90th) at each time step, as illustrated in Figure 4. The quantile forecasts are obtained through a linear transformation of the temporal fusion decoder's output. In this study, the model is configured for point forecasting. A complete description of the architecture is available in the original paper [58].

The TFT model has been applied in very recent literature, demonstrating state-of-the-art performance in electric load forecasting [59], [60], [61], [62], [63].

In our comparative study, the TFT model is implemented in *Python* using *Pytorch Lightning* library [11].

[11]https://pytorch-forecasting.readthedocs.io/en/v1.0.0/api/pytorch_forecasting.models.temporal_fusion_transformer.TemporalFusionTransformer.html

5 Methodology for models comparison

As explained in the objectives, the aim of this study is to compare state-of-the-art forecasting methods in the specific context of day-ahead electric load forecasting for residential loads at a low level of aggregation. To ensure a rigorous and fair comparison between algorithms, the evaluation methodology was designed based on the following principles:

- The algorithms are tested on data from open-access datasets to facilitate reproducibility and comparison with both existing and future research.
- All algorithms are evaluated on the same portion of the same datasets, as algorithm performance can vary not only across different datasets but also across different time periods within the same dataset.
- A cross-validation approach is employed. The goal of cross-validation is to assess the generalization performance of each algorithm on unseen data and to mitigate selection bias. Selection bias occurs when the performance of the model on the test set does not accurately reflect its performance on truly independent data. Multiple training and testing runs are particularly important for machine learning models, where random weight initialization can influence the final performance. The details of our cross-validation methodology are presented in Section 5.3.
- The algorithms are evaluated on two datasets, as their performance may vary between datasets with different characteristics.
- All algorithms are tuned using the same methodology, with minimal manual intervention to reduce user bias. The performance of an algorithm is highly dependent on its hyperparameters. Therefore, it is essential to avoid unfair comparisons between a model that has been thoroughly fine-tuned by an expert and one that has been only minimally adjusted. Unfortunately, this aspect is often overlooked or insufficiently reported in the literature. Our hyperparameter tuning strategy is detailed in Section 5.2.
- The algorithms are compared using multiple performance metrics commonly used in the literature, to provide a comprehensive evaluation across different aspects of forecasting accuracy.

The evaluation process consists of three main steps: first, dataset selection and preprocessing; second, hyperparameter tuning using a standardized methodology; and third, performance comparison using cross-validation.

5.1 Dataset pre-processing and feature selection

To create the data necessary for training and testing the algorithms, two open-access datasets were selected. These datasets contain real-world measurements from smart meters or digital meters at half-hourly intervals.

5.1.1 Low Carbon London project dataset

The first dataset is the Low Carbon London project dataset, which includes measurements from a representative sample of 5,567 households in London[12]. These readings were collected during a project led by UK Power Networks between November 2011 and February 2014. From this pool of households, 100 were selected to create a new aggregated dataset. The selection was based on the following criteria:

[12]The Dataset is available here: https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households

- The readings must cover at least two years, specifically between January 1, 2012, and January 1, 2014[13].
- The readings must be from customers on a standard tariff, excluding those on dynamic time-of-use tariffs.
- The readings must not contain data gaps longer than four periods (i.e., two hours).

Among the few hundred households that met these criteria, 100 were randomly selected to create the aggregated dataset. Prior to aggregation, any missing values corresponding to data gaps shorter than two hours were imputed using linear interpolation. The total aggregated consumption was then obtained by summing the individual household consumption time series. The datasets with 50, 25, and 10 households are created similarly by randomly selecting households from the 100.

Four weeks at the start of the aggregated dataset from 100 households are presented in Figure 5. On most days, a typical residential load profile can be observed, characterized by a prominent evening peak (corresponding to the time when residents return home and begin cooking), a smaller morning peak, and lower consumption during the night and midday periods[14]. The data displayed spans from Monday, January 6, to Sunday, February 2, 2012.

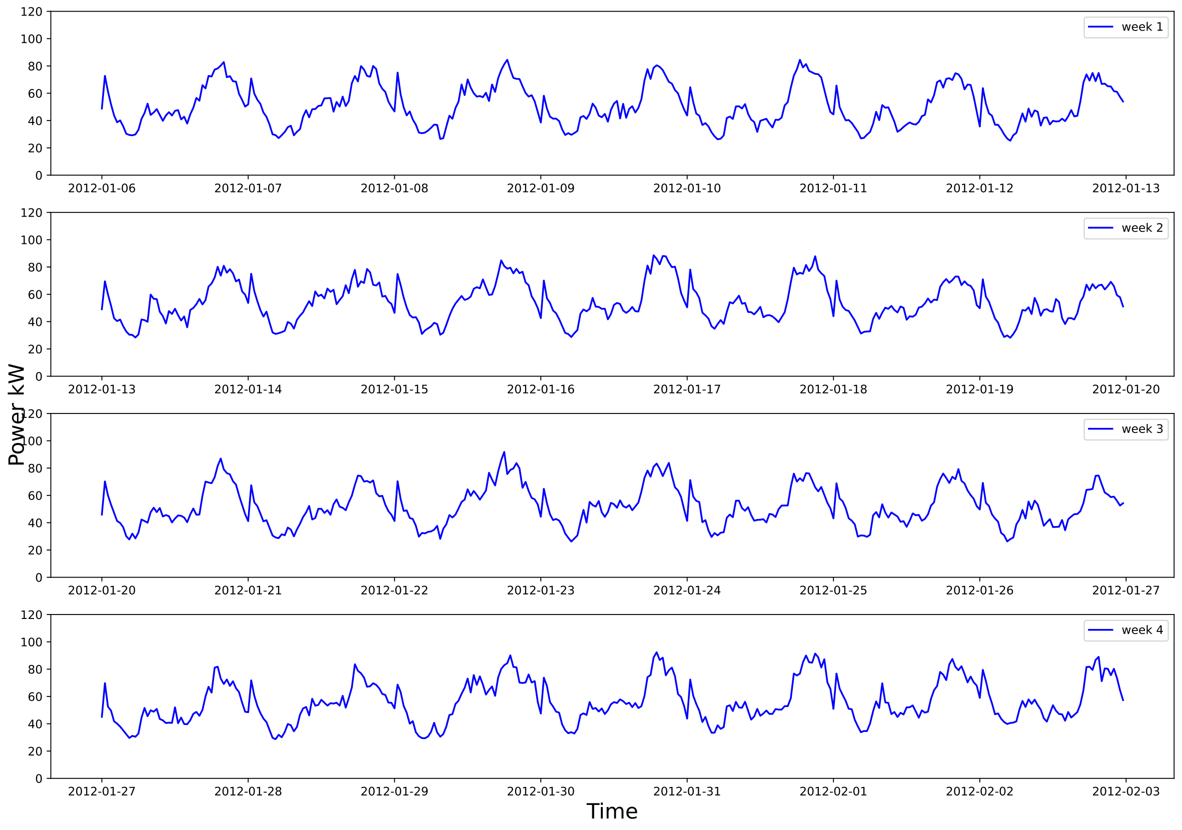


Figure 5: Power of 100 aggregated households of the London low carbon project.

[13]This period was selected because it represents the longest interval during which slightly more than 100 readings satisfying the other criteria overlap in time.

[14]A distinct consumption peak around midnight can also be observed in this dataset. This peak is likely attributable to the use of electric storage heaters during off-peak night tariffs, which, in London at the time of data collection, typically applied on weekdays from around midnight to 7 a.m., depending on the local network operator [64], [65], [66].

Various predictors or features can be used as inputs to forecasting algorithms for predicting future electric load. The most commonly used features in the literature include historical consumption data (i.e., lagged load values), calendar-related variables (such as day of the week, hour of the day, weekday or weekend, and public holidays), and meteorological variables (such as temperature, humidity, wind speed, and solar irradiation) [67], [68], [14], [13]. Calendar information is typically easy to derive for any dataset, provided the region of measurement is known. In contrast, meteorological data can be more difficult to obtain. For the Low Carbon London project, relevant meteorological data are available through an open-access *Kaggle* repository[15]. These data include maximum and minimum temperatures, dew point, wind speed, apparent temperature, cloud cover, and other variables. For our implementation, only the apparent temperature was selected as a meteorological input. Temperature is the most widely used meteorological predictor in electric load forecasting [68], [13]. In the case of the Low Carbon London dataset, apparent temperature also exhibited the highest Pearson correlation coefficient with the aggregated electric load. This may be because apparent temperature encapsulates both temperature and humidity information, making it a more comprehensive indicator of thermal comfort and energy usage.

For a realistic evaluation of algorithm performance in practical applications, forecast rather than observed temperature values should ideally serve as model inputs. However, obtaining historical weather forecasts aligned with the required prediction horizon is extremely challenging. Today, day-ahead temperature forecasts at ground level are nevertheless highly accurate, with reported root mean square errors typically between 1 and 3 K [69]. Furthermore, Haben et al. [70] showed that, for a region and period comparable to the London dataset, the effect of temperature on forecasting accuracy at low aggregation levels is relatively limited.[16] Taken together, these results suggest that the difference in forecasting performance between using actual and predicted temperature values is expected to be small and unlikely to affect the conclusions of this study.

The previous values of the load are referred to as endogenous inputs. Depending on the algorithm, these endogenous inputs can take different forms. For algorithms that accept fixed-length time series as input, such as N-BEATS or TFT, the endogenous inputs consist of the load values from the previous seven days. In contrast, models such as XGBoost rely only on specific lagged values, typically the load 24 hours and 7 days earlier. Algorithms like SARIMA and Holt-Winters use selected values from the previous 7 days. Prophet also accepts time series as input, but unlike the others, it does not require a fixed input length, and the time intervals between data points can vary. Exogenous features (features that are not previous load values) selected for each algorithm include: hour of the day, day of the week, day of the year, and apparent temperature. The N-BEATS model is the only to not accept exogenous features.

5.1.2 Smart-Grid Smart-City Customer Trial Data (SGSC)

The second dataset is the Smart-Grid Smart-City Customer Trial Data (SGSC), which includes measurements from more than 70 thousands households in Australia [17]. These readings were collected during the Smart Grid Smart City project between 2010 and 2014. From this pool of households, 100 were selected to create a new aggregated dataset. The selection was based on the following criteria:

- The readings must span at least 17 months, specifically from May 15, 2012, to December 5, 2013.[18]

[15]Meteorological data for the Low Carbon London project dataset: https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london

[16]Haben et al. examined the impact of temperature on low-level load forecasting using measurements collected between 2014 and 2015 in Bracknell, a town located less than 50 km west of London. This finding may not generalize to other regions, particularly in countries where residential heating and cooling are predominantly electricity-based.

[17]The Dataset is available here: https://www.data.gov.au/data/dataset/smart-grid-smart-city-customer-trial-data

[18]This period was selected because it represents the longest interval during which more than 100 readings satisfying

- Readings must be from customers in the control group, excluding those with non-standard tariffs such as dynamic peak pricing, seasonal time-of-use pricing, or those using feedback technologies.
- The readings must not contain data gaps longer than four periods (i.e., two hours).
- All customers must be located within the same climate zone (Warm temperate).
- Customers must not have photovoltaic panels or controllable appliances.

Among the few hundred households that met these criteria, 100 were randomly selected to create the aggregated dataset, similarly to the London dataset. Four weeks at the start of the aggregated dataset from 100 households are presented in Figure 6. The data displayed spans from Monday, May 21, to Sunday, June 17, 2012. For this dataset, a typical residential load profile can also be identified, characterized by a high evening peak, a smaller morning peak, and reduced consumption during the night and midday periods. This dataset does not contain meteorological data. The rest of the features are the same as for the London dataset.

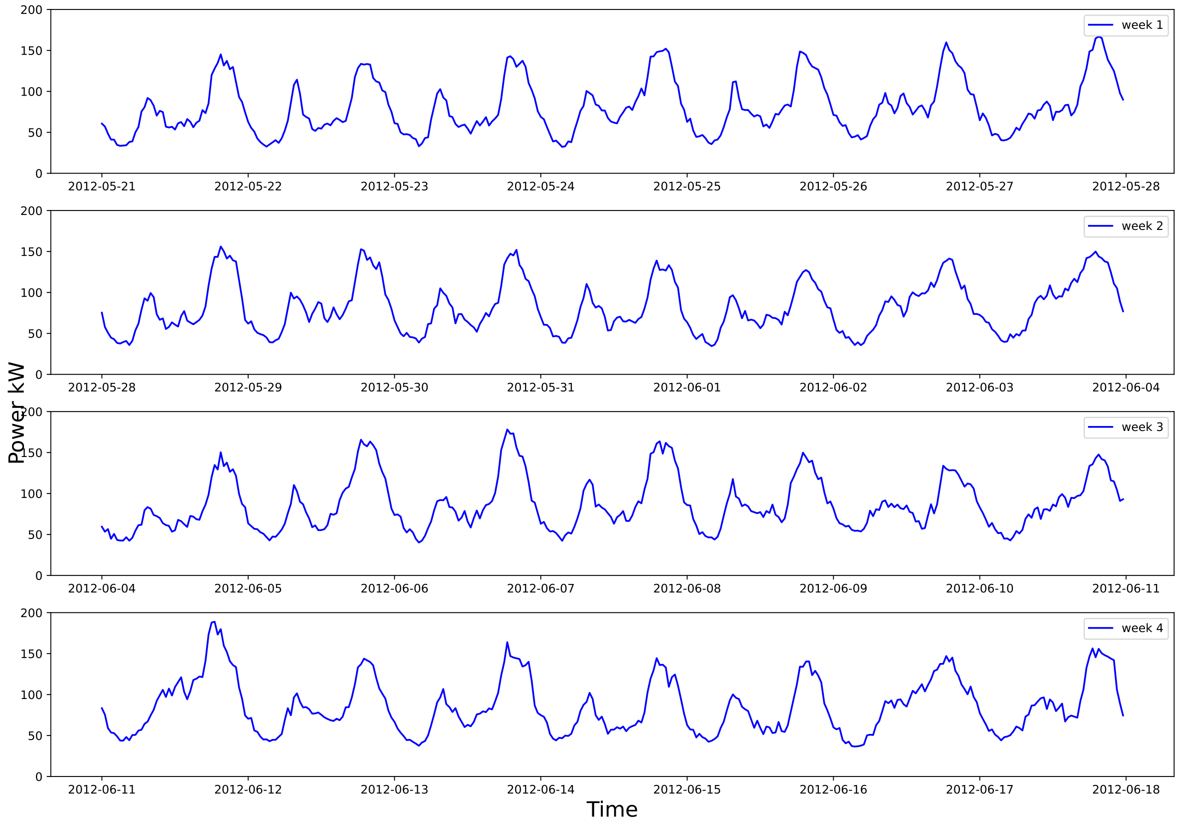


Figure 6: Power of 100 aggregated households of the SGSC project (Australia).

5.1.3 Dataset relevance

The two datasets used in this study, the London Low Carbon project dataset (2011–2014) and the Smart Grid Smart City dataset (2010–2014), are both relatively old. It is therefore important to assess their relevance in the context of our comparative study. As stated in the introduction, the aim of this

the other criteria overlap in time.

work is to evaluate multiple forecasting approaches and algorithms in order to identify those most effective for supporting residential load scheduling.

Residential load scheduling refers to the optimization of household appliance operation with objectives such as minimizing user energy costs, maximizing self-consumption, or enhancing grid efficiency. This is typically implemented through a Home Energy Management System (HEMS)[71]. For example, a HEMS may be tasked with scheduling controllable appliances such as an electric vehicle, a battery, and a heat pump in a household equipped with rooftop photovoltaic (PV) panels. The objective may be to reduce electricity costs while simultaneously alleviating stress on the distribution grid. The optimization is commonly performed on a rolling horizon basis: every hour, the schedule for the next 24 hours is updated using the most recent information. Inputs to the optimization include user preferences and constraints, the future price of electricity, forecasts of PV generation, and forecasts of the non-controllable (or "basic") household load. This non-controllable load refers to the total consumption of appliances outside HEMS control, such as lighting, kitchen appliances, and entertainment devices. Accurate forecasts of this non-controllable load are therefore essential to enable optimal scheduling that limits stress events in the distribution grid, such as peak demand or PV system disconnection caused by over-injection.

In this context, the selected datasets remain relevant despite their age. Unlike many more recent datasets, they do not include controllable loads such as electric vehicles, heat pumps, or batteries. Moreover, the households selected for this study do not feature rooftop PV systems, which could otherwise modify the load profile of non-controllable appliances. These characteristics make the datasets well-suited for evaluating forecasting methods at the residential level.

5.2 Hyperparameter tuning

Hyperparameter tuning is the process of selecting the optimal values for the hyperparameters of a machine learning algorithm. When this process is approached systematically using optimization techniques to minimize a predefined cost function, it is often referred to as hyperparameter optimization (HPO). A well-chosen set of hyperparameters can significantly influence the model's performance. For instance, for a neural network, typical hyperparameters include the number of layers and neurons, the learning rate, the regularization strength, and others. Hyperparameters define either the structure of the model or the process used to train it. They are distinct from model parameters such as weights and biases, which are learned during training. For statistical models like SARIMA, hyperparameters include, for example, the orders P and Q of the autoregressive and moving average components. In Holt-Winters models, hyperparameters include, for example, the presence and type (additive or multiplicative) of trend and seasonality. Hyperparameters often interact in complex ways, meaning they must be optimized jointly to achieve the best performance. Several strategies exist for hyperparameter tuning, including manual tuning, grid search, random search, and Bayesian optimization. Manual tuning is the most straightforward yet time-consuming approach. It involves adjusting hyperparameters based on experience, intuition, or trial-and-error, and observing the resulting model performance. While this method requires minimal computational resources, it is often impractical for large models or when the number of hyperparameters is high. However, it can be effective in early prototyping or when domain expertise guides the tuning process. Grid Search involves defining a discrete set of values for each hyperparameter and evaluating the model for every possible combination. While exhaustive, this method is computationally expensive, particularly when the search space is large or the training time is high. Additionally, it may miss optimal regions if the grid is too coarse. Random search, introduced by Bergstra and Bengio in 2012 [72], involves sampling random combinations of hyperparameters from defined distributions. After a fixed number of trials, the best-performing configuration is selected. The authors demonstrated that random search can outperform grid search in terms of finding better models with fewer evaluations. However, it may also miss optimal regions. Bayesian optimization gained significant attention for hyperparameter tuning in machine learning following the work of Snoek et al.

[73], but the mathematical foundation of Bayesian optimization dates back several decades. Bayesian optimization is an HPO techniques that can efficiently identifies high-performing hyperparameters and often outperforms traditional methods such as grid search and random search [74], [75], especially when the number of evaluations is limited. Therefore, we adopt Bayesian optimization to automatically tune the hyperparameters of all models in this study, with the exception of SARIMA and Holt-Winters. Given their relatively small and interpretable parameter spaces, we instead employ an exhaustive grid search to evaluate all possible parameter combinations and select the optimal configuration.

The goal of hyperparameter optimization is to identify the configuration $\boldsymbol{x}^*$ that yields the best performance, such as the lowest mean squared error in a regression task or the highest accuracy in a classification task. Hyperparameter optimization is thus an optimization problem where the goal is to minimize (or maximize) an objective function f as in equation (31), where the search space A is defined by the possible values of all the tunable hyperparameters. Typically for machine learning, the function f is a black-box function that has no simple closed form, is expensive to evaluate, and can be evaluated at any point in the search space.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in A}{\operatorname{argmin}} f(\boldsymbol{x}) \tag{31}$$

The general principle of Bayesian optimization (BO) is as follows: a prior distribution is first placed over the unknown objective function f. After each new observation, this prior is updated using Bayes' theorem to form a posterior distribution, which serves as the probabilistic surrogate model. Each iteration begins by selecting the next sample point x_{n+1}, chosen by maximizing an acquisition function. The objective function is then evaluated at this point, and the surrogate model is updated with the new data. The acquisition function guides the search by balancing exploration and exploitation: exploration favors sampling in regions where the surrogate model has high uncertainty, while exploitation focuses on areas near previously observed high-performing samples. This trade-off enables the efficient optimization of expensive black-box functions. The pseudocode for Bayesian optimization is provided in algorithm (1), inspired by [76] and [77].

Algorithm 1 Simple pseudo-code for Bayesian Optimization

```
1: Inputs: objective $f$, search space $A$, acquisition function $\alpha$, budget $n_f$
2: Place a prior over $f$ (e.g., GP prior)
3: Select $n_0$ points $\boldsymbol{x}_{i}{}_{i=1}^{n_0} \subset A$ and evaluate $\boldsymbol{y}_i = f(\boldsymbol{x}_i)$
4: Set $D_n = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n_0}, \quad n \leftarrow n_0$
5: Model update: compute the posterior (surrogate) given $D_n$
6: while $n < n_f$ do
7:     Acquisition step:
           $\boldsymbol{x}_{n+1} \in \arg\max_{\boldsymbol{x} \in A} \alpha(\boldsymbol{x}; D_n)$
8:     Evaluation: obtain $\boldsymbol{y}_{n+1} = f(\boldsymbol{x}_{n+1})$
9:     Augment data: $D_{n+1} \leftarrow D_n \cup \{(\boldsymbol{x}_{n+1}, \boldsymbol{y}_{n+1})\}$
10:    Model update: compute the posterior (surrogate) given $D_{n+1}$
11:    $n \leftarrow n + 1$
12: end while
13: Return: $\boldsymbol{x}^\star = \arg\min_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in D_n} \boldsymbol{y}_i$ (best observed point)
```

In this work, we use the Tree-structured Parzen Estimator (TPE), a Bayesian optimization method, to optimize hyperparameters [78]. TPE has gained widespread adoption and has recently demonstrated outstanding performance in various applications [79], [80], [81], [82]. In particular, to implement our methodology, we use the optimization software *Optuna* in *python* [83], implementing TPE.

Annex A presents the ranges of the hyperparameters tuned during optimization for each forecasting method, as well as some important fixed hyperparameters.

5.3 Models comparison with cross-validation

Cross-validation is a fundamental technique for evaluating the generalization performance of machine learning models in a statistically robust manner. It is particularly critical when proposing new models or comparing multiple alternatives, as it mitigates the risk of overfitting to a particular train-test split. By averaging performance over multiple folds, cross-validation reduces the variance associated with any single partition of the data and provides a more reliable estimate of how the model is expected to perform on unseen, real-world data. Cross-validation is also widely used for hyperparameter tuning, as it allows the selection of hyperparameters that yield consistently good performance across different data subsets, thereby enhancing model robustness.

In k-fold cross-validation, the dataset is partitioned into k equal-sized subsets, or "folds." The model is iteratively trained on $k-1$ folds and tested on the remaining fold. This process is repeated k times, with each fold serving as the testing set once. The final performance metric is computed as the average across all k iterations (or splits). This process is illustrated in Figure 7.

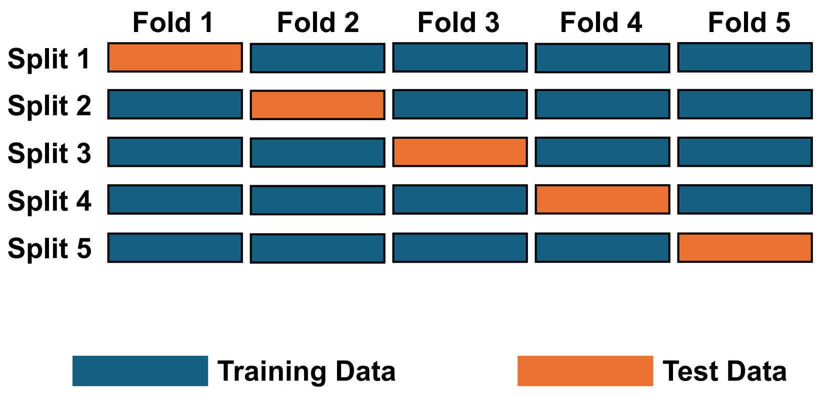
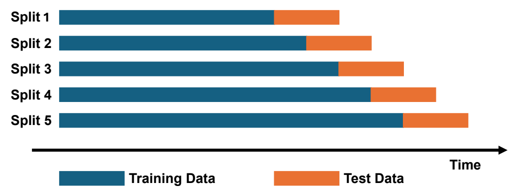
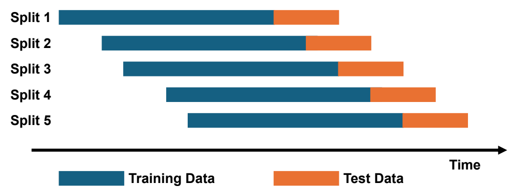


Figure 7: 5-folds cross-validation.

However, standard k-fold cross-validation, which relies on randomly splitting the data, is not suitable for time series forecasting tasks. This is due to the temporal dependency structure inherent in time series data, where future observations cannot be used to predict past events. Instead, time series forecasting requires a variant known as rolling origin evaluation [84], [85]. In time series analysis, the forecasting origin refers to the point from which forecasts are generated. In rolling origin evaluation, a moving-window approach is employed: the training set (or in-sample set) has a fixed size and shifts forward in time with each iteration. The test set (or hold-out set) consists of the subsequent time steps following each training window. If the training set does not maintain a fixed size but instead grows over time, the method is referred to as expanding origin evaluation. In this approach, the training set starts with an initial window and is incrementally extended forward in time at each iteration. As with rolling origin, the test set always comprises the time steps immediately following the training set. For clarity, we refer to these methods as rolling or expanding origin cross-validation with k iterations or splits. Rolling and expanding origin strategies are illustrated in Figure 8.



(a) Expanding origin



(b) Rolling origin

Figure 8: Expanding vs rolling origin evaluation.

In our experiments, we employ either a rolling origin or an expanding origin cross-validation strategy, depending on the characteristics of the forecasting algorithm. For models such as XGBoost, Prophet, N-BEATS, and TFT, we adopt an expanding origin approach. This choice is motivated by their ability to maintain or improve performance as the size of the training set increases. In contrast, for models like Holt-Winters and SARIMAX, we use a rolling origin approach. These models have a limited number of parameters and are more sensitive to the inclusion of outdated or irrelevant data. As the training set grows too large, their performance may degrade due to the inability to adapt to recent trends or seasonal variations. Consequently, for Holt-Winters and SARIMAX, we perform a search to identify an optimal training window size, which is then fixed for subsequent evaluations.

For several of our models, XGBoost, N-BEATS, and TFT, in addition to the conventional split between training and testing data, a separate validation set is required. In machine learning, the validation set plays a crucial role in monitoring model performance during training, enabling early stopping to prevent overfitting. As with the test set, in time-series forecasting, the validation set must be temporally located after the training set to preserve the causality and temporal order of the data. The division of the dataset into training, validation, and testing subsets is illustrated in Figure 9. More generally, the validation set can be considered a subset of the training data, used exclusively for model tuning and performance monitoring during training.

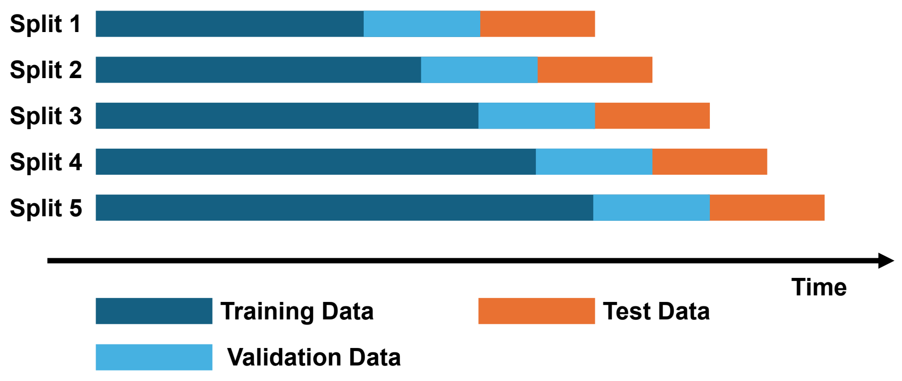


Figure 9: Training, validation, and test sets for 5 iterations expanding origin cross-validation.

In our methodology, we perform cross-validation in two distinct stages: first, for hyperparameter tuning, and second, for final model evaluation and comparison. To ensure the integrity of the evaluation and to prevent data leakage, the dataset is partitioned carefully, as illustrated in Figure 10. Initially, the full dataset is divided into two subsets: a hyperparameter tuning set and an evaluation set. The hyperparameter tuning set is then used in the first cross-validation loop, dedicated to optimizing model parameters. In the second stage, designed to assess the generalization ability of the models, a separate cross-validation is conducted using all available data. A crucial constraint is that the test sets in this phase must come from the evaluation set and exclude any data points from the hyperparameter tuning set, thereby ensuring an unbiased assessment of model performance.

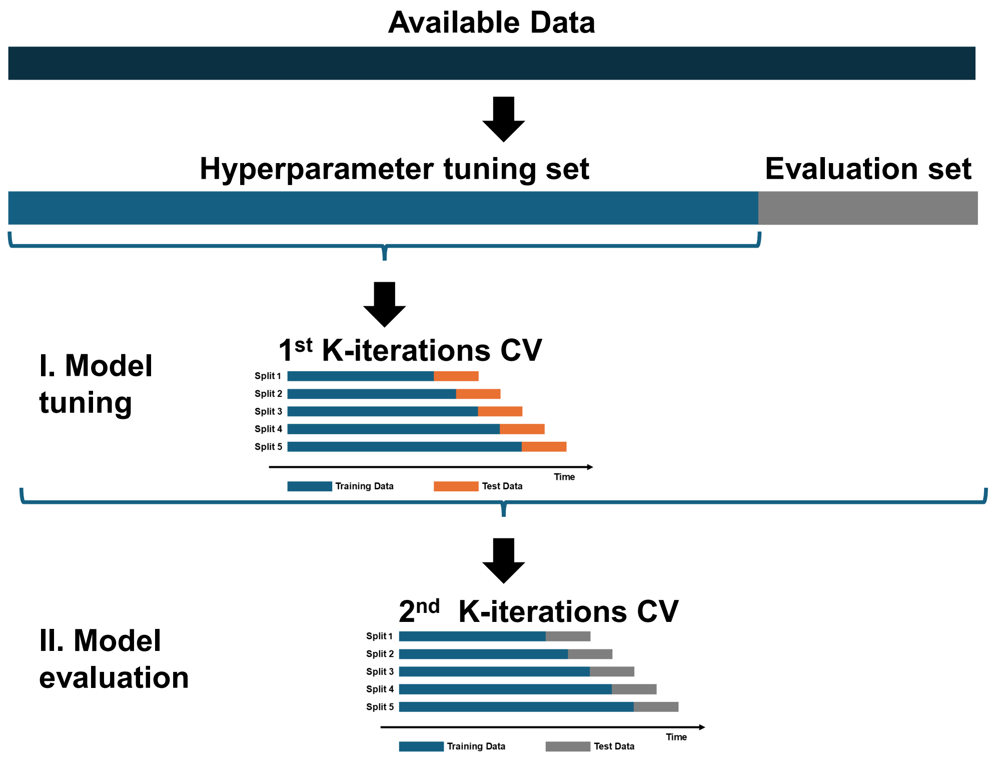


Figure 10: Dataset partitioning.

In our experiments, for every model except SARIMA and HW, we employ an expanding origin cross-validation strategy with 5 iterations at each stage, both for hyperparameter optimization and final model evaluation. For every iteration, the test set spans a duration of four weeks, with a one-week overlap between consecutive iterations. This overlap serves to reduce the total size of the test sets while introducing cyclical diversity in the forecasting origins [84]. As a result, the models are evaluated over a total of 16 distinct weeks, corresponding to 112 individual days, thereby providing a comprehensive assessment of their performance across a substantial portion of the year. SARIMAX is evaluated using rolling-origin cross-validation with eight iterations, each employing a two-week test set without overlap between iterations. HW is evaluated using rolling-origin cross-validation with sixteen iterations, each employing a one-week test set without overlap between iterations. This also results in 16 distinct evaluation weeks for SARIMA and HW. In our setup, the models have a forecast horizon of one day, equivalent to 48 time steps. Consequently, within each iteration of the cross-validation loop, an inner rolling origin evaluation is employed. For each iteration, the model is trained (or fitted) once and then sequentially updated with new data after each multi-horizon forecast. This is necessary because most of our models require lagged values of the endogenous feature as input. For models that cannot be updated without being refitted, which is the case with Prophet, the forecast is done for the entire test set at once for each iteration. This nested approach, combining rolling or expanding origin cross-validation with an inner rolling origin loop, is described in [86]. For the ML and DL models, the entire CV procedure is repeated ten times with different random seeds to capture stochastic variation.

5.4 Metrics

As explained in Section 5.3, the forecast horizon is one day, corresponding to 48 half-hour steps. The models are evaluated using cross-validation with five or more iterations, employing either an expanding-window or rolling-origin scheme. Within each iteration, a model is trained once and then updated after every multi-horizon forecast. Performance is reported using the standard metrics defined in equations (32), (33), and (34), averaged over all the evaluations. For the machine learning and deep learning algorithms, results are further averaged over ten cross-validation runs, each with a different random seed. The use of these widely adopted metrics facilitates comparison with existing and future studies. Among them, MAPE is the most popular in the literature. It expresses forecast errors as percentages of the actual values, which makes the results easy to interpret and communicate. Unlike RMSE or MAE, MAPE is normalized by the actual values at each time step, allowing comparisons not only across datasets with different aggregation levels (e.g., household, feeder, or system) but also across datasets of similar load scale collected in different regions or contexts. This property is particularly useful in electricity demand forecasting, where load magnitudes can differ substantially depending on geography, climate, or aggregation. Other metrics, such as normalized RMSE (obtained by dividing RMSE by the average load), symmetric MAPE (sMAPE), or the Mean Absolute Scaled Error (MASE), can also achieve scale independence, but they are generally less intuitive and less widely used in the literature.

$$MAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{|y_t - \hat{y}_t|}{y_t} \times 100 \tag{32}$$

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|y_t - \hat{y}_t| \tag{33}$$

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2} \tag{34}$$

6 Results analysis

Figure 11 shows a representative day-ahead forecast on the London dataset, generated by the two best-performing models. Tables 1 and 2 compare all algorithms on the London and SGSC datasets, respectively, at four aggregation levels (100, 50, 25, and 10 households). The magnitude of MAPE is similar to the magnitude reported in [12] with a similar but not identical set-up. This is due to the use of more advanced algorithms and the inclusion of exogenous predictors in this study. Figures 12 and 13 show the evolution of MAPE with respect to aggregation level. As expected, MAPE decreases as the number of aggregated meters increases. It can be observed that MAPE follows an asymptotic trend, decreasing rapidly for very small groups of households, then more gradually as the number of aggregated households increases. It is also notable that the MAPE values for the different methods evolve similarly, and their relative ranking remains mostly consistent across aggregation levels. MAPE is higher for the SGSC data set. The least good results on the SGSC dataset are due to a higher day-to-day variability visible with the Naive baseline. For London, XGBoost achieves the lowest error in almost all metrics, with N-BEATS a close second. For SGSC, N-BEATS leads, followed by SARIMAX and then XGBoost.

Table 3 lists the mean training duration per cross-validation iteration (as explained in section 5.3) and the mean inference duration for one day-ahead predictions. All ML and DL models ran on identical GPU hardware, whereas Holt–Winters and SARIMAX ran on a CPU node[19]. XGBoost, Prophet, and Holt–Winters are the fastest overall. Prediction is quite slow for HW and SARIMAX because each day-ahead forecast is followed by an update (but not a refit) of the algorithms with the new measured load. Taken together, the results indicate that N-BEATS and XGBoost offer the best trade-off between error and speed for low-aggregation residential load forecasting. Remarkably, the decades-old SARIMAX still rivals several state-of-the-art methods in error metrics, though not in computation time. By contrast, TFT delivers modest gains despite its higher complexity and long training time. While deeper HPO could slightly improve its performance, it would still be far less efficient than XGBoost or N-BEATS in terms of computational time.

[19]In our experiments, Holt–Winters and SARIMAX are implemented using the *statsmodels* library in Python, which does not support GPU acceleration. GPUs can provide substantial speed-ups for machine learning methods where computations can be parallelized. However, the fitting procedures for Holt–Winters and SARIMAX are inherently sequential, which greatly limits the potential benefit of GPU acceleration. The Python library, *RAPIDS cuML* [87], provides GPU-based implementations of these models that mirror those in *statsmodels*. These implementations can accelerate training when fitting many models across large collections of time series in parallel. By contrast, when training a single model on a single time series, GPU implementations are typically no faster than optimized CPU versions. Since our study focuses exclusively on the training time of a single model applied to a single time series, GPU implementations of SARIMAX and Holt–Winters are not relevant in this context.

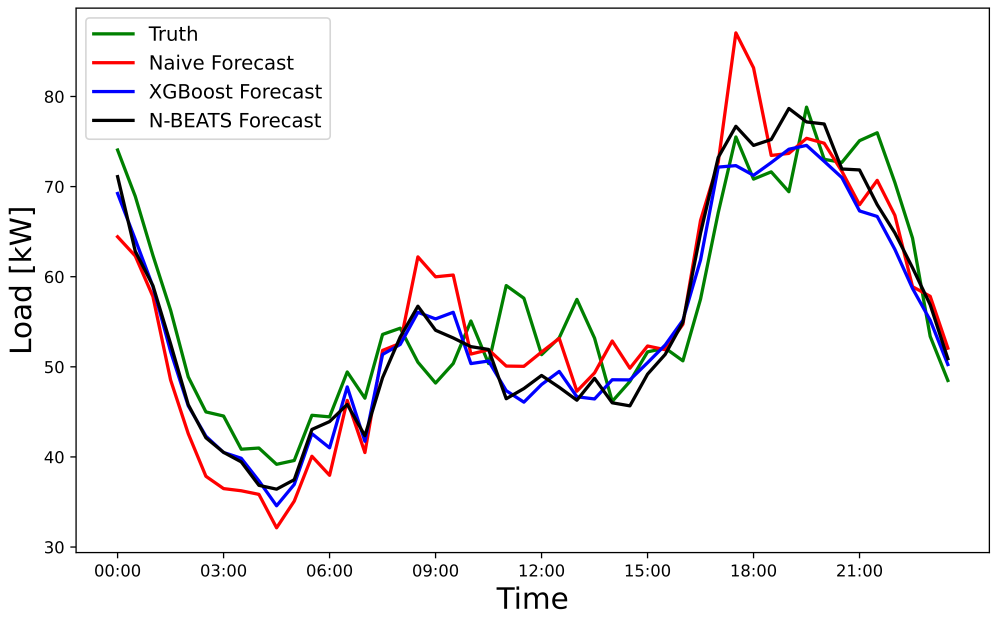


Figure 11: Day-ahead forecast for 100 aggregated households (London dataset).

Table 1: Comparison of forecasting models on the London dataset at different aggregation levels

	Metric	Naive	SARIMAX	HW	XGBoost	Prophet	N-BEATS	TFT
	MAPE [%]	11.0 ± 3.0	9.8 ± 2.6	12.9 ± 5.6	**8.0 ± 1.5**	9.0 ± 2.1	8.4 ± 2.1	9.6 ± 2.1
100H	MAE [kW]	5.1 ± 1.5	4.4 ± 1.2	6.0 ± 2.7	**3.7 ± 0.8**	4.1 ± 1.1	4.0 ± 1.1	4.5 ± 1.1
	RMSE [kW]	6.6 ± 1.9	5.5 ± 1.5	7.4 ± 3.0	**4.8 ± 0.0**	5.2 ± 1.3	5.0 ± 1.4	5.7 ± 1.3
	MAPE [%]	14.6 ± 3.8	12.8 ± 3.4	16.9± 7.1	**11.0 ± 2.3**	12.0 ± 2.7	11.4 ± 2.9	12.0 ± 2.7
50H	MAE [kW]	3.2 ± 0.7	2.6 ± 0.7	3.7 ± 1.6	**2.4 ± 0.5**	2.6 ± 0.6	2.5 ± 0.7	2.6 ± 0.7
	RMSE [kW]	4.2 ± 0.9	3.3 ± 0.9	4.7 ± 1.9	**3.1 ± 0.7**	3.3 ± 0.7	3.2 ± 0.9	3.4 ± 0.8
	MAPE [%]	19.6 ± 4.4	16.5 ± 4.0	21.3 ± 9.5	**15.1 ± 3.3**	16.2 ± 3.9	16.0 ± 3.7	16.6 ± 3.0
25H	MAE [kW]	2.3 ± 0.5	1.9 ± 0.5	2.5 ± 1.3	**1.7 ± 0.5**	1.8 ± 0.4	1.8 ± 0.5	1.9 ± 0.4
	RMSE [kW]	2.9 ± 0.7	2.3 ± 0.6	3.2 ± 1.5	**2.2 ± 0.6**	2.3 ± 0.5	2.3 ± 0.7	2.5 ± 0.5
	MAPE [%]	27.3 ± 14.3	23.6 ± 10.8	28.9 ± 20.1	**22.2 ± 7.1**	23.9 ± 11.0	23.1 ± 10.9	25.6 ± 12.6
10H	MAE [kW]	1.3 ± 0.5	**1.1 ± 0.3**	1.4 ± 0.7	1.2 ± 0.4	1.1 ± 0.4	**1.1 ± 0.3**	1.2 ± 0.4
	RMSE [kW]	1.8 ± 0.6	**1.4 ± 0.4**	1.8 ± 0.9	1.6 ± 0.5	1.5 ± 0.5	**1.4 ± 0.4**	1.6 ± 0.4

Table 2: Comparison of forecasting models on the SGSC dataset at different aggregation levels

	Metric	Naive	SARIMAX	HW	XGBoost	Prophet	N-BEATS	TFT
100H	MAPE [%]	12.7 ± 4.3	11.0 ± 3.4	13.4 ± 6.3	10.8 ± 3.2	12.4 ± 3.9	**10.5 ± 3.8**	11.0 ± 2.9
	MAE [kW]	7.4 ± 2.8	6.4 ± 2.3	7.9 ± 4.0	6.2 ± 2.4	7.1 ± 2.4	**6.0 ± 2.4**	6.3 ± 2.1
	RMSE [kW]	9.6 ± 3.8	8.1 ± 3.1	10.1 ± 4.8	8.0 ± 3.4	9.0 ± 3.1	**7.7 ± 3.1**	8.2 ± 2.9
50H	MAPE [%]	16.7 ± 3.1	**13.8 ± 2.8**	17.0 ± 7.1	15.1 ± 3.1	16.0 ± 4.3	**13.8 ± 2.7**	15.6 ± 2.6
	MAE [kW]	4.2 ± 0.9	3.4 ± 0.8	4.3 ± 1.8	3.6 ± 0.8	3.8 ± 0.9	**3.3 ± 0.7**	3.7 ± 0.7
	RMSE [kW]	5.4 ± 1.2	4.4 ± 1.1	5.5 ± 1.2	4.6 ± 1.0	4.8 ± 1.1	**4.3 ± 1.0**	4.7 ± 1.0
25H	MAPE [%]	26.5 ± 5.6	**22.3 ± 5.3**	27.9 ± 17.9	24.4 ± 7.2	25.6 ± 7.7	22.5 ± 5.7	24.2 ± 4.7
	MAE [kW]	2.8 ± 0.6	**2.3 ± 0.6**	3.0 ± 2.0	2.4 ± 0.6	2.5 ± 0.6	**2.3 ± 0.5**	2.4 ± 0.5
	RMSE [kW]	3.7 ± 0.8	3.0 ± 0.7	3.9 ± 2.5	3.1 ± 0.8	3.2 ± 0.7	**2.9 ± 0.6**	3.1 ± 0.6
10H	MAPE [%]	44.2 ± 10.4	**37.7 ± 11.0**	41.7 ± 19.3	42.0 ± 14.1	52.7 ± 21.7	40.3 ± 13.3	39.6 ± 12.4
	MAE [kW]	2.0 ± 0.4	**1.6 ± 0.4**	1.9 ± 0.7	1.7 ± 0.3	1.9 ± 0.5	**1.6 ± 0.4**	1.6 ± 0.4
	RMSE [kW]	2.6 ± 0.5	2.1 ± 0.5	2.4 ± 0.8	2.1 ± 0.4	2.4 ± 0.5	**2.0 ± 0.4**	2.1 ± 0.5

Table 3: Mean training and prediction time of each model

		SARIMAX	HW	XGBoost	Prophet	N-BEATS	TFT
London dataset	1 CV Iteration Training Time [s]	1408	**0.12**	1.14	13.2	44.8	696
	1-Day Forecast Time [ms]	150	26.7	**0.31**	11.5	8.69	62.8
SGSC dataset	1 CV Iteration Training Time [s]	686	**0.14**	0.66	2.96	31.3	558
	1-Day Forecast Time [ms]	103	30.8	**0.24**	11.0	6.85	60.8

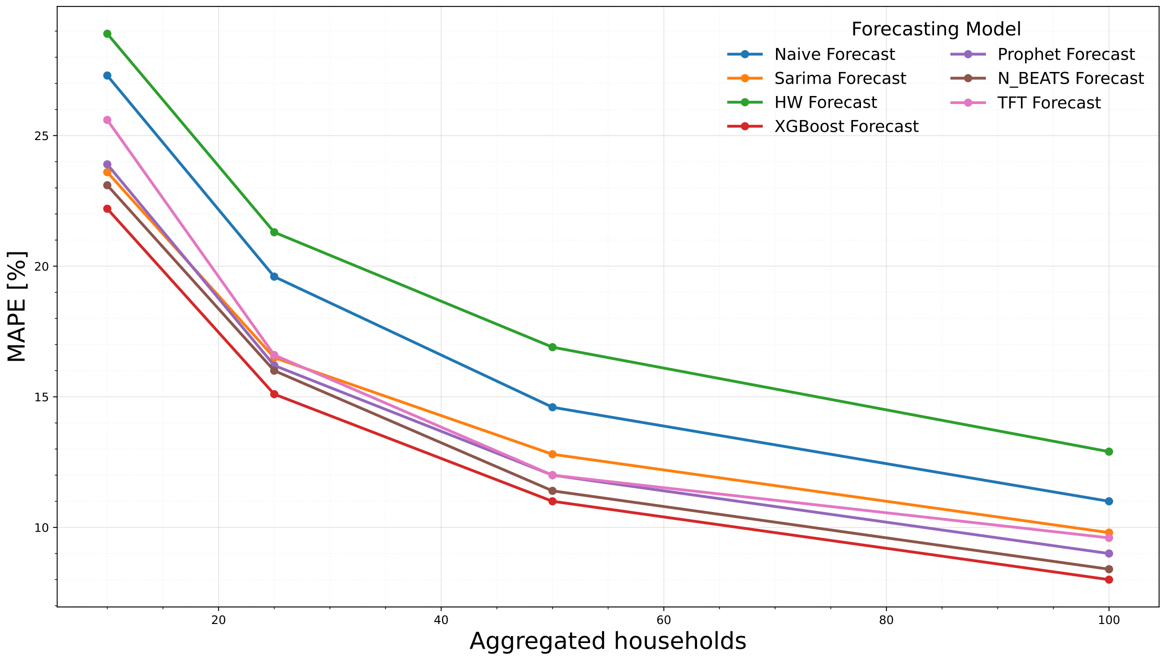


Figure 12: MAPE of forecasting models at different aggregation levels on the LONDON dataset.

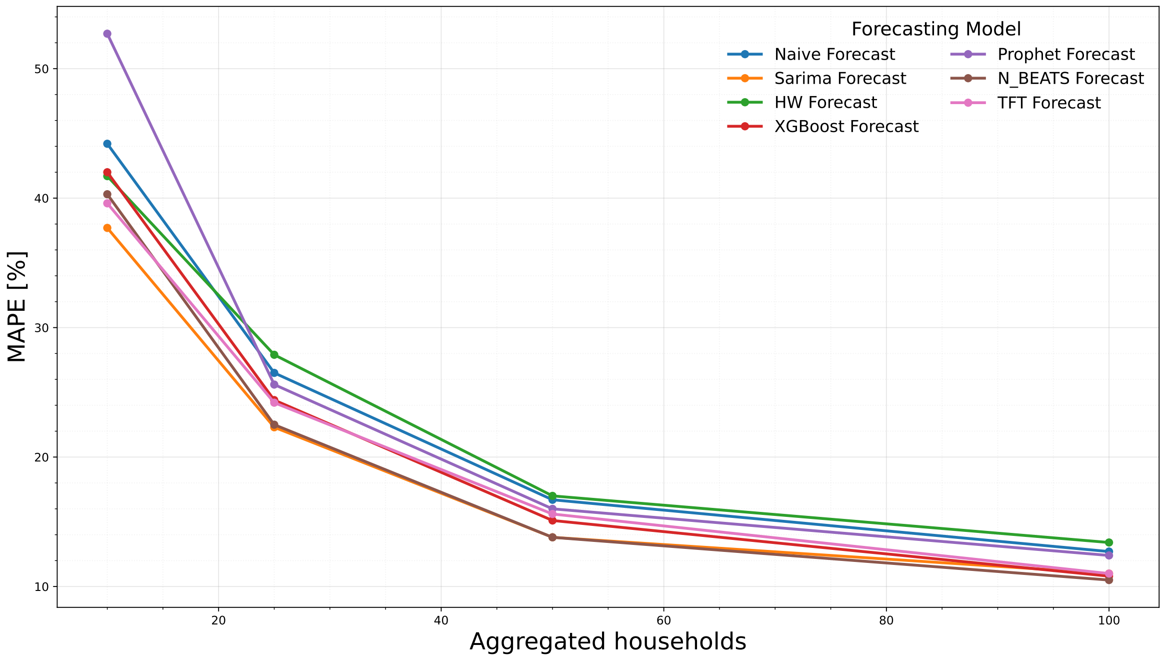


Figure 13: MAPE of forecasting models at different aggregation levels on the SGSC dataset.

7 Conclusions and perspectives

This report underscores the importance of residential load forecasting in the energy transition, reviewing its main concepts and methodologies. It focuses on day-ahead forecasting at low aggregation levels, which is increasingly vital for enhancing grid efficiency, mitigating local network issues, and supporting demand-side management at the distribution level.

Despite its practical significance, this specific forecasting problem remains underexplored in the literature. This study contributes to filling that gap by rigorously evaluating a diverse set of state-of-the-art forecasting models, including traditional statistical models, machine learning models, deep learning models, and others, on two open-access residential datasets. The results demonstrate that XGBoost and N-BEATS consistently achieve the best performance in terms of MAPE, RMSE, and MAE, while maintaining computational efficiency. These findings position them as the most promising approaches for day-ahead residential load forecasting at low aggregation levels, ranging from 10 to 100 households.

Our results also highlight how forecasting accuracy evolves with the level of aggregation: as expected, error metrics decrease rapidly with the increasing number of households. While this trend is well established in theory, empirical evidence at this level of residential aggregation remains relatively scarce. These findings therefore provide valuable insights for the design of optimal load scheduling strategies for small groups of aggregated households.

Building on the results of this study, a natural next step is the development and implementation of an optimal residential load scheduling algorithm that uses as input the forecasts generated by the best-performing models identified here.

Finally, this study focused exclusively on point forecasting. Future research should extend the analysis to probabilistic forecasting, which would enable optimization of residential load under uncertainty. A comparison between probabilistic load scheduling under uncertainty and deterministic scheduling based on point forecasts would then be valuable to assess whether the added complexity and computational burden of probabilistic approaches are justified by their potential benefits in this context.

Appendices

A Hyperparameters range

This appendix presents the ranges of the hyperparameters tuned during the optimization process, as well as important fixed hyperparameters.

A.1 Holt-Winters

Hyperparameter	**Search range or value**
Trend	$\{additive, multiplicative, none\}$
Damped trend	$\{True, False\}$
Seasonality	$\{additive, multiplicative, none\}$
Boxcox transform	$\{True, False, log\}$
Seasonal periods	$\{48\}$
Training periods (weeks)	$\{1, 2, 3, 4\}$
Test periods (weeks)	$\{1\}$

A.2 SARIMAX

Hyperparameter	**Search range or value**
p	$\{1, 2, 3\}$
q	$\{1, 2, 3\}$
P	$\{1, 2, 3\}$
Q	$\{1, 2, 3\}$
d	$\{1\}$
D	$\{0\}$
s	$\{48\}$
Training periods (weeks)	$\{4\}$
Test periods (weeks)	$\{2\}$

A.3 Prophet

Hyperparameter	**Search range or value**
Changepoint prior scale	$[0.001, 0.5]$
Changepoint range	$[0.8, 0.95]$
Seasonality prior scale	$[0.01, 10]$
Holidays prior scale	$[0.01, 10]$
Lag1 prior scale	$[0.01, 10]$
Lag7 prior scale	$[0.01, 10]$
Temperature prior scale	$[0.01, 10]$
Seasonality mode	$\{additive, multiplicative\}$
Lag1 mode	$\{additive, multiplicative\}$
Lag7 mode	$\{additive, multiplicative\}$
Temperature mode	$\{additive, multiplicative\}$
Number of BO trials	$\{400\}$

A.4 XGBoost

Hyperparameter	**Search range or value**
Number of boosting round	$[100, 5000]$
Learning rate	$[0.8, 0.95]$
Maximum depth	$[3, 12]$
Subsample	$[0.01, 10]$
colsample bytree	$[0.5, 1]$
Gamma	$[0, 5]$
Lambda	$[1e^{-4}, 100]$
Booster	$\{gbtree\}$
Tree method	$\{hist\}$
Objective	$\{reg : squarederror\}$
Number of BO trials	$\{400\}$

A.5 N-BEATS

Hyperparameter	**Search range or value**
Number of stacks	$\{1, 2, 3, 4\}$
Number of blocks per stack	$\{2, 3, 4\}$
Number of hidden layers per block	$\{2, 3, 4, 5\}$
Number of neurons per layer	$\{64, 128, 256, 512\}$
Share weights in stack	$\{True, False\}$
Batch size	$\{32, 64, 128\}$
Loss	$\{rmse\}$
Reduce learning rate on plateau factor	$\{0.5\}$
Reduce learning rate on plateau patience	$\{5\}$
Maximums epochs	$\{50\}$
Early stopping patience	$\{10\}$
Number of BO trials	$\{125\}$

A.6 TFT

Hyperparameter	**Search range or value**
Hidden size	$[8, 64]$
Attention head size	$\{2, 3, 4\}$
Gradient clipping	$[0.01, 0.6]$
Dropout	$[0.1, 0.22]$
Learning rate	$[1e^{-3}, 0.06]$
Loss	$\{rmse\}$
Reduce learning rate on plateau patience	$\{4\}$
Maximums epochs	$\{30\}$
Early stopping patience	$\{5\}$
Number of BO trials	$\{32\}$

References

[1] European Union, "Regulation (eu) 2021/1119 of the european parliament and of the council of 30 june 2021 establishing the framework for achieving climate neutrality and amending regulations (ec) no 401/2009 and (eu) 2018/1999 ('european climate law')," 2021. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32021R1119; accessed February 2025.

[2] European Commission, "Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions: European Wind Power Action Plan," 2023. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52023DC0669, Accessed: 2024-02-19.

[3] European Commission, "Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions: EU Solar Energy Strategy," 2022. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52022DC0221, Accessed: 2024-02-19.

[4] European Union, "Regulation (EU) 2023/851 of the European Parliament and of the Council of 19 April 2023 amending Regulation (EU) 2019/631 as regards strengthening the CO2 emission performance standards for new passenger cars and new light commercial vehicles in line with the Union's increased climate ambition," 2023. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32023R0851, Accessed: 2024-02-19.

[5] European Commission, "Communication from the Commission to the European Parliament, the European Council, the Council, the European Economic and Social Committee and the Committee of the Regions: REPowerEU Plan," 2022. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52022DC0230, Accessed: 2024-02-19.

[6] T. Verschueren, K. Mets, B. Meersman, M. Strobbe, C. Develder, and L. Vandevelde, "Assessment and mitigation of voltage violations by solar panels in a residential distribution grid," in *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 540–545, IEEE, 2011.

[7] K. Mets, R. D'hulst, and C. Develder, "Comparison of intelligent charging algorithms for electric vehicles to reduce peak load and demand variability in a distribution grid," *Journal of Communications and Networks*, vol. 14, no. 6, pp. 672–681, 2012.

[8] B. Williams, D. Bishop, G. Hooper, and J. Chase, "Driving change: Electric vehicle charging behavior and peak loading," *Renewable and Sustainable Energy Reviews*, vol. 189, p. 113953, 2024.

[9] European Commission, "Grids, the missing link – an EU action plan for grids: Communication from the commission to the european parliament, the council, the european economic and social committee and the committee of the regions." https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52023DC0757, Nov. 2023. Published 28 November 2023. Accessed 23 April 2025.

[10] S. Pelekis, I.-K. Seisopoulos, E. Spiliotis, T. Pountridis, E. Karakolis, S. Mouzakitis, and D. Askounis, "A comparative assessment of deep learning models for day-ahead load forecasting: Investigating key accuracy drivers," *Sustainable Energy, Grids and Networks*, vol. 36, p. 101171, 2023.

[11] M. Cai, M. Pipattanasomporn, and S. Rahman, "Day-ahead building-level load forecasts using deep learning vs. traditional time-series techniques," *Applied energy*, vol. 236, pp. 1078–1088, 2019.

[12] J. Coignard *et al.*, "Evaluating forecasting methods in the context of local energy communities," *International Journal of Electrical Power & Energy Systems*, vol. 131, p. 106956, 2021.

[13] S. Haben, S. Arora, G. Giasemidis, M. Voss, and D. V. Greetham, "Review of low voltage load forecasting: Methods, applications, and recommendations," *Applied Energy*, vol. 304, p. 117798, 2021.

[14] C. Kuster, Y. Rezgui, and M. Mourshed, "Electrical load forecasting models: A critical systematic review," *Sustainable cities and society*, vol. 35, pp. 257–270, 2017.

[15] H. Wang, K. A. Alattas, A. Mohammadzadeh, M. H. Sabzalian, A. A. Aly, and A. Mosavi, "Comprehensive review of load forecasting with emphasis on intelligent computing approaches," *Energy Reports*, vol. 8, pp. 13189–13198, 2022.

[16] R. Wazirali, E. Yaghoubi, M. S. S. Abujazar, R. Ahmad, and A. H. Vakili, "State-of-the-art review on energy and load forecasting in microgrids using artificial neural networks, machine learning, and deep learning techniques," *Electric power systems research*, vol. 225, p. 109792, 2023.

[17] W. Charytoniuk and M.-S. Chen, "Very short-term load forecasting using artificial neural networks," *IEEE transactions on Power Systems*, vol. 15, no. 1, pp. 263–268, 2000.

[18] B. F. Hobbs, S. Jitprapaikulsarn, S. Konda, V. Chankong, K. A. Loparo, and D. J. Maratukulam, "Analysis of the value for unit commitment of improved load forecasts," *IEEE Transactions on Power Systems*, vol. 14, no. 4, pp. 1342–1348, 1999.

[19] N. Amjady and F. Keynia, "Mid-term load forecasting of power systems by a new prediction method," *Energy Conversion and Management*, vol. 49, no. 10, pp. 2678–2687, 2008.

[20] K. Lindberg, P. Seljom, H. Madsen, D. Fischer, and M. Korpås, "Long-term electricity load forecasting: Current and future trends," *Utilities Policy*, vol. 58, pp. 102–119, 2019.

[21] X. Wang, H. Wang, S. Li, and H. Jin, "A reinforcement learning-based online learning strategy for real-time short-term load forecasting," *Energy*, vol. 305, p. 132344, 2024.

[22] J. Faraji, A. Ketabi, H. Hashemi-Dezaki, M. Shafie-Khah, and J. P. Catalao, "Optimal day-ahead self-scheduling and operation of prosumer microgrids using hybrid machine learning-based weather and load forecasting," *IEEE Access*, vol. 8, pp. 157284–157305, 2020.

[23] A. Bracale, P. Caramia, P. De Falco, and T. Hong, "Multivariate quantile regression for short-term probabilistic load forecasting," *IEEE Transactions on Power Systems*, vol. 35, no. 1, pp. 628–638, 2019.

[24] W. Zhang, H. Quan, and D. Srinivasan, "An improved quantile regression neural network for probabilistic load forecasting," *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 4425–4434, 2018.

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

[26] M. Gebetsberger, J. W. Messner, G. J. Mayr, and A. Zeileis, "Estimation methods for nonhomogeneous regression models: Minimum continuous ranked probability score versus maximum likelihood," *Monthly Weather Review*, vol. 146, no. 12, pp. 4323–4338, 2018.

[27] T. Duan, A. Anand, D. Y. Ding, K. K. Thai, S. Basu, A. Ng, and A. Schuler, "Ngboost: Natural gradient boosting for probabilistic prediction," in *International conference on machine learning*, pp. 2690–2700, PMLR, 2020.

[28] T. Hong and S. Fan, "Probabilistic electric load forecasting: A tutorial review," *International Journal of Forecasting*, vol. 32, no. 3, pp. 914–938, 2016.

[29] J. Zhang, X. Wen, Z. Zhang, S. Zheng, J. Li, and J. Bian, "Probts: Benchmarking point and distributional forecasting across diverse prediction horizons," *Advances in Neural Information Processing Systems*, vol. 37, pp. 48045–48082, 2024.

[30] H. Shi, M. Xu, and R. Li, "Deep learning for household load forecasting—a novel pooling deep rnn," *IEEE transactions on smart grid*, vol. 9, no. 5, pp. 5271–5280, 2017.

[31] B. Yildiz, J. I. Bilbao, J. Dore, and A. Sproul, "Household electricity load forecasting using historical smart meter data with clustering and classification techniques," in *2018 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia)*, pp. 873–879, IEEE, 2018.

[32] P. Ma, S. Cui, M. Chen, S. Zhou, and K. Wang, "Review of family-level short-term load forecasting and its application in household energy management system," *Energies*, vol. 16, no. 15, p. 5809, 2023.

[33] H. Chitsaz, H. Shaker, H. Zareipour, D. Wood, and N. Amjady, "Short-term electricity load forecasting of buildings in microgrids," *Energy and Buildings*, vol. 99, pp. 50–60, 2015.

[34] L. Xu, S. Wang, and R. Tang, "Probabilistic load forecasting for buildings considering weather forecasting uncertainty and uncertain peak load," *Applied energy*, vol. 237, pp. 180–195, 2019.

[35] C. J. Bennett, R. A. Stewart, and J. W. Lu, "Forecasting low voltage distribution network demand profiles using a pattern recognition based expert system," *Energy*, vol. 67, pp. 200–212, 2014.

[36] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960.

[37] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International journal of forecasting*, vol. 20, no. 1, pp. 5–10, 2004.

[38] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting methods and applications*. John wiley & sons, 2008.

[39] L. F. Tratar and E. Strmčnik, "The comparison of holt–winters method and multiple regression method: A case study," *Energy*, vol. 109, pp. 266–276, 2016.

[40] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*. San Francisco, CA: Holden-Day, 1970.

[41] M. Peixeiro, *Time series forecasting in python*. Simon and Schuster, 2022.

[42] C. Tarmanini, N. Sarma, C. Gezegin, and O. Ozgonenel, "Short term load forecasting based on arima and ann approaches," *Energy Reports*, vol. 9, pp. 550–557, 2023.

[43] C.-M. Lee and C.-N. Ko, "Short-term load forecasting using lifting scheme and arima models," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5902–5911, 2011.

[44] M. Cho, J. Hwang, and C. Chen, "Customer short term load forecasting by using arima transfer function model," in *Proceedings 1995 International Conference on Energy Management and Power Delivery EMPD'95*, vol. 1, pp. 317–322, IEEE, 1995.

[45] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[46] T. Bashir, C. Haoyong, M. F. Tahir, and Z. Liqiang, "Short term electricity load forecasting using hybrid prophet-lstm model optimized by bpnn," *Energy reports*, vol. 8, pp. 1678–1686, 2022.

[47] A. I. Almazrouee, A. M. Almeshal, A. S. Almutairi, M. R. Alenezi, and S. N. Alhajeri, "Long-term forecasting of electrical loads in kuwait using prophet and holt–winters models," *Applied Sciences*, vol. 10, no. 16, p. 5627, 2020.

[48] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[49] L. Zhang and D. Jánošík, "Enhanced short-term load forecasting with hybrid machine learning models: Catboost and xgboost approaches," *Expert Systems with Applications*, vol. 241, p. 122686, 2024.

[50] Y. Wang, S. Sun, X. Chen, X. Zeng, Y. Kong, J. Chen, Y. Guo, and T. Wang, "Short-term load forecasting of industrial customers based on svmd and xgboost," *International Journal of Electrical Power & Energy Systems*, vol. 129, p. 106830, 2021.

[51] W. Yin, J. Ji, T. Wen, and C. Zhang, "Study on orderly charging strategy of ev with load forecasting," *Energy*, vol. 278, p. 127818, 2023.

[52] R. A. Abbasi, N. Javaid, M. N. J. Ghuman, Z. A. Khan, S. Ur Rehman, and Amanullah, "Short term load forecasting using xgboost," in *Web, artificial intelligence and network applications: proceedings of the workshops of the 33rd international conference on advanced information networking and applications (WAINA-2019) 33*, pp. 1120–1131, Springer, 2019.

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

[54] B. N. Oreshkin, G. Dudek, P. Pełka, and E. Turkina, "N-beats neural network for mid-term electricity load forecasting," *Applied Energy*, vol. 293, p. 116918, 2021.

[55] N. P. Singh, A. R. Joshi, and M. N. Alam, "Short-term forecasting in smart electric grid using n-beats," in *2022 second international conference on power, control and computing technologies (ICPC2T)*, pp. 1–6, IEEE, 2022.

[56] H. Wen, J. Gu, J. Ma, L. Yuan, and Z. Jin, "Probabilistic load forecasting via neural basis expansion model based prediction intervals," *IEEE transactions on smart grid*, vol. 12, no. 4, pp. 3648–3660, 2021.

[57] P. Remy, "N-beats: Neural basis expansion analysis for interpretable time series forecasting." https://github.com/philipperemy/n-beats, 2020.

[58] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021.

[59] D. Li, Y. Tan, Y. Zhang, S. Miao, and S. He, "Probabilistic forecasting method for mid-term hourly load time series based on an improved temporal fusion transformer model," *International Journal of Electrical Power & Energy Systems*, vol. 146, p. 108743, 2023.

[60] A. Nazir, A. K. Shaikh, A. S. Shah, and A. Khalil, "Forecasting energy consumption demand of customers in smart grid using temporal fusion transformer (tft)," *Results in Engineering*, vol. 17, p. 100888, 2023.

[61] M. L. Santos, S. D. García, X. García-Santiago, A. Ogando-Martínez, F. E. Camarero, G. B. Gil, and P. C. Ortega, "Deep learning and transfer learning techniques applied to short-term load forecasting of data-poor buildings in local energy communities," *Energy and Buildings*, vol. 292, p. 113164, 2023.

[62] E. Saadipour-Hanzaie, M.-A. Pourmoosavi, and T. Amraee, "Deep learning based electrical load forecasting using temporal fusion transformer and trend-seasonal decomposition," in *2023 31st International Conference on Electrical Engineering (ICEE)*, pp. 283–288, IEEE, 2023.

[63] A. B. Ferreira, J. B. Leite, and D. H. Salvadeo, "Power substation load forecasting using interpretable transformer-based temporal fusion neural networks," *Electric Power Systems Research*, vol. 238, p. 111169, 2025.

[64] Ofgem, "The state of the market for customers with dynamically teleswitched meters." https://www.ofgem.gov.uk/sites/default/files/docs/2013/07/the-state-of-the-market-for-customers-with-dynamically-teleswitched-meters_0.pdf, July 2013. Accessed 11 September 2025.

[65] E. Energy, "Freedom for business terms & conditions." https://www.edfenergy.com/sites/default/files/b3001_edfe_freedom-ge_tcs_inline_a4_aw26_e1.pdf, 2014. Accessed 11 September 2025.

[66] I. P. N. Limited, "Use of system charging statement, final notice." https://www.gtc-uk.co.uk/wp-content/uploads/2020/07/IPNL-Tariffs-wef-1-April-2012-Final-Charges-Statement.pdf, April 2012. Accessed 11 September 2025.

[67] S. Khatoon, A. K. Singh, *et al.*, "Effects of various factors on electric load forecasting: An overview," in *2014 6th IEEE Power India International Conference (PIICON)*, pp. 1–5, IEEE, 2014.

[68] T.-H. Dang-Ha, F. M. Bianchi, and R. Olsson, "Local short term electricity load forecasting: Automatic approaches," in *2017 international joint conference on neural networks (ijcnn)*, pp. 4267–4274, IEEE, 2017.

[69] E. C. for Medium-Range Weather Forecasts, "Evaluation of ecmwf forecasts including the 2023 upgrade," tech. rep., ECMWF, 2023. Accessed March 21, 2025.

[70] S. Haben, G. Giasemidis, F. Ziel, and S. Arora, "Short term load forecasting and the effect of temperature at the low voltage level," *International Journal of Forecasting*, vol. 35, no. 4, pp. 1469–1484, 2019.

[71] B. Han, Y. Zahraoui, M. Mubin, S. Mekhilef, M. Seyedmahmoudian, and A. Stojcevski, "Home energy management systems: A review of the concept, architecture, and scheduling strategies," *Ieee Access*, vol. 11, pp. 19999–20025, 2023.

[72] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The journal of machine learning research*, vol. 13, no. 1, pp. 281–305, 2012.

[73] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[74] S. Putatunda and K. Rama, "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of xgboost," in *Proceedings of the 2018 international conference on signal processing and machine learning*, pp. 6–10, 2018.

[75] A. A. Chowdhury, A. Das, K. K. S. Hoque, and D. Karmaker, "A comparative study of hyperparameter optimization techniques for deep learning," in *Proceedings of International Joint Conference on Advances in Computational Intelligence: IJCACI 2021*, pp. 509–521, Springer, 2022.

[76] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[77] P. I. Frazier, “A tutorial on bayesian optimization,” *arXiv preprint arXiv:1807.02811*, 2018.

[78] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” *Advances in neural information processing systems*, vol. 24, 2011.

[79] S. Watanabe, “Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance,” *arXiv preprint arXiv:2304.11127*, 2023.

[80] S. Watanabe, N. Awad, M. Onishi, and F. Hutter, “Speeding up multi-objective hyperparameter optimization by task similarity-based meta-learning for the tree-structured parzen estimator,” *arXiv preprint arXiv:2212.06751*, 2022.

[81] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi, “Multiobjective tree-structured parzen estimator for computationally expensive optimization problems,” in *Proceedings of the 2020 genetic and evolutionary computation conference*, pp. 533–541, 2020.

[82] G. Rong, K. Li, Y. Su, Z. Tong, X. Liu, J. Zhang, Y. Zhang, and T. Li, “Comparison of tree-structured parzen estimator optimization in three typical neural network models for landslide susceptibility assessment,” *Remote Sensing*, vol. 13, no. 22, p. 4694, 2021.

[83] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

[84] L. J. Tashman, “Out-of-sample tests of forecasting accuracy: an analysis and review,” *International journal of forecasting*, vol. 16, no. 4, pp. 437–450, 2000.

[85] I. Svetunkov, *Forecasting and analytics with the augmented dynamic adaptive model (ADAM)*. CRC Press, 2023.

[86] O. Triebe, H. Hewamalage, P. Pilyugina, N. Laptev, C. Bergmeir, and R. Rajagopal, “Neuralprophet: Explainable forecasting at scale,” *arXiv preprint arXiv:2111.15397*, 2021.

[87] S. Raschka, J. Patterson, and C. Nolet, “Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence,” *arXiv preprint arXiv:2002.04803*, 2020.