IJRCS

International Journal of Robotics and Control Systems
Vol. 6, No. 1, 2026, pp. 670-692
ISSN 2775-2658
http://pubs2.ascee.org/index.php/ijrcs

ASCEE

Analyzing Hyperparameter Impact on TimeMixer Accuracy for Short-Term Load Forecasting

Tuan Anh Nguyen [a,1,*], Trung Dung Nguyen [a,2]

[a] Faculty of Electrical Engineering Technology, Industrial University of Ho Chi Minh City, Ho Chi Minh City, 700000, Vietnam
[1] nguyenanhtuan@iuh.edu.vn; [2] nguyentrungdung@iuh.edu.vn
* Corresponding Author

ARTICLE INFO

Article history
Received January 09, 2026
Revised February 14, 2026
Accepted March 07, 2026

Keywords
TimeMixer;
Hyperparameter Optimization;
Random Search;
Tree-Structured Parzen Estimator (TPE);
Genetic Algorithm

ABSTRACT

This paper examines the sensitivity of the TimeMixer forecasting model to key hyperparameters in short-term load forecasting (STLF). Experiments are conducted on the New South Wales (NSW) half-hourly load dataset under a fixed forecasting protocol with a 7-day input window (input_size = 336) and a 1-day-ahead horizon (h = 48). Models are trained using a recent-window retraining setup and evaluated on a time-ordered holdout set. The research contribution is twofold: To quantify the effects of four influential hyperparameters (learning_rate, dropout, d_model, and e_layers) on TimeMixer accuracy using trial-level error distributions and summary statistics, and to compare three hyperparameter optimization strategies, Random Search (RS), Tree-structured Parzen Estimator (TPE), and a Genetic Algorithm (GA) under an identical evaluation budget. Forecasting performance is assessed using mean absolute percentage error (MAPE), and hyperparameter effects are characterized through boxplots across trials and median performance across discrete hyperparameter levels. Results show that hyperparameter optimization consistently improves TimeMixer over the default configuration, reducing the best MAPE to 2.154% (RS), 2.119% (TPE), and 1.895% (GA), with GA achieving the most significant improvement. These findings provide practical guidance on selecting both an optimization strategy and robust hyperparameter settings when deploying TimeMixer for STLF.

1. Introduction

Short-term load forecasting (STLF) is a core component of modern power-system operation and planning because electrical demand must be continuously balanced with generation under security and economic constraints. Accurate STLF supports unit commitment and dispatch, reserve scheduling, congestion management, and market participation, thereby reducing operating costs and improving reliability. Conversely, forecast errors may lead to excessive reserve procurement, inefficient dispatch, or an increased risk of operating limit violations. For these reasons, improving STLF accuracy remains a necessary research and engineering objective.

Load forecasting methods have evolved from classical statistical models to machine learning (ML) and deep learning (DL). Traditional time-series approaches such as ARIMA and SARIMA [1]-[9], exponential smoothing [10]-[13], and linear regression [14]-[17] remain attractive due to their

simplicity, low computational cost, and interpretability. However, such models often rely on assumptions (linearity and stationary seasonality) that can be violated in real-world load data, which may exhibit nonlinear dynamics, abrupt regime changes, and complex interactions driven by operational and behavioral factors. As a result, purely statistical models may struggle when demand patterns are highly nonlinear or rapidly varying.

To better capture nonlinear relationships and improve practical generalization, ML models have been widely adopted. Representative approaches include Support Vector Regression (SVR) [18]-[20], k-Nearest Neighbors (kNN) [21]-[23], Random Forests [24]-[28], Gradient Boosting [29]-[34], and modern boosting frameworks such as LightGBM [35]-[41]. These methods can yield strong performance when appropriate features are engineered, and exogenous variables (such as calendar indicators or weather) are incorporated. Nevertheless, ML-based forecasting often depends heavily on feature engineering, lag selection, and preprocessing choices. Moreover, many ML models do not natively model long-range temporal dependencies unless temporal structure is carefully encoded in the input representation. With the increasing availability of high-resolution load measurements, DL models have become prominent because they can learn useful representations directly from data. Recurrent architectures such as RNN [42]-[45], LSTM [46]-[52], and GRU [53]-[60] are designed to model sequential dependence, while convolutional models such as CNN [61]-[64] and TCN [65]-[68] can efficiently extract temporal patterns and often offer stable training. Transformer-based models further improve the modeling of long-range dependencies via attention mechanisms, and specialized forecasting architectures such as N-BEATS [69]-[73] and N-HiTS [74]-[75] have demonstrated competitive performance on benchmark time-series tasks. Despite their expressiveness, DL models commonly face practical challenges: performance can be susceptible to architectural and training hyperparameters, overfitting may occur when regularization is insufficient, and convergence can become unstable when hyperparameter values are suboptimal. Consequently, selecting suitable hyperparameters is often as critical as choosing the model family itself.

In this study, we focus on the TimeMixer [76]-[80] forecasting model, implemented in the Nixtla NeuralForecast ecosystem. TimeMixer is designed to capture temporal patterns through structured mixing operations, and NeuralForecast provides a standardized pipeline for multi-step forecasting with panel-format data (unique_id, ds, y). From a practical standpoint, this ecosystem offers a consistent training and evaluation interface that supports reproducible comparisons across model configurations. However, even when the architecture is suitable for STLF, empirical accuracy can degrade substantially if key hyperparameters, such as the learning rate, dropout, representation size (d_model), or model depth (e_layers), are poorly matched to the dataset and forecasting protocol.

This motivates the research gap addressed in this paper. While TimeMixer has shown strong forecasting capability, there remains limited practical guidance on (i) which hyperparameters most strongly influence accuracy under a fixed STLF protocol and (ii) how different hyperparameter optimization strategies affect both the discovered configurations and the apparent sensitivity of TimeMixer to those hyperparameters. To address this gap, we systematically evaluate the influence of four core TimeMixer hyperparameters (learning_rate, dropout, d_model, and e_layers) on STLF accuracy and compare three widely used optimization strategies: Random Search (RS) [81], Tree-structured Parzen Estimator (TPE) [82], [83], and Genetic Algorithm (GA) [84] under a consistent experimental design. RS provides a strong baseline with broad coverage, TPE uses probabilistic modeling to guide trial selection, and GA explores the search space via population-based evolution. By analyzing trial-level error distributions and performance across hyperparameter values, we aim to identify the most influential hyperparameters and determine which optimizer yields the most favorable configurations in the considered STLF setting. The main contributions of this study are as follows:

- Unified experimental framework for TimeMixer hyperparameter analysis in STLF. We establish a consistent pipeline (data split, forecasting horizon, evaluation protocol, and search space) to ensure fair comparisons across optimization strategies.
- Systematic comparison of three hyperparameter optimization strategies (RS, TPE, and GA). Using the same tuned hyperparameter set (learning_rate, dropout, d_model, e_layers) under fixed

modeling settings, we compare optimizers based on best-achieved accuracy and the distribution of results across trials.

- Hyperparameter sensitivity assessment using trial-level error distributions. We quantify how forecasting error (MAPE) varies with each hyperparameter by analyzing trial outcomes across candidate values, highlighting median behavior and dispersion to identify the most influential hyperparameters.
- Practical insights for configuring TimeMixer in short-term load forecasting. Based on a comparative analysis across RS, TPE, and GA, we provide actionable guidance on which hyperparameters require the most careful tuning and how the choice of optimizer may affect the resulting configuration quality.

2. Method

2.1. TimeMixer Model and Architecture

TimeMixer is a decomposable multiscale mixing forecasting model that extracts past information and generates future predictions by exploiting representations at multiple temporal scales. Concretely, the architecture is built around two key modules: Past-Decomposable-Mixing (PDM) for multiscale feature extraction from the historical window, and Future-Multipredictor-Mixing (FMM) for combining multi-scale predictive skills into the final forecast.

Given a univariate load series y_t(or multivariate $\mathrm{y}_t \in \mathbb{R}^C$), the forecasting objective is to use the most recent input window of length $L$to predict the next horizon of length H:

$$\mathrm{x}_t = [y_{t-L+1}, y_{t-L+2}, \dots, y_t] \in \mathbb{R}^L \tag{1}$$

$$\hat{\mathrm{y}}_t = [\hat{y}_{t+1}, \hat{y}_{t+2}, \dots, \hat{y}_{t+H}] = f_\theta(\mathrm{x}_t) \tag{2}$$

In your experimental setting (Nixtla/NeuralForecast): $L = \text{input\_size} = 336$(7 days × 48 points/day) and $H = 48$(1 day ahead). A central idea of TimeMixer is to create multiple scaled views of the same past window, because fine scales preserve detailed fluctuations while coarse scales emphasize macroscopic trends. bet $\mathrm{x}_t^{(0)} = \mathrm{x}_t$ be the original scale. For scale $s \geq 1$, a downsampled series $\mathrm{x}_t^{(s)}$is produced from $\mathrm{x}_t^{(s-1)}$using a pooling operator with window size w(in your code: down_sampling_window $= 2$, method = avg):

$$x_{t,k}^{(s)} = \frac{1}{w}\sum_{i=1}^{w} x_{t,(k-1)w+i}^{(s-1)}, k = 1,2, \dots, \left\lfloor \frac{L}{w^s} \right\rfloor \tag{3}$$

Thus, TimeMixer forms a multiscale set $\left\{\mathrm{x}_t^{(0)}, \mathrm{x}_t^{(1)}, \dots, \mathrm{x}_t^{(S)}\right\}$that will be processed jointly. At each scale s, the model projects the raw sequence into a latent space of dimension d_model(in your tuning: d_model ∈ {16,24,32,64,128}). A standard linear embedding can be expressed as:

$$\mathrm{E}^{(s)} = \mathrm{X}^{(s)}\mathrm{W}_e + \mathrm{b}_e, \mathrm{X}^{(s)} \in \mathbb{R}^{T_s \times C}, \mathrm{E}^{(s)} \in \mathbb{R}^{T_s \times d} \tag{4}$$

Where $T_s = \left\lfloor \frac{L}{w^s} \right\rfloor$, $d = d_{\text{model}}$, and $C$is the number of variables (for univariate load, $C = 1$). TimeMixer explicitly leverages series decomposition to separate complex temporal variation into interpretable components (seasonal vs. trend), then performs multiscale mixing on these disentangled parts. A common decomposition form is:

$$\mathrm{X}^{(s)} = \mathrm{S}^{(s)} + \mathrm{T}^{(s)} \tag{5}$$

Whsuperscript denotes the seasonal (fluctuation) component, and bold cap T to the, open paren s, close paren, end superscript denotesere $\mathrm{S}^{(s)}$denotes the seasonal (fluctuation) component and

$T^{(s)}$denotes the trend (slow-moving) component. In practice, a moving-average trend extractor is often used:

$$T^{(s)} = MA\left(X^{(s)}\right), S^{(s)} = X^{(s)} - T^{(s)} \quad (6)$$

Applsuperscript, and bold cap T to the, open paren s, close paren, end superscript, separately (because they represent different dynamics). Stack the block e_layerstimes (your tuned depth), so deeper configurations can model more complex interactions across scales. A generic residual form for one PDM layer at scale $s$can can be written as:

$$\tilde{S}^{(s)} = S^{(s)} + \mathcal{M}_S^{(s)}(\{S^{(j)}\}_{j=0}^{S}), \tilde{T}^{(s)} = T^{(s)} + \mathcal{M}_T^{(s)}(\{T^{(j)}\}_{j=0}^{S}) \quad (7)$$

Where $\mathcal{M}_S^{(s)}(\cdot)$and $\mathcal{M}_T^{(s)}(\cdot)$denote learnable mixing functions (implemented by MLP/mixing blocks in the TimeMixer design).

After extracting multiscale representations, TimeMixer generates predictions at each scale and then ensembles/fuses them using Future-Multipredictor-Mixing (FMM), leveraging the complementary forecasting strengths across scales. Let $\hat{\mathbf{y}}^{(s)} \in \mathbb{R}^H$be the horizon prediction produced from scale s. A general fusion form is:

$$\hat{y} = \mathcal{G}\left(\hat{y}^{(0)}, \hat{y}^{(1)}, \ldots, \hat{y}^{(S)}\right) \quad (8)$$

A simple (but common) learnable fusion is a weighted sum:

$$\hat{y} = \sum_{s=0}^{S} \alpha_s \, \hat{y}^{(s)}, \sum_{s=0}^{S} \alpha_s = 1 \quad (9)$$

Where α_s are learnable or data-driven mixing weights. In your code (NeuralForecast–TimeMixer), the architecture is instantiated with: $H = 48$(forecast horizon),$L = 336$(input_size fixed), multiscale generation via average pooling with window $w = 2$(down_sampling_window=2, down_sampling_method='avg'). Model capacity controlled mainly by: d_model (latent dimension), e_layers (number of stacked mixing layers), dropout (regularization), and learning_rate (optimization dynamics).

2.2. Hyperparameters of the TimeMixer model

Within the Nixtla NeuralForecast framework, the TimeMixer model is typically specified through two major groups of hyperparameters: architectural settings (which govern representation capacity and mixing depth) and training settings (which govern the optimization dynamics). Because the official API exposes a relatively large set of options, this study deliberately focuses on a small subset of core hyperparameters most likely to influence forecasting accuracy, while keeping the remaining settings fixed. This controlled design is intended to ensure a fair and interpretable comparison among the three hyperparameter optimization strategies: Random Search (RS), Tree-structured Parzen Estimator (TPE), and Genetic Algorithm (GA).

Specifically, the study optimizes four hyperparameters using a centered-default search space, where the baseline (design) value is positioned near the middle of the candidate set to reduce bias toward extremes. Learning rate is included because it directly determines the step size of parameter updates and is often the most sensitive factor affecting convergence quality; an inappropriate learning rate can lead to unstable training or suboptimal minima, which is typically reflected in substantial variations in MAPE. Dropout represents a regularization mechanism that mitigates overfitting; this is particularly relevant for electric load series that exhibit seasonal patterns and noise, where excessive fitting to short-term fluctuations may degrade generalization on the test horizon. D_model controls the width of the latent representation and, therefore, the model's expressive capacity. Overly small values may under-represent temporal dependencies, whereas overly large values may increase optimization difficulty and raise the risk of overfitting. E_layers captures the adequate depth of the

mixing blocks; increasing depth can enhance the model's ability to extract hierarchical temporal structure, but may also introduce greater optimization complexity. E_layers is explicitly tuned to examine whether different optimization strategies discover consistent depth preferences. To reduce the dimensionality of the search and maintain a coherent representation scale, the feed-forward width is tied to the model width by enforcing d_ff = d_model throughout all experiments.

To ensure that performance differences arise only from the four tuned hyperparameters, all remaining settings are fixed according to the experimental protocol. The forecasting horizon is set to h = 48, corresponding to one day ahead for a 30-minute sampling frequency. The input context length is fixed at input_size = 336, corresponding to seven days of historical observations. The multi-scale processing mechanism is held constant by keeping the downsampling configuration unchanged, thereby preserving the model's information aggregation across resolutions. Likewise, the key training controls, such as the maximum number of training steps, the batch size, and the validation check schedule, are kept identical across RS, TPE, and GA. Finally, a fixed random seed is used to enhance reproducibility and limit stochastic variation that could otherwise confound comparisons among optimization methods.

To ensure that performance differences are attributable only to the four tuned hyperparameters, all remaining settings are held constant. The forecasting horizon is fixed = 48, and the input context is fixed at input_size = 336. The downsampling configuration is unchanged to preserve the multi-scale aggregation mechanism. Training controls such as maximum steps, batch size, and validation schedule are kept identical across all trials and across RS, TPE, and GA. A fixed random seed is used for reproducibility. Table 1 summarizes 10 core baseline settings consistently adopted across the experimental code.

Table 1. Default configuration of the TimeMixer model

Parameter	h	Input size	learning rate	Drop out	d model	e_ layers	down sampling window	Scaler type	Max steps	Batch size
Baseline	48	336	1e-3	0.1	32	4	2	standard	500	32

2.3. Hyperparameter Optimization Methods

This study compares three hyperparameter optimization strategies: Random Search, Tree-structured Parzen Estimator, and Genetic Algorithm under the same discrete centered-default search space and evaluation budget. All methods aim to minimize forecasting error, measured by MAPE.

$$\text{MAPE}(\%) = \frac{100}{N}\sum_{t=1}^{N}\left|\frac{y_t - \hat{y}_t}{y_t}\right| \tag{10}$$

To prevent information leakage, hyperparameter selection is performed using a time-ordered validation split, and the test set is used only once for final reporting of the selected best configuration for each method. Concretely, each trial is trained on the training portion, evaluated on the validation portion to obtain the objective value, and only the final best configuration is assessed on the test set.

Random Search samples candidate configurations uniformly at random from the discrete search space. Each sampled configuration is trained under the same fixed protocol and evaluated on the validation set to obtain its MAPE. RS is simple and robust, but it does not exploit information from previous trials to guide subsequent sampling, which can be inefficient under a limited budget.

TPE is a Bayesian optimization approach that proposes promising configurations based on the history of observed trials. In this study, TPE is implemented using Optuna's TPESampler, treating each hyperparameter as a categorical choice from predefined discrete values. After each completed trial, TPE updates its probabilistic models to increase the likelihood of sampling configurations associated with lower validation MAPE. To ensure a consistent reference point, the baseline configuration is enqueued as the first trial.

GA is a population-based optimization approach inspired by evolutionary processes. Each encodes a candidate set of the four hyperparameters, and fitness is defined as validation MAPE (lower is better). The GA procedure consists of: (i) initializing a population including the baseline and randomly generated individuals, (ii) selecting parents via tournament selection, (iii) generating offspring via uniform crossover, (iv) applying mutation with a fixed probability to maintain diversity, and (v) retaining the best individual using elitism. The evaluation budget is set to match the other methods; with POP_SIZE = 10 and N_GEN = 5, GA performs 50 evaluations.

For all three methods, each successful trial is recorded, including hyperparameter values and evaluation metrics, enabling subsequent sensitivity analysis and comparisons of optimizers based on best-achieved accuracy and trial-level performance distributions.

2.4. Flowchart of the Proposed Framework

Fig. 1 presents the overall workflow of the proposed forecasting framework, from data preparation to hyperparameter optimization and final evaluation. First, the raw load time-series $Y_1, Y_2, \ldots, Y_n$ is collected and carefully preprocessed to guarantee temporal integrity. This preprocessing stage includes (i) parsing timestamps into a unified datetime format, (ii) sorting all observations in strictly increasing chronological order, and (iii) verifying continuity of the sampling interval (30-minute resolution) to detect missing, duplicated, or irregular time points. When anomalies are identified, the series is corrected or filtered so that the final sequence maintains consistent spacing and can be reliably used for multi-step forecasting.

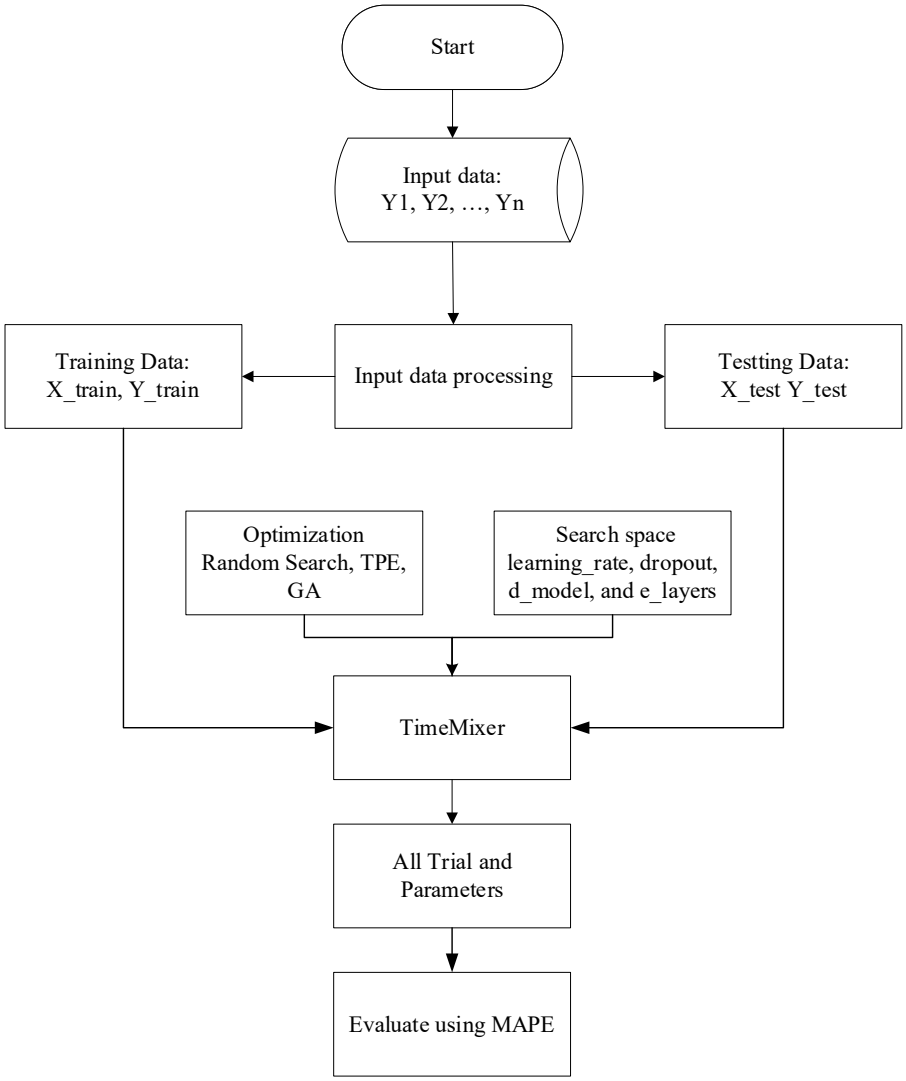

Fig. 1. Flowchart of the hyperparameter optimization search algorithm for the three methods

Next, the cleaned series is transformed into a supervised learning dataset by applying a sliding-window construction. Specifically, an input window $X$is formed from a fixed-length historical segment of the series, and the corresponding output $Y$is defined as the future target sequence over a pre-specified forecasting horizon H. In other words, each training sample consists of (X_t, Y_t), where $X_t = [Y_{t-L+1}, \dots, Y_t]$represents the past Lobservations (historical context), and $Y_t = [Y_{t+1}, \dots, Y_{t+H}]$denotes the next Hsteps to be predicted. This formulation enables TimeMixer to learn the mapping from historical demand patterns to future load trajectories in a multi-step setting.

After window construction, the dataset is partitioned into training and testing subsets, $\mathrm{X_{train}}, \mathrm{Y_{train}}, \mathrm{X_{test}}, \mathrm{Y\_test}$ using a strictly time-ordered split. Unlike random shuffling, this chronological separation preserves temporal causality and prevents information leakage from future observations into the training process. Consequently, the test set represents an unseen future period, providing a realistic evaluation of forecasting performance under operational conditions.

The TimeMixer model is then trained repeatedly under different hyperparameter configurations generated by three hyperparameter optimization strategies: Random Search (RS), Tree-structured Parzen Estimator (TPE), and Genetic Algorithm (GA). A predefined search space is specified for the most influential training and architectural hyperparameters, including the learning rate (learning_rate), the dropout ratio (dropout) for regularization, the model representation width (d_model) controlling latent dimensionality, and the encoder depth (e_layers) determining the number of stacked encoder blocks. For each trial, one candidate configuration is sampled (RS), suggested through probabilistic modeling (TPE), or evolved via population-based selection and mutation or crossover (GA), ensuring that all methods operate under the same evaluation budget for fairness.

For every training run (trial), the selected hyperparameters, training outputs, and evaluation results are logged to enable systematic analysis. The trained model produces multi-step forecasts over the horizon H for the test set, and forecasting accuracy is quantified using Mean Absolute Percentage Error (MAPE). By collecting MAPE scores across all trials, the procedure supports (i) direct comparison of optimization strategies in terms of best-achieved accuracy and overall result distributions, and (ii) selection of the best-performing hyperparameter configuration for TimeMixer under each optimization approach.

3. Experimental Setup

3.1. Dataset Description and Experimental Setup

The load demand data used in this study are a time-series dataset for the New South Wales (NSW) region, consisting of two fields: settlementdate and totaldemand. These field names are consistent with the data definitions in the NEM market data model (AEMO/MMS), where settlementdate denotes the applicable timestamp and totaldemand represents the regional total demand. The dataset was obtained from AEMO's public data source (nemweb/mms). It has a 30-minute resolution and spans from 01/01/2015 to 31/12/2021, comprising 122,735 observations with no missing values. Data preprocessing included timestamp format conversion, chronological sorting, and continuity checks before constructing the training and testing datasets. To provide a concise overview of the dataset's distribution and variability, Table 2 presents key descriptive statistics for the NSW half-hourly load-demand series.

Table 2. Key characteristics and distribution statistics of the NSW demand dataset (MW)

Observations	Time-step continuity	Min	Q1 (25%)	Median (50%)	Mean	Q3 (75%)	Max	Std (σ)
122,735	100% (30-min step)	4,316	6,977.57	7,798	7,890	8,625	13,985	1,260

Table 2 shows that the dataset contains 122,735 observations with 100% continuity at a 30-minute sampling interval, confirming that the time series is complete and suitable for time-series

modeling without gaps. The demand values range from 4,316 to 13,985, indicating substantial variability across the study period. The median demand is 7,798, while the mean is slightly higher at 7,890, suggesting a mildly right-skewed distribution driven by occasional high-demand peaks. The interquartile range (Q3–Q1) is 1,647.43, meaning that the middle 50% of demand values lie within a relatively concentrated band around the central tendency. The standard deviation of 1,260 (approximately 16% of the mean) further indicates moderate dispersion, highlighting the presence of both regular demand fluctuations and peak events that the forecasting model must capture effectively.

Based on the statistical characteristics reported in Table 2, the next step is to establish a preprocessing strategy and a time-ordered data-splitting scheme for model training and evaluation; Fig. 2 illustrates the train/test split used in this study.

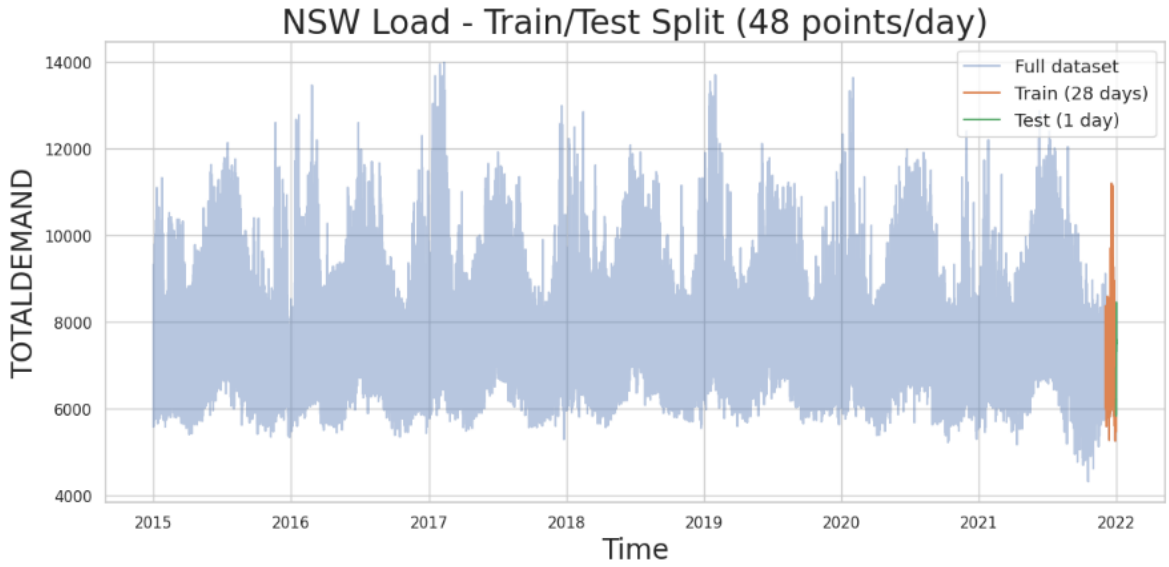

Fig. 2. NSW (AEMO) total demand train (28 days) vs test (1 day)

During preprocessing, the data are loaded from a CSV file, and the timestamp field settlementdate is converted to a standard datetime format (mm/dd/yyyy hh: mm). The entire series is then sorted in ascending chronological order, and the index is reset to preserve the time series' temporal sequence before extracting the training and testing segments. In addition, to ensure reproducibility, a fixed random seed (SEED = 42) is applied across the relevant libraries (NumPy, random, PyTorch, and PyTorch Lightning).

As shown in Fig. 2, the whole load series is displayed with a transparent curve (whole dataset), while the training and testing portions are highlighted at the end of the time series. With a 30-minute resolution, each day yields 48 data points; therefore, the forecasting horizon is set to H = 48, corresponding to a one-day-ahead forecast. The test set is the last 48 observations of the series (the most recent day). In contrast, the training set consists of the 28 days immediately preceding the test day, yielding 48 × 28 observations. This split preserves temporal validity (the test period always follows the training period) and mitigates information leakage, since the model is trained only on historical data to predict the subsequent future period. Moreover, the input window size is set to 7 historical days (48 × 7 points) to provide short-term trend information and daily/weekly seasonality patterns for the forecasting model.

3.2. Configuration of the Hyperparameter Search Space

In this study, to ensure a fair and controlled comparison among the three hyperparameter optimization strategies (Random Search, TPE, and Genetic Algorithm), the TimeMixer model implemented in the Nixtla NeuralForecast library is evaluated under the same experimental setup. Only four core hyperparameters that are expected to influence forecasting accuracy directly are allowed to vary, namely (learning_rate, dropout, d_model, and e_layers), while all other settings are

kept fixed. In addition, all three optimization methods operate on the same discrete "centered-default" search space, as summarized in Table 3.

Table 3. Hyperparameter search space for the three optimization methods

Hyperparameter	Discrete Value Set	Reference Value (Center)
learning_rate	{3e-4, 5e-4, 1e-3, 2e-3, 3e-3}	1e-3
dropout	{0.00, 0.05, 0.10, 0.15, 0.20}	0.10
d_model	{16, 24, 32, 64, 128}	32
e_layers	{2, 3, 4, 5, 6}	4

In Table 3, the reference configuration (executed independently as an evaluation benchmark) is placed at the center of each candidate range to avoid bias and to create two-sided "neighboring" levels. This design ensures that any performance differences observed among RS, TPE, and GA primarily reflect each method's search mechanism rather than changes in the training or evaluation procedures. It also facilitates group-wise impact analysis using median statistics and error distributions for each hyperparameter value. Specifically, learning_rate is sampled around 1×10^{-3} to examine convergence sensitivity (lower values enable more conservative updates, whereas higher values may converge faster but can introduce instability), dropout is selected from 0 to 0.2 to evaluate the role of regularization for noisy and seasonal load series, d_model is expanded from 16 to 128 to observe the trade-off between representational capacity and the risk of overfitting or optimization difficulty, and e_layers varies from 2 to 6 to verify the effect of mixing depth on forecasting error.

3.3. Google Colab Environment and Nixtla Library

All experiments were conducted on Google Colab with an NVIDIA A100 GPU to accelerate model training and evaluation across the datasets considered in this study. The experimental environment was implemented in Python, with the required libraries installed directly in Colab via pip. In particular, Nixtla NeuralForecast was used as the primary framework to build the time-series forecasting pipeline. Specifically, models were instantiated and trained via the NeuralForecast class, configured with a selected architecture (TimeMixer) and an optimization loss/metric (MAE). Core libraries such as PyTorch and PyTorch Lightning were employed to leverage GPU acceleration (CUDA), manage the training loop, and ensure experimental stability.

In addition, data processing and evaluation relied on numpy and pandas (data preprocessing and time-series formatting), matplotlib (visualization), and sklearn.metrics (computing performance indicators such as MSE, MAE, and MAPE). For hyperparameter optimization, the study integrated Random Search (RS), TPE (Tree-structured Parzen Estimator), and the Genetic Algorithm (GA) within a unified workflow: RS randomly samples hyperparameter combinations from the search space; TPE (via Optuna) performs Bayesian optimization by modeling the distributions of "good" versus "bad" configurations to propose more promising trials; and GA explores the search space using an evolutionary mechanism (selection–crossover–mutation) to progressively improve a population of candidate solutions according to the objective function. In each trial, the model is trained on the training set and evaluated on the validation set using the same criterion, after which the best configuration is selected for final testing. To enhance reproducibility and reduce randomness-induced variability, fixed seeds were set for the relevant libraries (random, numpy, torch), and memory management strategies (gc.collect() and releasing GPU memory when necessary) were applied to mitigate out-of-memory issues when running many consecutive trials in Colab.

4. Results And Discussion

4.1. Results Obtained Using Random Search

Random Search (RS) is first used as a baseline optimizer to explore the discrete centered-default hyperparameter space and to reveal how TimeMixer responds to each hyperparameter under the fixed STLF protocol. Rather than describing each boxplot separately, we synthesize the RS evidence into four sensitivity patterns and highlight practical settings that improve both accuracy and robustness.

The dropout analysis indicates that lower dropout values are consistently more favorable in this setting. Specifically, dropout = 0.0 and 0.1 achieve the lowest median MAPEs (approximately 3.548% and 3.621%), while dropout = 0.05 yields the highest median MAPE (about 4.798%). Higher dropout values (0.15 and 0.2) increase median errors (around 4.280% and 4.267%) and widen dispersion, suggesting reduced stability across trials. Overall, RS results indicated that dropout in the range 0.0–0.1 provides a better accuracy–robustness trade-off for the considered dataset and protocol.

The model-width results show a typical capacity trade-off. Intermediate dimensions provide the best balance: d_model = 32 and 64 yield lower median MAPE (approximately 3.652% and 3.612%) than both smaller and larger settings. In contrast, d_model = 16 and 128 produce higher medians (about 4.482% and 4.507%) and larger dispersion at higher width, suggesting that minimal width may under-represent temporal patterns while excessive width increases optimization difficulty and variability. Therefore, d_model = 32–64 emerges as a robust region under RS.

Encoder depth exhibits a similar "moderate is better" tendency. e_layers = 2 and 4 achieve the lowest medians (approximately 3.280% and 3.557%), whereas deeper configurations (e_layers = 3, 5, 6) yield noticeably higher median MAPE (around 4.324%–4.583%) and greater dispersion. The significant variability at e_layers = 6 suggests that deeper stacks do not necessarily improve forecasting accuracy in this setting and may increase instability or overfitting risk. Hence, RS evidence supports using e_layers = 2–4 as a stable depth range.

Learning rate shows the strongest link to both performance and stability, consistent with its role in optimization dynamics. The lowest medians occur at learning_rate = 0.0003 and 0.0005 (approximately 3.449% and 3.479%), while learning_rate = 0.001 remains competitive (median ≈ 3.539%). In contrast, higher learning rates (0.002 and 0.003) substantially worsen median MAPE (approximately 4.571% and 4.605%) and increase dispersion, which is consistent with unstable training and degraded generalization. Overall, RS suggests that learning_rate = 0.0003–0.001 is a robust region for this protocol.

Across Fig. 3, Fig. 4, Fig. 5, Fig. 6, the RS results provide two practical messages. First, TimeMixer accuracy is highly sensitive to hyperparameter choices, and the "safe" region is characterized by low dropout (0.0–0.1), moderate width (d_model 32–64), moderate depth (e_layers 2–4), and small-to-moderate learning rates (0.0003–0.001). Second, robustness matters: the most desirable settings are those that not only reduce the median MAPE but also limit dispersion, because low variance across trials indicates more reliable deployment behavior. From a practical standpoint, even seemingly minor differences in MAPE can accumulate in operational decisions (day-ahead planning and reserve scheduling), so selecting hyperparameters that improve both typical performance and stability can be more valuable than chasing a single best-case run.

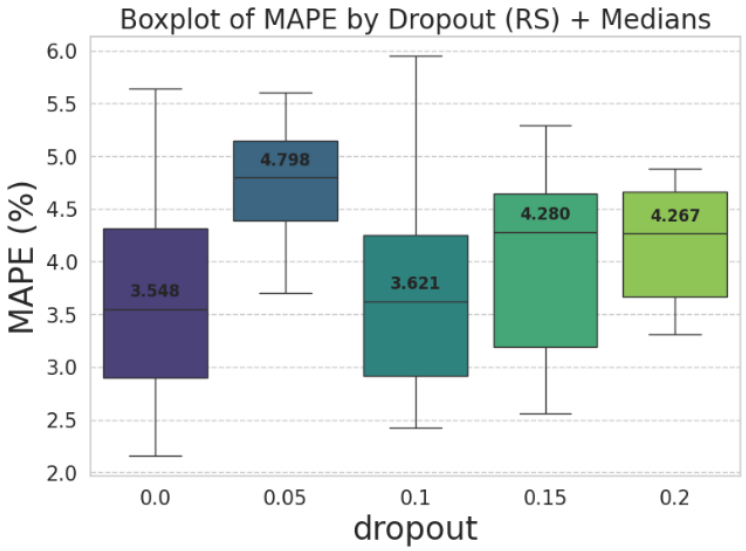

Fig. 3. Boxplot of MAPE for different dropout levels using random search

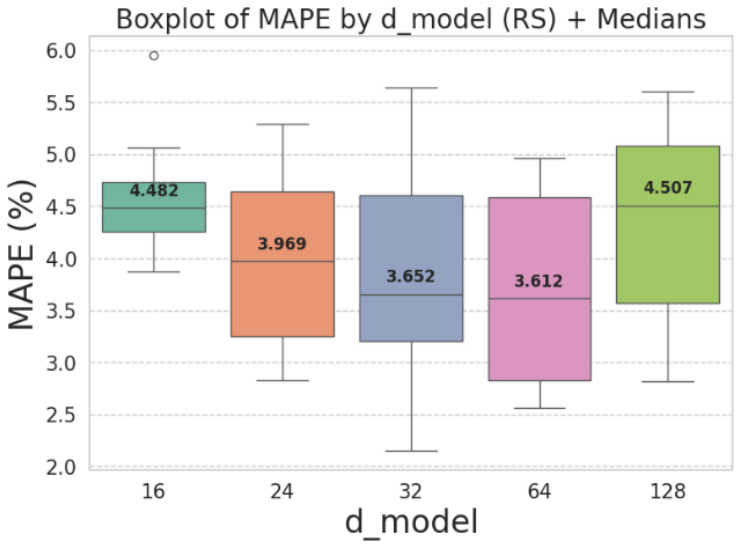

Fig. 4. Boxplot of MAPE for different d_model levels using random search

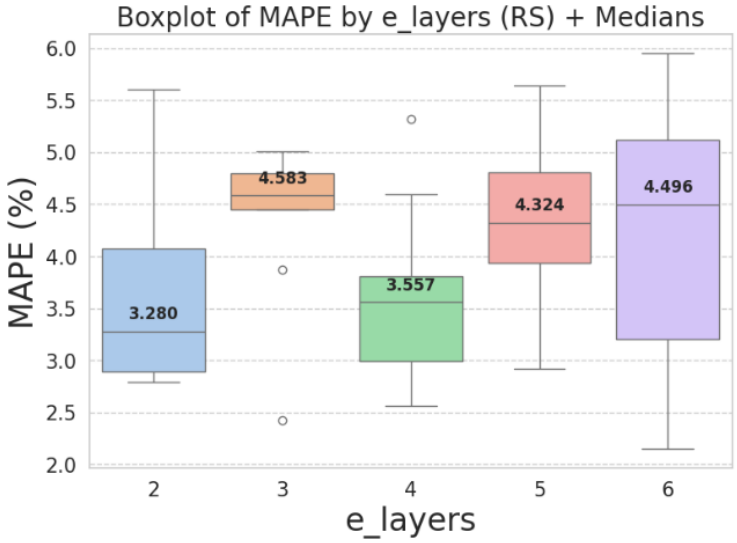

Fig. 5. Boxplot of MAPE for different e_layers levels using random search

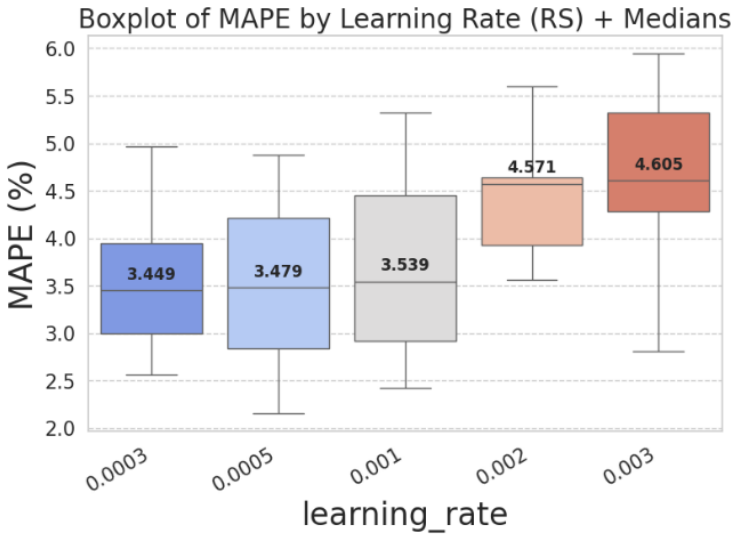

Fig. 6. Boxplot of MAPE for different learning_rate levels using random search

4.2. Results Obtained Using Tree-Structured Parzen Estimator

The Tree-structured Parzen Estimator (TPE) is expected to be more sample-efficient than Random Search because it leverages trial history to propose promising hyperparameter combinations. Fig. 7, Fig. 8, Fig. 9, Fig. 10 summarize how TPE explores the hyperparameter space and which regions are associated with lower and more stable MAPE. To avoid repetitive "figure-by-figure narration," we synthesize the key sensitivity patterns and provide practical ranges supported by the trial distributions.

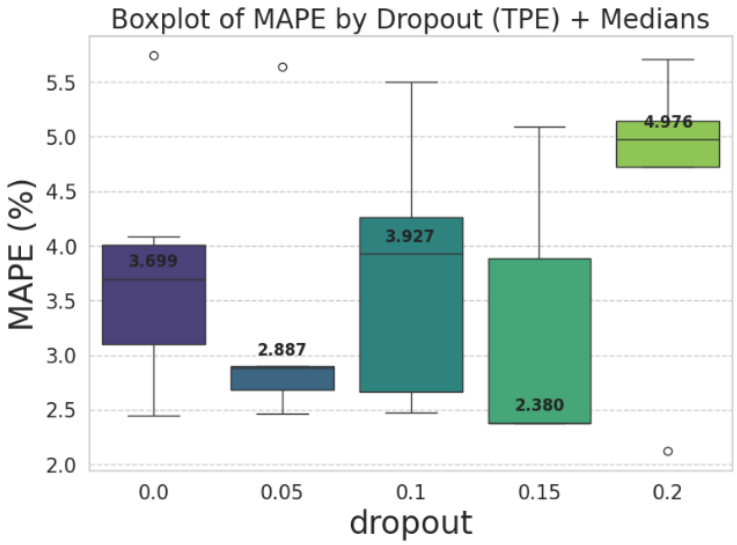

Fig. 7. Boxplot of MAPE for different dropout levels using TPE

The dropout results show that TPE favors an intermediate level of regularization. The lowest median MAPE is achieved at dropout = 0.15 (≈ 2.380%), while dropout = 0.05 also yields a relatively low median (≈ 2.887%). In contrast, smaller dropout values (0.0 and 0.1) yield higher medians (≈ 3.699% and 3.927%), whereas excessive regularization (dropout = 0.2) clearly degrades performance, with the highest median MAPE (≈ 4.976%) and increased dispersion. Overall, these results indicate that, under TPE-based tuning in this setting, moderate dropout improves generalization, whereas too little or too much regularization leads to less reliable performance.

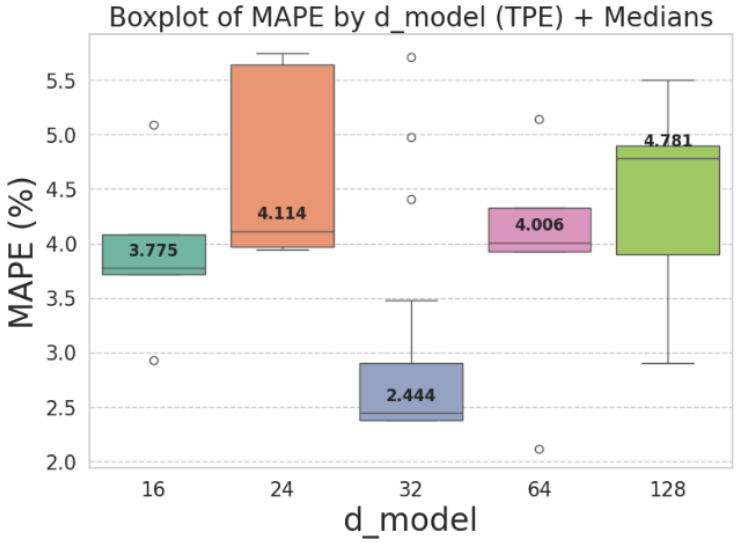

Fig. 8. Boxplot of MAPE for different d_model levels using TPE

For model width, Fig. 8 highlights a sharply preferred capacity region under TPE. The configuration d_model = 32 achieves the lowest median MAPE (≈ 2.444%) with relatively low

dispersion, indicating both high accuracy and strong stability. All other tested widths yield substantially higher median errors: d_model = 16 (≈ 3.775%), 24 (≈ 4.114%), 64 (≈ 4.006%), and 128 (≈ 4.781%), with d_model = 128 also showing greater variability, consistent with over-parameterization risk. This pattern suggests that TPE does not merely chase a single best trial; instead, it focuses on a capacity level that remains consistently robust under the current protocol.

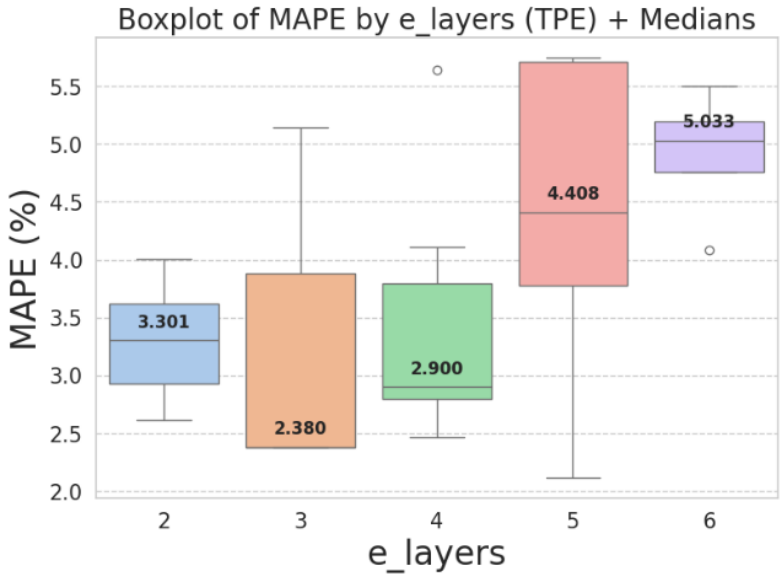

Fig. 9. Boxplot of MAPE for different e_layers levels using TPE

Encoder depth exhibits a similar "moderate is better" trend. The lowest medians occur at e_layers = 3 (≈ 2.380%) and e_layers = 4 (≈ 2.900%), while a shallower setting (e_layers = 2) is worse (≈ 3.301%). Deeper architectures lead to apparent degradation: e_layers = 5 yields a median of ≈ 4.408%, and e_layers = 6 increases to ≈ 5.033%, with greater dispersion. These findings indicate that, for the considered training window and forecasting horizon, increasing depth beyond four layers does not improve generalization and instead reduces stability.

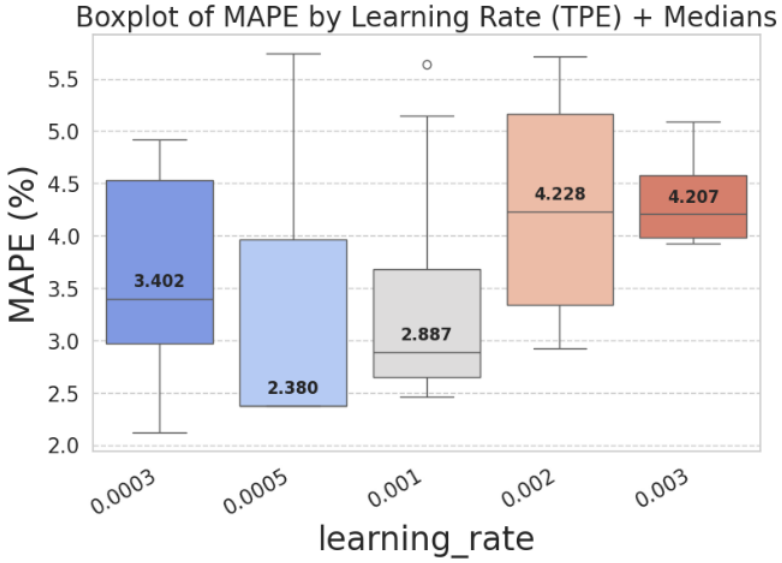

Fig. 10. Boxplot of MAPE for different learning_rate levels using TPE

Learning rate strongly shapes TPE outcomes, consistent with its role in convergence dynamics. The best medians are obtained at learning_rate = 0.0005 (≈ 2.380%) and 0.001 (≈ 2.887%), reflecting stable and effective optimization in this setting. By contrast, both lower values (0.0003, ≈ 3.402%) and higher values (0.002, ≈ 4.228%; 0.003, ≈ 4.207%) yield noticeably worse medians and increased

dispersion, suggesting suboptimal or unstable training behavior. Overall, TPE favors a moderate learning-rate regime (0.0005–0.001).

Taken together, Fig. 7, Fig. 8, Fig. 9, Fig. 10 show that TPE identifies a compact, high-performing region of the search space characterized by dropout ≈ 0.15, d_model = 32, e_layers = 3–4, and learning_rate = 0.0005–0.001. Compared with RS, TPE shows stronger preferences for specific hyperparameter levels and tends to avoid unstable regions (excessive depth, high learning rates, and overly large width) that lead to high dispersion. From a practical perspective, these results emphasize that achieving low MAPE with TimeMixer is not only about locating a single best trial; it also requires selecting hyperparameters that yield consistent performance across trials, which is particularly important for reliable deployment.

4.3. Results Obtained Using Genetic Algorithm

The Genetic Algorithm (GA) explores the discrete hyperparameter space through population-based evolution, thereby improving global exploration compared with purely sequential sampling. Fig. 11, Fig. 12, Fig. 13, Fig. 14 summarize the GA trial distributions and reveal the hyperparameter regions associated with lower and more stable MAPE. To reduce redundancy, we synthesize the GA evidence into sensitivity patterns and provide practical ranges supported by the boxplots.

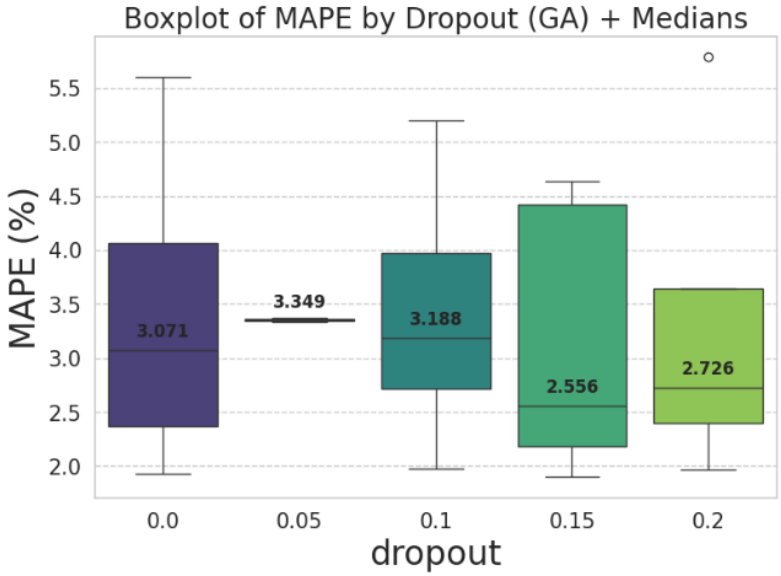

Fig. 11. Boxplot of MAPE for different dropout levels using GA

The dropout results indicate that GA favors intermediate regularization. The lowest median MAPE occurs at dropout = 0.15 (≈ 2.556%), while dropout = 0.2 also yields a relatively low median (≈ 2.726%), suggesting that moderate-to-high regularization can be beneficial under GA-guided exploration. In contrast, lower dropout values produce higher medians: 0.0 (≈ 3.071%), 0.05 (≈ 3.349%), and 0.1 (≈ 3.188%) and show larger dispersion, indicating reduced stability across trials. Overall, the GA results support dropout = 0.15 as the most reliable choice in this setting.

For model width, GA identifies a stronger preference for moderately larger capacity than TPE. The best median performance is achieved at d_model = 64 (≈ 2.559%), indicating high accuracy and robustness under GA optimization. Although d_model = 32 yields a relatively low median (≈ 2.970%), its larger dispersion suggests less stable trial outcomes. In contrast, other values lead to apparent degradation: d_model = 16 (≈ 3.486%), 24 (≈ 4.636%), and 128 (≈ 4.326%). Notably, d_model = 24 performs worst, reinforcing that specific "intermediate" values can still be suboptimal depending on training dynamics and the overall configuration. These results suggest that, under GA, d_model = 64 provides the best trade-off between accuracy and robustness.

Encoder depth exhibits a non-monotonic pattern under GA. The lowest median MAPE is achieved at e_layers = 4 (≈ 2.559%), while e_layers = 6 also yields a low median (≈ 2.766%) but with

slightly larger dispersion, indicating reduced stability compared with the 4-layer setting. By contrast, e_layers = 2 (≈ 3.455%), 3 (≈ 4.492%), and 5 (≈ 3.524%) perform worse, and e_layers = 3 shows both the highest median and substantial variability. Overall, GA results suggest that e_layers = 4 is the most robust depth choice for this protocol, while deeper stacks may occasionally achieve competitive outcomes but are less consistent.

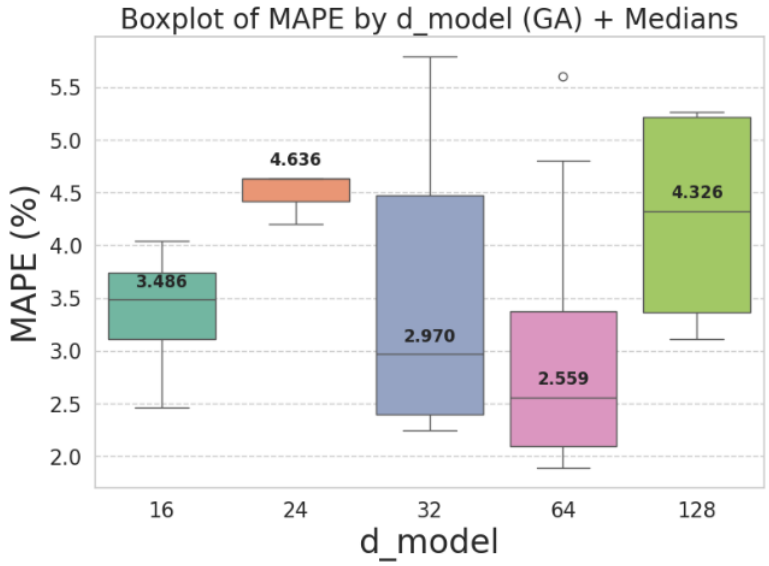

Fig. 12. Boxplot of MAPE for different d_model levels using GA

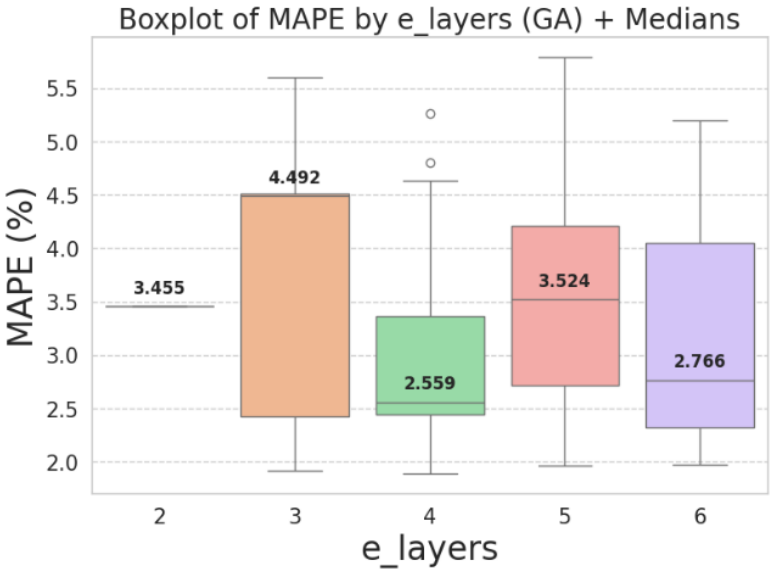

Fig. 13. Boxplot of MAPE for different e_layers levels using GA

Learning rate remains a dominant factor affecting both performance and stability. The best medians occur at small learning rates: 0.0003 (≈ 2.556%) and 0.0005 (≈ 2.597%), indicating stable convergence under GA-selected configurations. Increasing the learning rate to 0.001 yields a higher median (≈ 3.369%) and greater dispersion, suggesting reduced stability. Larger learning rates further degrade performance: 0.002 (≈ 4.340%) and 0.003 (≈ 3.823%). These findings indicate that GA consistently benefits from a learning rate of 0.0003–0.0005 in the considered STLF setting.

Taken together, Fig. 11, Fig. 12, Fig. 13, Fig. 14 show that GA tends to select hyperparameters that balance regularization and capacity while avoiding unstable optimization regimes. The most consistently favorable region under GA is characterized by dropout ≈ 0.15, d_model = 64, e_layers = 4, and learning_rate = 0.0003–0.0005. Importantly, GA's advantage should not be interpreted solely

in terms of best-case outcomes; the trial distributions also show that GA can locate regions that deliver low median error with relatively controlled dispersion, which is crucial for robust deployment.

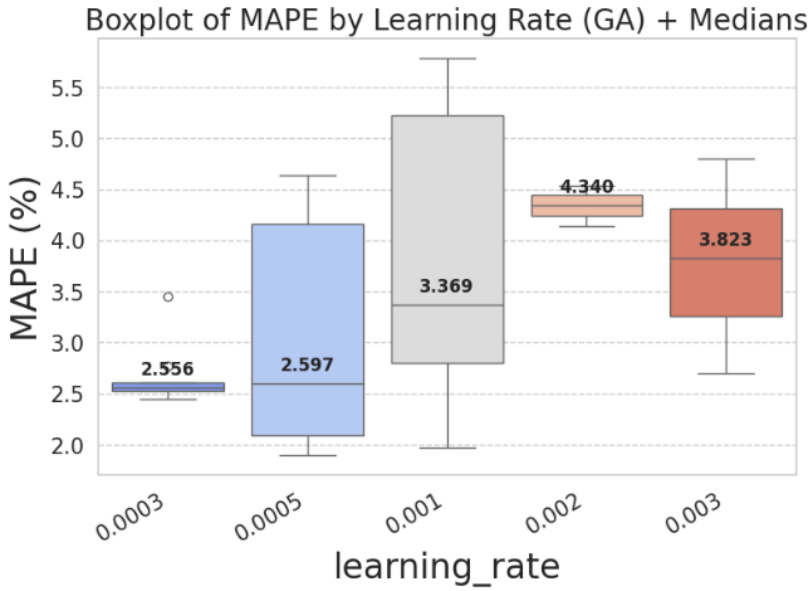

Fig. 14. Boxplot of MAPE for different learning_rate levels using GA

4.4. Comparative Evaluation of RS–TPE–GA

Fig. 15 compares the best (minimum) MAPE achieved by TimeMixer under the default configuration and three hyperparameter-tuning strategies (RS, TPE, and GA).

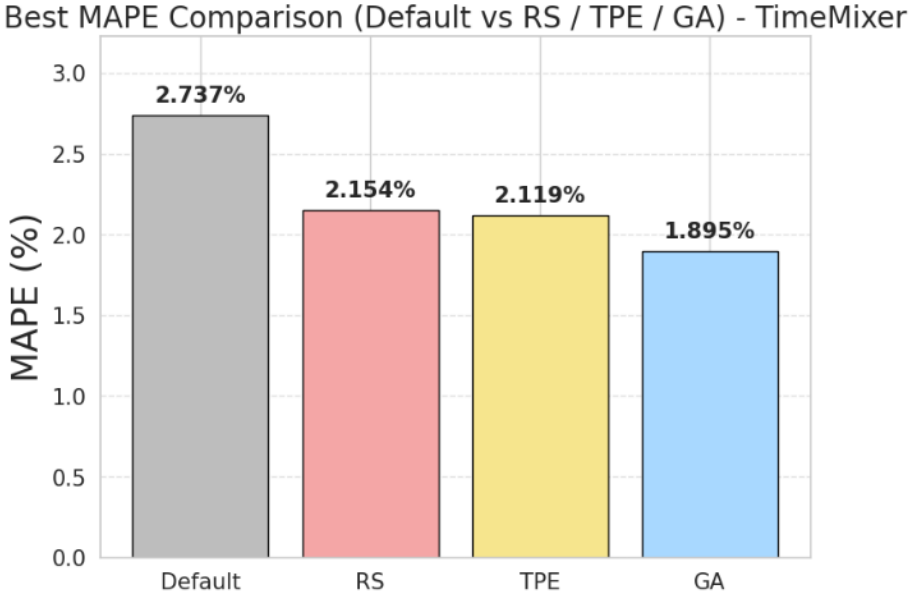

Fig. 15. Best MAPE comparison for TimeMixer (Default vs. RS/TPE/GA)

All tuning strategies deliver clear improvements over the default setting, confirming that TimeMixer is highly sensitive to hyperparameter selection and benefits from systematic optimization. Quantitatively, the default model attains a best MAPE of 2.737%, while RS and TPE reduce it to 2.154% and 2.119%, corresponding to absolute reductions of 0.583 and 0.618 percentage points, respectively; GA achieves the lowest best MAPE of 1.895%, yielding the most considerable absolute decrease of 0.842 percentage points relative to the baseline. In relative terms, these improvements correspond to approximately 21.3% (RS), 22.6% (TPE), and 30.8% (GA) compared with the default best MAPE. Although RS and TPE produce similar best-case outcomes, TPE is slightly better than

RS (2.119% vs. 2.154%), suggesting that model-based Bayesian optimization can marginally outperform uniform exploration under the same trial budget. GA provides the strongest best-case result, outperforming RS and TPE by 0.259 and 0.224 percentage points, respectively (equivalent to additional relative gains of about 12.0% vs. RS and 10.6% vs. TPE when measured against their best MAPE values). From a practical perspective, even sub-percentage reductions in MAPE can be meaningful in STLF, as minor improvements repeated across many time steps and operational cycles can translate into better scheduling and reserve-related decisions. However, to address reviewer concerns that "GA is best" should not be concluded from best-case performance alone, the comparison should be interpreted jointly with the trial-level distributions discussed in Section 4.1, Section 4.2, Section 4.3, where robustness is reflected by dispersion and consistency across trials; therefore, conclusions about optimizer effectiveness should consider both accuracy and stability. Finally, to support the reliability of the observed differences on the held-out evaluation horizon, a paired significance assessment (Diebold–Mariano or Wilcoxon signed-rank test on per-step forecast errors) can be reported at a conventional significance level (α=0.05) to verify whether GA's advantage over TPE/RS is statistically significant rather than driven by isolated points.

5. Conclusion

This study shows that TimeMixer performance in short-term load forecasting is strongly influenced by a small set of core hyperparameters, and systematic optimization is therefore essential for reliable accuracy. Under an identical evaluation budget, the best-case results indicate the ranking GA > TPE > RS > Default, with GA achieving the lowest best MAPE (1.895%) compared with 2.119% (TPE), 2.154% (RS), and 2.737% (default). Across the sensitivity analyses, learning rate and capacity-related settings (d_model and e_layers) drive the most significant changes in error levels and stability. At the same time, dropout primarily affects the accuracy–robustness trade-off via regularization. As a practical rule of thumb within the considered dataset and protocol, practitioners should prioritize tuning learning_rate first to avoid unstable convergence, then tune d_model and e_layers to balance underfitting and over-parameterization, and finally adjust dropout to improve robustness once a stable capacity regime is found.

Several limitations should be noted. First, the conclusions are dataset- and protocol-dependent, and the recommended hyperparameter regions may not transfer directly to other load systems or different forecasting horizons. Second, the evaluation reflects a constrained retraining and trial-budget setting; different training-window lengths or larger budgets may alter optimizer behavior and sensitivity rankings. Third, this work does not yet quantify seasonal transferability or concept drift effects. whether hyperparameters tuned in one season remain optimal in another.

Future work will address these points by (i) validating findings across multiple datasets and additional modern baselines under the same held-out testing protocol, (ii) conducting season-aware experiments (separate tuning and testing in summer vs. winter, plus cross-season transfer) to evaluate drift and robustness of the selected hyperparameters explicitly, and (iii) studying interaction effects among hyperparameters to derive more general tuning guidelines beyond marginal sensitivity.

Author Contribution: All authors contributed equally to the main contributor to this paper. All authors read and approved the final paper.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] U. M. Sirisha, M. C. Belavagi and G. Attigeri, "Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison," *IEEE Access*, vol. 10, pp. 124715-124727, 2022, https://doi.org/10.1109/ACCESS.2022.3224938.

[2] U. Samal and A. Kumar, "Enhancing Software Reliability Forecasting Through a Hybrid ARIMA-ANN Model," *Arabian Journal for Science and Engineering*, vol. 49, no. 5, pp. 7571-7584, 2024, https://doi.org/10.1007/s13369-023-08486-1.

[3] J. A. C. Dias *et al*., "Enhanced Carbon Flux Forecasting via STL Decomposition and Hybrid ARIMA-ES-LSTM Model in Amazon Forest," *IEEE Access*, vol. 13, pp. 84713-84726, 2025, https://doi.org/10.1109/ACCESS.2025.3561166.

[4] Y. -C. Jin *et al*., "Models for COVID-19 Data Prediction Based on Improved LSTM-ARIMA Algorithms," *IEEE Access*, vol. 12, pp. 3981-3991, 2024, https://doi.org/10.1109/ACCESS.2023.3347403.

[5] S. Karamolegkos and D. E. Koulouriotis, "Advancing short-term load forecasting with decomposed Fourier ARIMA: A case study on the Greek energy market," *Energy*, vol. 325, p. 135854, 2025, https://doi.org/10.1016/j.energy.2025.135854.

[6] S. Cantillo-Luna, R. Moreno-Chuquen and J. A. Lopez Sotelo, "Intra-day Electricity Price Forecasting Based on a Time2Vec-LSTM Neural Network Model," *2023 IEEE Colombian Conference on Applications of Computational Intelligence (ColCACI)*, pp. 1-6, 2023, https://doi.org/10.1109/ColCACI59285.2023.10225803.

[7] S. Chen, R. Lin and W. Zeng, "Short-Term Load Forecasting Method Based on ARIMA and LSTM," *2022 IEEE 22nd International Conference on Communication Technology (ICCT)*, pp. 1913-1917, 2022, https://doi.org/10.1109/ICCT56141.2022.10073051.

[8] M. Abdurohman and A. G. Putrada, "Forecasting Model for Lighting Electricity Load with a Limited Dataset using XGBoost," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, vol. 8, no. 2, pp. 571-580, 2023, https://doi.org/10.22219/kinetik.v8i2.1687.

[9] U. I. Lezama Lope, A. Benavides-Vázquez, G. Santamaría-Bonfil and R. Z. Ríos-Mercado, "Fast and Efficient Very Short-Term Load Forecasting Using Analogue and Moving Average Tools," *IEEE Latin America Transactions*, vol. 21, no. 9, pp. 1015-1021, 2023, https://doi.org/10.1109/TLA.2023.10251808.

[10] A. P. Wibawa, A. B. P. Utama, H. Elmunsyah, U. Pujianto, F. A. Dwiyanto, and L. Hernandez, "Time-series analysis with smoothed Convolutional Neural Network," *Journal of Big Data*, vol. 9, no. 1, 2022, https://doi.org/10.1186/s40537-022-00599-y.

[11] A. Jarraya *et al*., "On-Site temperature and irradiance forecast tuning for improved load prediction in buildings," *Energy and Buildings*, vol. 337, p. 115642, 2025, https://doi.org/10.1016/j.enbuild.2025.115642.

[12] R. Bitit, A. Derhab, M. Guerroumi, and F. A. Khan, "DDoS attack forecasting based on online multiple change points detection and time series analysis," *Multimedia Tools and Applications*, vol. 83, no. 18, pp. 53655–53685, 2024, https://doi.org/10.1007/s11042-023-17637-3.

[13] D. Kiruthiga and V. Manikandan, "Levy flight-particle swarm optimization-assisted BiLSTM + dropout deep learning model for short-term load forecasting," *Neural Computing and Applications*, vol. 35, no. 3, pp. 2679-2700, 2023, https://doi.org/10.1007/s00521-022-07751-y.

[14] S. S. Subbiah and J. Chinnappan, "Deep learning based short term load forecasting with hybrid feature selection," *Electric Power Systems Research*, vol. 210, p. 108065, 2022, https://doi.org/10.1016/j.epsr.2022.108065.

[15] A. Irankhah, M. H. Yaghmaee, and S. Ershadi-Nasab, "Optimized short-term load forecasting in residential buildings based on deep learning methods for different time horizons," *Journal of Building Engineering*, vol. 84, p. 108505, 2024, https://doi.org/10.1016/j.jobe.2024.108505.

[16] M. Choubey, R. K. Chaurasiya, and J. S. Yadav, "Predicting Electrical Load Demands Using Neural Prophet-Based Forecasting Model," *SN Computer Science*, vol. 6, no. 1, p. 46, 2025, https://doi.org/10.1007/s42979-024-03587-6.

[17] W. Li, Q. Shi, M. Sibtain, D. Li and D. E. Mbanze, "A Hybrid Forecasting Model for Short-Term Power Load Based on Sample Entropy, Two-Phase Decomposition and Whale Algorithm Optimized Support Vector Regression," *IEEE Access*, vol. 8, pp. 166907-166921, 2020, https://doi.org/10.1109/ACCESS.2020.3023143.

[18] Z. Zhang, W. -C. Hong and J. Li, "Electric Load Forecasting by Hybrid Self-Recurrent Support Vector Regression Model With Variational Mode Decomposition and Improved Cuckoo Search Algorithm," *IEEE Access*, vol. 8, pp. 14642-14658, 2020, https://doi.org/10.1109/ACCESS.2020.2966712.

[19] H. She, S. Li and L. Wang, "The Short-Term Load Forecasting Method Based on Particle Swarm Optimization Support Vector Machine," *2024 International Conference on New Power System and Power Electronics (NPSPE)*, pp. 182-186, 2024, https://doi.org/10.1109/NPSPE62515.2024.00037.

[20] N. -C. Yang and K. -L. Sung, "Non-Intrusive Load Classification and Recognition Using Soft-Voting Ensemble Learning Algorithm With Decision Tree, K-Nearest Neighbor Algorithm and Multilayer Perceptron," *IEEE Access*, vol. 11, pp. 94506-94520, 2023, https://doi.org/10.1109/ACCESS.2023.3311641.

[21] D. Bairrão, D. Ramos, P. Faria, and Z. Vale, "Improving Load Forecasting with Data Partitioning: A K-Means Approach to an Office Building," *IFAC-PapersOnLine*, vol. 58, no. 13, pp. 314-319, 2024, https://doi.org/10.1016/j.ifacol.2024.07.501.

[22] M. Y. Junior, R. Z. Freire, L. O. Seman, S. F. Stefenon, V. C. Mariani, and L. dos S. Coelho, "Optimized hybrid ensemble learning approaches applied to very short-term load forecasting," *International Journal of Electrical Power & Energy Systems*, vol. 155, p. 109579, 2024, https://doi.org/10.1016/j.ijepes.2023.109579.

[23] R. Banik and A. Biswas, "Enhanced renewable power and load forecasting using RF-XGBoost stacked ensemble," *Electrical Engineering*, vol. 106, no. 4, pp. 4947–4967, 2024, https://doi.org/10.1007/s00202-024-02273-3.

[24] M. Dostmohammadi, M. Z. Pedram, S. Hoseinzadeh, and D. A. Garcia, "A GA-stacking ensemble approach for forecasting energy consumption in a smart household: A comparative study of ensemble methods," *Journal of Environmental Management*, vol. 364, p. 121264, 2024, https://doi.org/10.1016/j.jenvman.2024.121264.

[25] R. R. Onteru and V. Sandeep, "An intelligent model for efficient load forecasting and sustainable energy management in sustainable microgrids," *Discover Sustainability*, vol. 5, no. 1, p. 170, 2024, https://doi.org/10.1007/s43621-024-00356-6.

[26] A. Wang, Q. Yu, J. Wang, X. Yu, Z. Wang, and Z. Hu, "Electric Load Forecasting Based on Deep Ensemble Learning," *Applied Sciences*, vol. 13, no. 17, p. 9706, 2023, https://doi.org/10.3390/app13179706.

[27] V. Veeramsetty, K. R. Reddy, M. Santhosh, A. Mohnot, and G. Singal, "Short-term electric power load forecasting using random forest and gated recurrent unit," *Electrical Engineering*, vol. 104, no. 1, pp. 307-329, 2022, https://doi.org/10.1007/s00202-021-01376-5.

[28] M. Mohamed, F. E. Mahmood, M. A. Abd, M. Rezkallah, A. Hamadi and A. Chandra, "Load Demand Forecasting Using eXtreme Gradient Boosting (XGboost)," *2023 IEEE Industry Applications Society Annual Meeting (IAS)*, pp. 1-7, 2023, https://doi.org/10.1109/IAS54024.2023.10406613.

[29] T. Kavzoglu and A. Teke, "Predictive Performances of Ensemble Machine Learning Algorithms in Landslide Susceptibility Mapping Using Random Forest, Extreme Gradient Boosting (XGBoost) and Natural Gradient Boosting (NGBoost)," *Arabian Journal for Science and Engineering*, vol. 47, no. 6, pp. 7367-7385, 2022, https://doi.org/10.1007/s13369-022-06560-8.

[30] Z. Qinghe, X. Wen, H. Boyan, W. Jong, and F. Junlong, "Optimised extreme gradient boosting model for short-term electric load demand forecasting of regional grid system," *Scientific Reports*, vol. 12, no. 1, pp. 1–12, 2022, https://doi.org/10.1038/s41598-022-22024-3.

[31] W. Niu, X. Song, H. Wang and J. Chen, "Forecasting the load flow of Engine Driven Pump based on Light Gradient Boosting Machine Model," *2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*, pp. 771-774, 2021, https://doi.org/10.1109/CEI52496.2021.9574553.

[32] T. Zhang, Y. Huang, H. Liao, and Y. Liang, "A hybrid electric vehicle load classification and forecasting approach based on GBDT algorithm and temporal convolutional network," *Applied Energy*, vol. 351, p. 121768, 2023, https://doi.org/10.1016/j.apenergy.2023.121768.

[33] M. M. Hasan, N. El-Tazi, R. Moawad and A. H. B. Eissa, "TSB-Forecast: A Short-Term Load Forecasting Model in Smart Cities for Integrating Time Series Embeddings and Large Language Models," *IEEE Access*, vol. 13, pp. 141694-141716, 2025, https://doi.org/10.1109/ACCESS.2025.3597421.

[34] S. Lei *et al*., "Net Load Segmented Forecasting Method For Data Center Based on GS-LightGBM Model," *2023 IEEE IAS Global Conference on Renewable Energy and Hydrogen Technologies (GlobConHT)*, pp. 1-6, 2023, https://doi.org/10.1109/GlobConHT56829.2023.10087415.

[35] Z. Fang, J. Zhan, J. Cao, L. Gan and H. Wang, "Research on Short-Term and Medium-Term Power Load Forecasting Based on STL-LightGBM," *2022 2nd International Conference on Electrical Engineering and Control Science (IC2ECS)*, pp. 1047-1051, 2022, https://doi.org/10.1109/IC2ECS57645.2022.10088145.

[36] Y. Miao, J. Zhu, H. Dong, Z. Chen, S. Li and X. Wen, "Short-term Load Forecasting Based on Echo State Network and LightGBM," *2023 IEEE International Conference on Predictive Control of Electrical Drives and Power Electronics (PRECEDE)*, pp. 1-6, 2023, https://doi.org/10.1109/PRECEDE57319.2023.10174609.

[37] X. Yao, X. Fu and C. Zong, "Short-Term Load Forecasting Method Based on Feature Preference Strategy and LightGBM-XGboost," *IEEE Access*, vol. 10, pp. 75257-75268, 2022, https://doi.org/10.1109/ACCESS.2022.3192011.

[38] X. Liang, Y. Feng, J. Jiang, W. Wang, X. Liu and Z. Gong, "Short-term Load Forecasting of a Technology Park Based on a LightGBM-LSTM Fusion Algorithm," *2022 IEEE 5th International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, pp. 151-155, 2022, https://doi.org/10.1109/AUTEEE56487.2022.9994355.

[39] S. Lei *et al*., "A Short-term Net Load Forecasting Method Based on Two-stage Feature Selection and LightGBM with Hyperparameter Auto-Tuning," *2023 IEEE/IAS 59th Industrial and Commercial Power Systems Technical Conference (I&CPS)*, pp. 1-6, 2023, https://doi.org/10.1109/ICPS57144.2023.10142095.

[40] E. Dolgintseva, H. Wu, O. Petrosian, A. Zhadan, A. Allakhverdyan, and A. Martemyanov, "Comparison of multi-step forecasting methods for renewable energy," *Energy Systems*, vol. 17, pp. 95–126, 2026, https://doi.org/10.1007/s12667-024-00656-w.

[41] M. Abumohsen, A. Y. Owda, and M. Owda, "Electrical Load Forecasting Using LSTM, GRU, and RNN Algorithms," *Energies*, vol. 16, no. 5, p. 2283, 2023, https://doi.org/10.3390/en16052283.

[42] A. Ajitha, M. Goel, M. Assudani, S. Radhika, and S. Goel, "Design and development of Residential Sector Load Prediction model during COVID-19 Pandemic using LSTM-based RNN," *Electric Power Systems Research*, vol. 212, p. 108635, 2022, https://doi.org/10.1016/j.epsr.2022.108635.

[43] A. K. Mishra, P. Mishra, and H. D. Mathur, "A deep learning assisted adaptive nonlinear deloading strategy for wind turbine generator integrated with an interconnected power system for enhanced load frequency control," *Electric Power Systems Research*, vol. 214, p. 108960, 2023, https://doi.org/10.1016/j.epsr.2022.108960.

[44] A. O. Aseeri, "Effective RNN-Based Forecasting Methodology Design for Improving Short-Term Power Load Forecasts: Application to Large-Scale Power-Grid Time Series," *Journal of Computational Science*, vol. 68, p. 101984, 2023, https://doi.org/10.1016/j.jocs.2023.101984.

[45] N. A. Nguyen, T. D. Dang, E. Verdú, and V. Kumar Solanki, "Short-term forecasting electricity load by long short-term memory and reinforcement learning for optimization of hyper-parameters," *Evolutionary Intelligence*, vol. 16, no. 5, pp. 1729-1746, 2023, https://doi.org/10.1007/s12065-023-00869-5.

[46] S. M. Praminta, H. Aji and E. Y. Ikhsan, "Emerging and Evaluating Long Short Term Memory (LSTM) Network for Load Forecast in Java Bali System," *2023 4th International Conference on High Voltage Engineering and Power Systems (ICHVEPS)*, pp. 797-801, 2023, https://doi.org/10.1109/ICHVEPS58902.2023.10257331.

[47] Z. Sheng, Z. An, H. Wang, G. Chen, and K. Tian, "Residual LSTM-based short-term load forecasting," *Applied Soft Computing*, vol. 144, p. 110461, 2023, https://doi.org/10.1016/j.asoc.2023.110461.

[48] H. Kuang, Q. Guo, S. Li and H. Zhong, "Short-term Power Load Forecasting Method in Rural Areas Based on CNN-LSTM," *2021 IEEE 4th International Electrical and Energy Conference (CIEEC)*, pp. 1-5, 2021, https://doi.org/10.1109/CIEEC50170.2021.9510777.

[49] F. Liu and C. Liang, "Short-term power load forecasting based on AC-BiLSTM model," *Energy Reports*, vol. 11, pp. 1570–1579, 2024, https://doi.org/10.1016/j.egyr.2024.01.026.

[50] M. Taheri, M. Abedini and F. Aminifar, "A Novel Centralized Load Shedding Approach to Assess Short-Term Voltage Stability: A Model-Free Using Time Series Forecasting," *IEEE Transactions on Power Delivery*, vol. 38, no. 5, pp. 3076-3083, 2023, https://doi.org/10.1109/TPWRD.2023.3266265.

[51] M. Alhussein, K. Aurangzeb and S. I. Haider, "Hybrid CNN-LSTM Model for Short-Term Individual Household Load Forecasting," *IEEE Access*, vol. 8, pp. 180544-180557, 2020, https://doi.org/10.1109/ACCESS.2020.3028281.

[52] S. Luo, Z. Ni, X. Zhu, P. Xia, and H. Wu, "A Novel Methanol Futures Price Prediction Method Based on Multicycle CNN-GRU and Attention Mechanism," *Arabian Journal for Science and Engineering*, vol. 48, no. 2, pp. 1487–1501, 2023, https://doi.org/10.1007/s13369-022-06902-6.

[53] I. Jrhilifa, H. Ouadi, A. Jilbab, and N. Mounir, "Forecasting smart home electricity consumption using VMD-Bi-GRU," *Energy Efficiency*, vol. 17, no. 4, p. 35, 2024, https://doi.org/10.1007/s12053-024-10205-0.

[54] Z. Chen, T. Jin, X. Zheng, Y. Liu, Z. Zhuang, and M. A. Mohamed, "An innovative method-based CEEMDAN–IGWO–GRU hybrid algorithm for short-term load forecasting," *Electrical Engineering*, vol. 104, no. 5, pp. 3137-3156, 2022, https://doi.org/10.1007/s00202-022-01533-4.

[55] Y. Chen, C. Lin, Y. Zhang, J. Liu, and D. Yu, "Day-ahead load forecast based on Conv2D-GRU_SC aimed to adapt to steep changes in load," *Energy*, vol. 302, p. 131814, 2024, https://doi.org/10.1016/j.energy.2024.131814.

[56] H. Eskandari, M. Imani, and M. P. Moghaddam, "Best-tree wavelet packet transform bidirectional GRU for short-term load forecasting," *The Journal of Supercomputing*, vol. 79, no. 12, pp. 13545-13577, 2023, https://doi.org/10.1007/s11227-023-05193-4.

[57] Y. Li, Y. Ye, Y. Xu, L. Li, X. Chen, and J. Huang, "Two-stage forecasting of TCN-GRU short-term load considering error compensation and real-time decomposition," *Earth Science Informatics*, vol. 17, pp. 5347-5357, 2024, https://doi.org/10.1007/s12145-024-01456-7.

[58] R. Liu, J. Shi, G. Sun, S. Lin, and F. Li, "A Short-term net load hybrid forecasting method based on VW-KA and QR-CNN-GRU," *Electric Power Systems Research*, vol. 232, p. 110384, 2024, https://doi.org/10.1016/j.epsr.2024.110384.

[59] H. Hua, M. Liu, Y. Li, S. Deng, and Q. Wang, "An ensemble framework for short-term load forecasting based on parallel CNN and GRU with improved ResNet," *Electric Power Systems Research*, vol. 216, p. 109057, 2023, https://doi.org/10.1016/j.epsr.2022.109057.

[60] X. Bu, Q. Wu, B. Zhou, and C. Li, "Hybrid short-term load forecasting using CGAN with CNN and semi-supervised regression," *Applied Energy*, vol. 338, p. 120920, 2023, https://doi.org/10.1016/j.apenergy.2023.120920.

[61] Q. Lu and M. Li, "VMD and CNN-Based Classification Model for Infrasound Signal," *Archives of Acoustics*, vol. 48, no. 3, pp. 403-412, 2023, https://acoustics.ippt.pan.pl/index.php/aa/article/view/3723.

[62] C. Wang, X. Li, Y. Shi, W. Jiang, Q. Song, X. Li, "Load forecasting method based on CNN and extended LSTM," *Energy Reports*, vol. 12, pp. 2452-2461, 2024, https://doi.org/10.1016/j.egyr.2024.07.030.

[63] L. Hu, J. Wang and D. Juan, "Medium and Long-term Load Forecasting Based on EMD-CNN-BLSTM Hybrid Deep Learning Model," *2023 35th Chinese Control and Decision Conference (CCDC)*, pp. 640-645, 2023, https://doi.org/10.1109/CCDC58219.2023.10326627.

[64] D. Tan, Z. Tang, F. Zhou and Y. Xie, "A Novel Hybrid Model Based on EMD-Improved TCN-Improved TST for Short-Term Railway Traction Load Forecasting," *IEEE Transactions on Transportation Electrification*, vol. 11, no. 2, pp. 6418-6427, 2025, https://doi.org/10.1109/TTE.2024.3509855.

[65] J. Wen, Y. Peng, W. Zhang, X. Huang and Z. Wang, "Short-term power load forecasting based on TCN-LSTM model," *2024 IEEE 6th International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*, pp. 734-738, 2024, https://doi.org/10.1109/ICCASIT62299.2024.10827868.

[66] Z. Xu *et al*., "PhaCIA-TCNs: Short-Term Load Forecasting Using Temporal Convolutional Networks With Parallel Hybrid Activated Convolution and Input Attention," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 1, pp. 427-438, 2024, https://doi.org/10.1109/TNSE.2023.3300744.

[67] Y. Feng, J. Zhu, P. Qiu, X. Zhang, and C. Shuai, "Short-term Power Load Forecasting Based on TCN-BiLSTM-Attention and Multi-feature Fusion," *Arabian Journal for Science and Engineering*, vol. 50, pp. 5475-5486, 2024, https://doi.org/10.1007/s13369-024-09351-5.

[68] S. Cen and C. G. Lim, "Multi-Task Learning of the PatchTCN-TST Model for Short-Term Multi-Load Energy Forecasting Considering Indoor Environments in a Smart Building," *IEEE Access*, vol. 12, pp. 19553-19568, 2024, https://doi.org/10.1109/ACCESS.2024.3355448.

[69] A. Binte Habib, M. G. R. Alam and M. Z. Uddin, "AUNET (Attention-Based Unified Network): Leveraging Attention-Based N-BEATS for Enhanced Univariate Time Series Forecasting," *IEEE Access*, vol. 13, pp. 95184-95217, 2025, https://doi.org/10.1109/ACCESS.2025.3574459.

[70] A. Motavali, K. -C. Yow, N. Hansmeier and T. -C. Chao, "DSA-BEATS: Dual Self-Attention N-BEATS Model for Forecasting COVID-19 Hospitalization," *IEEE Access*, vol. 11, pp. 137352-137365, 2023, https://doi.org/10.1109/ACCESS.2023.3318931.

[71] P. Nedić, I. Djurović, M. Ćalasan, S. Kovačević, and K. Pavlović, "Electrical energy load forecasting using a hybrid N-BEATS - CNN Approach: Case study Montenegro," *Electric Power Systems Research*, vol. 247, p. 135486, 2025, https://doi.org/10.1016/j.epsr.2025.111749.

[72] F. C. de L. Duarte, P. S. G. de M. Neto, and P. R. A. Firmino, "A hybrid recursive direct system for multi-step mortality rate forecasting," *The Journal of Supercomputing*, vol. 80, no. 13, pp. 18430-18463, 2024, https://doi.org/10.1007/s11227-024-06182-x.

[73] B. Wu, J. Xiao, S. Wang, Z. Zhang, and R. Wen, "Enhancing short-term net load forecasting with additive neural decomposition and Weibull Attention," *Energy*, vol. 322, p. 135486, 2025, https://doi.org/10.1016/j.energy.2025.135486.

[74] A. Faustine, N. J. Nunes and L. Pereira, "Efficiency Through Simplicity: MLP-Based Approach for Net-Load Forecasting With Uncertainty Estimates in Low-Voltage Distribution Networks," *IEEE Transactions on Power Systems*, vol. 40, no. 1, pp. 46-56, 2025, https://doi.org/10.1109/TPWRS.2024.3400123.

[75] J. Zhang, X. Fu, J. Fu and S. Qu, "Multi-time resolution load unified prediction method based on improved TimeMixer," *2024 IEEE 6th International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*, pp. 1171-1176, 2024, https://doi.org/10.1109/ICCASIT62299.2024.10828070.

[76] D. Wang, T. Wang, and B. Zhang, "Fault diagnosis in rolling element bearings based on timemixer and prototypical network," *2025 Prognostics and System Health Management Conference (PHM)*, pp. 2-6, 2025, https://doi.org/10.1049/icp.2025.2349.

[77] B. Jeong and Y. -S. Jeong, "ARAScaler: Adaptive Resource Autoscaling Scheme Using ETimeMixer for Efficient Cloud-Native Computing," *IEEE Transactions on Services Computing*, vol. 18, no. 1, pp. 72-84, 2025, https://doi.org/10.1109/TSC.2024.3522815.

[78] S. Wang *et al*., "A general time series pattern machine for universal predictive analysis," *arXiv*, 2025, https://doi.org/10.48550/arXiv.2410.16032.

[79] D. Wu, Y. Yang, D. Qiu, and G. Li, "A Dual-Memory Enhanced Architecture for Time Series Forecasting," *Frontiers in Computing and Intelligent Systems (FCIS)*, vol. 12, no. 1, pp. 35-42, 2025, https://doi.org/10.54097/g0504662.

[80] D. Navon and A. M. Bronstein, "Random Search Hyper-Parameter Tuning: Expected Improvement Estimation and the Corresponding Lower Bound," *arXiv*, 2022, https://doi.org/10.48550/arXiv.2208.08170.

[81] T. N. Tran and T. A. Nguyen, "Hyperparameter Optimization for Deep Learning Modeling in Short-Term Load Forecasting," *International Journal of Electrical and Computer Engineering Systems*, vol. 16, no. 6, pp. 443–450, 2025, https://doi.org/10.32985/ijeces.16.6.2.

[82] H. Hu and B. Zheng, "Short-term electricity load forecasting based on CEEMDAN-FE-BiGRU-Attention model," *International Journal of Low-Carbon Technologies*, vol. 19, pp. 988–995, 2024, https://doi.org/10.1093/ijlct/ctae040.

[83] M. Sakib, T. Siddiqui, S. Mustajab, R. M. Alotaibi, N. M. Alshareef, and M. Z. Khan, "An ensemble deep learning framework for energy demand forecasting using genetic algorithm-based feature selection," *PLOS ONE*, vol. 20, no. 1, p. e0310465, 2025, https://doi.org/10.1371/journal.pone.0310465.

[84] Y. Da Jhong, C. S. Chen, B. C. Jhong, C. H. Tsai, and S. Y. Yang, "Optimization of LSTM Parameters for Flash Flood Forecasting Using Genetic Algorithm," *Water Resources Management*, vol. 38, no. 3, pp. 1141-1164, 2024, https://doi.org/10.1007/s11269-023-03713-8.