International Journal of Robotics and Control Systems
Vol. 6, No. 1, 2025, pp. 741-761
ISSN 2775-2658
http://pubs2.ascee.org/index.php/ijrcs

A Study on the Impact of Hyperparameters on the Temporal Convolutional Network Model for Electric 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 06, 2026
Revised February 14, 2026
Accepted March 24, 2026

Keywords
Electric Load Forecasting;
Temporal Convolutional Network;
Hyperparameter Impact;
Time Series Forecasting

ABSTRACT

Electric load forecasting is essential for reliable power-system planning and operation. Temporal Convolutional Networks (TCNs) have gained attention for time-series prediction due to causal and dilated convolutions with residual connections, which help capture long-range temporal dependencies. However, TCN accuracy is highly sensitive to hyperparameter settings, and systematic evidence on which hyperparameters matter most for load forecasting remains limited. This study investigates major architectural and training hyperparameters in a TCN-based short-term load forecasting pipeline using 30-minute electricity-load data from New South Wales (NSW), Australia (2015-2021). The task is one-day-ahead forecasting with a 48-step horizon, and performance is evaluated primarily using Mean Absolute Percentage Error (MAPE) under a controlled experimental protocol with a fixed reference configuration. Results show that training hyperparameters-especially the learning rate-have the most substantial impact on accuracy and stability: a learning rate of 9×10^{-4}achieves the lowest median MAPE of 2.375%, improving the reference configuration's median MAPE of 3.212% by approximately 26% (relative). Architectural choices such as the dilation schedule and input window length have moderate effects, while encoder-decoder capacity parameters are comparatively less sensitive within the tested ranges. These findings provide practical guidance on prioritizing hyperparameters when configuring TCNs for short-term load forecasting, balancing forecasting accuracy and computational efficiency.

1. Introduction

Short-term load forecasting (STLF) is a fundamental task in 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 mitigation, and market participation, while reducing operational costs and improving system reliability. Conversely, forecasting errors may lead to unnecessary reserve procurement, inefficient dispatch decisions, or increased risk of operating-limit violations. For these reasons, improving STLF accuracy remains an essential research and engineering objective.

Load forecasting methods have evolved from classical statistical modeling to machine-learning and deep-learning approaches. Traditional statistical models-such as AR/MA/ARMA, ARIMA/SARIMA [1]-[6], exponential smoothing [7]-[11], and linear regression [12]-[15] are attractive due to their simplicity and interpretability, but they can be limited by assumptions (linearity and stationary seasonal patterns) that are often violated in real load series with nonlinear behaviors, regime shifts, and complex temporal interactions. To better capture nonlinear relationships, machine-learning (ML) models have been widely adopted in practical STLF applications, including Support Vector Regression (SVR) [16]-[18], k-Nearest Neighbors (kNN) [19], [20], Random Forests [21], [22], and modern boosting frameworks such as LightGBM [23]-[27]. These methods can perform strongly with appropriate feature engineering and exogenous variables (weather and calendar effects), yet performance often depends on preprocessing choices and the extent to which temporal structure is encoded in the input.

With the growing availability of high-resolution load measurements, deep-learning (DL) models have become prominent because they can learn representations directly from data. Common DL architectures for STLF include convolutional and recurrent models such as CNN [28]-[32], RNN [33]-[37], LSTM [38]-[45], and GRU [46]-[53], while Transformer-based models [54]-[62] further enhance long-range dependency modeling through attention mechanisms. Despite their expressive power, DL models can be sensitive to architectural and training hyperparameters and may overfit or converge unstably when hyperparameter choices are suboptimal.

In this study, we focus on the Temporal Convolutional Network (TCN) [63]-[66] for STLF. TCN is a convolutional architecture designed for sequence modeling through causal and dilated convolutions combined with residual connections, enabling long-range temporal dependency modeling with stable gradients and efficient parallel computation. Importantly, its effective receptive field can be controlled through architectural hyperparameters (kernel size and dilation settings), while model capacity and complexity are shaped by hidden sizes and the depth of encoder-decoder components. These characteristics make TCN a suitable candidate for studying how hyperparameter choices translate into forecasting accuracy and computational cost under a fixed forecasting task.

This leads to a practical research gap: although automated hyperparameter optimizers (Bayesian optimization and Optuna) are widely used to search for a single best configuration, there remains limited evidence-under a controlled and reproducible protocol-on which TCN hyperparameters most strongly influence STLF accuracy and stability, which hyperparameters have limited impact within reasonable ranges, and how error distributions change across candidate values. In other words, this work aims to provide sensitivity-driven guidance by analyzing how forecasting errors shift in distribution (median dispersion and outliers) under controlled comparisons, rather than reporting only a single optimum configuration. In addition, while recent benchmarks often highlight Transformer-family models, this paper does not claim TCN is universally superior; instead, it provides actionable hyperparameter sensitivity insights for a widely used and computationally efficient sequence model, which can be extended to other architectures in future work.

To address this gap, we systematically evaluate the impact of key TCN hyperparameters [67]-[70] on short-term load forecasting performance using a unified experimental framework, including a consistent data split strategy, forecasting horizon, evaluation protocol, and a predefined hyperparameter search space. The main contributions of this study are as follows:

- A controlled experimental framework for analyzing TCN hyperparameter sensitivity in STLF to ensure fair comparisons and reproducibility.
- A structured investigation of core structural hyperparameters (kernel_size, dilations, encoder_hidden_size, context_size, decoder_hidden_size, decoder_layers) and their influence on forecasting accuracy.
- An assessment of training hyperparameters (learning_rate, batch_size, max_steps) in relation to convergence behavior and predictive performance.

- Practical sensitivity-based insights that help prioritize hyperparameters for configuring TCN models in short-term electric load forecasting based on empirical error patterns observed in the experiments.

2. Method

2.1. TCN Model and Architecture

Temporal Convolutional Network (TCN) formulates short-term load forecasting as a mapping from a fixed-length historical window to a multi-step horizon. Let $y_t \in \mathbb{R}$denote the electric load at time step t. Given an input history of length L, the model predicts the next Hvalues as $\mathbf{x}_t = [y_{t-L+1}, \dots, y_t]$and $\hat{\mathbf{y}}_t = [\hat{y}_{t+1}, \dots, \hat{y}_{t+H}]$. In our experimental setup, Lcorresponds to the context window (context_size/input_size) and $H$is the forecasting horizon.

$$\mathbf{x}_t = [y_{t-L+1}, y_{t-L+2}, \dots, y_t], \hat{\mathbf{y}}_t = [\hat{y}_{t+1}, \hat{y}_{t+2}, \dots, \hat{y}_{t+H}] \quad (1)$$

To prevent information leakage from future observations, TCN employs causal convolution so that the output at time $t$depends only on inputs up to time t. For a 1D causal convolution with kernel size k, the convolution response is expressed as $z_t = \sum_{i=0}^{k-1} w_i \, x_{t-i}$, where $\{w_i\}_{i=0}^{k-1}$are learnable filter weights. The hyperparameter kernel_size directly controls $k$and thus the local temporal span captured by each convolutional filter.

$$z_t = \sum_{i=0}^{k-1} w_i \; x_{t-i} \quad (2)$$

To capture long-range temporal dependencies efficiently, TCN extends causal convolution using dilation. With dilation factor d, the convolution samples inputs at intervals of d, and the operation becomes $z_t = \sum_{i=0}^{k-1} w_i \, x_{t-i \cdot d}$. In practice, the dilation schedule ($d \in \{1,2,4,8, \dots\}$) is controlled by the hyperparameter dilations, allowing the network to expand its temporal coverage without a proportional increase in depth.

$$z_t = \sum_{i=0}^{k-1} w_i \; x_{t-i \cdot d} \quad (3)$$

The adequate temporal coverage of a TCN is commonly characterized by the receptive field R, i.e., the number of past time steps that can influence the output at time t. For a stack of Bdilated convolutional blocks with kernel size $k$and dilation factors $\{d_1, d_2, \dots, d_B\}$, an approximate receptive field is given by $R = 1 + (k-1)\sum_{b=1}^{B} d_b$. This expression highlights that kernel_size and dilations are key structural hyperparameters that directly shape the model's ability to learn dependencies across multiple time scales.

$$R = 1 + (k-1)\sum_{b=1}^{B} d_b \quad (4)$$

To improve training stability and enable deep stacks of dilated convolutions, TCN adopts residual learning. A typical residual block applies a nonlinear transformation $\mathcal{F}(\cdot)$(implemented by one or more dilated causal convolutions and activations) and adds the input through an identity shortcut, producing $\mathbf{h}^{(b)} = \mathcal{F}(\mathbf{h}^{(b-1)}; \theta^{(b)}) + \mathbf{h}^{(b-1)}$, where $\theta^{(b)}$denotes the learnable parameters of block b. Residual connections help preserve gradient flow and reduce degradation when increasing depth.

$$\mathrm{h}^{(b)} = \mathcal{F}\left(\mathrm{h}^{(b-1)}; \theta^{(b)}\right) + \mathrm{h}^{(b-1)} \quad (5)$$

From an implementation perspective, the convolutional stack can be interpreted as an encoder that maps the input window $\mathbf{x}_t$ into a latent representation $\mathbf{H}$. In this study, the representational capacity of this latent space is controlled by encoder_hidden_size, which determines the dimensionality of intermediate features used to summarize the historical context. This mapping can be expressed as:

$$\mathrm{H} = \text{TCN-Encoder}(\mathbf{x}_t; \text{kernel\_size,dilations,encoder\_hidden\_size}) \quad (6)$$

The multi-step forecasting output can be produced by a decoding component that transforms $\mathbf{H}$ into $\hat{\mathbf{y}}_t$. The decoding capacity is governed by decoder_hidden_size and the depth parameter decoder_layers, which controls the number of stacked transformations applied before output projection. This process can be written as $\hat{\mathbf{y}}_t = \text{Decoder}(\mathbf{H}; \text{decoder\_hidden\_size,decoder\_layers})$, where context_size determines the historical length L fed into the overall encoder-decoder pipeline.

$$\hat{y}_t = \text{Decoder}(\mathrm{H}; \text{decoder\_hidden\_size,decoder\_layers}) \quad (7)$$

In direct multi-step forecasting, the final latent representation at the last input time step can be projected to the horizon output through a linear head, $\hat{y}_t = \mathrm{W}_o \mathrm{h}_t + \mathrm{b}_o$, where $\hat{y}_t \in \mathbb{R}^H$. This formulation matches practical implementations in which the network produces a vector of length H as the forecast for the next horizon.

$$\hat{y}_t = \mathrm{W}_o\, \mathrm{h}_t + \mathrm{b}_o \quad (8)$$

Consistent with our implementation using mean squared error, model parameters are optimized by minimizing the average squared forecasting error across the horizon. The training objective is defined as:

$$\mathcal{L}(\theta) = \frac{1}{H}\sum_{j=1}^{H}\left(y_{t+j} - \hat{y}_{t+j}\right)^2 \quad (9)$$

Which encourages the model to reduce deviations between the predicted and actual load values over all forecast steps.

2.2. Hyperparameters of the TCN Model

In this study, we analyze two groups of hyperparameters: structural hyperparameters and training hyperparameters. Structural hyperparameters define the model architecture and temporal modeling capacity (width, depth, and temporal coverage), whereas training hyperparameters govern the optimization process (convergence speed, stability, and generalization). For clarity and reproducibility, we explicitly separate tuned hyperparameters from fixed parameters (including Nixtla NeuralForecast defaults and study-level fixed settings).

2.2.1. Tuned Hyperparameters

Structural hyperparameters (tuned). We tune kernel_size, dilations, encoder_hidden_size, context_size, decoder_hidden_size, and decoder_layers because they directly determine the TCN architecture, its temporal coverage, and its representational capacity. In a TCN, the pair (kernel_size, dilations) primarily controls the receptive field, i.e., the maximum length of historical input that can influence each forecasted value through stacked causal dilated convolutions. Specifically, kernel_size determines how many adjacent time steps each convolutional filter aggregates at a layer, thereby shaping the model's ability to capture local temporal patterns. In contrast, dilations specify the dilation schedule across layers ([1,2,4,8,16]) to efficiently expand the receptive field and enable learning of longer-range dependencies without requiring excessive depth. An overly small receptive field may prevent the model from exploiting relevant past information, whereas overly aggressive dilation or excessive capacity can increase optimization difficulty and the risk of overfitting. The remaining structural hyperparameters mainly regulate model capacity. encoder_hidden_size sets the width of the encoder representation and controls how rich the learned feature space can be. context_size determines

how much recent encoded context is used when producing multi-step forecasts, thereby affecting stability and coherence across the prediction horizon. Finally, decoder_hidden_size and decoder_layers determine the width and depth of the decoder (forecasting head) that maps encoded features to multi-step outputs, thereby influencing nonlinear modeling capacity, parameter count, and computational cost.

Training hyperparameters (tuned). We tune learning_rate, batch_size, and max_steps because they govern the optimization dynamics and can substantially affect convergence stability and generalization even under the same architecture. learning_rate controls the magnitude of parameter updates; values that are too large may lead to unstable training or divergence, whereas overly small values can slow convergence and yield underfitting within a finite training budget. batch_size specifies the number of samples per gradient update and influences gradient stochasticity, computational efficiency, and generalization behavior; smaller batches introduce noisier gradients that may regularize training but can slow convergence, while larger batches typically improve throughput and stability but may require careful adjustment of learning_rate. max_steps defines the total optimization budget in training steps; too few steps may prevent the model from learning sufficiently expressive representations, whereas excessive steps increase computation and may exacerbate overfitting if not controlled by validation or regularization. By tuning these training hyperparameters while keeping other settings fixed, we isolate their effects on forecasting accuracy and stability within the proposed evaluation protocol.

2.2.2. Fixed Parameters

To avoid confounding effects and ensure fair, interpretable comparisons, we keep all non-target parameters fixed throughout the experiments. This design choice is critical for a sensitivity study: if multiple settings change simultaneously (scaling, validation frequency, window sampling strategy), observed performance differences could no longer be confidently attributed to the tuned hyperparameters. By freezing all other options, any change in forecasting accuracy or stability can be linked primarily to the hyperparameter under investigation rather than to hidden implementation or training-side variations.

Concretely, the fixed settings comprise two parts. (i) Library-level defaults (Nixtla NeuralForecast). We retain the default training and pipeline controls provided by the Nixtla NeuralForecast implementation, including the scaling/normalization choice (robust scaling), validation-check frequency (val_check_steps), window-based batching controls for training and inference (windows_batch_size and inference_windows_batch_size), learning-rate decay configuration (num_lr_decays), and other auxiliary options that govern how batches are formed, how often validation is evaluated, and how training proceeds. These defaults act as a constant "background" so that tuned configurations differ only in the selected hyperparameters. (ii) Study-level fixed settings required by the forecasting task. We also fix parameters that define the problem setting itself, such as the sampling frequency (30 minutes) and the forecast horizon ($h = 48$ for one-day-ahead prediction). Fixing these task-defining parameters ensures that every configuration solves the same forecasting problem under the same evaluation conditions.

Table 1 reports the complete reference configuration used in this study and explicitly separates (a) the tuned structural hyperparameters, (b) the tuned training hyperparameters, and (c) the fixed/default parameters. This separation improves transparency and reproducibility, and it prevents unintended shifts in preprocessing or training behavior from biasing the hyperparameter-impact analysis.

2.3. Proposed Method for Analyzing Hyperparameter Impact

In this study, we develop a controlled, reproducible sensitivity analysis protocol to quantify how TCN hyperparameters affect short-term load forecasting performance. The guiding principle is that only the target hyperparameter(s) under investigation are varied. At the same time, all remaining settings are held constant-either as study-defined constants or as Nixtla NeuralForecast default options. This design minimizes confounding effects and ensures that any observed performance

differences are primarily attributable to hyperparameter changes rather than to uncontrolled implementation details or training-side variations.

Table 1. Basic hyperparameters of the TCN mode

Hyperparameter	Brief role	Value / Setting
kernel_size	Convolution kernel length	2
dilations	Dilation schedule	[1, 2, 4, 8, 16]
encoder_hidden_size	Encoder hidden representation size	128
context_size	Context size used for forecasting	10
decoder_hidden_size	Decoder hidden representation size	128
decoder_layers	Decoder depth	2
learning_rate	Update step size	0.001
batch_size	Samples per gradient update	32
max_steps	Training budget	1000
h	Forecast horizon	48
input_size	Lookback window length	336
loss	Training loss	MSE()
random_seed	Reproducibility seed	42
freq *(NeuralForecast)*	Sampling frequency	30min
scaler_type	Scaling/normalization option	'robust'
val_check_steps	Validation checking frequency	100
windows_batch_size	Window-based training batch size	128
inference_windows_batch_size	Window-based inference batch size	1024
num_lr_decays	Learning-rate decay count	-1

The procedure starts by defining a reference (baseline) TCN configuration that serves as the anchor for all comparisons. The baseline fixes the task-defining settings (sampling frequency and forecast horizon), windowing settings (input_size), the loss function, and the random seed used by the training pipeline. In addition, non-target pipeline and training controls are retained at their Nixtla NeuralForecast defaults, including scaling/normalization, validation check frequency, window-based batching for training and inference, and learning rate decay settings. By explicitly separating tuned hyperparameters from fixed/default parameters, we maintain a consistent experimental background across all trials.

Next, we specify a predefined hyperparameter search space spanning two groups: structural hyperparameters (kernel_size, dilations, encoder_hidden_size, context_size, decoder_hidden_size, decoder_layers) and training hyperparameters (learning_rate, batch_size, max_steps). A trial corresponds to one complete training-evaluation run under a specific hyperparameter setting (or a targeted setting combination if explicitly tested), with all other parameters held constant at the baseline values. All trials are executed using the same NeuralForecast training pipeline and the same multi-step forecasting interface, ensuring direct comparability across configurations.

To isolate the effects of each hyperparameter group and maintain clear interpretation, we adopt an OFAT-style controlled comparison at the group level. When analyzing structural hyperparameters, all training hyperparameters are fixed at their baseline values so that performance variation reflects architectural changes rather than optimization differences. Conversely, when analyzing training hyperparameters, all structural hyperparameters are fixed at baseline so that observed differences are driven by optimization settings rather than changes in model structure. Within each group, the target hyperparameter is varied across its candidate values, while the remaining hyperparameters are held constant, enabling a direct and fair assessment of sensitivity. Because this OFAT design does not fully characterize interactions among hyperparameters, we explicitly acknowledge this limitation and, where applicable, report only targeted interaction checks (learning_rate × batch_size or kernel_size × dilations) rather than a full factorial exploration.

For each trial, forecasts are generated on the same held-out test segment and accuracy is quantified using standard error metrics (MAE, MSE, RMSE, and MAPE). We treat MAPE as the primary metric because it is scale-free and facilitates comparison across different load levels. At the

same time, MAE/RMSE provide complementary views of absolute error magnitude and the influence of larger deviations. To characterize the impact of hyperparameters beyond a single average value, we analyze trial-level error distributions. Specifically, for each candidate value of a hyperparameter, we compare the distribution of errors in terms of central tendency (median) and dispersion (IQR and outliers), thereby capturing both typical performance and robustness/stability.

Finally, the impact of hyperparameters is summarized from two complementary perspectives. First, we identify the best-performing configuration within the tested ranges using the lowest median MAPE as the primary selection criterion, with MAE/RMSE used as supporting evidence when needed. Second, we quantify sensitivity by measuring how strongly performance varies across candidate values (changes in median error and dispersion) and rank hyperparameters accordingly. Structural hyperparameters are interpreted by their influence on temporal coverage (receptive field) and representational capacity, whereas training hyperparameters are interpreted by their influence on convergence behavior, optimization stability, and generalization. This procedure yields empirical, sensitivity-driven guidance on which hyperparameters require careful tuning, which choices tend to be robust, and how architectural and training decisions shape TCN forecasting performance under the proposed protocol.

2.4. Flowchart of the Hyperparameter Impact Evaluation Process

Fig. 1 presents the end-to-end workflow for building, tuning, and evaluating a Temporal Convolutional Network (TCN) model for time-series forecasting. The process starts at the Start block, where the input time-series data (Y1, Y2, …, Yn) are fed into the system through the Input data block. This sequence (for example, electrical load measured hourly or at 30-minute intervals) serves as the primary data source for all subsequent stages. In the Input data processing block, the raw series is preprocessed to ensure consistency and suitability for deep learning.

Specifically, the data are time-aligned by sorting chronologically and ensuring a consistent sampling frequency, missing values are handled (if any), and optional transformations or normalization may be applied to facilitate optimization. The time series is then converted into supervised learning samples using a sliding-window strategy: a historical segment is used as input X, and the corresponding future segment as target Y, consistent with multi-step forecasting.

From the processed data, the workflow constructs two datasets using a chronological split: (i) the Training Data set (X_train, Y_train), which is used to train the model and update its weights, and (ii) the Testing Data set (X_test, Y_test), which is used to evaluate the model's generalization performance on unseen data. This time-ordered split prevents information leakage from the future into training and reflects realistic deployment conditions for time-series forecasting.

In parallel with data preparation, the Search space (Structural and Training) block defines the hyperparameter search space, which comprises two main groups. Structural hyperparameters describe the TCN architecture (input window length, kernel size, dilation settings, encoder hidden size, decoder hidden size, number of decoder layers, and context size), whereas training hyperparameters control the optimization process (learning rate, batch size, and the maximum number of training steps). Based on this search space, the Optimization (Grid Search) block generates a set of candidate hyperparameter configurations, where each configuration specifies a concrete combination of both structural and training hyperparameters.

Each hyperparameter configuration is then passed to the TCN block, where the model is trained on the Training Data set (X_train, Y_train). The training procedure estimates the model parameters (weights) by optimizing the learning objective using gradient-based optimization. After training under a given configuration, the resulting model produces forecasts on the Testing Data set (X_test, Y_test), yielding the corresponding forecasting errors. All trials-including both the hyperparameter settings and their evaluation outcomes-are stored in the All Trial and Parameters block. This block supports direct comparison among candidate configurations, selection of the best-performing configuration, and sensitivity analysis to identify which hyperparameters most strongly influence forecasting performance.

Finally, the Evaluate using MAPE block quantifies the forecasting performance of each configuration using MAPE (Mean Absolute Percentage Error) as the unified evaluation metric. MAPE is used to rank configurations and summarize performance differences. By aggregating MAPE across all trials, the workflow enables the selection of an appropriate configuration and provides empirical evidence for analyzing the effects of both structural and training hyperparameters on the forecasting accuracy of the TCN model in the short-term load forecasting task.

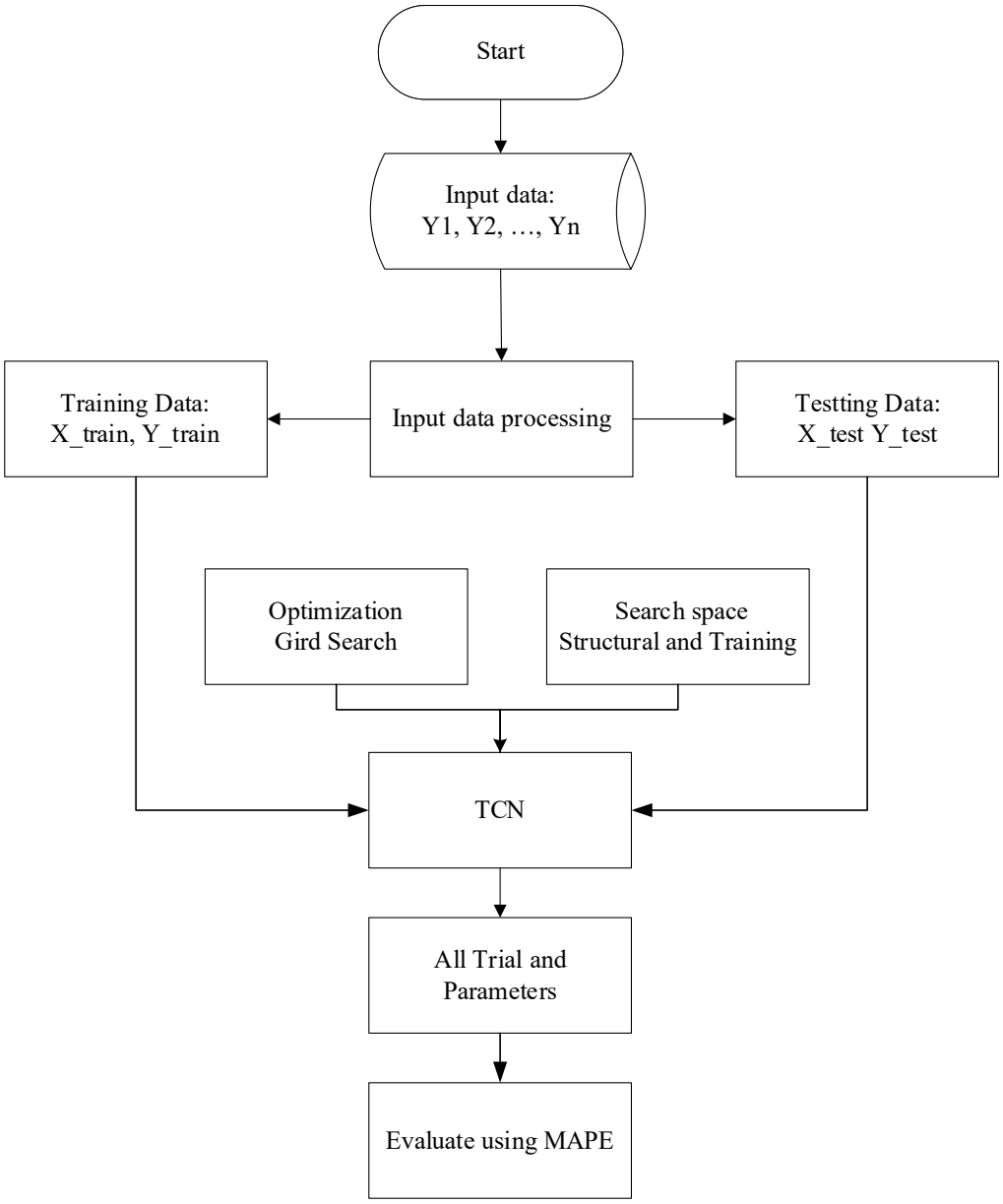

Fig. 1. Flowchart of the hyperparameter optimization search process for the TCN model

3. Experimental Setup

3.1. Dataset Description and Experimental Setup

The study uses the NSW (New South Wales) electricity load dataset as a univariate time series, consisting of two main fields: SETTLEMENTDATE (timestamp) and TOTALDEMAND (load). The data are recorded at a 30-minute resolution, continuously over time, and are sorted chronologically before being fed into the model. The dataset covers the period from 2015-01-01 00:30:00 to 2021-12-31 23:30:00, with a total of 122,735 observations. In terms of descriptive statistics, TOTALDEMAND

has an average of approximately 7,890.88, a standard deviation of 1,260.70, a minimum of 4,316.63, and a maximum of 13,985.87 (in the dataset's original units). No exogenous variables are included; therefore, the forecasting task relies entirely on the historical values of the load series itself. Based on this dataset, the experimental setup is designed to reflect a practical short-term load-forecasting scenario and to ensure consistent model evaluation. Specifically, the task is configured as multi-step forecasting with a fixed forecasting horizon of 48 time steps, corresponding to 1 day at a 30-minute resolution. The data are split chronologically, with the most recent 28 days used for training and the next day for testing. This split emulates the real operational setting, where the model is trained only on past data and produces forecasts for the immediate future. Fig. 2 illustrates the data split at the end segment of the time series, including the 28-day training period and the 1-day testing period, together with the train/test boundary.

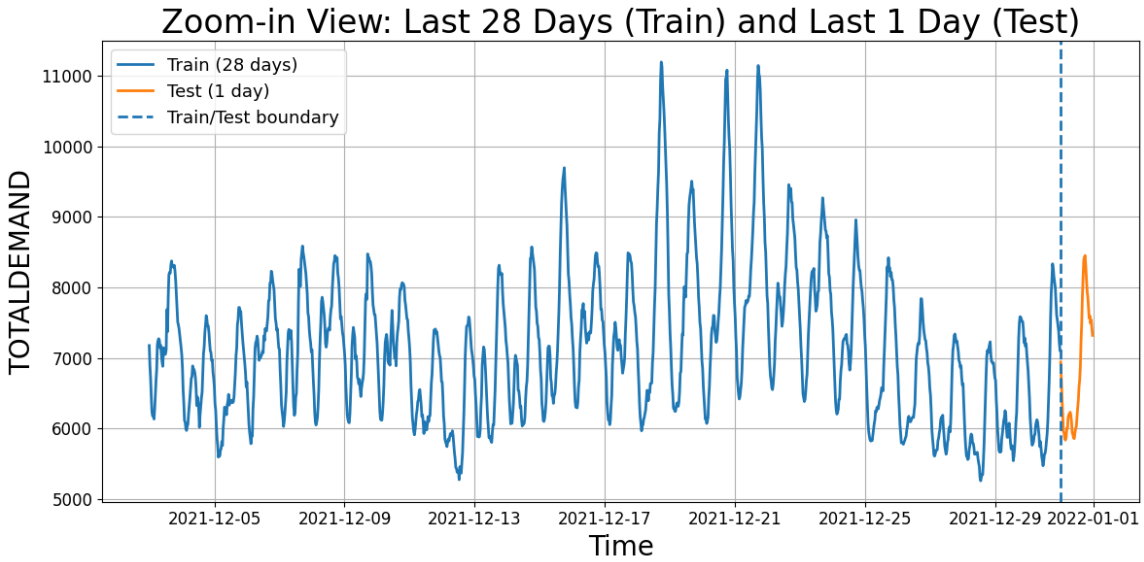

Fig. 2. Zoomed view of the last 28 training days and the last 1 testing day

The figure clearly shows daily load fluctuations as well as several load peaks occurring within the training segment. Selecting a 28-day training window allows the model to learn both short-term trends and recurring daily patterns, while the 1-day testing window directly reflects the model's short-term forecasting capability under practical conditions. Fig. 3 presents the average daily load profile aggregated over all days using 30-minute intervals. The mean curve highlights a typical load shape, with lower demand during the night and early morning, increasing during daytime hours, and reaching a peak in the late afternoon to evening.

The min-max band indicates the variability of load at each time-of-day, reflecting the inherent uncertainty and randomness of electricity demand. These observations confirm that the load series exhibits a clear daily cyclical structure while still showing considerable fluctuations, which requires forecasting models to capture both stable periodic patterns and irregular variations. Overall, the experimental setup and the visual analyses in Fig. 2 and Fig. 3 indicate that the NSW dataset is suitable for short-term load forecasting and sufficiently challenging for a comprehensive evaluation of how hyperparameters affect the performance of the TCN model.

3.2. Configuration of the Hyperparameter Search Space

Table 2 presents the hyperparameter search space used to evaluate the impact of TCN hyperparameters on forecasting performance. As shown in Table 2, the search space is organized into two groups-architecture hyperparameters and training hyperparameters-so that the effects of model design choices and optimization settings can be examined in a controlled and interpretable manner.

For the architecture group, the candidate sets are constructed to cover the main factors that determine temporal coverage and model capacity. The historical input length is examined through

input_size with three window choices [48×3, 48×5, 48×7], enabling a comparison of short, medium, and longer contexts under the same forecasting setup. The temporal receptive field is mainly governed by the convolutional configuration; therefore, kernel_size is tested at [2, 3], and dilations is tested using two commonly used dilation schedules [[1,2,4,8], [1,2,4,8,16]] to compare a smaller versus a larger receptive field expansion strategy. To assess representational capacity, encoder_hidden_size and decoder_hidden_size are varied in [64, 128], and the depth of the decoding component is varied using decoder_layers in [1, 2]. In addition, context_size is evaluated in [5, 10] to analyze how the amount of contextual information used in multi-step forecasting affects accuracy.

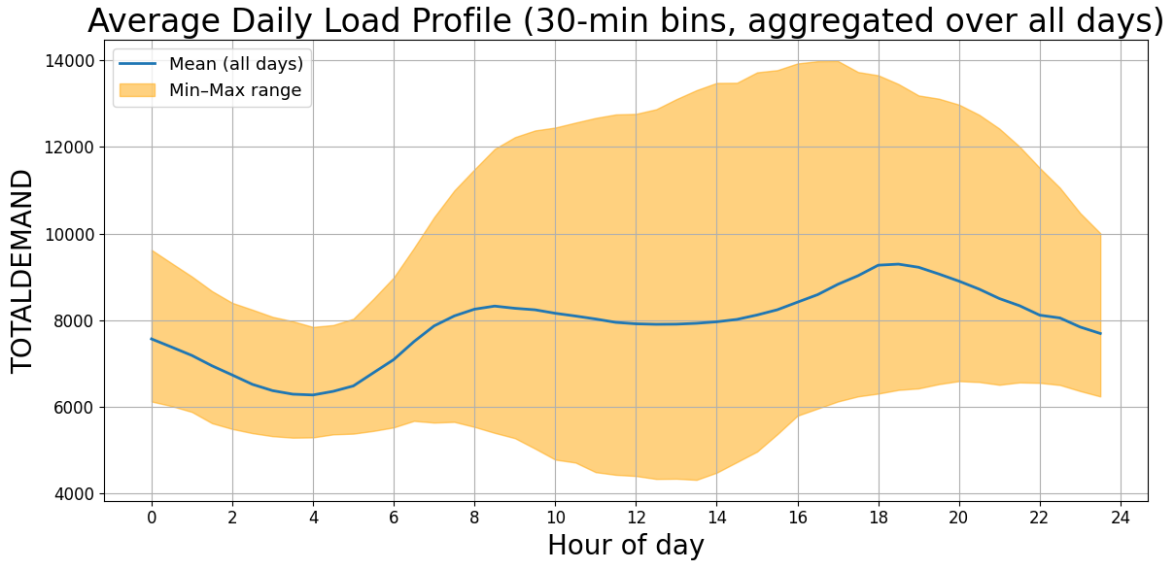

Fig. 3. Average daily load profile with min-max range (30-minute resolution)

Table 2. Hyperparameter search space

Group	Parameter	Candidate Value Set	Status
Architecture	input_size	[48×3, 48×5, 48×7]	Tuned
	kernel_size	[2, 3]	Tuned
	dilations	[[1,2,4,8], [1,2,4,8,16]]	Tuned
	encoder_hidden_size	[64, 128]	Tuned
	context_size	[5, 10]	Tuned
	decoder_hidden_size	[64, 128]	Tuned
	decoder_layers	[1, 2]	Tuned
Training	learning_rate	[7e-4, 8e-4, 9e-4, 1.0e-3, 1.1e-3, 1.2e-3, 1.4e-3, 1.6e-3]	Tuned
	batch_size	[16, 32, 64, 128]	Tuned
	max_steps	[900, 1000, 1100, 1200, 1300]	Tuned

For the training group, the candidate sets are selected to probe optimization sensitivity around standard settings. The learning_rate is explored in a narrow range centered on 1.0e-3 using [7e-4, 8e-4, 9e-4, 1.0e-3, 1.1e-3, 1.2e-3, 1.4e-3, 1.6e-3], which helps identify whether minor adjustments in step size improve convergence stability and generalization. The batch_size is varied across [16, 32, 64, 128] to capture the trade-off between gradient stochasticity and computational efficiency. The training budget is examined through max_steps in [900, 1000, 1100, 1200, 1300] to determine whether slightly shorter or longer training schedules lead to meaningful performance changes under the same training pipeline.

Importantly, the candidate grids intentionally include values that match the Nixtla default settings summarized in Table 2 to provide a reliable reference point within the tuning process. Specifically, the architecture grids include kernel_size = 2 and dilations = [1,2,4,8,16], as well as encoder_hidden_size = 128, context_size = 10, decoder_hidden_size = 128, and decoder_layers = 2, while the training grids include learning_rate = 1.0e-3, batch_size = 32, and max_steps = 1000.

Including these default values in the search space ensures that baseline-consistent configurations are always evaluated, improves comparability across trials, and increases the reliability of conclusions regarding hyperparameter impact.

3.3. Google Colab Environment and Nixtla Library

All experiments were conducted in the Google Colab environment to ensure an accessible and reproducible execution platform. The runtime configuration followed a standardized setup. Specifically, the runtime type was set to Python 3, the hardware accelerator was enabled and configured with a GPU (A100), and the high-RAM option was activated to support model training and repeated hyperparameter trials. In addition, the runtime version was kept at the latest recommended option to reduce compatibility issues across library dependencies. For implementation, the study adopted the Nixtla NeuralForecast ecosystem, which provides a consistent pipeline for time-series forecasting models, including the TCN model used in this work. NeuralForecast is built on a PyTorch backend; therefore, PyTorch and its related packages were installed together with the NeuralForecast library to ensure full compatibility with GPU acceleration. The installation step was performed directly in Colab using the following commands: pip install neuralforecast and pip install torch torchvision. After installation, model training and forecasting were executed through the unified

NeuralForecast interface, enabling a consistent procedure for fitting the model, generating multi-step forecasts, and collecting evaluation metrics across different hyperparameter configurations. To support reproducibility, the experimental code set a random seed at the model level and maintained consistent software settings across runs. Overall, using Google Colab with GPU acceleration and the Nixtla NeuralForecast library provided an efficient and standardized environment for training the TCN model and performing systematic hyperparameter impact analysis.

4. Results and Discussion

4.1. Structural Hyperparameters

This subsection analyzes the sensitivity of TCN forecasting performance to structural hyperparameters using the Grid A experiments. Grid A exhaustively enumerates 192 structural configurations in the predefined structural search space (3×2×2×2×2×2×2). All configurations are evaluated under the same forecasting setup and fixed non-target settings; therefore, each boxplot in Fig. 4, Fig. 5, Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10 summarizes MAPE across many architectural combinations sharing the same hyperparameter value, highlighting not only typical accuracy (median) but also robustness (IQR and outliers). Fig. 4 illustrates the MAPE distributions for three lookback window lengths (input sizes of 144, 240, and 336).

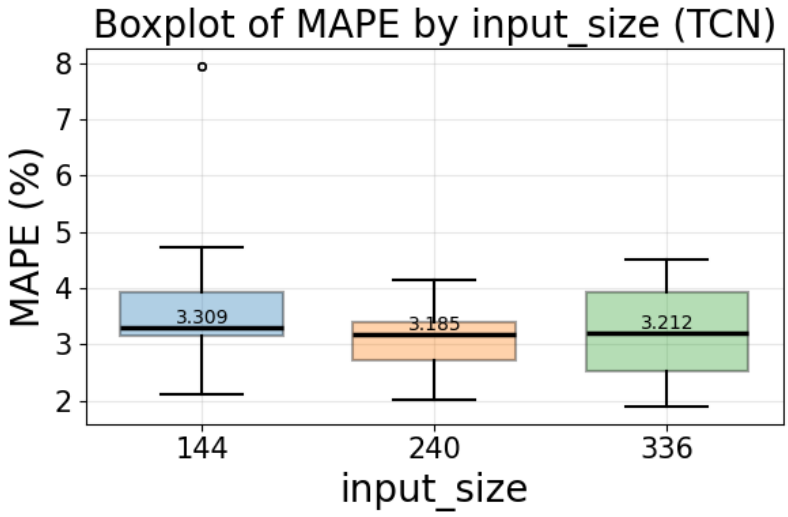

Fig. 4. Distribution of MAPE across different input sizes in TCN

Among the tested values, input_size = 240 achieves the lowest median MAPE (3.1848), the smallest IQR (0.6588), and no outliers, while input_size = 144 yields a higher median (3.3087) and four high-error outliers. Although input_size = 336 produces a median close to the reference level (3.2118), it shows the largest dispersion (IQR = 1.3923), indicating that a longer window does not necessarily improve robustness. Fig. 5 compares kernel_size = 2 and 3.

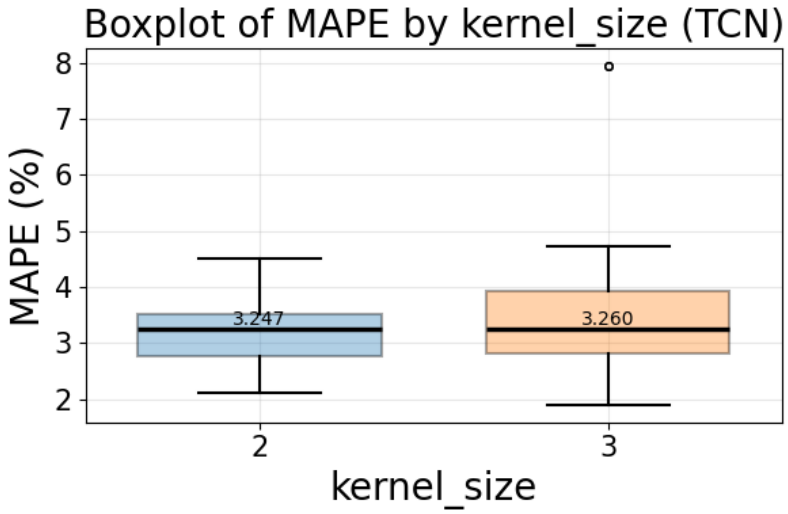

Fig. 5. Distribution of MAPE across different kernel_size in TCN

The medians are close (3.2470 vs. 3.2604), but kernel_size = 2 shows lower dispersion (IQR = 0.7454) with no outliers, whereas kernel_size = 3 has a wider IQR (1.1158) and four high-error outliers. Fig. 6 compares two dilation schedules, [1, 2, 4, 8] and [1, 2, 4, 8, 16].

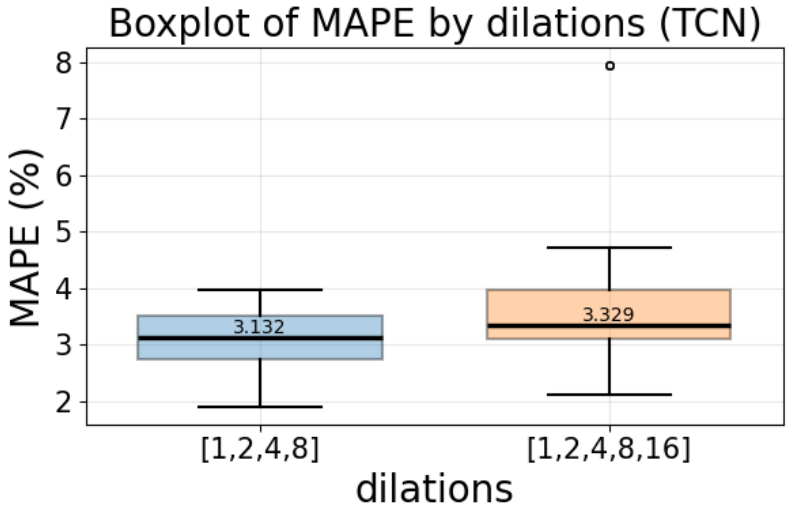

Fig. 6. Distribution of MAPE across different dilations in TCN

Fig. 6 shows the distribution of MAPE for two dilation settings: [1, 2, 4, 8] and [1, 2, 4, 8, 16]. The results of the [1, 2, 4, 8] schedule achieve a lower median MAPE (3.1322) than [1, 2, 4, 8, 16] (3.3293), with tighter dispersion (IQR = 0.7728 vs. 0.8798) and no outliers (versus four outliers when adding dilation 16). Fig. 7, Fig. 8, Fig. 9, Fig. 10 report the distributions for encoder_hidden_size, context_size, decoder_hidden_size, and decoder_layers.

Within the explored ranges, these capacity-related parameters show limited shifts in median MAPE compared with input_size and dilations. For example, encoder_hidden_size = 64 yields a median MAPE of 3.2054, versus 3.2882 for 128, while decoder_hidden_size = 64 and 128 yield

3.2470 and 3.2649, respectively. Notably, context_size (5 vs. 10) and decoder_layers (1 vs. 2) exhibit identical distributions (median = 3.2489 and IQR = 1.0506), indicating negligible sensitivity within the tested ranges.

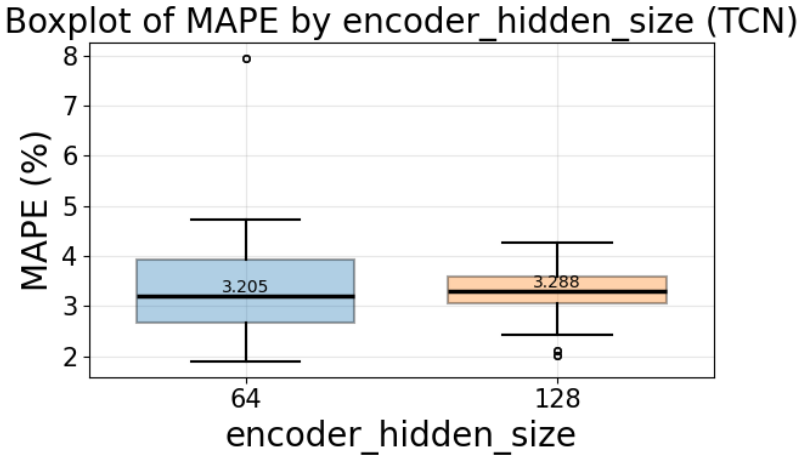

Fig. 7. Distribution of MAPE across different encoder_hidden_size in TCN

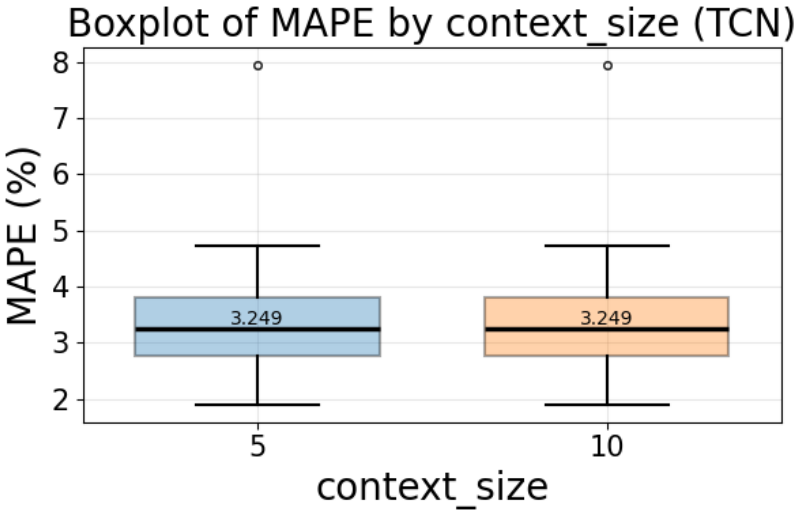

Fig. 8. Distribution of MAPE across different context_size in TCN

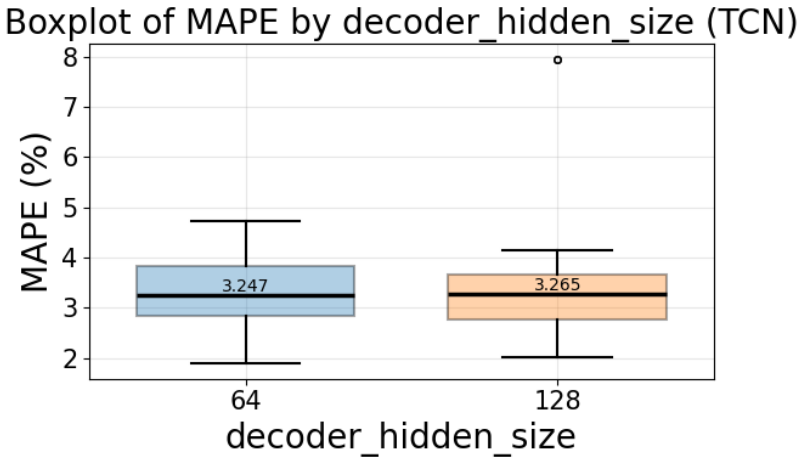

Fig. 9. Distribution of MAPE across different decoder_hidden_size in TCN

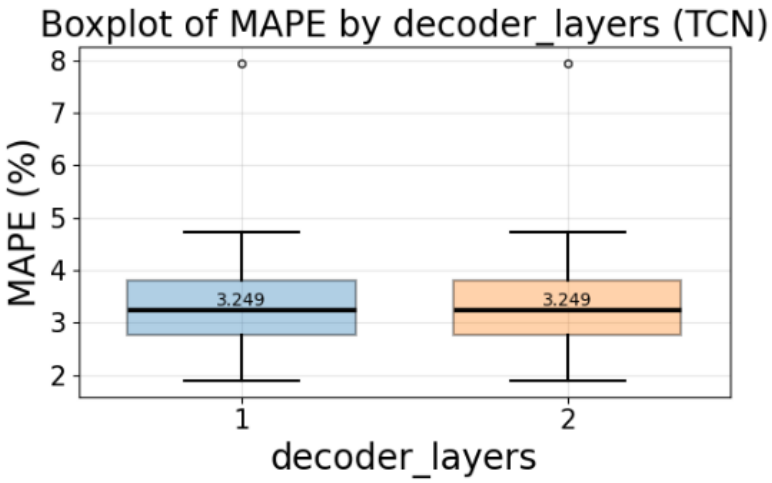

Fig. 10. Distribution of MAPE across different decoder_layers in TCN

To summarize the overall structural sensitivity in Grid A, Table 3 reports the best and worst median MAPE for each structural hyperparameter and the corresponding dispersion/outlier information. The most enormous median differences are observed for dilations (Δmedian = 0.1971) and input_size (Δmedian = 0.1239), while the remaining parameters exhibit more minor median shifts and primarily affect dispersion.

Table 3. Summary of structural hyperparameter impact in Grid A (MAPE)

Hyperparameter	Candidate Values	Best Median (IQR; Outliers)	Worst Median (IQR; Outliers)	Median
input_size	144, 240, 336	240: 3.1848 (0.6588; 0)	144: 3.3087 (0.7679; 4)	0.1239
kernel_size	2, 3	2: 3.2470 (0.7454; 0)	3: 3.2604 (1.1158; 4)	0.0134
dilations	[1,2,4,8], [1,2,4,8,16]	[1,2,4,8]: 3.1322 (0.7728; 0)	[1,2,4,8,16]: 3.3293 (0.8798; 4)	0.1971
encoder_hidden_size	64, 128	64: 3.2054 (1.2598; 4)	128: 3.2882 (0.5334; 8)	0.0829
decoder_hidden_size	64, 128	64: 3.2470 (0.9954; 0)	128: 3.2649 (0.9058; 4)	0.0179
context_size	5, 10	5: 3.2489 (1.0506; 2)	5: 3.2489 (1.0506; 2)	0.0000
decoder_layers	1, 2	1: 3.2489 (1.0506; 2)	1: 3.2489 (1.0506; 2)	0.0000

Overall, Grid A indicates that structural tuning should prioritize receptive-field controls (especially the dilation schedule) and lookback length. At the same time, several encoder-decoder capacity parameters appear relatively insensitive within the tested ranges.

4.2. Training Hyperparameters

This subsection examines the impact of training hyperparameters using the Grid B experiments. Grid B exhaustively enumerates 160 training configurations (8×4×5) in the predefined training search space. All configurations are evaluated under the same forecasting setup and fixed non-target settings; therefore, each boxplot in Fig. 11, Fig. 12, Fig. 13 summarizes MAPE across many training combinations sharing the same hyperparameter value, highlighting both typical accuracy (median) and robustness (IQR and outliers). Fig. 11 shows the MAPE distributions across eight learning-rate values.

The learning rate shows the most significant shift in median performance among the training hyperparameters. In particular, learning_rate = 9×10^{-4} achieves the lowest median MAPE (2.3751) with IQR = 0.3741 and no outliers, while learning_rate = 8×10^{-4} yields the highest median MAPE (3.0638) (IQR = 0.4703) with one outlier. Although several intermediate values (1.1×10^{-3}) also perform competitively (median = 2.4395, IQR = 0.3253), huge learning rates (1.6×10^{-3}) exhibit increased instability, including outliers. Fig. 12 compares the MAPE distributions across four batch sizes (16, 32, 64, and 128).

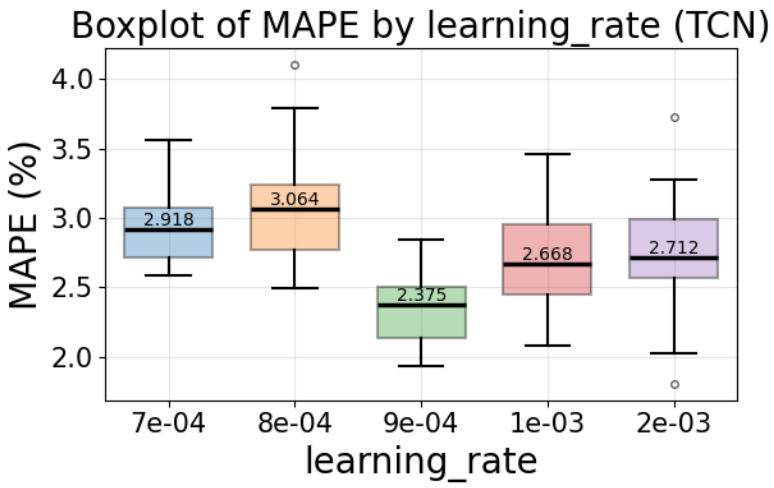

Fig. 11. Boxplot of MAPE across different learning rates for the TCN model

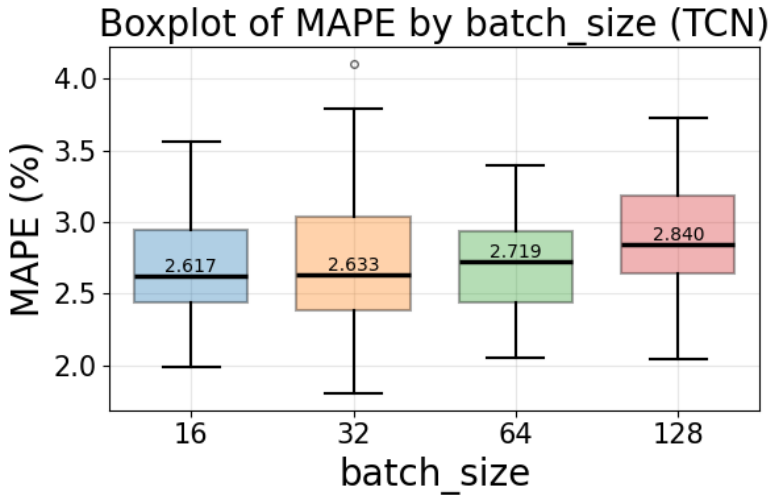

Fig. 12. Boxplot of MAPE across different batch_size for the TCN model

Batch size shows a clear, though more negligible, effect than learning rate. The lowest median MAPE is achieved by batch_size = 16 (2.6172, IQR = 0.5066, no outliers), whereas batch_size = 128 produces the highest median (2.8404, IQR = 0.5432). The differences are mainly reflected in median shifts rather than in dramatic changes in dispersion, suggesting that smaller batches are more favorable within the tested range. Fig. 13 presents the MAPE distributions across five training budgets (max_steps = 900, 1000, 1100, 1200, and 1300).

Compared with learning rate and batch size, max_steps has a weaker influence on median accuracy within the examined range. The lowest median MAPE occurs at max_steps = 1200 (2.6566, IQR = 0.5465), while max_steps = 1000 yields the highest median (2.7623, IQR = 0.4787). Some settings (1100 and 1300) exhibit outliers, suggesting that increasing the training budget does not necessarily improve robustness.

To summarize overall training sensitivity in Grid B, Table 4 reports the best- and worst-case median MAPE for each training hyperparameter, along with IQR and outlier counts. The most considerable median difference is observed for learning_rate (Δmedian = 0.6886), followed by batch_size (Δmedian = 0.2233), whereas max_steps shows a more minor median shift (Δmedian = 0.1057) within the tested range.

Overall, Grid B indicates that training hyperparameters have a more direct effect on model performance than most capacity-related structural parameters in Grid A, with learning_rate being the

dominant factor within the tested ranges. Consequently, training-time tuning should prioritize the learning rate, followed by the batch size, while max_steps primarily serves as a convergence budget control once a sufficient range is reached.

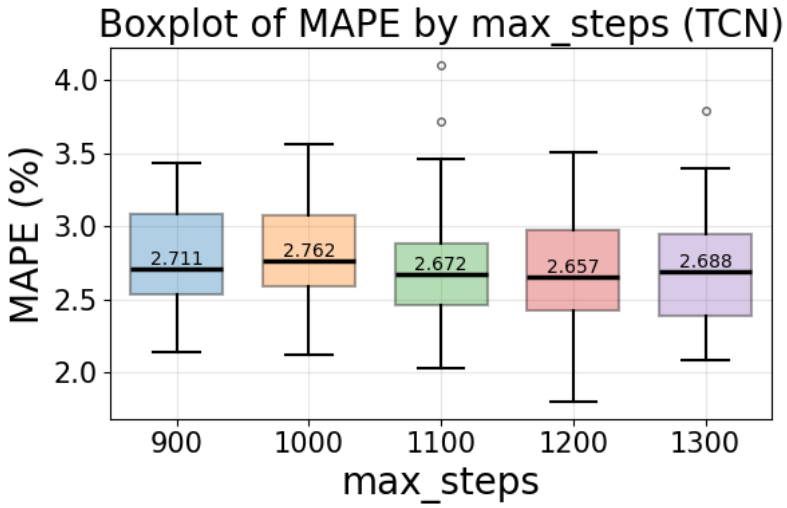

Fig. 13. Boxplot of MAPE across different max_step for the TCN model

Table 4. Summary of training hyperparameter impact in Grid B (MAPE)

Hyperparameter	Candidate Values	Best Median (IQR; Outliers)	Worst Median (IQR; Outliers)	Median
learning_rate	7e-4, 8e-4, 9e-4, 1e-3, 1.1e-3, 1.2e-3, 1.4e-3, 1.6e-3	9e-4: 2.3751 (0.3741; 0)	8e-4: 3.0638 (0.4703; 1)	0.6886
batch_size	16, 32, 64, 128	16: 2.6172 (0.5066; 0)	128: 2.8404 (0.5432; 0)	0.2233
max_steps	900, 1000, 1100, 1200, 1300	1200: 2.6566 (0.5465; 0)	1000: 2.7623 (0.4787; 0)	0.1057

5. Conclusion

This study examined how structural and training hyperparameters affect TCN performance for short-term load forecasting, using distribution-level evaluation (median MAPE, IQR, and outliers) over the tested configuration grids (Fig. 4, Fig. 5, Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10, Fig. 11, Fig. 12, Fig. 13). The results show that the most consequential structural choices are those controlling temporal coverage, particularly the dilation schedule and input window length (input_size), whereas several encoder-decoder capacity parameters (context_size, decoder_layers, and, to a lesser extent, hidden sizes) exhibit limited sensitivity within the explored ranges. For training, the model is most sensitive to learning_rate (dominant effect), followed by batch_size, while max_steps mainly acts as a convergence budget control once a sufficient range is reached. Within the tested ranges, the best setting achieves a median MAPE of 2.375%, improving the reference median MAPE of 3.212% by approximately 26% (relative).

From a practical standpoint, these findings suggest a clear tuning priority: first select a stable learning rate and an appropriate batch size, then focus on receptive-field design (especially dilations) and a moderate lookback window, while avoiding unnecessary increases in encoder-decoder capacity when comparable accuracy is achieved with simpler settings. This strategy supports an accuracy-robustness-efficiency trade-off that is directly relevant to operational short-term load forecasting pipelines.

Limitations of this work include evaluation on a single regional dataset and finite hyperparameter ranges; therefore, the reported conclusions should be interpreted as sensitivity patterns within the tested grids. Future work will extend the analysis to additional regions and seasons, incorporate

repeated-seed and/or statistical testing where appropriate, and benchmark against strong non-deep-learning baselines (boosting models) under the same protocol. Overall, the study provides an evidence-based sensitivity map that helps practitioners prioritize TCN hyperparameters that matter most for reliable short-term load forecasting.

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. 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.

[2] 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.

[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] 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.

[5] A. P. Piyal, S. Ahmed, K. F. Rahman and A. S. M. Mohsin, "Energy Demand Forecasting Using Machine Learning Perspective Bangladesh," *2023 IEEE IAS Global Conference on Renewable Energy and Hydrogen Technologies (GlobConHT)*, pp. 1-5, 2023, https://doi.org/10.1109/GlobConHT56829.2023.10087679.

[6] A. Gupta and A. Kumar, "Mid Term Daily Load Forecasting using ARIMA, Wavelet-ARIMA and Machine Learning," *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, pp. 1-5, 2020, https://doi.org/10.1109/EEEIC/ICPSEurope49358.2020.9160563.

[7] N. T. Tran and V. D. Lai, "Grid search of exponential smoothing method: a case study of ho chi minh city load demand," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, no. 3, pp. 161-167, 2020, http://doi.org/10.11591/ijeecs.v19.i3.pp1121-1130.

[8] M. Pleños, "Time series forecasting using holt-winters exponential smoothing: Application to abaca fiber data," *Zeszyty Naukowe SGGW w Warszawie-Problemy Rolnictwa Światowego*, vol. 22, no. 2, pp. 17-29, 2022, https://doi.org/10.22630/PRS.2022.22.2.6.

[9] N. T. Dieudonné, T. K. F. Armel, A. K. C. Vidal, and T. René, "Prediction of electrical energy consumption in Cameroon through econometric models," *Electric Power Systems Research*, vol. 210, p. 108102, 2022, https://doi.org/10.1016/j.epsr.2022.108102.

[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, p. 44, 2022, https://doi.org/10.1186/s40537-022-00599-y.

[11] G. Dudek, "Pattern-based local linear regression models for short-term load forecasting," *Electric Power Systems Research*, vol. 130, pp. 139-147, 2016, https://doi.org/10.1016/j.epsr.2015.09.001.

[12] J. Gan, L. Pan, Y. Jin, Q. Liu and X. Liu, "A Load Forecasting Approach Based on Graph Convolution Neural Network," *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber*

Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), pp. 1-3, 2022, https://doi.org/10.1109/DASC/PiCom/CBDCom/Cy55231.2022.9927829.

[13] Y. Mohamed, M. M. Fouda, Z. M. Fadlullah, R. Abdelfattah and M. I. Ibrahem, "BOL-LPP: A Bayesian-Optimized LSTM Model for Day-Ahead Load Price Forecasting in the ERCOT Market," *IEEE Open Journal of the Computer Society*, vol. 6, pp. 1001-1011, 2025, https://doi.org/10.1109/OJCS.2025.3580107.

[14] 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, https://doi.org/10.1016/j.energy.2024.132344.

[15] C. Xu and G. Chen, "Interpretable transformer-based model for probabilistic short-term forecasting of residential net load," *International Journal of Electrical Power & Energy Systems*, vol. 155, p. 109515, 2024, https://doi.org/10.1016/j.ijepes.2023.109515.

[16] A. Ahmad, X. Xiao, H. Mo, and D. Dong, "TFTformer: A novel transformer based model for short-term load forecasting," *International Journal of Electrical Power & Energy Systems*, vol. 166, p. 110549, 2025, https://doi.org/10.1016/j.ijepes.2025.110549.

[17] Y. Wang, S. Sun, G. Fathi, and M. Eslami, "Improving the method of short-term forecasting of electric load in distribution networks using wavelet transform combined with ridgelet neural network optimized by self-adapted Kho-Kho optimization algorithm," *Heliyon*, vol. 10, no. 7, 2024, https://doi.org/10.1016/j.heliyon.2024.e28381.

[18] D. Kumar, S. Ganguly, B. Acherjee, and A. S. Kuar, "Performance evaluation of TWIST welding using machine learning assisted evolutionary algorithms," *Arabian Journal for Science and Engineering*, vol. 49, no. 2, pp. 2411–2441, 2024, https://doi.org/10.1007/s13369-023-08238-1.

[19] R. Zhang, Y. Xu, Z. Y. Dong, W. Kong and K. P. Wong, "A composite k-nearest neighbor model for day-ahead load forecasting with limited temperature forecasts," *2016 IEEE Power and Energy Society General Meeting (PESGM)*, pp. 1-5, 2016, https://doi.org/10.1109/PESGM.2016.7741097.

[20] G. F. Fan, Y. H. Guo, J. M. Zheng, and W. C. Hong, "Application of the weighted k-nearest neighbor algorithm for short-term load forecasting," *Energies*, vol. 12, no. 5, p. 916, 2019, https://doi.org/10.3390/en12050916.

[21] N. Shabbir, R. Ahmadiahangar, A. Rosin, M. Jawad, J. Kilter and J. Martins, "XgBoost based Short-term Electrical Load Forecasting Considering Trends & Periodicity in Historical Data," *2023 IEEE International Conference on Energy Technologies for Future Grids (ETFG)*, pp. 1-6, 2023, https://doi.org/10.1109/ETFG55873.2023.10407926.

[22] A. Agarwal, "Load forecast anomaly detection under cyber attacks using a novel approach," *2022 IEEE 4th International Conference on Cybernetics, Cognition and Machine Learning Applications (ICCCMLA)*, pp. 1-6, 2022, https://doi.org/10.1109/ICCCMLA56841.2022.9988990.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] K. Ullah *et al*., "Short-Term Load Forecasting: A Comprehensive Review and Simulation Study With CNN-LSTM Hybrids Approach," *IEEE Access*, vol. 12, pp. 111858-111881, 2024, https://doi.org/10.1109/ACCESS.2024.3440631.

[29] O. Rubasinghe, X. Zhang, T. K. Chau, Y. H. Chow, T. Fernando and H. H. -C. Iu, "A Novel Sequence to Sequence Data Modelling Based CNN-LSTM Algorithm for Three Years Ahead Monthly Peak Load Forecasting," *IEEE Transactions on Power Systems*, vol. 39, no. 1, pp. 1932-1947, 2024, https://doi.org/10.1109/TPWRS.2023.3271325.

[30] M. Aouad, H. Hajj, K. Shaban, R. A. Jabr, and W. El-Hajj, "A CNN-Sequence-to-Sequence network with attention for residential short-term load forecasting," *Electric Power Systems Research*, vol. 211, p. 108152, 2022, https://doi.org/10.1016/j.epsr.2022.108152.

[31] H. Sun, "The Research of Wind Farm Power Forecasting Based on VMD-IPSO-CNN-LSTM Hybrid Approach," *IEEE Access*, vol. 13, pp. 192577-192588, 2025, https://doi.org/10.1109/ACCESS.2025.3614722.

[32] C. Ren, L. Jia and Z. Wang, "A CNN-LSTM Hybrid Model Based Short-term Power Load Forecasting," *2021 Power System and Green Energy Conference (PSGEC)*, pp. 182-186, 2021, https://doi.org/10.1109/PSGEC51302.2021.9542404.

[33] 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.

[34] D. Díaz-Bedoya, A. Philippon, M. González-Rodríguez and J. -M. Clairand, "Enhancing Building Energy Forecasting: A Comparative Analysis of LSTM, GRU, and RNN Models for Active Power, Reactive Power, and Power Factor Prediction," *IEEE Access*, vol. 13, pp. 186030-186041, 2025, https://doi.org/10.1109/ACCESS.2025.3625374.

[35] 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.

[36] M. A. Majeed, S. Phichaisawat, F. Asghar and U. Hussan, "Data-Driven Optimized Load Forecasting: An LSTM-Based RNN Approach for Smart Grids," *IEEE Access*, vol. 13, pp. 99018-99031, 2025, https://doi.org/10.1109/ACCESS.2025.3576303.

[37] A. Ali and J. E. A., "Deep Learning Networks for Short Term Load Forecasting," *2023 International Conference on Control, Communication and Computing (ICCC)*, pp. 1-5, 2023, https://doi.org/10.1109/ICCC57789.2023.10165216.

[38] T. H. Bao Huy, D. N. Vo, K. P. Nguyen, V. Q. Huynh, M. Q. Huynh and K. H. Truong, "Short-Term Load Forecasting in Power System Using CNN-LSTM Neural Network," *2023 Asia Meeting on Environment and Electrical Engineering (EEE-AM)*, pp. 01-06, 2023, https://doi.org/10.1109/EEE-AM58328.2023.10395221.

[39] Y. Yang, E. U. Haq and Y. Jia, "A Novel Deep Learning Approach for Short and Medium-Term Electrical Load Forecasting Based on Pooling LSTM-CNN Model," *2020 IEEE/IAS Industrial and Commercial Power System Asia (I&CPS Asia)*, pp. 26-34, 2020, https://doi.org/10.1109/ICPSAsia48933.2020.9208557.

[40] C. Li, R. Hu, C. -Y. Hsu and Y. Han, "Short-term Power Load Forecasting based on Feature Fusion of Parallel LSTM-CNN," *2022 IEEE 4th International Conference on Power, Intelligent Computing and Systems (ICPICS)*, pp. 448-452, 2022, https://doi.org/10.1109/ICPICS55264.2022.9873566.

[41] M. Sharma, N. Mittal, A. Mishra, and A. Gupta, "An efficient approach for load forecasting in agricultural sector using machine learning," *E-Prime-Advances in Electrical Engineering, Electronics and Energy*, vol. 6, p. 100337, 2023, https://doi.org/10.1016/j.prime.2023.100337.

[42] A. Kumar and M. N. Alam, "Bidirectional LSTM Network-Based Short-Term Load Forecasting Method in Smart Grids," *2023 5th International Conference on Energy, Power and Environment: Towards Flexible Green Energy Technologies (ICEPE)*, pp. 1-6, 2023, https://doi.org/10.1109/ICEPE57949.2023.10201537.

[43] J. -W. Xiao, X. -Y. Cui, X. -K. Liu, H. Fang and P. -C. Li, "Improved 3-D LSTM: A Video Prediction Approach to Long Sequence Load Forecasting," *IEEE Transactions on Smart Grid*, vol. 16, no. 2, pp. 1885-1896, 2025, https://doi.org/10.1109/TSG.2024.3458989.

[44] C. Wang, X. Li, Y. Shi, W. Jiang, Q. Song, and 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.

[45] 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.

[46] 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.

[47] 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.

[48] G. Yan, J. Wang, and M. Thwin, "A new Frontier in electric load forecasting: The LSV/MOPA model optimized by modified orca predation algorithm," *Heliyon*, vol. 10, no. 2, p. e24183, 2024, https://doi.org/10.1016/j.heliyon.2024.e24183.

[49] 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, no. 6, pp. 5347–5357, 2024, https://doi.org/10.1007/s12145-024-01456-7.

[50] 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.

[51] H. Eskandari, M. Imani, and M. Parsa 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.

[52] 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.

[53] S. Kumar, L. Hussain, S. Banarjee and M. Reza, "Energy Load Forecasting using Deep Learning Approach-LSTM and GRU in Spark Cluster," *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 1-4, 2018, https://doi.org/10.1109/EAIT.2018.8470406.

[54] B. Jiang, Y. Wang, Q. Wang and H. Geng, "A Novel Interpretable Short-Term Load Forecasting Method Based on Kolmogorov-Arnold Networks," *IEEE Transactions on Power Systems*, vol. 40, no. 1, pp. 1180-1183, 2025, https://doi.org/10.1109/TPWRS.2024.3498452.

[55] G. F. Rosa, J. O. Avelino, M. C. Cavalcanti and J. C. Duarte, "TForMIX: A Method That Combines LLM and Multidimensional Modeling for Technological Foresight," *IEEE Access*, vol. 13, pp. 153320-153339, 2025, https://doi.org/10.1109/ACCESS.2025.3605116.

[56] E. Poongulali and K. Selvaraj, "Improved load demand prediction for cluster microgrids using modified temporal convolutional feed forward network," *Telecommunication Systems*, vol. 87, no. 3, pp. 561-574, 2024, https://doi.org/10.1007/s11235-024-01187-6.

[57] H. Mirshekali, M. Reza Shadi, F. Ghanadi Ladani and H. Reza Shaker, "A Review of Large Language Models for Energy Systems: Applications, Challenges, and Future Prospects," *IEEE Access*, vol. 13, pp. 163162-163188, 2025, https://doi.org/10.1109/ACCESS.2025.3610994.

[58] J. Li, W. Chen, Y. Liu, J. Yang, Z. Zhou and D. Zeng, "Integrating Ordinary Differential Equations With Sparse Attention for Power Load Forecasting," *IEEE Transactions on Instrumentation and Measurement*, vol. 74, pp. 1-12, 2025, https://doi.org/10.1109/TIM.2025.3581667.

[59] X. Wen, J. Liao, Q. Niu, N. Shen, and Y. Bao, "Deep learning-driven hybrid model for short-term load forecasting and smart grid information management," *Scientific Reports*, vol. 14, no. 1, p. 13720, 2024, https://doi.org/10.1038/s41598-024-63262-x.

[60] M. A. A. Sarker, B. Shanmugam, S. Azam, and S. Thennadil, "Enhancing smart grid load forecasting: An attention-based deep learning model integrated with federated learning and XAI for security and interpretability," *Intelligent Systems with Applications*, vol. 23, p. 200422, 2024, https://doi.org/10.1016/j.iswa.2024.200422.

[61] T. Shoji and T. Noda, "A study of harmonics in a dedicated cable supply system to feed EV fast chargers," *Electric Power Systems Research*, vol. 221, p. 109421, 2023, https://doi.org/10.1016/j.epsr.2023.109421.

[62] 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.

[63] Y. Liu, X. Wang, S. Wang and Z. Xu, "Short-term Power Load Forecasting Based on Temporal Convolutional Network," *2022 International Conference on Information, Control, and Communication Technologies (ICCT)*, pp. 1-4, 2022, https://doi.org/10.1109/ICCT56057.2022.9976543.

[64] 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.

[65] D. Li, T. Wang, D. Feng, Y. Zhou, and F. Ji, "Temporal Convolutional Network with Adaptive Diffusion Model for Generation–Load Probabilistic Forecasting," *Energies*, vol. 18, no. 23, p. 6179, 2025, https://doi.org/10.3390/en18236179.

[66] K. Gu and L. Jia, "Temporal Convolutional Network Based Short-term Load Forecasting Model," *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*, pp. 584-589, 2020, https://doi.org/10.1109/DDCLS49620.2020.9275148.

[67] N. T. N. Anh, D. T. Dat and L. A. Ngoc, "Reinforcement learning for optimization hyperparameters of Long Short-Term Memory applied to Electricity load forecasting," *2021 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT)*, pp. 1-6, 2021, https://doi.org/10.1109/ICMLANT53170.2021.9690555.

[68] K. T. Nguyen, T. N. Tran, and H. T. Nguyen, "Research on the influence of hyperparameters on the LightGBM model in load forecasting," *Engineering, Technology & Applied Science Research*, vol. 14, no. 5, pp. 17005-17010, 2024, https://doi.org/10.48084/etasr.

[69] T. T. Ngoc, C. M. T. Le Van Dai, and C. M. Thuyen, "Support vector regression based on grid search method of hyperparameters for load forecasting," *Acta Polytechnica Hungarica*, vol. 18, no. 2, pp. 143-158, 2021, https://doi.org/10.12700/APH.18.2.2021.2.8.

[70] A. Parizad and C. J. Hatziadoniu, "A Real-Time Multistage False Data Detection Method Based on Deep Learning and Semisupervised Scoring Algorithms," *IEEE Systems Journal*, vol. 17, no. 2, pp. 1753-1764, 2023, https://doi.org/10.1109/JSYST.2023.3265021.