Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Evaluating Hyperparameter Sensitivity in Temporal Convolutional Networks for Short-Term Electric Load Forecasting Using Grid Search

Tuan Anh Nguyen

nguyenanhtuan@iuh.edu.vn

Industrial University of Ho Chi Minh City

Thanh Ngoc Tran

Industrial University of Ho Chi Minh City

Research Article

Keywords: Short-term electric load forecasting, Temporal Convolutional Network, hyperparameter sensitivity analysis, grid search, NeuralForecast, deep learning

Posted Date: August 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10330697/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Evaluating Hyperparameter Sensitivity in Temporal Convolutional Networks for Short-Term Electric Load Forecasting Using Grid Search

Tuan Anh Nguyen [1,*], Thanh Ngoc Tran [2]
[1,2] Faculty of Electrical Engineering Technology, Industrial University of Ho Chi Minh City, Ho Chi Minh City, 700000, Vietnam.
*Corresponding author. E-mail: nguyenanhtuan@iuh.edu.vn

Abstract

Short-term electric load forecasting is essential for the reliable and efficient operation of modern power systems. Although Temporal Convolutional Networks have demonstrated strong capabilities for modeling sequential patterns, their forecasting performance can vary considerably with different hyperparameter settings. This study presents a systematic grid-search-based sensitivity analysis of Temporal Convolutional Network hyperparameters for one-day-ahead electric load forecasting using half-hourly demand data from Queensland, Australia. A reproducible forecasting framework implemented using the NeuralForecast library with a PyTorch backend evaluates 243 configurations generated from three candidate values for each of five hyperparameters: input size, learning rate, encoder hidden size, decoder hidden size, and batch size. All configurations are evaluated under the same chronological data partition and 48-step forecasting horizon using Mean Absolute Percentage Error as the selection criterion. The evaluated configurations produce MAPE values ranging from 1.841% to 5.455%, with an average of 3.499%. The best configuration, consisting of an input size of 336, a learning rate of 0.0005, an encoder hidden size of 64, a decoder hidden size of 128, and a batch size of 16, achieves a MAPE of 1.841%. This result represents a 55.05% reduction in error compared with the reference configuration, which yields a MAPE of 4.096%. The learning rate shows the clearest marginal effect, whereas increasing the number of hidden dimensions does not consistently improve forecasting accuracy. The results demonstrate that effective TCN performance depends on coordinated hyperparameter settings rather than simply increasing model capacity, providing practical guidance for configuring TCN-based short-term load forecasting models.

Keywords: Short-term electric load forecasting; Temporal Convolutional Network; hyperparameter sensitivity analysis; grid search; NeuralForecast; deep learning

1 Introduction

Accurate short-term electric load forecasting is essential for the secure, economical, and reliable operation of modern power systems. Forecasts of near-future electricity demand provide important information for generation scheduling, unit commitment, reserve allocation, energy trading, demand-response management, and the coordination of renewable energy resources. Underestimating electricity demand may result in inadequate generation or reserve capacity, whereas overestimating it can lead to unnecessary resource commitment and increased operating costs [1-3]. The growing penetration of distributed energy resources, electric vehicles, demand-side technologies, and weather-dependent renewable generation has further increased the variability and uncertainty of power-system operation, making accurate short-term forecasting increasingly important for real-time and day-ahead decision-making [4-6].

Electricity load data exhibit nonlinear, nonstationary, and multi-periodic characteristics. These properties make short-term load forecasting challenging because the underlying demand patterns may vary across different temporal scales and operating conditions [7-8]. A forecasting model must therefore learn both local variations and longer temporal dependencies while remaining sufficiently robust to changes in load behavior.

Short-term electricity demand is affected by recent consumption, intraday patterns, weekday–weekend differences, seasonal behavior, weather conditions, consumer activities, and irregular operating events [9-10]. The relative importance of these factors may vary across aggregation levels, consumer categories, and forecasting horizons. Consequently, the selected forecasting model and historical input representation must capture both recurring demand patterns and short-term fluctuations.

In addition to the forecasting architecture, prediction accuracy is strongly affected by model hyperparameters [11-12]. In neural forecasting models, the historical input length determines the amount of temporal information available to the model, the learning rate controls the magnitude of parameter updates, the hidden-layer dimensions determine representational capacity, and the batch size affects gradient estimation and training behavior. Inappropriate settings may lead to underfitting, unstable convergence, unnecessary model complexity, or poor generalization. Hyperparameter configuration should therefore be considered an integral component of forecasting-model development rather than a secondary implementation detail.

Short-term load forecasting has been investigated using a broad range of statistical, machine-learning, deep-learning, hybrid, and optimization-assisted approaches. Traditional statistical methods include linear and multiple regression [13-15], exponential smoothing [16], autoregressive models, autoregressive moving-average models, autoregressive integrated moving-average models [17], seasonal ARIMA models [18-19], and Kalman-filter-based techniques. These methods remain attractive because of their relatively simple mathematical formulation, low computational requirements,

and interpretability. They can provide reliable forecasts when the load series follows stable trends and seasonal patterns. However, their performance often depends on assumptions concerning linearity, stationarity, and predefined temporal structures. Consequently, statistical models may have difficulty representing complex nonlinear interactions, abrupt load variations, and multiple temporal patterns occurring at different time scales. Their performance may also deteriorate when the relationship between historical observations and future demand changes.

Machine-learning methods provide greater flexibility by learning nonlinear relationships from historical load observations and explanatory variables. Frequently applied approaches include artificial neural networks, multilayer perceptrons [20], support vector regression [21], decision trees [22], random forests [23], gradient boosting machines [24], extreme gradient boosting [25], LightGBM [26], CatBoost [27], and k-nearest-neighbor regression [28]. These models can incorporate lagged demand values, calendar information, temperature, humidity, rolling statistics, and other engineered predictors. Support vector regression is effective for nonlinear regression with limited samples, while tree-based ensemble methods can model complex feature interactions and are relatively robust to differences in data scale. Boosting methods such as XGBoost and CatBoost have also achieved competitive forecasting performance by sequentially correcting prediction errors and handling heterogeneous explanatory variables. Nevertheless, these approaches generally depend on carefully designed lagged and exogenous features. Their performance is sensitive to the selected input representation, model complexity, regularization settings, and optimization parameters. In multi-step forecasting, conventional machine learning models may also require separate models, recursive prediction, or specially constructed multi-output strategies.

Deep learning approaches reduce this dependence on manual feature engineering by learning temporal representations directly from sequential data. Recurrent neural networks [29], long short-term memory networks [30], gated recurrent units [31], convolutional neural networks [32], temporal convolutional networks [33], attention-based architectures, and Transformer-derived [34] models have all been applied to electric-load forecasting.

LSTM and GRU models are designed to capture temporal dependencies and alleviate the vanishing gradient problem in conventional recurrent networks. However, their sequential computation can limit parallelism during training, and their performance remains dependent on sequence length, hidden-state dimension, network depth, and optimization settings. CNN-based models provide efficient parallel computation and are effective at extracting local load patterns. However, their ability to represent long-term temporal relationships depends on network depth, receptive-field design, or the use of dilations. Transformer-based models use attention mechanisms to capture long-range relationships and interactions among time steps or variables. Still, they may require larger datasets and careful configuration of embedding dimensions, attention heads, learning rates, and regularization mechanisms. The increased representational capacity of deep models, therefore, does not automatically guarantee superior forecasting performance, particularly when their hyperparameters are selected empirically.

Hybrid forecasting methods have been developed to combine the complementary advantages of multiple techniques. Common combinations include CNN–LSTM [35], CNN–GRU [36], TCN–LSTM [37], bidirectional recurrent networks with attention [38], statistical–machine-learning ensembles, and combinations of decomposition algorithms with neural predictors. Wavelet transform, empirical mode decomposition, ensemble empirical mode decomposition, complete ensemble empirical mode decomposition with adaptive noise, variational mode decomposition, and singular-spectrum analysis are often used to separate the original load series into smoother components before forecasting. Ensemble and stacking methods combine predictions from several base models, whereas residual-correction approaches train a secondary model to compensate for the errors of an initial forecast. Other studies have employed genetic algorithms [39], particle swarm optimization, grey wolf optimization [40], Bayesian optimization [41], random search [42], and related metaheuristics to determine model configurations. These hybrid frameworks can improve accuracy by reducing noise, extracting multi-scale patterns, or combining diverse predictive behaviors. However, they usually introduce additional processing stages, parameters, and model components. This increases methodological complexity, makes it more difficult to isolate the contribution of each component, and may reduce reproducibility if the configuration procedure is not sufficiently detailed.

Among deep sequence models, the Temporal Convolutional Network offers an attractive alternative to recurrent architectures. A typical TCN employs causal convolution to ensure that predictions at a given time depend only on current and previous observations [43], [44]. Dilated convolution increases the receptive field without requiring a proportional increase in network depth, while residual connections facilitate information flow and support the training of deeper networks. In contrast to recurrent models, convolutional operations can be computed in parallel across a sequence. These characteristics make TCNs suitable for learning local changes, daily periodicity, and longer temporal dependencies in electric-load data. TCN-based and TCN-hybrid models have consequently been used in several load-forecasting applications and have demonstrated the ability to represent nonlinear temporal patterns.

Despite its advantages, TCN performance is highly sensitive to hyperparameter configuration. The input size determines the available historical context; the encoder and decoder hidden sizes control model capacity; the learning rate affects convergence; and the batch size influences gradient estimation and generalization. These parameters may also interact, meaning that larger input windows or hidden dimensions do not necessarily improve accuracy and may instead increase noise sensitivity or optimization difficulty [45].

Most previous studies focus on proposing new architectures or identifying a single best configuration through Grid Search, Random Search, Bayesian optimization, or metaheuristic methods. However, reporting only the best result provides limited insight into the individual and interaction effects of hyperparameters. Comparisons across studies are also difficult

because of differences in datasets, forecasting horizons, data splits, evaluation metrics, and training settings. Therefore, a controlled sensitivity analysis is needed to understand TCN behavior and provide reproducible configuration guidance [46].

To address this gap, this study evaluates five TCN hyperparameters: input size, learning rate, encoder hidden size, decoder hidden size, and batch size. Three candidate values are assigned to each parameter, resulting in 243 configurations. All models are assessed using the same chronological data split, training procedure, random seed, and 48-step forecasting horizon on half-hourly Queensland electricity-demand data. The models are implemented using the TCN available in Nixtla NeuralForecast with a PyTorch backend, and MAPE is the primary evaluation metric.

The experimental results demonstrate that TCN performance varies substantially across the evaluated hyperparameter space. The 243 configurations produce MAPE values ranging from 1.841% to 5.455%, with an average value of 3.499%. The best-performing configuration uses an input size of 336, a learning rate of 0.0005, an encoder hidden size of 64, a decoder hidden size of 128, and a batch size of 16, achieving a MAPE of 1.841%. In comparison, the reference configuration with an input size of 144, a learning rate of 0.001, encoder and decoder hidden sizes of 128, and a batch size of 32 obtains a MAPE of 4.096%. The best configuration, therefore, reduces the forecasting error by approximately 55.05% relative to the reference setting. The aggregated results also indicate that the learning rate yields the clearest marginal improvement in MAPE, whereas increasing the encoder or decoder hidden size does not yield consistent improvement. These findings confirm that effective TCN configuration depends on coordinated hyperparameter selection rather than simply increasing model capacity.

The principal contributions of this study are summarized as follows:

- A controlled and reproducible framework is established to investigate TCN hyperparameter sensitivity in 48-step-ahead short-term electric load forecasting.
- A full factorial Grid Search evaluates 243 combinations of input size, learning rate, encoder hidden size, decoder hidden size, and batch size under identical experimental conditions.
- The study examines both the individual influence and combined effects of the selected hyperparameters instead of reporting only the best-performing configuration.
- The analysis demonstrates quantitatively that inappropriate hyperparameter selection can produce substantial differences in forecasting accuracy and that larger hidden dimensions do not necessarily provide lower prediction errors.
- Practical evidence is provided to support the configuration of TCN models implemented through the NeuralForecast framework for short-term electric load forecasting applications.

2 Related Work

Short-term electric load forecasting has been investigated using statistical, machine-learning, and deep-learning methods. Traditional approaches, including regression, ARIMA, and exponential smoothing, remain useful for their simplicity and interpretability, but their performance may degrade when electricity demand exhibits nonlinear and time-varying behavior [47]. Machine-learning methods such as support vector regression, random forests, gradient boosting, and feedforward neural networks offer greater flexibility. However, they often depend strongly on manually selected lagged, calendar, and weather features [48].

Deep-learning models reduce this dependence by learning temporal representations directly from sequential data. LSTM, GRU, CNN, N-BEATS, and Transformer-based models have been widely applied to short-term and day-ahead load forecasting [49-50]. Comparative studies have shown that forecasting accuracy depends not only on the model architecture but also on the dataset, input representation, forecasting horizon, and training configuration [51]. Hybrid models combining convolutional, recurrent, attention-based, decomposition, and residual-correction components have also achieved competitive results. However, their additional processing stages and parameters make it difficult to isolate the contribution of each component [52-53].

Temporal Convolutional Networks have emerged as a competitive alternative to recurrent models. TCNs employ causal convolutions to prevent the use of future information, dilated convolutions to enlarge the receptive field, and residual connections to support stable training[54]. Their convolutional operations can be processed in parallel, making them suitable for representing both short-term fluctuations and longer temporal dependencies. TCNs have therefore been used as standalone predictors and as components of hybrid load-forecasting frameworks [55-56]

Despite these advantages, TCN implementations vary considerably across studies in terms of input length, hidden dimensions, dilation structure, kernel size, learning rate, and batch size. These settings directly affect model capacity, receptive field, convergence, and computational cost. Consequently, improvements in forecasting cannot always be attributed solely to the TCN architecture, as they may also result from favorable hyperparameter configurations [57].

Hyperparameter selection in load forecasting has been performed using manual tuning, Grid Search, Random Search, Bayesian optimization, Particle Swarm Optimization, genetic algorithms, and other metaheuristic methods [58-60]. Most studies focus on identifying the configuration that produces the lowest forecasting error. Although this strategy improves predictive performance, it provides limited information about the individual influence of each hyperparameter and the interactions among parameters [61].

Hyperparameter optimization and sensitivity analysis address different objectives. Optimization aims to identify the best-performing configuration, whereas sensitivity analysis examines how model performance changes across the investigated search space. In TCN forecasting, the effects of input size, learning rate, encoder hidden size, decoder hidden size, and batch size may not be independent. For example, the benefit of a longer input window may depend on the available encoder capacity, while larger hidden dimensions may require an appropriate learning rate to ensure stable optimization. Table 1 summarizes representative TCN-based load-forecasting

studies and highlights their main objectives for parameter selection and the extent of sensitivity analysis.

Table 1. Comparison of representative studies on TCN-based load forecasting and hyperparameter selection.

Ref.	Model	Forecast horizon	Parameter-selection focus	Sensitivity analysis
[54]	TCN	Multiple horizons	Comparison of architectures and configurations	Partial
[55]	Attention-based TCN	Day-ahead	Feature selection and model configuration	No
[56]	Multiscale TCN	Day- and week-ahead	Comparison of historical input scales	Limited
[62]	Multi-task TCN–LSTM	Short-term	Hybrid architecture and parameter sharing	No
[63]	Residual attentive LSTM–TCN	Short-term	Empirical hybrid-model configuration	No
[64]	Graph-TCN	Short-term	Architecture and module ablation	Limited
Present study	NeuralForecast TCN	48 steps	Full factorial Grid Search of five hyperparameters	Yes

As summarized in Table 1, previous TCN-based studies have mainly focused on architectural enhancement, feature selection, hybridization, or the identification of a high-performing configuration. Although some studies compared model variants or input scales, systematic investigation of the individual and interaction effects of TCN hyperparameters remains limited. To address this gap, the present study evaluates all 243 combinations of five TCN hyperparameters using a full factorial Grid Search on the same Queensland dataset, with a chronological data split, a 48-step forecasting horizon, a training procedure, a random seed, and an evaluation metric. Unlike studies that report only the best-performing configuration, this work analyzes both the marginal effects and interactions of the investigated hyperparameters, thereby providing more reproducible guidance for TCN configuration.

3 Methodology

This section presents the methodological framework for investigating the sensitivity of Temporal Convolutional Network hyperparameters in short-term electric load forecasting. The forecasting task is formulated as a multistep time-series prediction problem in which historical half-hourly electricity demand observations are used to forecast the following 48 time steps, corresponding to a one-day-ahead horizon. The TCN model is implemented using the NeuralForecast framework, and a full grid search is employed to systematically assess five key hyperparameters: input size, learning rate, encoder hidden size, decoder hidden size, and batch size. Each hyperparameter configuration is trained and evaluated on the same chronological data split, under the same experimental conditions, to ensure a fair comparison. Forecasting performance is assessed primarily using the Mean Absolute Percentage Error, while training and prediction times are also recorded to examine the relationship between forecasting accuracy and computational cost. The following subsections describe the problem formulation, TCN implementation, hyperparameter search space, grid-search-based sensitivity assessment, evaluation metric, and overall experimental workflow.

3.1 Problem Formulation

Let y_t denote the total electricity demand observed in Queensland at time step t. The demand data are recorded at 30-minute intervals and arranged chronologically. After the timestamp and load columns are converted into valid formats, missing or invalid observations are removed, and the remaining records are sorted by time. The resulting univariate electricity-demand series is expressed as:

$$\mathbf{Y} = \{y_1, y_2, \dots, y_t, \dots, y_N\} \tag{1}$$

where $\mathbf{Y}$ denotes the complete load series, y_t is the electricity demand at time step t, and N is the total number of valid observations. The forecasting task is formulated as a univariate multistep time-series problem, as the future load profile is predicted solely from historical demand values. Weather variables, calendar indicators, and other exogenous variables are not included in the model input.

At each forecast origin t, the TCN receives a fixed-length historical sequence containing the most recent L load observations. The input vector is defined as:

$$\mathbf{x}_t^{(L)} = [y_{t-L+1}, y_{t-L+2}, \dots, y_{t-1}, y_t]^{\mathrm{T}} \tag{2}$$

Here, $\mathbf{x}_t^{(L)}$ represents the historical input sequence, and L denotes the input size. The input size determines the amount of historical information available to the model. A shorter window emphasizes recent demand variations, whereas a longer window may include repeated daily and weekly patterns. Therefore, L is treated as a hyperparameter in the sensitivity analysis rather than being fixed beforehand.

The forecasting target associated with each input sequence consists of the subsequent H electricity-demand observations:

$$\mathbf{y}_t^{(H)} = [y_{t+1}, y_{t+2}, \dots, y_{t+H-1}, y_{t+H}]^{\mathrm{T}} \tag{3}$$

where $\mathbf{y}_t^{(H)}$ is the actual future-load vector and H is the forecasting horizon. In this study, H is fixed at 48. Since each observation represents 30 minutes, the 48 forecast steps correspond to a complete 24-hour period. The model therefore performs one-day-ahead load forecasting and produces the full demand profile for the following day.

The Temporal Convolutional Network learns a nonlinear mapping from the historical input sequence to the corresponding future-load vector:

$$\hat{\mathbf{y}}_t^{(H)} = f_{\boldsymbol{\theta}}\left(\mathbf{x}_t^{(L)}\right) \tag{4}$$

where $f_{\boldsymbol{\theta}}(\cdot)$ denotes the TCN forecasting function, $\boldsymbol{\theta}$ represents the trainable model parameters, and $\hat{\mathbf{y}}_t^{(H)}$ is the predicted 48-step load vector. All future values are generated in a single multistep forecasting operation rather than by recursively feeding each predicted value back into the model. This direct forecasting strategy avoids the progressive accumulation of recursive prediction errors across the 48-step horizon.

During training, the model parameters are estimated by minimizing the Mean Squared Error over all valid training samples and forecasting steps:

$$\mathcal{L}_{\mathrm{MSE}}(\boldsymbol{\theta}) = \frac{1}{KH}\sum_{k=1}^{K}\sum_{h=1}^{H}\left(y_{k,h} - \hat{y}_{k,h}\right)^2 \tag{5}$$

where K is the number of valid training samples, H is the forecasting horizon, $y_{k,h}$ is the actual load at forecasting step h for sample k, and $\hat{y}_{k,h}$ is the corresponding predicted load. MSE is used as the optimization loss because it is differentiable and places a larger penalty on large forecast deviations. The hyperparameter configurations themselves are evaluated and ranked using the Mean Absolute Percentage Error on the holdout test set, as described in Section 3.5.

To preserve the temporal dependency of the electricity-demand series and prevent future information leakage, the data are divided chronologically rather than randomly. The training set contains the 1,344 observations immediately preceding the test period, equivalent to 28 days of half-hourly demand data. The final 48 observations, corresponding to one complete day, are retained as the holdout test set. The same training and test periods are used for every grid search trial. Consequently, differences in forecasting accuracy are primarily attributable to the investigated TCN hyperparameters rather than to changes in data partitioning or evaluation conditions.

3.2 Temporal Convolutional Network Architecture and Neural Forecast Implementation

The forecasting model was implemented using the TCN class from NeuralForecast version 3.1.9, with PyTorch as the underlying deep learning backend. The selected implementation follows an encoder–adapter–decoder structure. First, a temporal convolutional encoder extracts representations from the historical load sequence. The encoded sequence is then transformed from the input length to the required 48-step forecasting horizon by a linear context adapter. Finally, a multilayer perceptron decoder converts the horizon-specific context vectors into predicted electricity demand values.

Because the experiment is univariate, the encoder receives only the historical Queensland demand series, and no historical, future, or static exogenous variables are supplied.

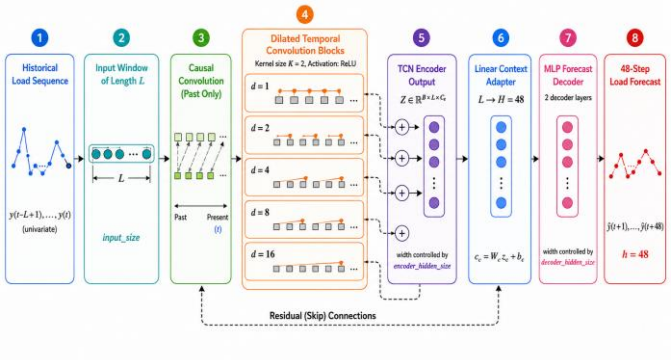


Fig. 1. Architecture of the Temporal Convolutional Network used for 48-step-ahead load forecasting.

As illustrated in Fig. 1, the model receives a univariate historical load sequence and extracts an input window of length L, where L is determined by the selected input_size. The window is processed by causal convolutions so that each representation depends only on the current and preceding observations. A stack of dilated temporal convolutional blocks with dilation factors 1, 2, 4, 8, and 16 progressively enlarges the receptive field while maintaining the temporal ordering of the input. Residual connections facilitate information propagation across the convolutional stack and support stable optimization. The resulting encoder output has dimensions $B \times L \times C_e$, where C_e is controlled by encoder_hidden_size. A linear context adapter then maps the encoded temporal dimension from L to the forecasting horizon $H = 48$, producing horizon-aligned context vectors. Finally, a shared two-layer MLP decoder, whose hidden width is controlled by decoder_hidden_size, converts these context vectors into the complete 48-step load forecast in a single forward pass.

For a training batch, the model input has the form $\mathbf{X} \in \mathbb{R}^{B \times L \times 1}$, where B denotes the number of series instances processed by the data loader, L is the selected input size. The final dimension corresponds to the single target variable. Before each convolution, NeuralForecast internally rearranges the tensor to the channel-first form required by one-dimensional convolution and restores the time-first arrangement after the encoder. This operation does not change the chronological order of the observations.

The main operation in the historical encoder is dilated causal convolution. For the convolutional layer ℓ, the hidden representation at time position t can be expressed as:

$$\mathbf{z}_t^{(\ell)} = \phi\left(\mathbf{b}^{(\ell)} + \sum_{k=0}^{K-1} \mathbf{W}_k^{(\ell)} \mathbf{z}_{t-kd_\ell}^{(\ell-1)}\right) \tag{6}$$

where $\mathbf{z}_t^{(\ell)}$ is the hidden output of layer ℓ, $\mathbf{z}_{t-kd_\ell}^{(\ell-1)}$ is the input from the preceding layer, $\mathbf{W}_k^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ are trainable weights and biases, K is the convolutional kernel size, d_ℓ is the dilation factor, and $\phi(\cdot)$ is the activation function. For the first layer, $\mathbf{z}^{(0)}$ corresponds to the historical load input. The index $t - kd_\ell$ ensures that the representation at position t depends only on the current and earlier observations. Future values are therefore excluded from the encoder computation, preventing information leakage.

Dilation increases the temporal distance between consecutive kernel elements. A dilation factor of one produces a conventional causal convolution. In contrast, larger values allow the filter to access more widely separated historical observations without increasing the number of trainable parameters proportionally. For a stack of M causal convolutional layers with kernel size K and dilation factors $d_1, d_2, \ldots, d_M$, the theoretical receptive field is:

$$R = 1 + (K-1)\sum_{\ell=1}^{M} d_\ell \tag{7}$$

where R is the number of historical positions covered by the deepest convolutional representation. In the implemented model, parameters not explicitly included in the grid search retain the NeuralForecast defaults. Accordingly, the kernel size is 2, the dilation sequence is [1, 2, 4, 8, 16], and the encoder uses the ReLU activation function. These settings produce five causal convolutional layers and a theoretical receptive field of 32 half-hourly positions. Although each deepest-layer representation is constructed from this dilated neighborhood, representations are generated across the complete input sequence, and the subsequent context adapter uses the full encoded length when producing the 48-step forecast context.

The width of every convolutional layer is controlled by encoder_hidden_size, denoted by C_e. After the final causal convolution, the encoder output has dimensions $\mathbf{Z} \in \mathbb{R}^{B \times L \times C_e}$.

larger-capacity C sub e arger C_e provides more channels for representing temporal patterns but also increases the number of model parameters and computational requirements. Because this parameter directly controls the historical encoder's representational capacity, it is included in the sensitivity analysis.

NeuralForecast then applies a linear context adapter independently to each encoder channel. The adapter maps the encoded temporal dimension from the historical input length L to the forecasting the horizon H. For encoder channel c, this transformation is written as:

$$\mathbf{c}_c = \mathbf{W}_c \mathbf{z}_c + \mathbf{b}_c, \qquad \mathbf{W}_c \in \mathbb{R}^{H \times L} \tag{8}$$

where $\mathbf{z}_c \in \mathbb{R}^L$ contains the encoded values of channel c across the complete input window and $\mathbf{c}_c \in \mathbb{R}^H$ is the resulting horizon-aligned context. In this study, $H = 48$, so the adapter converts each encoded channel into 48 context values, one for each half-hourly forecasting step. After rearranging the tensor dimensions, the context supplied to the decoder has the form $\mathbf{C} \in \mathbb{R}^{B \times H \times C_e}$.

The final predictions are produced by an MLP decoder applied separately to the context vector of each forecasting step. The NeuralForecast default of two decoder layers is retained. Since no future exogenous variables are used and the MSE loss produces one point estimate per step, the forecast at horizon position h can be represented as:

$$\hat{y}_{t+h} = \mathbf{w}_o^{\mathrm{T}} \mathrm{ReLU}(\mathbf{W}_d \mathbf{c}_h + \mathbf{b}_d) + b_o \tag{9}$$

where $h = 1,2,\dots,H$, $\mathbf{c}_h \in \mathbb{R}^{C_e}$ is the context vector for forecasting step h, $\mathbf{W}_d$ and $\mathbf{b}_d$ are the parameters of the decoder hidden layer, and $\mathbf{w}_o$ and b_o form the output projection. The dimension of the hidden decoder layer is controlled by decoder_hidden_size. The same decoder is shared across the 48 horizon positions, and its outputs collectively form the direct multistep forecast vector $\hat{\mathbf{y}}_t^{(H)}$. Thus, the model generates the complete next-day load profile in one forward pass rather than through recursive one-step prediction.

For every grid-search trial, the TCN was instantiated with h = 48, the selected values of input_size, learning_rate, encoder_hidden_size, decoder_hidden_size, and batch_size, an MSE training loss, and random_seed = 42. The model was wrapped in a NeuralForecast object with a sampling frequency of 30min. The fit method was applied to the training dataframe containing the required unique_id, ds, and y columns, and the predict method generated the subsequent 48 observations. No exogenous-variable lists were specified.

All non-searched model settings remained at their NeuralForecast defaults, including max_steps = 1000, decoder_layers = 2, windows_batch_size = 128, and robust temporal scaling. The default optimizer was used with the learning rate supplied by each grid-search configuration. No separate validation data frame, user-defined optimizer, learning rate scheduler, or early stopping rule was introduced in the code.

It is important to distinguish the two batching parameters used by NeuralForecast. The batch_size argument controls the number of separate time series included in a data-loader batch, whereas windows_batch_size controls the number of temporal windows sampled during a training update. The supplied implementation varied batch_size based on the search space, while keeping windows_batch_size fixed at its default value of 128. Because the experiment contains one QLD series, batch_size should therefore be interpreted according to the NeuralForecast API rather than as the conventional number of sliding-window samples.

To improve reproducibility, the random seed was fixed at 42 for Python, NumPy, and PyTorch. CUDA deterministic behavior was enabled by defining CUBLAS_WORKSPACE_CONFIG, enabling deterministic cuDNN execution, turning off cuDNN benchmarking, and limiting PyTorch to a single CPU thread. The same model-construction and forecasting procedure was repeated for all hyperparameter combinations so that performance differences reflected the investigated settings under otherwise identical implementation conditions.

3.3 Hyperparameter Search Space

The sensitivity analysis considered five TCN hyperparameters: the amount of historical information supplied to the model, the optimization step size, the representational capacities of the encoder and decoder, and the data-loader batching behavior. The hyperparameter vector represents a candidate configuration:

$$\boldsymbol{\lambda} = (L, \eta, C_e, C_d, B) \tag{10}$$

where L is input_size, η is learning_rate, C_e is encoder_hidden_size, C_d is decoder_hidden_size, and B is batch_size. These parameters were selected because they influence different stages of the forecasting process and can therefore affect predictive accuracy and computational demand differently.

Table 2. Hyperparameter search space used for the TCN sensitivity analysis

Hyperparameter	Symbol	Candidate values	Methodological meaning
input_size	L	96, 144, 336	Number of half-hourly historical observations supplied to the model. The three values correspond to 2, 3, and 7 days of load history, respectively.
learning_rate	η	0.0005, 0.001, 0.005	Step size used by the optimizer when updating the trainable TCN parameters.
encoder_hidden_size	C_e	64, 128, 256	The number of hidden channels in the temporal convolutional encoder, and therefore a control on its representation capacity.
decoder_hidden_size	C_d	64, 128, 256	Width of the hidden representation used by the MLP decoder to transform horizon-aligned temporal features into load forecasts.
batch_size	B	16, 32, 64	NeuralForecast data-loader series-batch parameter. The temporal-window batch size remained fixed at its default value and was not included in the search.

Let $\mathcal{S}_L$, $\mathcal{S}_\eta$, $\mathcal{S}_{C_e}$, $\mathcal{S}_{C_d}$, and $\mathcal{S}_B$ denote the candidate sets shown in Table 1. The complete hyperparameter domain is the Cartesian product:

$$\mathcal{G} = \mathcal{S}_L \times \mathcal{S}_\eta \times \mathcal{S}_{C_e} \times \mathcal{S}_{C_d} \times \mathcal{S}_B \tag{11}$$

Each candidate set contains three values. Consequently, the total number of unique configurations is:

$$N_{\mathcal{G}} = |\mathcal{G}| = 3 \times 3 \times 3 \times 3 \times 3 = 3^5 = 243 \tag{12}$$

The selected values provide low, intermediate, and high settings for each investigated factor while keeping the exhaustive experiment computationally manageable. In particular, the input-size candidates cover short recent histories and a complete weekly history; the learning-rate candidates span conservative to comparatively aggressive updates; and the encoder and decoder sizes allow the influence of model capacity to be examined without changing the remaining TCN architecture. All hyperparameters not listed in Table 2 were held constant at the NeuralForecast settings described in Section 3.2. Thus, observed performance differences were associated with the five-dimensional search space rather than simultaneous changes in unrelated implementation options.

3.4 Grid-Search-Based Hyperparameter Sensitivity Assessment

A full grid search was adopted to evaluate every configuration in $\mathcal{G}$. Unlike random or sequential optimization methods, exhaustive grid search applies the same predefined evaluation procedure to all possible combinations of the selected values. This property is useful for sensitivity assessment because each level of a hyperparameter is repeatedly evaluated across all combinations of the remaining factors, enabling both individual and interaction effects to be examined from a complete and balanced result set.

For each trial, one hyperparameter vector, the bold lambda element of script cap G, was assigned to the TCN model. The model was trained on the same 1,344-point chronological training period and then used to generate a direct 48-step forecast for the same holdout test day. The forecasting horizon, training loss, data frequency, random seed, and non-searched model settings remained unchanged across trials. No random train-test reshuffling, cross-validation, repeated-seed averaging, or separate validation-based model selection was applied. Therefore, the assessment represents a controlled comparison of hyperparameter configurations within a single fixed-chronological-forecasting experiment.

The execution procedure for each trial consisted of the following stages: model construction, model fitting, generation of the 48 future values, calculation of the stepwise errors and MAPE, recording of computational time, and storage of the trial outputs. Training time was measured around the fit operation, prediction time was measured around the predict operation, and total trial time covered the complete trial execution, including minor model-construction and result-processing overhead. The saved record for every successful trial contained the five hyperparameter values, MAPE, training time, prediction time, total time, random seed, and execution status. In addition, the 48 timestamps, actual loads, forecasts, absolute errors, and absolute percentage errors were retained, allowing the best configuration and horizon-specific behavior to be examined after the search.

Sensitivity to an individual hyperparameter was assessed by grouping completed trials by each candidate value, while allowing the other four hyperparameters to vary over their full candidate sets. Let $\mathcal{G}_{j,v}$ denote the subset of configurations in which hyperparameter j takes value v. The marginal mean forecasting error for that value is defined as:

$$\overline{E}_{j,v} = \frac{1}{|\mathcal{G}_{j,v}|} \sum_{\boldsymbol{\lambda} \in \mathcal{G}_{j,v}} \mathrm{MAPE}\,(\boldsymbol{\lambda}) \tag{13}$$

where $|\mathcal{G}_{j,v}|$ is the number of configurations containing value v. Because the grid is balanced, every candidate value of a given hyperparameter occurs in the same number of trials. The grouped MAPE distributions, including their central tendency, spread, minimum, and maximum, can therefore be compared without unequal representation. A large change in the grouped error distribution across candidate values indicates greater sensitivity, whereas substantial overlap suggests a weaker isolated effect within the tested range.

The complete grid also permits interaction analysis. Pairwise groupings can be used to determine whether the influence of one hyperparameter depends on the level of another. For example, the effect of a larger encoder may depend on the learning rate, while a long input window may require sufficient representation capacity. Accordingly, conclusions are not based solely on the lowest individual marginal error; joint configurations, ranked results, and top-performing combinations are also considered.

The best-performing configuration is selected solely according to the minimum test-set MAPE:

$$\boldsymbol{\lambda}^{*} = \arg\min_{\boldsymbol{\lambda} \in \mathcal{G}} \mathrm{MAPE}(\boldsymbol{\lambda}) \tag{14}$$

where $\boldsymbol{\lambda}^{*}$ is the selected TCN configuration. If multiple trials yield the same minimum MAPE, they are considered equivalent with respect to the primary selection criterion, while their computational times can serve as a secondary practical consideration. Since the experiment uses a single deterministic run per configuration, the resulting sensitivity analysis quantifies configuration-dependent variation under the fixed seed and data split; it does not estimate the uncertainty introduced by repeated stochastic training runs.

3.5 Forecasting Accuracy Metric

Mean Absolute Percentage Error (MAPE) was used as the primary forecasting-accuracy metric and as the objective for ranking the 243 TCN configurations. MAPE expresses the magnitude of the forecasting error relative to the corresponding actual demand, making the results interpretable on a percentage scale and comparable across different load levels within the test day.

For forecasting step h, the Absolute Percentage Error (APE) of configuration $\boldsymbol{\lambda}$ is calculated as:

$$\mathrm{APE}_h(\boldsymbol{\lambda}) = 100 \left| \frac{y_{t+h} - \hat{y}_{t+h}(\boldsymbol{\lambda})}{y_{t+h}} \right| \qquad h = 1,2,\dots,H \tag{15}$$

where y_{t+h} is the actual electricity demand, $\hat{y}_{t+h}(\boldsymbol{\lambda})$ is the forecast produced by the TCN configuration, and $H = 48$ is the day-ahead forecasting horizon. The absolute operator prevents positive and negative forecasting deviations from canceling each other out.

To avoid division by zero, the implementation calculates the average only over forecasting positions with nonzero actual demand. Let $\mathcal{J} = \{h \in \{1,\dots,H\} : y_{t+h} \neq 0\}$ denote the set of valid positions. MAPE is then defined as:

$$\mathrm{MAPE}(\boldsymbol{\lambda}) = \frac{1}{|\mathcal{J}|} \sum_{h \in \mathcal{J}} \mathrm{APE}_h\,(\boldsymbol{\lambda}) \tag{16}$$

where $|\mathcal{I}|$ is the number of valid test observations. For the investigated QLD electricity-demand series, the loads are positive, so the metric is normally calculated over all 48 forecasting steps. Lower MAPE values indicate greater forecasting accuracy, and a value of zero would represent a perfect forecast.

MAPE serves a different purpose from the Mean Squared Error used during model training. MSE provides the differentiable loss required to update the model parameters and gives greater weight to large absolute deviations. In contrast, MAPE is applied after prediction to compare configurations using an intuitive relative-error scale. The same MAPE implementation and the same 48-point holdout target are used for every trial, ensuring that the ranking is internally consistent. Although percentage errors can become unstable when actual values approach zero, this limitation is not critical for the present dataset because system-level electricity demand remains well above zero throughout the evaluated period.

3.6 Overall Experimental Workflow

The following figure presents the complete experimental workflow adopted to identify an appropriate Temporal Convolutional Network (TCN) configuration for 48-step-ahead short-term electricity load forecasting. The procedure is organized as an algorithmic flowchart beginning with the acquisition of half-hourly Queensland electricity demand data and continuing through data preprocessing, supervised-sample construction, chronological train–test splitting, hyperparameter-space definition, model training, forecasting, evaluation, and final result generation. The iterative section of the flowchart explicitly represents the grid search process, in which each candidate hyperparameter combination is trained and evaluated under the same experimental conditions.

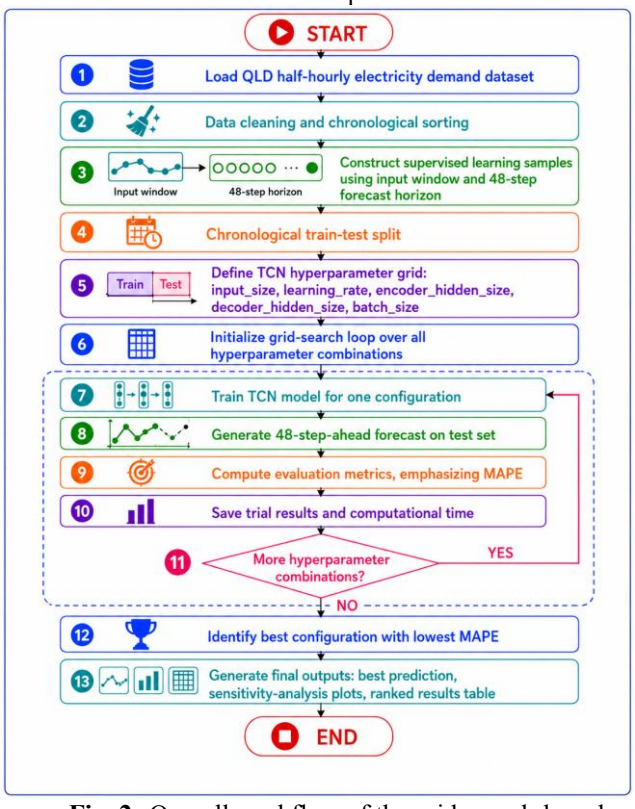


Fig. 2. Overall workflow of the grid-search-based hyperparameter sensitivity analysis for TCN forecasting.

The procedure begins by loading the half-hourly electricity demand series for Queensland. This dataset constitutes the primary input to the forecasting framework and provides observations at 30-minute intervals. Because the forecasting horizon is fixed at 48 steps, the model produces predictions for the entire subsequent 24-hour period. This setting is suitable for day-ahead operational planning while preserving intraday variations in electricity demand.

The second stage performs data cleaning and chronological sorting. Invalid timestamps, missing demand observations, and non-numeric values are removed or converted into valid formats. The remaining observations are then ordered by timestamp. Preserving chronological order is essential in time-series forecasting because future observations must not be used to train the model. Randomly shuffling the data could introduce information leakage and produce unrealistically optimistic forecasting results.

The cleaned time series is subsequently transformed into supervised learning samples. Each sample consists of an historical input window and a 48-step target sequence. The input_size hyperparameter controls the length of the historical window. Therefore, different configurations allow the TCN to use different amounts of historical information when predicting the next 48 half-hourly demand values. This formulation converts the original univariate time series into a multi-output supervised forecasting problem.

The dataset is then divided chronologically into training and testing subsets. Earlier observations are used to estimate the model parameters, whereas the most recent observations are reserved for out-of-sample evaluation. Unlike conventional machine-learning datasets, the split is not performed randomly. A chronological division more accurately reflects the practical forecasting situation, in which a model is trained on past measurements and then applied to previously unseen future demand values.

After the data have been prepared, the TCN hyperparameter grid is defined. Five principal hyperparameters are investigated: input_size, learning_rate, encoder_hidden_size, decoder_hidden_size, and batch_size. These parameters influence different aspects of the forecasting model. The input_size determines the historical context available to the network. The learning_rate controls the magnitude of parameter updates during optimization. The encoder and decoder hidden sizes determine the representational capacity of the corresponding network components, whereas the batch_size affects gradient estimation, memory consumption, and computational efficiency.

All possible combinations of the predefined hyperparameter values are generated to form the complete grid-search space. The procedure then initializes an iterative loop in which each candidate configuration is evaluated independently. This exhaustive strategy ensures that every combination within the specified search space is examined under the same dataset, forecasting horizon, loss function, and reproducibility settings.

For each iteration, a new TCN model is constructed using the current hyperparameter combination. The model is trained on the same chronological training subset. Within the TCN architecture, causal convolutions ensure that predictions at a given time depend only on current and past observations, not on future information. Dilated convolutions expand the network's

receptive field, enabling it to represent both short-term fluctuations and longer temporal dependencies without requiring an excessively deep architecture. Residual connections further facilitate stable optimization and improve information propagation across multiple temporal convolutional blocks.

Once training is complete, the fitted model generates a 48-step-ahead forecast for the test period. The predicted demand sequence aligns with the observed values, enabling forecasting errors to be calculated at every half-hourly step. This evaluation design assesses the model over the complete day-ahead horizon rather than only at a single future time point.

The resulting predictions are evaluated using forecasting error measures, with the Mean Absolute Percentage Error (MAPE) serving as the primary criterion for model selection. MAPE expresses the average absolute forecasting deviation relative to the observed demand and is therefore readily interpretable as a percentage. Its scale-independent form also facilitates direct comparison among all candidate configurations. Additional information, including point-wise errors, training time, prediction time, and total computational time, is retained to support a broader assessment of forecasting performance and computational cost.

The outputs of each trial are then stored. These outputs include the trial identifier, hyperparameter values, forecasted demand values, actual observations, absolute errors, absolute percentage errors, MAPE, and computational-time measurements. Retaining complete trial-level information is important because the study's purpose is not limited to identifying the single best configuration. The stored results also enable systematic sensitivity analysis of how individual hyperparameters and their interactions affect TCN forecasting accuracy.

The decision node determines whether additional hyperparameter combinations remain unevaluated. When untested configurations are still available, the procedure returns to the model-training stage and evaluates the next combination. This iterative cycle continues until the entire grid-search space has been exhausted. Consequently, the final comparison is based on a complete and consistent evaluation of all candidate configurations rather than on a randomly selected subset.

After all trials have been completed, the configuration producing the lowest MAPE is identified as the best-performing TCN within the predefined search space. This selection should be interpreted as conditional on the dataset, the forecasting horizon, the search ranges, and the experimental protocol used in the study. It does not imply that the configuration is universally optimal for every electricity-demand series or forecasting task.

Finally, the framework generates the principal study outputs, including the best forecast, the ranked configuration table, hyperparameter-sensitivity plots, interaction analyses, error-distribution figures, and computational trade-off results. These outputs support both predictive evaluation and methodological interpretation. In particular, they enable determining not only which TCN configuration performs best, but also which hyperparameters exert the greatest influence on forecasting accuracy and whether increased computational requirements accompany improvements in MAPE.

Overall, the flowchart demonstrates a sequential, reproducible, and controlled experimental framework. The first part establishes reliable time-series preparation, the central part implements exhaustive hyperparameter evaluation, and the final part performs model selection and sensitivity analysis. This structure reduces the risk of data leakage, ensures fair comparison among configurations, and provides a transparent basis for evaluating TCN hyperparameter effects in 48-step-ahead short-term load forecasting.

4 Experimental Design

This section describes the dataset, data preparation procedure, hyperparameter domain, reference configuration, model training settings, and the computational environment used in the experiments. The design was constructed as a controlled full-factorial study. Every TCN configuration was trained on the same chronological data window, evaluated on the same one-day holdout period, and executed under identical fixed settings except for the five hyperparameters included in the grid search. This protocol isolates configuration-dependent changes in forecasting accuracy and computational demand.

4.1 Dataset Description

The experiments used the Queensland regional electricity-demand series contained in the QLD.csv file. The two fields required for the analysis were SETTLEMENTDATE, which identifies the settlement interval, and TOTALDEMAND, which reports total regional demand in megawatts. The observations are nominally sampled every 30 minutes; therefore, a complete day contains 48 load measurements.

After invalid timestamp and load entries were removed, the dataset contained 122,735 valid observations covering the period from 1 January 2015 at 00:30 to 31 December 2021 at 23:30. Across the complete series, electricity demand ranged from 3,687.58 MW to 9,988.09 MW, with an average value of 6,170.49 MW. These values indicate a broad operating range and provide a long historical context for visualizing changes in the magnitude and temporal structure of Queensland electricity consumption.

Table 3. Summary of the Queensland electricity-demand dataset

Characteristic	Value	Meaning
Region	Queensland (QLD)	Regional electricity-demand series used in the experiment.
Target variable	TOTALDEMAND	Total electricity demand expressed in megawatts.
Sampling interval	30 minutes	Two observations per hour and 48 observations per complete day.
Valid observations	122,735	Number of records remaining after validity filtering.
Data period	2015-01-01 00:30 to 2021-12-31 23:30	Chronological coverage of the complete cleaned series.
Minimum demand	3,687.58 MW	Lowest observed total demand in the complete series.
Maximum demand	9,988.09 MW	Highest observed total demand in the complete series.
Mean demand	6,170.49 MW	Arithmetic mean of all valid load observations.

Figure 3 presents the complete cleaned half-hourly load series. The full record is used to describe the dataset and its long-term variation, whereas the controlled forecasting experiment uses the final contiguous 29-day segment defined in Section 4.2.

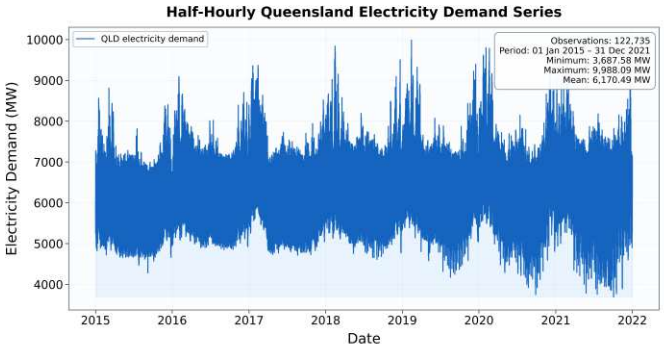


Fig. 3. Half-hourly electricity demand series of Queensland used in this study.

4.2 Data Preprocessing

Data preparation followed a deterministic chronological procedure. First, SETTLEMENTDATE was converted to a datetime representation using the month/day/year hour:minute format. Entries that could not be converted were marked as invalid. TOTALDEMAND was independently converted to a numeric variable, and rows with an invalid timestamp or load value were removed. The remaining observations were then sorted in ascending temporal order and their row indices were reset. No random shuffling, synthetic interpolation, or exogenous-feature construction was applied.

For the full grid-search experiment, the final 1,392 consecutive observations were retained. The last 48 observations formed the holdout test set, and the immediately preceding 1,344 observations formed the training set. At the 30-minute sampling frequency, the number of observations per day and in each partition are related as follows:

$$N_{day} = 24/0.5 = 48 \quad (17)$$

$$N_{train} = 28N_{day} = 1{,}344 \quad (18)$$

$$N_{test} = N_{day} = 48 \quad (19)$$

where N_day is the number of half-hourly observations in one day, N_train is the training-set size, and N_test is the test-set size. The training set therefore covers 28 complete days from 3 December 2021 at 00:00 to 30 December 2021 at 23:30. The test set covers the following complete day, from 31 December 2021 at 00:00 to 23:30. Inspection of the selected modeling interval confirmed a constant 30-minute spacing between consecutive observations.

Table 4. Chronological training and testing periods

Set	Period	Observations	Duration	Load range and mean
Training	2021-12-03 00:00 to 2021-12-30 23:30	1,344	28 days	4,601.66-9,117.43 MW; mean 6,294.90 MW
Testing	2021-12-31 00:00 to 2021-12-31 23:30	48	1 day	5,255.11-7,135.83 MW; mean 5,979.37 MW

The training data were subsequently reformatted according to the NeuralForecast input structure. A constant series identifier, unique_id = QLD, was assigned to all observations; the timestamp column was renamed ds; and TOTALDEMAND was renamed y. Only this training dataframe was passed to the model-fitting operation. The actual test loads were kept outside the fitting process and used only after the model generated its 48-step forecast. Manual normalization was not performed before constructing the dataframe because the TCN implementation applied its fixed robust scaler internally.

The chronological arrangement shown in Fig. 4 prevents future test information from entering model estimation. The same training and test timestamps were used across all 243 trials, and no validation-based model selection or random train-test partitioning was employed.

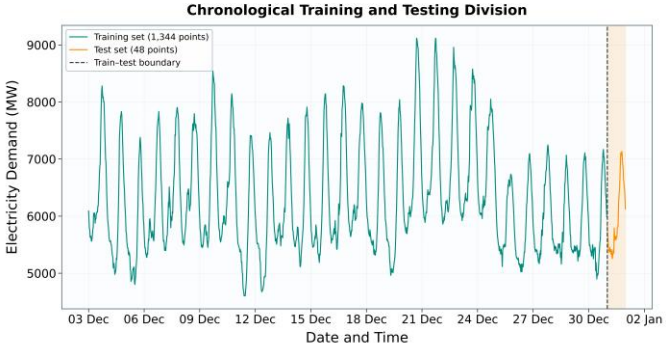


Fig. 4. Chronological division of the QLD load series into training and testing periods.

4.3 Hyperparameter Search Space

Table 5 specifies the exact factor levels evaluated in the experiment. Three candidate values were assigned to each of the five investigated hyperparameters. The input sizes of 96, 144, and 336 observations correspond to 2, 3, and 7 days of half-hourly history, respectively. The remaining factors control the optimization step size, encoder capacity, decoder capacity, and data loader batch size. The Cartesian product of the five three-level sets produced 243 unique TCN configurations, as formulated in Section 3.3.

Table 5. Hyperparameter levels used in the full grid search

Hyperparameter	Candidate values	Experimental interpretation
input_size	96, 144, 336	Historical context of 2, 3, or 7 days.
learning_rate	0.0005, 0.001, 0.005	Conservative, intermediate, and relatively aggressive parameter-update steps.
encoder_hidden_size	64, 128, 256	Low, intermediate, and high encoder representation capacities.
decoder_hidden_size	64, 128, 256	Low, intermediate, and high decoder representation capacities.
batch_size	16, 32, 64	Alternative NeuralForecast series-level loader batch settings.

All 243 combinations were evaluated without pruning, adaptive sampling, or early rejection. Parameters not listed in Table 5 were held constant at the values reported in Section 4.5. Consequently, the resulting dataset of trial outcomes is balanced: each candidate value of a given hyperparameter occurs with equal frequency across the entire experiment.

4.4 Reference TCN Configuration

A reference configuration was defined using the intermediate level of each investigated hyperparameter. This center-point configuration provides a consistent descriptive anchor for comparing changes toward lower or higher settings and is included as one ordinary trial within the complete grid. It is not assumed to be optimal, does not replace the exhaustive search,

and should not be confused with the best configuration selected from the test-set MAPE results.

Table 6. Reference TCN configuration

Setting	Reference value	Rationale or role
input_size	144	Three days of historical half-hourly demand; center level of the tested set.
learning_rate	0.001	Intermediate optimization step size and NeuralForecast default value.
encoder_hidden_size	128	Intermediate encoder capacity and NeuralForecast default value.
decoder_hidden_size	128	Intermediate decoder capacity and NeuralForecast default value.
batch_size	32	Intermediate tested value and NeuralForecast default series-batch value.
Forecast horizon	48	Fixed one-day-ahead output for all configurations.
Training loss	MSE	Fixed differentiable point-forecast objective.
Random seed	42	Fixed initialization and data-processing seed.

Because the reference vector uses only values within the search space, it is trained and assessed using the same procedure as all other trials. Thus, any comparison with the reference configuration does not introduce an additional model, dataset split, or evaluation condition.

4.5 Model Training Configuration

For each grid point, a new TCN instance was constructed and trained from scratch using the NeuralForecast framework. The model generated all 48 future values directly, and MSE was used as the training loss. Training was limited to 1,000 optimization steps for every trial. No early stopping, separate validation set, learning rate scheduler, repeated-seed averaging, warm start, or parameter transfer between trials were used.

The five searched hyperparameters were supplied to the TCN constructor for each trial, whereas the remaining architecture and training options were fixed. In particular, the encoder used a kernel size of 2, ReLU activation, and the dilation sequence [1, 2, 4, 8, 16]. The decoder contained two layers. Robust scaling was applied internally, and the temporal window batch size remained fixed at 128. This distinction is important because the searched batch_size parameter is the NeuralForecast series-level loader setting, whereas windows_batch_size controls the number of temporal windows used during optimization.

Table 7. Fixed TCN training and architecture settings

Parameter or option	Value	Role in the experiment
freq	30min	Defines the half-hourly timestamp frequency.
h	48	Produces one complete day of future demand values.
loss	MSE	Training objective for point forecasting.
max_steps	1,000	The common maximum number of optimization steps for every trial.
kernel_size	2	Width of each temporal convolutional kernel.
dilations	[1, 2, 4, 8, 16]	Expands the temporal receptive field across encoder layers.
encoder_activation	ReLU	Nonlinear activation used by the encoder.
context_size	10	Fixed context representation setting used by the implementation.
decoder_layers	2	Number of layers in the MLP decoder.
scaler_type	robust	Internal scaling that reduces sensitivity to extreme load values.
windows_batch_size	128	Number of temporal windows sampled per training update; not searched.
inference_windows_batch_size	1,024	Window batch setting used during inference.
step_size	1	Stride between consecutive training windows.
early_stop_patience_steps	-1	Turns off early stopping.
Learning-rate scheduler	None	No scheduled learning-rate decay was applied.
Exogenous variables	None	The experiment used only historical load values.

After fitting, NeuralForecast generated the next 48 timestamps and their associated TCN predictions. The predicted column was identified by its TCN model name, renamed y_pred for storage, and aligned with the 48 holdout observations. The model object was discarded before the next configuration was initialized, ensuring that each trial began with an independent model instance under the same fixed seed.

4.6 Implementation and Reproducibility Settings

The experiment was implemented in Python in a GPU-enabled Google Colab environment. The recorded software and hardware versions are provided in Table 8. CUDA acceleration was available, and all TCN trials were executed on an NVIDIA A100-SXM4-80GB GPU.

Table 8. Software and hardware environment

Component	Recorded setting
Python	3.12.13
NumPy	2.0.2
Pandas	2.2.2
PyTorch	2.11.0+cu128
NeuralForecast	3.1.9
Computing device	NVIDIA A100-SXM4-80GB
CUDA availability	True
Execution platform	Google Colab with Google Drive storage

A common seed of 42 was assigned to the Python random module, NumPy, PyTorch, and all available CUDA devices. PYTHONHASHSEED was set to 42, and CUBLAS_WORKSPACE_CONFIG was set to :4096:8 before model execution. CuDNN deterministic behavior was enabled, CuDNN benchmarking was disabled, and the number of PyTorch CPU threads was restricted to one. The TCN constructor also received random_seed = 42 in every trial. These controls reduce variability arising from initialization, algorithm selection, and multithreaded execution.

The stricter torch.use_deterministic_algorithms(True) option was not forcibly enabled because it can reject operations for which a deterministic CUDA implementation is unavailable. Accordingly, the reported settings support strong run-to-run reproducibility in the recorded environment. However, bitwise-identical outputs cannot be guaranteed after changes in hardware, CUDA libraries, or package versions.

Training, prediction, and total trial durations were measured separately using time.perf_counter. After each trial, the model

and NeuralForecast objects were deleted, Python garbage collection was invoked, and the CUDA cache was cleared when a GPU was available. For auditability, the output workbook stored the run information, training and test records, search-space definition, unsorted and MAPE-ranked trial summaries, the selected best configuration, the best 48-step forecast, and all trial-level forecasts and errors. Each trial record also retained its hyperparameter values, MAPE, timing measurements, random seed, and execution status.

5 Results and Discussion

This section evaluates the forecasting behavior of all 243 TCN configurations from complementary perspectives. The analysis first examines the overall error landscape and distribution of MAPE values, then quantifies the marginal influence of each investigated hyperparameter. Pairwise interaction effects are subsequently considered to determine whether individual parameter values remain beneficial when combined with other settings. The discussion then evaluates the relationship between forecasting accuracy and computational time before examining the 48-step predictions produced by the best-performing TCN configuration. This organization links the global search results to parameter-level explanations and, finally, to the practical behavior of the selected forecasting model.

5.1. Overall Grid-Search Performance

The first perspective is provided by Fig. 5, which orders all 243 completed trials from the lowest to the highest test-set MAPE. The ranked curve reveals the full range of accuracy across the predefined hyperparameter space. It shows whether the best result is isolated or part of a broader group of competitive configurations.

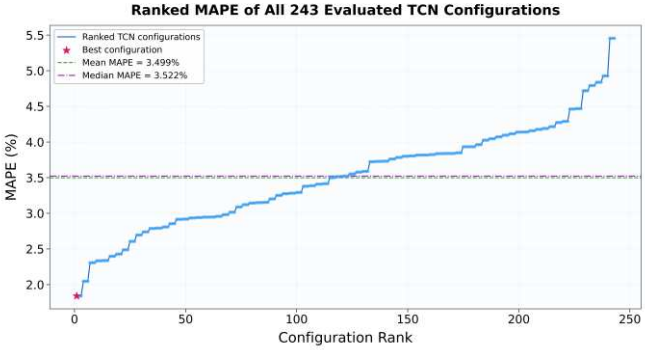


Fig. 5. Ranked MAPE values of all evaluated TCN hyperparameter configurations.

As shown in Fig. 5, forecasting accuracy varies substantially across the evaluated configurations. MAPE ranges from 1.8409% to 5.4553%, producing a difference of 3.6144 percentage points between the best and worst trials. The mean and median errors are 3.4993% and 3.5220%, respectively, indicating that a typical configuration lies near 3.5% MAPE. The ranked curve increases gradually through most of the search space but becomes noticeably steeper near both extremes. This pattern indicates that a limited set of configurations provides clearly superior forecasts, while another small group produces pronounced degradation. The minimum MAPE is shared by three configurations that differ only in the tested series-level batch_size, whereas the reference configuration obtains 4.0955%. Consequently, the minimum-error setting reduces MAPE by 55.05% relative to the reference configuration. These results demonstrate that the TCN architecture alone does not determine forecasting performance; the selected hyperparameter combination has a major influence on the final accuracy.

Although the ranked curve establishes the magnitude of configuration-dependent variation, it does not show how densely the trials are concentrated around typical error levels. The distributional view in Fig. 6 therefore provides the next layer of evidence.

Figure 6 summarizes the frequency distribution of MAPE across the complete grid and marks the best result, mean, and median. This representation is useful for determining whether a few isolated trials drive the observed range or whether the observed range reflects broad dispersion throughout the search space.

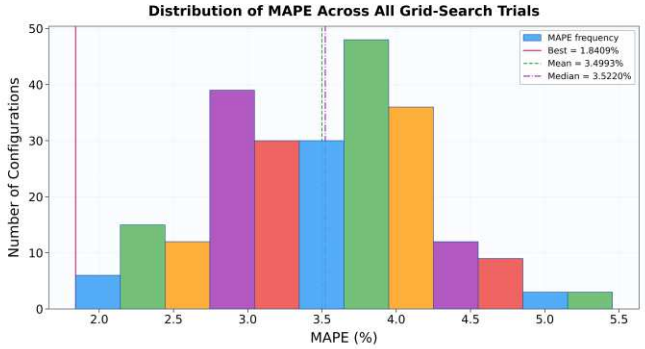


Fig. 6. Distribution of forecasting MAPE obtained from all TCN configurations.

The histogram confirms that most configurations are concentrated between approximately 2.8% and 4.2% MAPE, while relatively few trials occur at the extremes of the range. The small difference between the mean of 3.4993% and the median of 3.5220% suggests that the distribution is reasonably symmetric, although the best configuration lies far from the main concentration of trials. The sample standard deviation is 0.7053 percentage points, which is large relative to the best MAPE and confirms meaningful sensitivity to the investigated settings. In practice, selecting a configuration near the center of the search space without systematic assessment could easily yield an error of 3.5% or higher, despite the availability of configurations below 2%. Having established this global variability, the analysis now separates the contributions of the five investigated hyperparameters.

5.2. Marginal Hyperparameter Sensitivity

Figure 7 evaluates the influence of input_size by aggregating the MAPE values obtained from all model configurations that use 96, 144, or 336 historical observations, while the remaining four hyperparameters are allowed to vary across their predefined candidate values. This aggregation provides an overall assessment of how the length of the historical input window affects forecasting accuracy under different model settings, rather than relying on the result of a single configuration. Because the electricity-demand data are recorded at 30-minute intervals, the three selected input sizes correspond to 2, 3, and 7 days of historical demand information, respectively. These window lengths therefore represent short-,

medium-, and longer-term historical contexts used by the model to generate its forecasts.

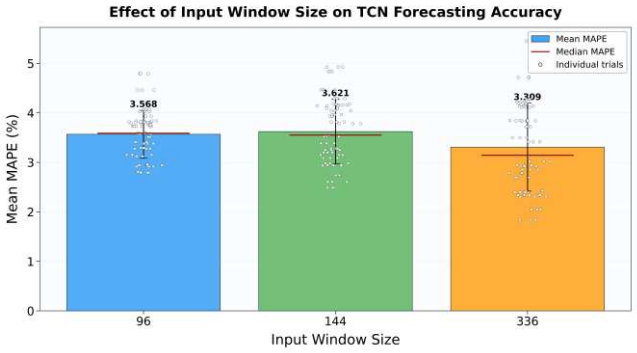


Fig. 7. Effect of input window size on TCN forecasting accuracy.

The seven-day input window provides the lowest marginal mean MAPE of 3.3087%, compared with 3.5678% for 96 observations and 3.6214% for 144 observations. This result suggests that a complete weekly history can supply useful recurring information that is not fully represented by two- or three-day windows. However, the effect is not monotonic: increasing the input from 96 to 144 observations slightly worsens the mean error rather than improving it. Moreover, the 336-step group has the largest dispersion, with errors ranging from 1.8409% to 5.4553%. Thus, a long historical context offers the greatest performance potential but also becomes more dependent on the accompanying learning rate and model-capacity settings. Input length should therefore be selected jointly with the optimization configuration rather than treated as an independently beneficial increase in historical information.

Because the usefulness of a longer window depends on whether the model can learn effectively from that information, the optimization step size is examined next.

Figure 8 compares the grouped MAPE distributions for the three learning-rate values. This parameter directly controls the magnitude of each optimizer update and can therefore affect convergence stability and the quality of the final solution.

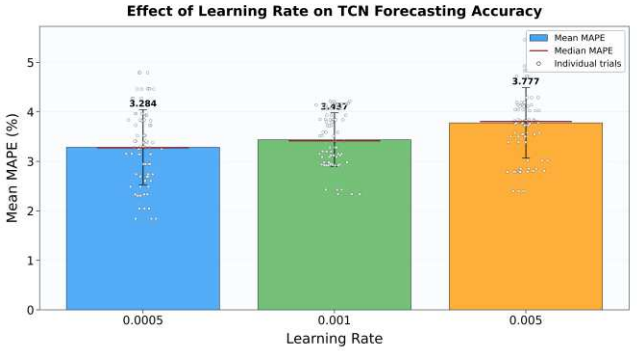


Fig. 8. Effect of learning rate on TCN forecasting accuracy.

Learning rate exhibits the clearest marginal trend among the investigated hyperparameters. The smallest value, 0.0005, achieves the lowest mean MAPE of 3.2842%, followed by 3.4367% at 0.001 and 3.7771% at 0.005. The progressive increase in average error indicates that the relatively aggressive update size of 0.005 is generally unsuitable for this TCN forecasting task under the fixed 1,000-step training budget. A smaller learning rate appears to support more stable parameter refinement and is also present in the minimum-MAPE configuration. Nevertheless, the dispersion at 0.0005 remains substantial, showing that a conservative update size cannot compensate for every unfavorable combination of input length and hidden dimensions. The marginal pattern, therefore, identifies the learning rate as a strong driver of accuracy, while the interaction analysis in Section 5.3 determines when this advantage is preserved.

The discussion now shifts from optimization behavior to the representational capacity of the temporal encoder. Figure 9 groups the trials by encoder_hidden_size, which determines the number of channels used to represent temporal information after causal and dilated convolutions.

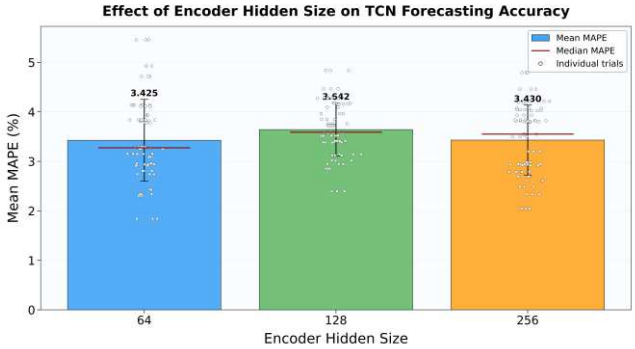


Fig. 9. Effect of encoder hidden size on TCN forecasting accuracy.

The encoder results do not support the assumption that a wider representation consistently improves forecasting accuracy. The mean MAPE is 3.4254%, 3.6423%, and 3.4303% for encoder sizes of 64, 128, and 256, respectively. The intermediate size of 128 performs worst on average, whereas the smallest and largest settings produce nearly identical marginal means. The 64-channel encoder includes both the global best and global worst trials, which explains its comparatively large variability and demonstrates that its effectiveness is strongly conditional on the other hyperparameters. The best configuration uses only 64 encoder channels, indicating that a compact temporal representation is sufficient when paired with a long input window, a small learning rate, and a suitable decoder. Increasing the encoder width to 256 may partially recover lost accuracy in some combinations. Still, it provides no systematic advantage that would justify selecting it solely based on model capacity.

A related but distinct question is whether expanding the decoder improves the transformation of encoded temporal features into the 48-step forecast vector.

Figure 10 presents the marginal effect of decoder_hidden_size by summarizing the forecasting performance of all configurations associated with each candidate decoder dimension while allowing the remaining hyperparameters to vary. The decoder_hidden_size determines the width and representational capacity of the nonlinear mapping that transforms the horizon-aligned features extracted by the TCN into the final electricity-load predictions. A smaller decoder may provide a more compact transformation with lower model complexity, whereas a larger decoder offers greater capacity to capture nonlinear relationships between the

extracted temporal features and the predicted load values. Therefore, this analysis helps determine whether increasing the decoder width consistently improves forecasting accuracy or instead introduces unnecessary model complexity..

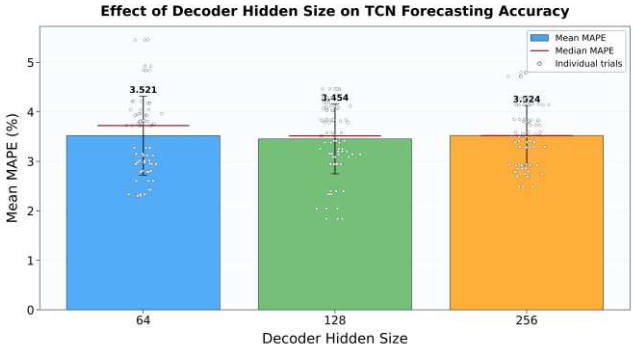


Fig. 10. Effect of decoder hidden size on TCN forecasting accuracy.

The decoder-size effect is comparatively weak. Mean MAPE is 3.5207% for 64 units, decreases modestly to 3.4535% for 128 units, and returns to 3.5237% for 256 units. The central setting, therefore, provides the lowest average error, but the difference between the best and worst marginal means is only about 0.07 percentage points. The result again contradicts a simple "larger is better" interpretation: doubling the decoder width from 128 to 256 does not improve accuracy and instead produces a slightly higher mean MAPE. The global best configuration uses 128 decoder units, suggesting that moderate output-mapping capacity is adequate for converting the compact 64-channel encoder representation into a one-day-ahead forecast.

The final marginal comparison concerns batch_size; however, its interpretation requires attention to the specific NeuralForecast data-loading mechanism used in this study.

Figure 11 compares the three tested batch_size values. In NeuralForecast, this argument controls the number of separate series processed in a loader batch, whereas windows_batch_size controls the number of temporal windows used during an optimization update.

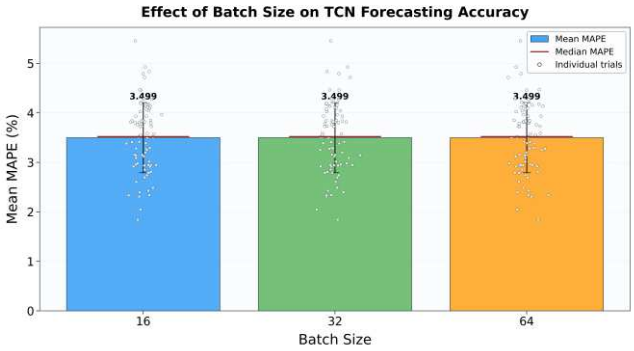


Fig. 11. Effect of batch size on TCN forecasting accuracy.

All three batch_size values produce the same mean MAPE (3.4993%), median (3.5220%), and minimum and maximum errors. This equality occurs because the experiment contains only one QLD series; therefore, changing the series-level loader capacity from 16 to 32 or 64 does not change the effective training data presented to the model. The identical MAPE values across the three settings also explain why the three globally best trials differ only in batch_size. This result should not be generalized to conventional window batching or multi-series datasets, where batch construction can affect gradient estimates and optimization behavior. In the current implementation, the findings instead show that windows_batch_size—not the searched series-level batch_size—would be the more meaningful batching parameter for a future single-series sensitivity study.

The marginal results identify broad tendencies, but they cannot explain why some settings with favorable averages still produce poor individual trials. Pairwise heatmaps are therefore used to reveal how the effects change when hyperparameters are combined.

5.3. Hyperparameter Interaction Effects

Figure 12 examines the joint influence of input_size and learning_rate by averaging over all encoder, decoder, and batch-size combinations for each pair of values. The heatmap tests whether the preferred learning rate remains consistent across different amounts of historical context.

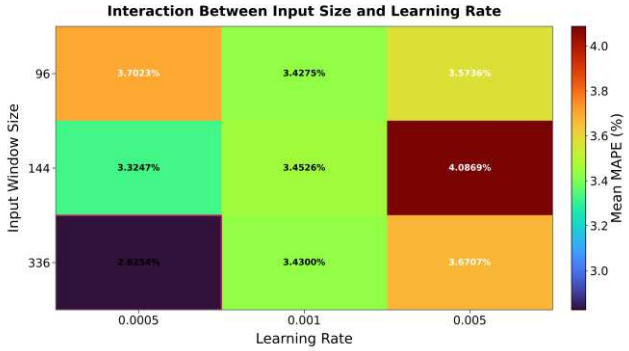


Fig. 12. Interaction effect of input size and learning rate on forecasting MAPE.

The interaction is pronounced. The lowest pairwise mean MAPE, 2.8254%, occurs when the weekly input window of 336 observations is combined with the smallest learning rate of 0.0005. For the 336-step window, increasing the learning rate to 0.001 and 0.005 raises the mean error to 3.4300% and 3.6707%, respectively. A similar deterioration is observed for the 144-step window, where the combination with 0.005 produces the highest cell mean of 4.0869%. In contrast, the 96-step window performs better at 0.001 than at 0.0005, with mean MAPEs of 3.4275% and 3.7023%, respectively. This exception confirms that the marginal advantage of the smallest learning rate is not universal. The main implication is that additional historical information becomes beneficial only when optimization proceeds with a sufficiently controlled step size; a long input sequence paired with an overly aggressive learning rate fails to exploit the available temporal context.

While Fig. 12 links temporal context to optimization behavior, Fig. 13 evaluates whether encoder and decoder capacity should be increased together or arranged asymmetrically.

Figure 13 reports the mean MAPE obtained for each combination of encoder_hidden_size and decoder_hidden_size, with the results averaged across all tested values of input size, learning rate, and batch size. This aggregation isolates the joint effect of the encoder and decoder dimensions while reducing the influence of the remaining hyperparameters. The figure therefore provides a systematic comparison of how different

capacity allocations between the encoder and decoder affect forecasting accuracy. By examining the complete set of encoder–decoder combinations, it is possible to determine whether balanced hidden dimensions, a larger encoder, or a larger decoder produces more reliable performance across the broader hyperparameter search space..

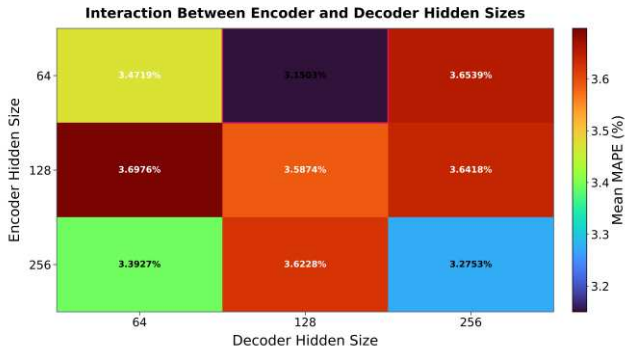


Fig. 13. Interaction effect of encoder and decoder hidden sizes on forecasting MAPE.

The best encoder–decoder pairing is 64–128, with an average MAPE of 3.1503%. This asymmetric structure combines a compact temporal encoder with a moderately sized decoder and is also the hidden-size combination used by the global minimum-MAPE configuration. The second-lowest cell mean is obtained by the 256–256 combination at 3.2753%, showing that a larger balanced architecture can also be competitive, although it does not surpass the more compact 64–128 design. The poorest pair is 128–64, which yields 3.6976% MAPE. Furthermore, the three equal-width designs do not follow a monotonic pattern: 64–64, 128–128, and 256–256 obtain 3.4719%, 3.5874%, and 3.2753%, respectively. These results demonstrate that representational balance is governed by interaction rather than by an absolute hidden dimension. In particular, increasing only one component may create an inefficient mapping between encoded features and forecast outputs.

After identifying the combinations that improve accuracy, the next analysis considers whether these gains require a meaningful increase in computational time.

5.4. Accuracy–Computational-Cost Trade-off

Figure 14 plots test-set MAPE against total trial time for all configurations and highlights the Pareto frontier, the fastest trial, and a minimum-MAPE trial. Point color represents the input window size.

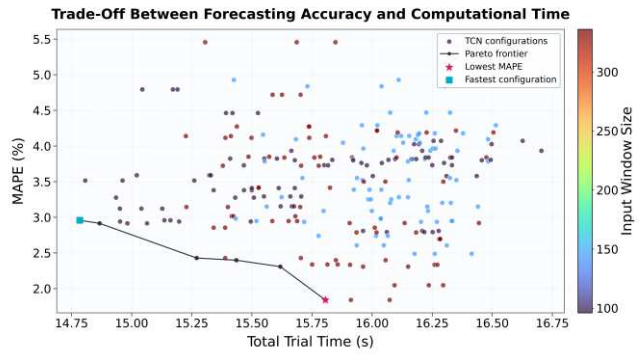


Fig. 14. Trade-off between forecasting accuracy and computational time across TCN configurations.

Total execution time ranges from approximately 14.784 to 16.704 s, while the correlation between total time and MAPE is only 0.034. Thus, longer computation does not systematically lead to greater forecasting accuracy within the evaluated space. The fastest configuration completes in 14.784 s with a MAPE of 2.9580%, whereas the fastest member of the minimum-MAPE tie completes in 15.806 s with a MAPE of 1.8409%. An additional 1.022 s therefore reduces error by 1.117 percentage points relative to the fastest trial. The Pareto frontier shows a sequence of configurations that progressively improve MAPE over time, but many points above the frontier take similar or longer times while producing substantially larger errors. This dispersion indicates that computational cost is influenced by runtime variability and architecture settings, yet accuracy depends primarily on selecting an effective hyperparameter combination rather than simply allocating more time.

A clarification is necessary when interpreting the highlighted best point. Trials 166, 167, and 168 share the same minimum MAPE because the searched batch_size does not affect forecasting in this single-series experiment. Figure 14 highlights trial 167 as the fastest of these tied trials, whereas the current abstract reports trial 166. Once this tie-handling rule is made consistent throughout the manuscript, the selected configuration can be examined at the level of its 48 individual forecasts.

5.5. Forecasting Behavior of the Best TCN Configuration

Figure 15 compares the actual Queensland load profile with the direct 48-step prediction produced by the reported best TCN configuration. The plot assesses whether the low aggregate MAPE indicates faithful tracking of the intraday demand pattern.

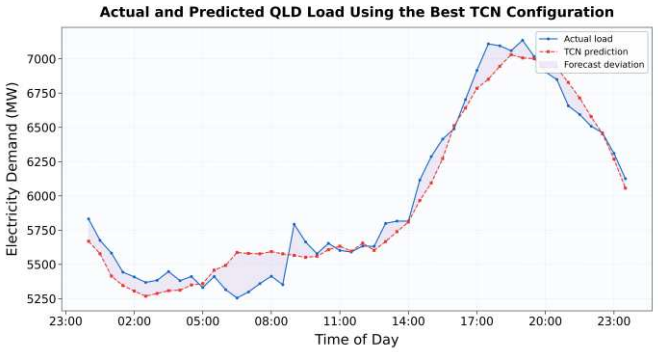


Fig. 15. Actual and predicted electricity demand over the 48-step test horizon using the best TCN configuration.

The predicted series follows the principal daily trajectory, including the overnight decline, the relatively stable morning demand, the rapid afternoon increase, the evening peak, and the subsequent reduction. Across the 48 half-hourly observations, the model obtains a MAPE of 1.8409%, a MAE of 107.77 MW, an RMSE of 133.18 MW, and a mean signed error of −22.71 MW. The small negative bias indicates mild overall underprediction. The model captures the timing of the high-demand period well, although the predicted maximum occurs approximately one-half hour before the actual maximum. At the actual peak of 7,135.83 MW near 19:00, the forecast is about 129.12 MW lower, corresponding to an error of approximately

1.81%. The most visible deviations occur during abrupt transitions: the forecast is smoother than the actual load around the early-morning change and underestimates parts of the afternoon rise. These localized differences suggest that the TCN effectively captures the overall temporal pattern but responds less precisely to rapid, short-duration changes.

To identify exactly where these deviations occur, Fig. 16 decomposes the aggregate MAPE into step-specific percentage errors.

Figure 16 presents the Absolute Percentage Error at each of the 48 forecast positions. Bars are distinguished according to whether their errors fall below or at/above the overall MAPE of 1.8409%.

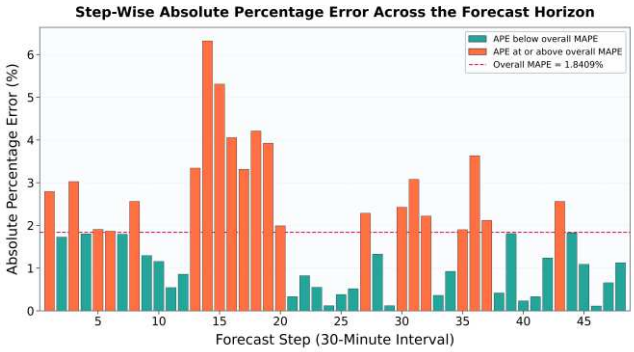


Fig. 16. Step-wise absolute percentage error over the 48-step forecasting horizon.

Twenty-seven of the 48 steps have APE below the overall MAPE, while 21 steps are at or above it. The largest error is 6.3202% at step 14, corresponding to approximately 06:30, followed by 5.3091% at step 15. A cluster of elevated errors extends from roughly steps 13 to 19, covering the early-morning transition from the low-load period to renewed daytime activity. The forecast initially overestimates the low morning observations and then underestimates the abrupt increase around step 19, indicating a temporal smoothing effect. A second, smaller concentration appears during the afternoon and early evening, including an APE of 3.6334% at step 36 near 17:30 as demand rises toward its daily peak. By contrast, several midday and late-horizon steps produce errors well below 1%. The error profile, therefore, shows that the low daily MAPE is not uniformly distributed: accuracy is strongest during relatively smooth intervals and decreases around rapid ramps and turning points.

The horizon-wise analysis reveals when errors occur; the final diagnostic examines whether the predictions remain quantitatively aligned with actual demand across low, medium, and high load levels.

Figure 17 presents a scatter plot comparing the predicted electricity demand with the corresponding actual demand values across all 48 steps of the day-ahead forecasting horizon. Each point represents one forecast–observation pair, allowing the overall agreement between the model outputs and the measured load values to be assessed visually. The dashed diagonal line indicates the ideal condition in which the predicted demand is exactly equal to the actual demand. Therefore, points located close to this reference line represent accurate forecasts, whereas larger deviations indicate greater prediction errors. In addition, the color of each point identifies its forecast position within the day, making it possible to examine whether forecasting accuracy changes systematically from the earlier to the later steps of the 48-step horizon..

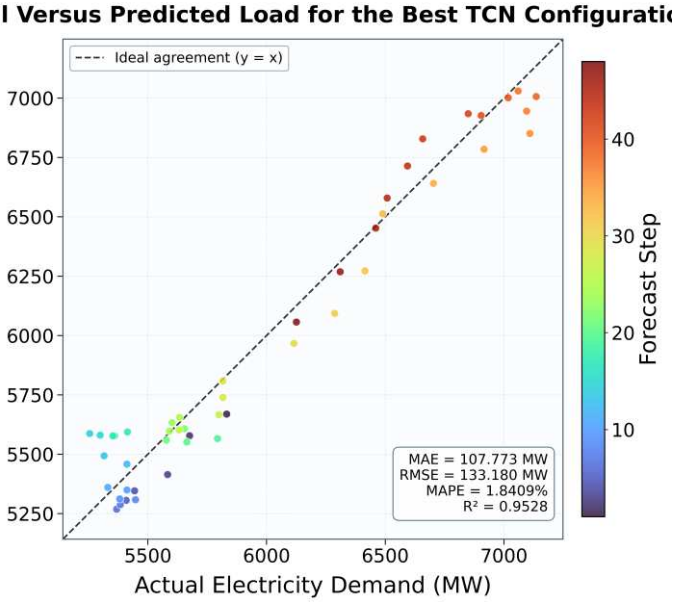


Fig. 17. Scatter plot of actual versus predicted load values for the best TCN model.

Most points lie close to the ideal y = x line, confirming strong agreement across the complete load range. The coefficient of determination is 0.9528, and the actual–predicted correlation is approximately 0.9768, indicating that the model explains most of the observed variation in the test-day profile. The remaining departures from the diagonal are systematic rather than random. Several low-load morning points lie above the line, reflecting overprediction during the morning trough. In contrast, several high-load afternoon and evening points lie below the line, reflecting underprediction during rapid growth and near the peak. This pattern is consistent with the slightly negative overall bias and suggests compression of the forecast range toward the daily mean. Even so, the limited scatter around the diagonal, together with the low MAPE and high R^2, demonstrates that the selected TCN provides a reliable representation of both the magnitude and shape of the one-day-ahead load profile.

Taken together, Figs. 15–17 show that the selected configuration performs well at the daily-profile level while retaining identifiable limitations around abrupt load transitions. These observations support a final synthesis of the sensitivity findings and their practical implications.

5.6. Synthesis and Practical Implications

The complete analysis leads to four principal findings. First, TCN forecasting accuracy is highly configuration-dependent: the evaluated MAPE range of 1.8409%–5.4553% is wide despite using the same architecture, data partition, forecasting horizon, loss function, and random seed. Second, the learning rate is the clearest marginal factor: 0.0005 produces the lowest average error, while 0.005 generally degrades performance. Third, a weekly input window yields the lowest mean MAPE and contains the global best configuration. Still, its high

dispersion indicates that long temporal context is beneficial only when accompanied by appropriate optimization and representation settings. Fourth, hidden dimensions exhibit nonlinear interaction effects: a compact encoder with a moderate decoder performs better than many larger alternatives, demonstrating that increasing parameter capacity alone is not an effective configuration strategy.

For a practical NeuralForecast implementation, the results support using a conservative learning rate and testing longer historical contexts before increasing the number of hidden dimensions. The 64–128 encoder–decoder pair provides a favorable balance between accuracy and model size in this experiment. The absence of a batch_size effect must be interpreted within the single-series loader design; future studies should investigate windows_batch_size or use multiple series if the objective is to assess batching behavior. The accuracy–time analysis further shows that the minimum error can be achieved with only a modest increase in runtime over the fastest trial. At the same time, many slower configurations remain dominant in both accuracy and computational efficiency.

These findings should nevertheless be interpreted under the study's controlled scope. The sensitivity results are based on one regional demand series, one 28-day training interval, one holdout day, and one fixed random seed. In addition, configuration ranking is performed on the final holdout period rather than through repeated rolling-origin validation. The results, therefore, quantify relative behavior under the selected experiment but do not establish universal parameter values for every season, region, or TCN implementation. This qualification provides the appropriate basis for the study conclusion and future research directions.

6 Conclusion

This study presented a controlled Grid-Search-based sensitivity analysis of Temporal Convolutional Network hyperparameters for one-day-ahead short-term electric load forecasting. A total of 243 configurations were generated from three levels of input size, learning rate, encoder hidden size, decoder hidden size, and batch size. Every model was implemented using the same NeuralForecast–PyTorch framework, trained on the same 28-day chronological QLD load segment, and evaluated on the same 48-step holdout day using MAPE as the primary accuracy criterion.

The evaluated configurations produced MAPE values from 1.8409% to 5.4553%, with a mean of 3.4993%, confirming that TCN forecasting performance is strongly affected by hyperparameter selection. The reported minimum-error configuration uses an input size of 336, a learning rate of 0.0005, an encoder hidden size of 64, and a decoder hidden size of 128. Its MAPE of 1.8409% represents a 55.05% reduction relative to the reference configuration, which has a MAPE of 4.0955%. The same minimum MAPE is obtained for batch sizes of 16, 32, and 64 because this NeuralForecast parameter controls series-level batching, and the experiment contains only one series. If runtime is adopted as an explicit secondary criterion, the batch_size = 32 trial is the fastest member of this tied group.

Among the investigated factors, the learning rate shows the clearest marginal effect, with 0.0005 yielding the lowest average MAPE and 0.005 yielding the highest. The weekly input window of 336 observations also yields the lowest marginal mean and the best individual result. However, it exhibits greater variability and is strongly dependent on the selected learning rate. The encoder and decoder dimensions do not follow a monotonic relationship between capacity and accuracy. In particular, the 64–128 encoder–decoder combination outperforms many larger structures, demonstrating that coordinated architectural settings are more important than simply increasing hidden size.

The selected TCN accurately follows the principal intraday load pattern, achieving a test-day MAE of 107.77 MW, RMSE of 133.18 MW, MAPE of 1.8409%, and R^2 of 0.9528. Most forecast points lie close to ideal agreement, while the largest errors occur during rapid morning and afternoon load transitions. These findings indicate that the model captures the overall daily profile effectively but tends to smooth abrupt changes and slightly underpredict high-demand periods.

The study is limited to a single Queensland series, a fixed 28-day training period, a single holdout day, and a single random seed. Future research should evaluate the observed sensitivities using rolling-origin experiments across multiple seasons and regions, repeated stochastic runs, and longer test periods. Calendar and weather variables may also be incorporated, and sensitivity to windows_batch_size, convolutional dilation, kernel size, dropout, and training duration should be examined. Comparisons with Random Search, Bayesian optimization, and metaheuristic approaches under a common computational budget would further clarify the trade-off between exhaustive sensitivity analysis and search efficiency.

Declarations

Ethical approval

Not applicable.

Consent to participate

Not applicable.

Consent to publish

Not applicable.

Availability of data and materials

The dataset used in this study is publicly available from the Australian Energy Market Operator (AEMO) open data repository.

Competing interests

The authors declare that they have no competing interests.

Funding

Not applicable.

References

[1] I. K. Nti, M. Teimeh, O. Nyarko-Boateng, and A. F. Adekoya, "Electricity load forecasting: a systematic review," *J. Electr. Syst. Inf. Technol.*, vol. 7, no. 1, 2020, doi: 10.1186/s43067-020-00021-8.

[2] M. G. Pinheiro, S. C. Madeira, and A. P. Francisco, "Short-term electricity load forecasting—A systematic approach from system level to secondary substations," *Appl. Energy*, vol. 332, no. November 2022, p. 120493, 2023, doi: 10.1016/j.apenergy.2022.120493.

[3] M. Short, T. Crosbie, M. Dawood, and N. Dawood, "Load forecasting and dispatch optimization for decentralized cogeneration plant with dual energy storage," *Appl. Energy*, vol.

186, pp. 304–320, 2017, doi: 10.1016/j.apenergy.2016.04.052.
[4] M. Haugen, H. Farahmand, S. Jaehnert, and S. E. Fleten, *Representation of uncertainty in market models for operational planning and forecasting in renewable power systems: a review*, no. 0123456789. Springer Berlin Heidelberg, 2023. doi: 10.1007/s12667-023-00600-4.
[5] A. Akrami, M. Doostizadeh, and F. Aminifar, "Power system flexibility: an overview of emergence to evolution," *J. Mod. Power Syst. Clean Energy*, vol. 7, no. 5, pp. 987–1007, 2019, doi: 10.1007/s40565-019-0527-4.
[6] S. S. G. Acharige *et al.*, "Grid integration of electric vehicles - Impact assessment and remedial measures," *J. Power Sources*, vol. 650, no. November 2024, p. 236697, 2025, doi: 10.1016/j.jpowsour.2025.236697.
[7] G. Dudek, "Multilayer perceptron for short-term load forecasting: from global to local approach," *Neural Comput. Appl.*, vol. 32, no. 8, pp. 3695–3707, 2020, doi: 10.1007/s00521-019-04130-y.
[8] S. Pelekis, I. Seisopoulos, and E. Spiliotis, "Sustainable Energy , Grids and Networks: A comparative assessment of deep learning models for day-ahead load forecasting : Investigating key accuracy drivers," *Sustain. Energy, Grids Networks*, vol. 36, no. July, p. 101171, 2023, doi: 10.1016/j.segan.2023.101171.
[9] P. Lusis, K. R. Khalilpour, L. Andrew, and A. Liebman, "Short-term residential load forecasting: Impact of calendar effects and forecast granularity," *Appl. Energy*, vol. 205, no. July, pp. 654–669, 2017, doi: 10.1016/j.apenergy.2017.07.114.
[10] A. Tascikaraoglu and B. M. Sanandaji, "Short-term residential electric load forecasting: A compressive spatiooral approach," *Energy Build.*, vol. 111, pp. 380–392, 2016, doi: 10.1016/j.enbuild.2015.11.068.
[11] H. Shi, L. Wang, R. Scherer, M. Wozniak, P. Zhang, and W. Wei, "Short-Term Load Forecasting Based on Adabelief Optimized Temporal Convolutional Network and Gated Recurrent Unit Hybrid Neural Network," *IEEE Access*, vol. 9, pp. 66965–66981, 2021, doi: 10.1109/ACCESS.2021.3076313.
[12] A. Morteza, A. A. Yahyaeian, M. Mirzaeibonehkhater, S. Sadeghi, A. Mohaimeni, and S. Taheri, "Deep learning hyperparameter optimization: Application to electricity and heat demand prediction for buildings," *Energy Build.*, vol. 289, p. 113036, 2023, doi: 10.1016/j.enbuild.2023.113036.
[13] F. Li, J. Ma, X. Yang, and W. Deng, "A short-term forecasting for multi-factor time series with multiple linear trend fuzzy information granule and cross-association," *Eng. Appl. Artif. Intell.*, vol. 137, no. PB, p. 109232, 2024, doi: 10.1016/j.engappai.2024.109232.
[14] Z. Dong, Z. Tian, and S. Lv, "A short-term power load forecasting system based on data decomposition, deep learning, and weighted linear error correction with a feedback mechanism," *Appl. Soft Comput.*, vol. 162, no. June, p. 111863, 2024, doi: 10.1016/j.asoc.2024.111863.
[15] Y. Bi, M. Guo, Q. Sui, J. Liang, K. Zhang, and K. Chen, "A hybrid method based on linear regression and genetic programming-based symbolic regression for short-term load forecasting," *Appl. Soft Comput.*, vol. 196, no. February, p. 115111, 2026, doi: 10.1016/j.asoc.2026.115111.
[16] M. Pleños, "Time Series Forecasting Using Holt-Winters Exponential Smoothing: Application to Abaca Fiber Data," *Zesz. Nauk. SGGW w Warszawie - Probl. Rol. Światowego*, vol. 22, no. 2, pp. 17–29, 2022, doi: 10.22630/prs.2022.22.2.6.
[17] S. Valero, C. Senabre, and L. Miguel, "Impact of COVID-19 lockdown in short-term load forecasting," vol. 237, no. May, pp. 344–354, 2025, doi: 10.1016/j.matcom.2025.04.035.
[18] S. Karamolegkos and D. E. Koulouriotis, "Advancing short-term load forecasting with decomposed Fourier ARIMA : A case study on the Greek energy market," vol. 325, no. February, 2025.
[19] M. M. Eljazzar and E. E. Hemayed, "Enhancing electric load forecasting of ARIMA and ANN using adaptive Fourier series," *2017 IEEE 7th Annu. Comput. Commun. Work. Conf. CCWC 2017*, no. January 2017, 2017, doi: 10.1109/CCWC.2017.7868457.
[20] M. Wanek, "Variable renewable energy forecasting in Germany : Reassessing simplicity with Bayesian-optimized multilayer perceptrons," *Renew—energy*, vol. 262, no. February, p. 125409, 2026, doi: 10.1016/j.renene.2026.125409.
[21] W. Tang and F. Kong, "Short-term Power Load Forecasting Based on Spatio-temporal Multicore Support Vector Regression," *2023 2nd Asia Power Electr. Technol. Conf.*, pp. 750–755, 2023, doi: 10.1109/APET59977.2023.10489025.
[22] 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, no. November 2023, p. 131814, 2024, doi: 10.1016/j.energy.2024.131814.
[23] I. Rafiq, A. Mahmood, U. Ahmed, I. Aziz, A. R. Khan, and S. Razzaq, "A Novel Approach for Forecasting and Scheduling Building Load through Real-Time Occupant Count Data," *Arab. J. Sci. Eng.*, 2024, doi: 10.1007/s13369-024-09296-9.
[24] P. Saini and S. K. Parida, "A novel probabilistic gradient boosting model with multi-approach feature selection and iterative seasonal trend decomposition for short-term load forecasting," *Energy*, vol. 294, no. August 2023, p. 130975, 2024, doi: 10.1016/j.energy.2024.130975.
[25] M. Mohamed, F. E. Mahmood, M. A. Abd, M. Rezkallah, and A. Hamadi, "Load Demand Forecasting Using eXtreme Gradient Boosting (XGboost)," *2023 IEEE Ind. Appl. Soc. Annu. Meet.*, pp. 1–7, 2023, doi: 10.1109/IAS54024.2023.10406613.
[26] H. Auto-tuning, S. Lei, and F. Wang, "A Short-term Net Load Forecasting Method Based on Two-stage Feature Selection and LightGBM," *2023 IEEE/IAS 59th Ind. Commer. Power Syst. Tech. Conf.*, pp. 1–6, doi: 10.1109/ICPS57144.2023.10142095.
[27] L. Zhang and D. Jánošík, "Enhanced short-term load forecasting with hybrid machine learning models: CatBoost and XGBoost approaches," *Expert Syst. Appl.*, vol. 241, no. November 2023, 2024, doi: 10.1016/j.eswa.2023.122686.
[28] 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, no. August, pp. 94506–94520, 2023, doi: 10.1109/ACCESS.2023.3311641.
[29] S. Navidi, K. Bajarunas, M. Arias, and C. Hu, "Forecasting battery capacity for second-life applications using physics-informed recurrent neural networks," *eTransportation*, vol. 25, no. May, p. 100432, 2025, doi: 10.1016/j.etran.2025.100432.
[30] F. Yang, X. Fu, Q. Yang, and Z. Chu, "Decomposition strategy and attention-based long short-term memory network for multi-step ultra-short-term agricultural power load forecasting[Formula presented]," *Expert Syst. Appl.*, vol. 238, no. PF, p. 122226, 2024, doi: 10.1016/j.eswa.2023.122226.
[31] J. Dong, Y. Jiang, P. Chen, J. Li, Z. Wang, and S. Han, "International Journal of Electrical Power and Energy Systems Short-term power load forecasting using bidirectional gated recurrent units-based adaptive stacked autoencoder," *Int. J. Electr. Power Energy Syst.*, vol. 165, no. August 2024, p. 110459, 2025, doi: 10.1016/j.ijepes.2025.110459.
[32] Y. Y. Hong and Y. H. Chan, "Short-term electric load forecasting using particle swarm optimization-based convolutional neural network," *Eng. Appl. Artif. Intell.*, vol. 126, no. PA, p. 106773, 2023, doi: 10.1016/j.engappai.2023.106773.
[33] Z. Xu *et al.*, "PhaCIA-TCNs: Short-Term Load Forecasting Using Temporal Convolutional Networks With Parallel Hybrid

Activated Convolution and Input Attention," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 1, pp. 427–438, 2023, doi: 10.1109/TNSE.2023.3300744.

[34] G. Huang, H. Li, X. Mo, and X. Jian, "Energy Conversion and Management : X A novel dynamic spatio-temporal graph convolutional network with multi-scale transformer for high-resolution wind vector forecasting," *Energy Convers. Manag. X*, vol. 30, no. February, p. 101684, 2026, doi: 10.1016/j.ecmx.2026.101684.

[35] Y. Jiang, D. Wang, Y. Zhang, M. Wang, J. Chen, and K. Yuan, "Prediction of freeze– thaw interface depth in a permafrost highway embankment using variational mode decomposition and a CNN– LSTM hybrid model," *Transp. Geotech.*, p. 102139, 2026, doi: 10.1016/j.trgeo.2026.102139.

[36] Z. Wang, L. Chen, and C. Wang, "Weekly periodicity feature-driven CNN-GRU and LightGBM hybrid model for short-term multi-step building load forecasting," *Energy*, vol. 355, no. April, p. 141163, 2026, doi: 10.1016/j.energy.2026.141163.

[37] J. Wen, Y. Peng, W. Zhang, X. Huang, and Z. Wang, "Short-term power load forecasting based on TCN-LSTM model," *Proc. 2024 IEEE 6th Int. Conf. Civ. Aviat. Saf. Inf. Technol. ICCASIT 2024*, pp. 734–738, 2024, doi: 10.1109/ICCASIT62299.2024.10827868.

[38] E. Avdevičius, M. Eskander, and D. Schulz, "Energy Price Prediction with CEEMDAN-STL and Hybrid Neural Network using Time2Vec Attention-based CNN-biGRU," *IEEE PES Innov. Smart Grid Technol. Eur. ISGT Eur. 2024*, pp. 1–5, 2024, doi: 10.1109/ISGTEUROPE62998.2024.10863496.

[39] P. F. Pai and W. C. Hong, "Forecasting regional electricity load based on recurrent support vector machines with genetic algorithms," *Electr. Power Syst. Res.*, vol. 74, no. 3, pp. 417–425, 2005, doi: 10.1016/j.epsr.2005.01.006.

[40] Q. Zhang *et al.*, "Multi-load forecasting for regional integrated energy systems using a hybrid ResNet-GRU-MTL model with dynamic grey wolf optimization," *Renew. Energy*, vol. 256, no. PI, p. 124690, 2026, doi: 10.1016/j.renene.2025.124690.

[41] A. A. Alsumaiei, "Modular hybrid framework for groundwater forecasting in arid urban environments : Integrating SWLR, CNN, and FNN with Bayesian optimization," *J. Eng. Res.*, no. August 2025, doi: 10.1016/j.jer.2025.11.010.

[42] A. C. Florea and R. Andonie, "Weighted Random Search for hyperparameter optimization," *Int. J. Comput. Commun. Control*, vol. 14, no. 2, pp. 154–169, 2019, doi: 10.15837/ijccc.2019.2.3514.

[43] S. Han, Z. Liang, and S. Li, "Knowledge-Based Systems MFF-Mixer : A multi-perspective feature fusion method for long-term multivariate time series forecasting," *Knowledge-Based Syst.*, vol. 335, no. December 2025, p. 115155, 2026, doi: 10.1016/j.knosys.2025.115155.

[44] D. Li, T. Wang, D. Feng, and Y. Zhou, "Temporal Convolutional Network with Adaptive Diffusion Model for Generation – Load Probabilistic Forecasting," pp. 1–17, 2025.

[45] A. Wan, Z. Xiong, K. Al-bukhaiti, R. Yin, and J. Yuan, "An integrated TCN-ECA-BiLSTM and TD 3 framework for high-accuracy forecasting and optimal load allocation in combined heat and power systems," *Energy*, vol. 350, no. February, p. 140670, 2026, doi: 10.1016/j.energy.2026.140670.

[46] R. Mamat and M. Fairusham, "Sustainable Chemistry for Climate Action: Emerging role of generative AI in renewable energy forecasting and system optimization," *Sustain. Chem. Clim. Action*, vol. 7, no. May, p. 100099, 2025, doi: 10.1016/j.scca.2025.100099.

[47] "Mô Hình Dự Báo Phụ Tải Ngắn Hạn Dựa Trên Phương Pháp Xử Lý Dữ Liệu Nhóm.Pdf."

[48] E. A. Madrid and N. Antonio, "Short-term electricity load forecasting with machine learning," *Inf.*, vol. 12, no. 2, pp. 1–21, 2021, doi: 10.3390/info12020050.

[49] A. Gasparin, S. Lukovic, and C. Alippi, "Deep learning for time series forecasting: The electric load case," *CAAI Trans. Intell. Technol.*, vol. 7, no. 1, pp. 1–25, 2022, doi: 10.1049/cit2.12060.

[50] S. Pelekis *et al.*, "A comparative assessment of deep learning models for day-ahead load forecasting: Investigating key accuracy drivers," *Sustain—Energy, Grids Networks*, vol. 36, no. August, p. 101171, 2023, doi: 10.1016/j.segan.2023.101171.

[51] Ş. Özdemir, Y. Demir, and Ö. Yildirim, "The Effect of Input Length on Prediction Accuracy in Short-Term Multi-Step Electricity Load Forecasting: A CNN-LSTM Approach," *IEEE Access*, no. February, 2025, doi: 10.1109/ACCESS.2025.3540636.

[52] X. Ji *et al.*, "A Hybrid Residential Short-Term Load Forecasting Method Using Attention Mechanism and Deep Learning," *Buildings*, vol. 13, no. 1, pp. 1–19, 2023, doi: 10.3390/buildings13010072.

[53] Z. Zhang, J. Liu, S. Pang, M. Shi, and H. Hwang, "International Journal of Electrical Power and Energy Systems General short-term load forecasting based on multi-task temporal convolutional network in COVID-19," *Int. J. Electr. Power Energy Syst.*, vol. 147, no. October 2022, p. 108811, 2023, doi: 10.1016/j.ijepes.2022.108811.

[54] P. Lara-Benítez, M. Carranza-García, J. M. Luna-Romera, and J. C. Riquelme, "Temporal convolutional networks applied to energy-related time series forecasting," *Appl. Sci.*, vol. 10, no. 7, pp. 1–17, 2020, doi: 10.3390/app10072322.

[55] A. K. Shaikh, A. Nazir, N. Khalique, A. S. Shah, and N. Adhikari, "A new approach to seasonal energy consumption forecasting using temporal convolutional networks," *Results Eng.*, vol. 19, no. March, p. 101296, 2023, doi: 10.1016/j.rineng.2023.101296.

[56] Q. Xiong, M. Liu, Y. Li, C. Zheng, and S. Deng, "Short-Term Load Forecasting Based on VMD and Deep TCN-Based Hybrid Model with Self-Attention Mechanism," *Appl. Sci.*, vol. 13, no. 22, 2023, doi: 10.3390/app132212479.

[57] N. Bacanin, C. Stoean, M. Zivkovic, M. Rakic, R. Strulak-Wójcikiewicz, and R. Stoean, "On the Benefits of Using Metaheuristics in the Hyperparameter Tuning of Deep Learning Models for Energy Load Forecasting," *Energies*, vol. 16, no. 3, pp. 1–21, 2023, doi: 10.3390/en16031434.

[58] I. Gridin, *Hyperparameter Optimization*. 2022. doi: 10.1007/978-1-4842-8149-9_2.

[59] A. Morales-Hernández, I. Van Nieuwenhuyse, and S. Rojas Gonzalez, *A survey on multi-objective hyperparameter optimization algorithms for machine learning*, vol. 56, no. 8. Springer Netherlands, 2023. doi: 10.1007/s10462-022-10359-2.

[60] M. Zulfiqar, K. A. A. Gamage, M. Kamran, and M. B. Rasheed, "Hyperparameter Optimization of Bayesian Neural Network Using Bayesian Optimization and Intelligent Feature Engineering for Load Forecasting," 2022.

[61] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," *31st Int. Conf. Mach. Learn. ICML 2014*, vol. 2, pp. 1130–1144, 2014.

[62] Z. Zhang *et al.*, "General short-term load forecasting based on multi-task temporal convolutional network in COVID-19," *Int. J. Electr. Power Energy Syst.*, vol. 147, no. November 2022, p. 108811, 2023, doi: 10.1016/j.ijepes.2022.108811.

[63] H. Li, S. Li, Y. Wu, Y. Xiao, Z. Pan, and M. Liu, "Short-term power load forecasting for integrated energy system based on a residual and attentive LSTM-TCN hybrid network," *Front. Energy Res.*, vol. 12, no. May, pp. 1–21, 2024, doi: 10.3389/fenrg. 2024.1384142.

[64] C. Sun, Y. Ning, D. Shen, and T. Nie, "Graph Neural Network-Based Short-Term Load Forecasting with Temporal

Convolution," *Data Sci. Eng.*, vol. 9, no. 2, pp. 113–132, 2024, doi: 10.1007/s41019-023-00233-8.