Siddiqui *et al. Journal of Cloud Computing* (2025) 14:75
https://doi.org/10.1186/s13677-025-00815-z

Journal of Cloud Computing

RESEARCH Open Access

SpectraNet: a lightweight hybrid time–frequency deep learning framework for sustainable cloud workload forecasting

Nahin F. Siddiqui[1], Zarif Safwan Hoque[1], Md. Ehsanul Haque[1], Abu Raihan Mostofa Kamal[1], A. K. M. Azad[2*], Salem A. Alyami[2] and Md Azam Hossain[1*]

Abstract

The rapid proliferation of cloud computing has created the demand for sustainable resource management strategies to address the increasing energy consumption and carbon footprint of data centers. Accurate CPU workload forecasting is a cornerstone of this effort, enabling proactive resource allocation that minimizes energy waste from over-provisioning and prevents performance degradation from under-provisioning. This study introduces **SpectraNet**, a lightweight hybrid model designed to advance sustainability in cloud computing by delivering accurate CPU usage predictions with minimal computational overhead. **SpectraNet** integrates time-domain and frequency-domain analysis to capture both transient and periodic workload dynamics, enabling robust and fine-grained forecasting. **SpectraNet** processes time-series data through parallel time- and frequency-domain branches, which are then fused and refined using a CNN with attention to capture local patterns. Finally, an LSTM models long-term dependencies to generate the forecast. Extensive experiments on the **vmCloud Dataset** and **Alibaba Cluster Trace** datasets demonstrate that **SpectraNet** achieves a Mean Absolute Error (MAE) of 0.0549 on long-range forecasts, delivering comparable or superior accuracy to larger baselines while using roughly twenty times fewer parameters and offering noticeably faster inference. The model's efficiency and small footprint make it a practical and scalable solution for real-world, resource-constrained cloud environments. By enabling more precise and energy-efficient resource management, **SpectraNet** provides a valuable tool for building more sustainable and cost-effective cloud infrastructures. The implementation code for this study has been made publicly available.

Keywords Cloud computing, Sustainable resource management, CPU workload forecasting, Time-Frequency analysis, Hybrid deep learning models, Lightweight al models

Code Repository: https://github.com/lingdaizon-crypto/SpectraNet.git.

*Correspondence:
A. K. M. Azad
kazad@imamu.edu.sa
Md Azam Hossain
azam@iut-dhaka.edu
[1]Department of Computer Science and Engineering, Islamic University of Technology, K B Bazar, Gazipur 1700, Bangladesh
[2]Department of Mathematics and Statistics, Faculty of Science, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh 13318, Saudi Arabia

© The Author(s) 2025. **Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

Introduction

Cloud computing forms the backbone of modern digital infrastructure, but its rapid expansion has led to a significant increase in the energy consumption and carbon footprint of global data centers [1]. Consequently, **sustainable cloud management** has emerged as an imperative, driving research towards strategies that enhance operational efficiency while minimizing environmental impact [2–4]. A cornerstone of this effort is the accurate forecasting of resource usage, particularly CPU demand. Recent studies have demonstrated that leveraging predictive models for CPU scaling can significantly reduce energy consumption [5, 6]. Reliable forecasts enable dynamic resource allocation, preventing both energy-wasting over-provisioning and performance-degrading under-provisioning [7, 8]. This not only ensures compliance with Service Level Agreements (SLAs) but also directly contributes to more stable, cost-effective, and energy-efficient cloud ecosystems, a goal actively pursued by the latest generation of carbon-aware schedulers [9, 10]. However, the highly dynamic and non-linear nature of modern cloud workloads presents a formidable challenge to achieving such forecasts [11].

Traditional statistical methods like Autoregressive Integrated Moving Average (ARIMA) [12], while foundational, are ill-suited for these complex environments. Their reliance on linearity assumptions, sensitivity to noise, and inability to scale make them inadequate for today's high-dimensional cloud data [13, 14]. Seasonal ARIMA (SARIMA) extends ARIMA to capture repeating seasonal patterns, but it inherits many of the same limitations, including sensitivity to noise, manual specification of seasonal parameters, and rapidly increasing complexity with higher seasonal periods, rendering it less effective for non-stationary or multi-dimensional cloud workloads. While deep learning models such as LSTMs and CNNs have advanced the field by capturing non-linear patterns, they often operate in a single domain, limiting their effectiveness [15]. Time-domain models frequently miss underlying periodicities, whereas frequency-domain models can overlook sudden, localized demand spikes. Hybrid architectures that integrate both domains are essential for a holistic understanding but require careful design to avoid introducing noise and redundancy [16, 17]. A significant, unaddressed challenge remains: many advanced models, despite their accuracy, are too computationally demanding for practical, real-time deployment, thereby undermining the very goal of resource efficiency.

Existing research on cloud workload forecasting has explored various hybrid models combining time and frequency domain information [16–23]. However, most existing methods transform data to the frequency domain only as an intermediate feature extraction step before returning to the time domain for final forecasting [17–20]. In contrast, our approach directly integrates both domains in a parallel manner, enabling **SpectraNet** to jointly exploit temporal and spectral dependencies without reverting to a single-domain representation. This design redefines multidomain time series analysis by simplifying the modeling process while capturing richer long-term dependencies critical for cloud workload prediction. This research directly confronts this challenge by developing **SpectraNet**, a novel and lightweight hybrid model designed for **sustainable CPU usage prediction** [24, 25]. The scope of this work is to introduce a computationally efficient architecture that strategically integrates both time and frequency domain information to maximize predictive accuracy while minimizing resource overhead. Our model is engineered to be practical for resource-constrained cloud environments, making it a viable tool for real-world energy conservation [26]. The key contributions of this work are threefold:

1. **A Novel Hybrid Time-Frequency Architecture:** We propose a dual-branch model that fuses spectral features (via Fourier Transform) with raw temporal data. This comprehensive representation captures both long-term periodic trends and short-term, localized fluctuations in cloud workloads.
2. **Adaptive Attention for Feature Refinement:** Our design incorporates a hierarchical Squeeze-and-Excitation Network (SENet) to dynamically weigh and select the most salient features from the fused data, enhancing signal clarity while suppressing noise and redundancy [27].
3. **A Focus on Lightweight, Sustainable Deployment:** We introduce an efficient framework that significantly reduces the computational overhead associated with complex deep learning models. By prioritizing a smaller footprint, **SpectraNet** offers a practical solution that improves resource management, reduces operational costs, and contributes to a more **sustainable and cost-effective cloud infrastructure**.

The remainder of this paper is organized as follows: Section 2 reviews related work in cloud workload forecasting. Section 3 details the proposed **SpectraNet** methodology. Section 4 presents the experimental setup and results. Finally, Sect. 5 concludes the paper by summarizing our findings and discussing their implications for sustainable cloud computing.

Literature review

Accurate prediction of CPU usage in dynamic cloud environments is a challenging task due to workload volatility, non-stationarity, and periodicity. Traditional time series

forecasting models such as ARIMA and SARIMA, while foundational, are often unable to represent the complex, non-linear dynamics of modern cloud workloads and are limited by their reliance on time-domain information [28]. To address these limitations, recent research has integrated frequency-domain analysis to capture global periodicities and reduce noise. However, purely frequency-based approaches risk losing essential temporal dependencies, establishing hybrid time-frequency models as a more promising solution.

In cloud CPU usage forecasting, deep learning architectures—particularly LSTM and GRU networks—excel at modeling non-linear trends and long-term dependencies [29, 30]. Hybrid methods, such as ARIMA–LSTM, aim to capture both linear and non-linear patterns, while more advanced approaches like VTGAN combine CNN and LSTM within adversarial frameworks to enhance prediction robustness [30]. These models demonstrate the power of deep learning but often focus on improving accuracy without explicitly considering computational overhead.

Beyond direct CPU forecasting, hybrid time-frequency models have proven effective in related applications. Architectures like MoFE-Time [31] and TFEGRU [32] have achieved state-of-the-art results on various benchmarks by integrating sophisticated components like Mixture of Experts networks [33], attention mechanisms [34], and Short-Time Fourier Transforms [35]. Similarly, in power and load forecasting, fused models like CNN-LSTM [36] and DWT-BiGRU [19] have successfully combined parallel signal processing and wavelet transforms to handle non-stationary data. Recent lightweight architectures have also sought to balance accuracy with efficiency. DLinear [37] provides a strong baseline through trend–seasonal decomposition and simple linear mappings, achieving competitive accuracy with minimal complexity. TimeMixer [38] extends this direction using multiscale mixing to model temporal patterns across different granularities. Meanwhile, MSCNet [39] employs multi-scale convolution and transformer blocks to jointly capture periodic and contextual patterns, while SparseTSF [40] achieves high efficiency through sparse cross-period forecasting with fewer than 1K parameters. PETformer [41] further advances transformer-based forecasting using structural optimizations that enhance temporal continuity and information density.

Collectively, these studies validate that integrating complementary time and frequency perspectives leads to more comprehensive and accurate forecasts. However, a critical trade-off often emerges: advanced models frequently achieve higher accuracy through increased architectural complexity, leading to significant computational overhead. This presents a major barrier to their practical deployment in resource-constrained cloud environments, where efficiency and low latency are paramount for sustainable operations. This reveals a distinct research gap for a model that harnesses the power of hybrid-domain analysis within a lightweight and efficient framework. Our work addresses this gap by proposing **SpectraNet**, a dual-branch architecture designed to provide a robust forecasting solution that explicitly balances high performance with low computational cost, making it suitable for sustainable, real-world cloud resource management.

Methodology

Problem formulation

Accurately forecasting CPU utilization in cloud environments remains a challenging problem due to the inherently non-linear, high-dimensional, and periodic characteristics of workload patterns. Formally, given a multivariate time series $X = \{x_t \in \mathbb{R}^d \mid t = 1, 2, \ldots, T\}$, where x_t represents a vector of cloud resource metrics (e.g., CPU, memory, network I/O) at time t, our objective is to learn a lightweight forecasting model f_θ. This model, $f_\theta : \mathbb{R}^{T \times d} \to \mathbb{R}^H$, aims to predict CPU usage over a future horizon H such that it minimizes the forecasting error, defined by the Mean Squared Error (MSE) [42]:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{H} \sum_{h=1}^{H} (y_{t+h} - \hat{y}_{t+h})^2 \tag{1}$$

where y_{t+h} is the true CPU usage and $\hat{y}_{t+h}$ is the predicted value. Our core objectives are to: (i) leverage both frequency-domain periodicity and time-domain locality; (ii) enhance feature relevance through adaptive attention; (iii) promote sustainability by ensuring a small model footprint; and (iv) maintain high efficiency for real-time forecasting.

SpectraNet: model architecture and rationale

To address the forecasting challenge, we propose **SpectraNet**, a hybrid deep learning architecture designed to integrate complementary time-domain and frequency-domain information. The model's strength lies in its ability to capture both global periodic trends (e.g., daily cycles) and local sequential patterns (e.g., sudden bursts in activity), thereby improving accuracy and robustness. The model is designed to perform direct multi-step forecasting, where multiple future time steps are predicted simultaneously by configuring the output layer to match the desired horizon. This differs from recursive strategies, where one-step predictions are repeatedly fed back as inputs. The direct approach helps mitigate error accumulation across forecasting horizons and enables stable multi-step predictions. A visual overview of the

architecture is provided in Fig. 1. The following subsections detail the step-by-step rationale and design of each component.

Step-by-step design and rationale

To capture the complete behavior of cloud workloads, our model processes data from both the time and frequency domains. This dual-domain approach allows it to capture both short-term events and long-term cyclical trends. The following numbered paragraphs detail the eight primary components of the **SpectraNet** architecture. Each component corresponds to the numbered dotted boxes in the model diagram shown in Fig. 1.

1) Frequency-domain extraction:

Cloud workloads often contain strong cyclical patterns (e.g., diurnal or weekly usage cycles) that are more easily identified in the frequency domain [43]. The raw input time series X is transformed into the frequency domain using the Fast Fourier Transform (FFT) algorithm [44] which helps to isolate these global periodicities and can mitigate the effects of random noise. We take the absolute magnitude of the 2D FFT applied across the time and feature dimensions to obtain a stable spectral representation:

$$X_{\text{freq}} = |\mathcal{F}(X)| \tag{2}$$

where $\mathcal{F}$ denotes the 2D Discrete Fourier Transform, computed for each batch element as:

$$X_{k,l} = \sum_{t=0}^{T-1} \sum_{j=0}^{d-1} x_{t,j} \cdot e^{-i2\pi\left(\frac{kt}{T} + \frac{lj}{d}\right)} \tag{3}$$

for frequency indices $k = 0, \ldots, T-1$ and $l = 0, \ldots, d-1$. This spectral representation, illustrated in Fig. 2 [45], highlights cyclical patterns essential for modeling cloud workloads.

2) Adaptive Spectral Feature Refinement:

The raw frequency features may include redundant or uninformative channels. An attention mechanism is needed to dynamically select the most relevant spectral components for the forecasting task. To adaptively emphasize these components, a Squeeze-and-Excitation Network (SENet) [27] is applied. The process involves three steps:

i. **Squeeze:** Global information from each channel is aggregated using global average pooling across the time dimension. For each channel c, we compute a statistic z:

$$z = \frac{1}{T} \sum_{t=1}^{T} X_{\text{freq},t} \tag{4}$$

This equation calculates a channel descriptor by averaging the feature values $X_{freq,t}$ across the entire time dimension T. This effectively "squeezes" the temporal information for each channel into a single scalar value, representing that channel's global summary.

ii. **Excitation:** The aggregated information is fed through two fully-connected layers to learn channel-wise relationships and produce a vector of attention weights s:

$$s = \sigma(W_2 \cdot \delta(W_1 \cdot z)) \tag{5}$$

where W_1 and W_2 are weight matrices, δ is the ReLU activation function, and σ is the sigmoid activation function.

iii. **Recalibration:** The original frequency features are re-weighted using the learned attention scores:

$$\hat{X}_{\text{freq}} = s \cdot X_{\text{freq}} \tag{6}$$

The SENet adaptively re-weights each frequency channel based on its global importance, effectively amplifying significant cyclical patterns and suppressing noise. The output is a refined feature map, $\hat{X}_{\text{freq}}$.

3) Hybrid Time-Frequency Fusion:

To create a comprehensive input representation, we must combine the global periodic information from the frequency domain with the precise, local event information from the time domain.

The refined frequency-domain features, $\hat{X}_{\text{freq}}$, are concatenated with the original time-domain input X, along the feature dimension to create a unified tensor,

$$X_{\text{combined}} = \text{concat}(X, \hat{X}_{\text{freq}}, \text{dim} = -1) \in \mathbb{R}^{B \times T \times (2d)} \tag{7}$$

This dual-domain representation enables the model to integrate abrupt changes (e.g., load spikes) with underlying cycles.

4) Local Context Capture via Framing:

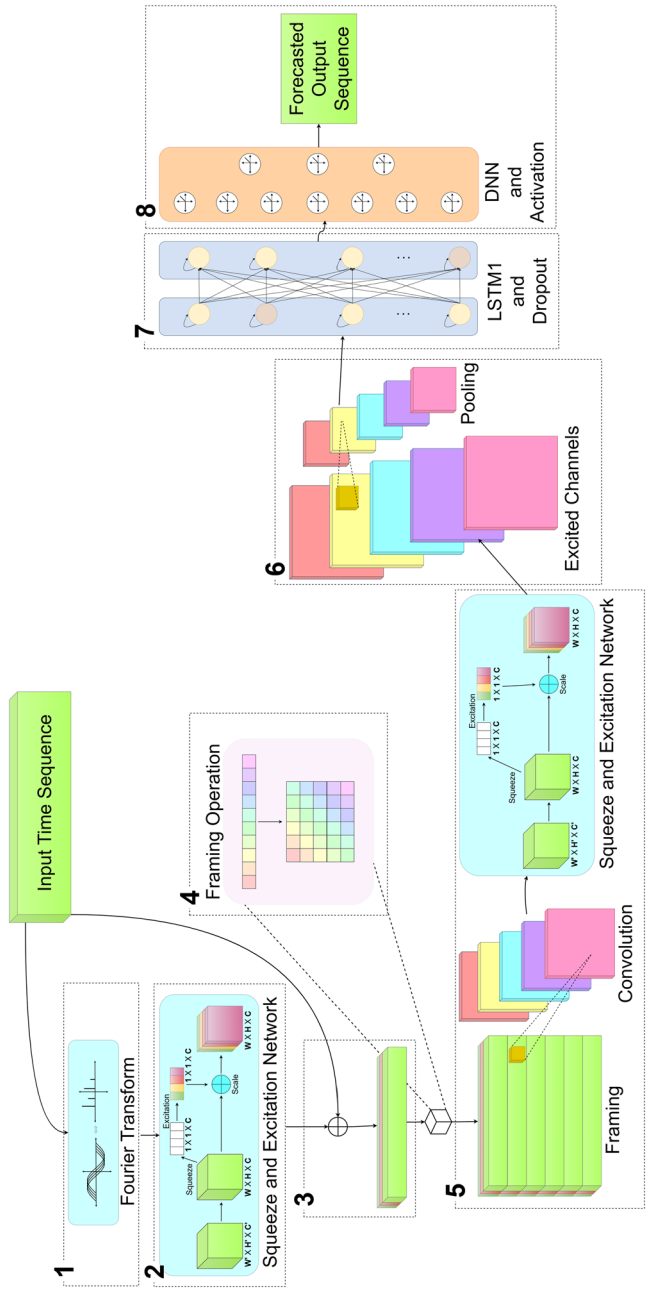


Fig. 1 Proposed hybrid architecture: ***SpectraNet***. The model processes the input time series through two parallel branches (time and frequency), fuses them, and uses a CNN-LSTM pipeline with attention to generate the final forecast. The numbered, dotted boxes (1–8) in the diagram correspond to the components described in Sect. 3.3

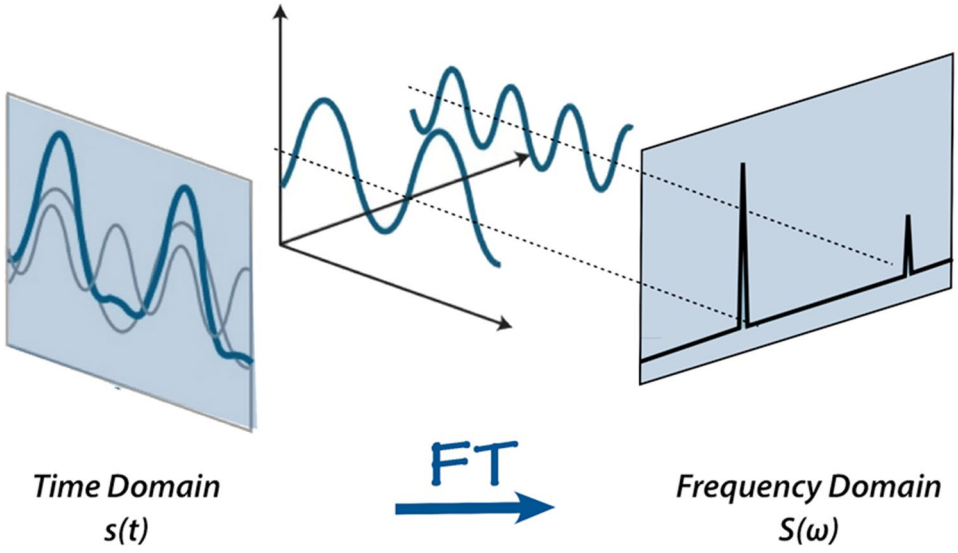


Fig. 2 Conversion from time domain to frequency domain

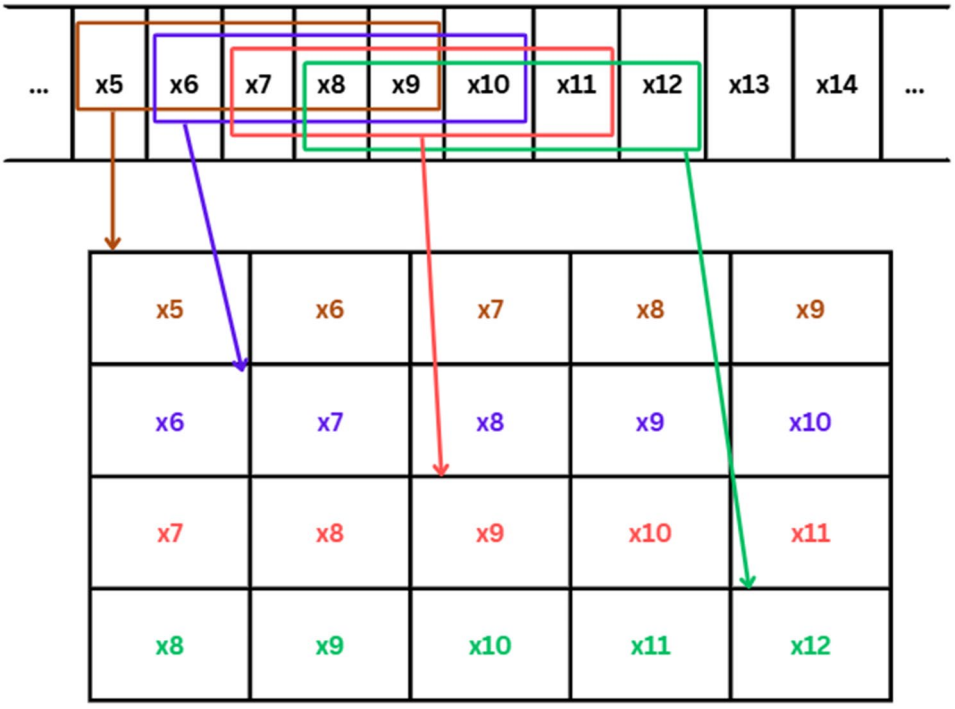


Fig. 3 Framing layer working process [Fig.1(4)]

Convolutional Neural Networks (CNNs) are highly effective in extracting spatial patterns from grid-like data. To apply CNNs on input time series, it must be converted into a format that captures local sequential contexts, similar to how an image is composed of overlapping patches. For that the combined tensor is transformed using a sliding window operation (*framing*) with frame length $L = 30$ and step size 1:

$$X_{\text{framed}} = \text{Unfold}(X_{\text{combined}}, \text{dim} = 1, \text{size} = 30, \text{step} = 1) \in \mathbb{R}^{B \times N_w \times 30 \times 2F} \tag{8}$$

where $N_w = T - 29$ is the number of windows. The tensor is then permuted to image-like format for convolutional operations. This framing, depicted in Fig. 3, transforms the $1D$ time series into a set of overlapping $2D$ patches, preserving short-term dependencies and making it suitable for processing by 2D convolution layers.

5) Spatio-Temporal Feature Extraction:

A convolutional neural network (CNN) is applied to the framed input in order to capture localized short-term patterns within each window. The extracted representations are further refined by an attention mechanism. Specifically, the CNN module consists of two layers: $X_{\text{cnn}} = \text{CNN}(X_{\text{framed}})$; The first convolutional layer employs 64 output channels and 3×3 kernels, followed by ReLU and batch normalization; the second layer uses 128 output channels and 1×1 kernels, again followed by ReLU activation and batch normalization. Subsequently, a second SENet block is introduced to enhance the resulting feature maps (X_{cnn}) through channel-wise attention:

$$X_{\text{refined}} = \text{SENet}(X_{\text{cnn}}) \tag{9}$$

This step adaptively re-weights the feature channels, emphasizing the most informative spatio-temporal patterns for downstream processing.

6) Feature Aggregation and Dimensionality Reduction:

To prepare the features for the final modeling stage while ensuring computational efficiency, the high-dimensional feature maps are condensed into a more compact representation. This is accomplished through Adaptive Global Average Pooling (GAP) [46], which reduces each spatial feature map to a single representative value. Specifically, for every channel in the refined feature maps $X_{refined}$, GAP computes the mean across all spatial dimensions, making a concise yet information-preserving representation suitable for subsequent processing. This operation can be formally expressed for each batch element b and channel c as:

$$X_{\text{pool}} = \frac{1}{H \times W} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} X_{\text{refined},b,c,h,w} \in \mathbb{R}^{B \times 128 \times 1 \times 1} \tag{10}$$

where H and W denote the height and width of the input feature map's spatial dimensions (in this architecture, $H = N_w$ and $W = 30$). This process reduces the tensor $\mathbb{X}_{\text{refined}} \in \mathbb{R}^{B \times 128 \times N_w \times 30}$ to $X_{\text{pool}} \in \mathbb{R}^{B \times 128 \times 1 \times 1}$. The pooled tensor is then reshaped as $X_{\text{seq}} = \text{View}(X_{\text{pool}}, (B, 1, 128))$, forming a feature vector suitable for recurrent processing. By aggregating spatio-temporal information across the entire look-back window into a single fixed-size representation, this pooling step not only improves computational efficiency but also helps mitigate overfitting. It is important to note that the sequence of length one resulting from the reshaped pooled tensor is not the main temporal input to the LSTM. The primary recurrent processing operates on sequences generated by the `FramingLayer`, which segments the raw time series into overlapping frames, resulting in an effective sequence length of approximately 971. This enables the LSTM stack to learn long-range temporal dependencies across these frames. The globally pooled representation is used solely within a skip connection to inject aggregated context, not as the temporal sequence fed into the LSTM.

7) Long-Range Sequential Modeling:

The CNN module captures localized short-term patterns; however, modeling long-range temporal dependencies and capturing the overall sequence dynamics requires a recurrent neural network. To achieve this, the aggregated feature sequence is processed by a stack of two Long Short-Term Memory (LSTM) layers [47], with dropout applied between layers for regularization:

$$X_{\text{lstm1}}, _ = \text{LSTM}_1(X_{\text{seq}}) \quad (\text{input: } 128, \text{ hidden: } 64), \tag{11}$$

$$X_{\text{lstm1}} = \text{Dropout}(X_{\text{lstm1}}, p = 0.2), \tag{12}$$

$$X_{\text{lstm2}}, _ = \text{LSTM}_2(X_{\text{lstm1}}) \quad (\text{input: } 64, \text{ hidden: } 32), \tag{13}$$

$$X_{\text{lstm2}} = \text{Dropout}(X_{\text{lstm2}}, p = 0.2). \tag{14}$$

LSTMs are well-suited for processing sequential information, and stacking them allows for learning more complex feature hierarchies.

8) Final Prediction:

Finally, the learned high-level feature representations are fed into a small dense neural network (DNN) to map to the desired multi-step forecast horizon:

$$\hat{y} = \text{DNN}(X_{\text{lstm2}}[:, -1, :]) \tag{15}$$

This acts as the prediction head, comprises a linear layer ($32 \to 32$), ReLU activation, and another linear layer ($32 \to H$), with H being the forecast horizon. This network outputs a vector of length H, corresponding to the CPU usage forecast.

Experimental setup

Datasets

To ensure a robust and comprehensive evaluation, our proposed model was tested on two large-scale, real-world datasets that represent diverse cloud computing workloads: **Alibaba Cluster Trace (2018)** [48] and **vmCloud Dataset** [49]. The selection criteria focused on datasets

with high-resolution, multivariate time series data that exhibit the complex periodic and non-stationary patterns characteristic of production environments.

Alibaba Cluster Trace is a public, large-scale datacenter workload trace that provides insight into production system behavior. We utilized *Version 2*, which contains 8 days of monitoring data from more than 4,000 machines. Its richness, including fine-grained resource metrics and metadata for containers and batch jobs, makes it ideal for testing model scalability and its ability to handle complex high-dimensional data. For these experiments, a feature set encompassing temporal context (e.g., `hour`, `weekday`), core resource utilization (`cpu_util_percent`, `mem_util_percent`), network I/O (`net_in`, `net_out`), and disk activity (`disk_io_percent`) were selected.

vmCloud Dataset provides a multivariate time series of cloud computing performance metrics, with approximately 2 million observations capturing virtual machine (VM) activity. Core resource features include CPU usage, memory utilization, network traffic, execution time, number of executed instructions, power consumption, and derived measures of energy efficiency. Each record is time-stamped and linked to a unique VM identifier, enabling temporal analysis and per VM tracking. In addition, the dataset includes task metadata (e.g., `task type`, `priority`, and `status`), which supports evaluating models across heterogeneous workload types. The explicit inclusion of power and efficiency metrics makes this dataset particularly valuable for studies focused on energy-aware cloud management and sustainable computing.

Data preprocessing

A multi-stage preprocessing pipeline was applied to clean the data, handle missing values, and engineer relevant features to improve model performance and training stability. While several steps were shared across datasets, others were tailored to the specific characteristics of the **Alibaba Cluster Trace** and **vmCloud** datasets.

For data cleaning and missing value handling, the **Alibaba Cluster Trace** dataset required imputing missing entries in the `mem_gps` and `mkpi` columns with a constant placeholder to signify absence. In contrast, the **vmCloud** dataset contained irregular timestamps, which were forward-filled to ensure temporal continuity. The series was then sorted to maintain chronological order, and any remaining numerical gaps were filled using median imputation.

The temporal structuring of the datasets also differed. The **Alibaba Cluster Trace** dataset was resampled to a fixed 30-minute interval to ensure a consistent time series resolution. The **vmCloud** dataset did not require resampling, but the forward-filling [50] of missing timestamps served to maintain a regular sequence.

Feature engineering [51] was particularly dataset-specific. For the **Alibaba Cluster Trace** dataset, instances with a coefficient of variation greater than 0.1 were filtered to remove potential outliers or unstable periods. The **vmCloud** dataset, on the contrary, underwent extensive feature engineering. time-based features were extracted from timestamps, including month, day, hour, minute, weekday, and weekend indicators, enabling the model to capture cyclical patterns. Additionally, domain-informed features were created, such as first-order differences of CPU usage to capture rate changes, rolling averages to smooth short-term fluctuations, lag features, and exponential moving averages. The trend and seasonality components were also explicitly decomposed to better characterize workload dynamics.

Finally, after these dataset-specific operations, both datasets underwent a normalization step. All numerical features were scaled to the range $[0, 1]$ using a `MinMaxScaler` [52]. This step ensured that features with larger scales did not disproportionately influence the model and helped achieve faster convergence and greater stability during training. Table 1 summarizes the preprocessing workflow.

Baseline and comparative models

To rigorously evaluate the performance of proposed **SpectraNet**, we selected a diverse set of baseline models representing the state-of-the-art in time series forecasting. The baselines were chosen to benchmark our model against a range of architectural approaches, from established recurrent and convolutional networks to more recent hybrid designs. The models are summarized in Table 2 and are grouped into the following categories for our comparative analysis:

- **Recurrent and Convolutional Baselines:** This group includes foundational deep learning models such as a simple 1D CNN (`cnn_1d`) [54], an LSTM with dense layers (`lstm_dnn`) [54, 55], and

Table 1 Summary of data preprocessing steps

Step	Description
Imputation	Missing values replaced using linear interpolation on each feature.
Normalization	`MinMaxScaler` fitted on training set, applied to validation and test sets.
Splitting Strategy	Chronological split: 70% training, 15% validation, 15% testing.
Feature Fusion	Combination of original features with Fourier-transformed frequency components.

Table 2 Baseline models for time series forecasting

Model Name	Type / Architecture
MCDFN [53]	Multi-scale convolutional deep forecasting network
SE LSTM DNN [54, 55]	LSTM with dense layers + attention
LSTM DNN [54, 55]	LSTM with dense layers
TCN [56]	Temporal Convolutional Network
FreTS [57]	Hybrid frequency-temporal sequence model
T CNN SE LSTM [54, 55]	CNN-LSTM with SE attention
C LSTM AE [55]	Convolutional LSTM with autoencoder
C LSTM [55]	Convolutional LSTM without autoencoder
CNN 1D [54]	Simple 1D CNN
DLinear [37]	Season-Trend Decomposition with Linear Layers
TimeMixer [38]	Hybrid Multi-block MLP-based architecture

a Temporal Convolutional Network (`tcn`) [56]. These models serve to establish a performance baseline using standard single-domain (time-only) architectures.

- **Attention-Enhanced and Hybrid Architectures:** To compare against more advanced models, we include architectures that incorporate attention mechanisms or hybrid components. This includes an LSTM with Squeeze-and-Excitation attention [27] (`se_lstm_dnn`), a CNN-LSTM with SE attention (`t_cnn_se_lstm`), and Convolutional LSTM variants (`c_lstm`, `c_lstm_ae`) [55]. These models allow us to assess the performance contribution of our specific hybrid design against other fusion and attention strategies.
- **State-of-the-Art and Competing Hybrid Models:** This category includes recent, high-performing models designed for complex forecasting tasks. We compare against `MCDFN` [53], a multi-scale convolutional network, `FreTS`, a hybrid frequency-temporal model [57], `DLinear` [37], which leverages seasonal-trend decomposition with lightweight linear layers, and `TimeMixer` [38], a hybrid multi-block MLP-based model that captures both temporal dependencies and frequency-domain patterns. These models represent a direct comparison against other sophisticated architectures that, like **SpectraNet**, aim to capture complex data patterns.

This comprehensive set of baselines allows us to situate **SpectraNet**'s performance in the broader landscape of time series forecasting and specifically to demonstrate its advantages in balancing accuracy and computational efficiency.

Evaluation metrics

The performance of the forecasting models was assessed using widely adopted error-based evaluation metrics, including *Mean Squared Error* (MSE), *Mean Absolute Error* (MAE), and *Root Mean Squared Error* (RMSE) [42]. These metrics quantify the deviation between the predicted values and the ground truth observations, providing complementary perspectives on prediction accuracy. Formally, the metrics are defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{16}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{17}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{18}$$

Here, y_i denotes the actual value, $\hat{y}_i$ the predicted value, and n the total number of observations. The MSE metric emphasizes larger errors due to the squaring of differences, making it sensitive to outliers. MAE measures the average absolute deviation between predictions and actual values, treating all errors linearly. RMSE provides an interpretable error in the same units as the original data, combining sensitivity to larger errors with direct interpretability.

In practice, MAE is useful for capturing general prediction deviation, while MSE and RMSE are more appropriate when larger errors need to be penalized. RMSE is often preferred when reporting results in the original scale of the variable, facilitating intuitive interpretation. By combining these metrics, we ensure a comprehensive evaluation of forecasting accuracy, balancing sensitivity to outliers and overall prediction deviation [42].

Training and evaluation protocol

A rigorous protocol was established for training and evaluating all models to ensure fair comparison and reproducible results.

Data splitting and validation

To maintain temporal integrity and prevent look-ahead bias, which is critical for time series forecasting, the dataset was split chronologically into training (70%), validation (20%), and testing (10%) sets. All feature scaling models (e.g., MinMaxScaler) were fitted exclusively on the training data and then applied to the validation and test sets to prevent data leakage. The validation set was used for hyperparameter tuning and to identify the best-performing model epoch during training.

Model training

All models were trained using the Adam optimizer, chosen for its adaptive learning rate capabilities, with an initial learning rate of 1×10^{-3} and a weight decay of 1×10^{-5} for regularization. The Mean Squared Error (MSE) was used as the loss function, as its sensitivity to larger errors is highly desirable in CPU usage forecasting, where significant prediction misses can lead to severe resource provisioning issues. A batch size of 64 was used for all experiments, and early stopping was employed based on validation loss with a patience of 10 epochs to prevent overfitting. Random seeds were fixed to ensure reproducibility across all training runs. Training was conducted for a maximum of 100 epochs, with model checkpoints saved every two epochs for consistent evaluation and recovery.

Regularization and overfitting prevention

Several techniques were employed to promote generalization and prevent overfitting. **Early stopping** was implemented with a patience of 10 epochs; training was halted if no improvement in validation loss was observed over this period. This ensures the model with the best generalization performance is selected. Additionally, **dropout** with a rate of $p = 0.2$ was applied after each LSTM block to further regularize the model.

Evaluation

The final evaluation was performed on the held-out test set using the model weights that yielded the lowest validation loss. To provide a comprehensive assessment of prediction accuracy, performance was measured using three metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). We evaluated performance across multiple forecast horizons ($H = 1, 5, 10, 30$) to understand the model's capabilities for both short-term and long-term prediction scenarios in cloud resource management. Fig. 4 illustrates the training and validation loss during the training process with a forecast horizon of $H = 30$. Additionally, a random sample from the test set is presented in Fig. 5, comparing the actual values against the trained model's predictions.

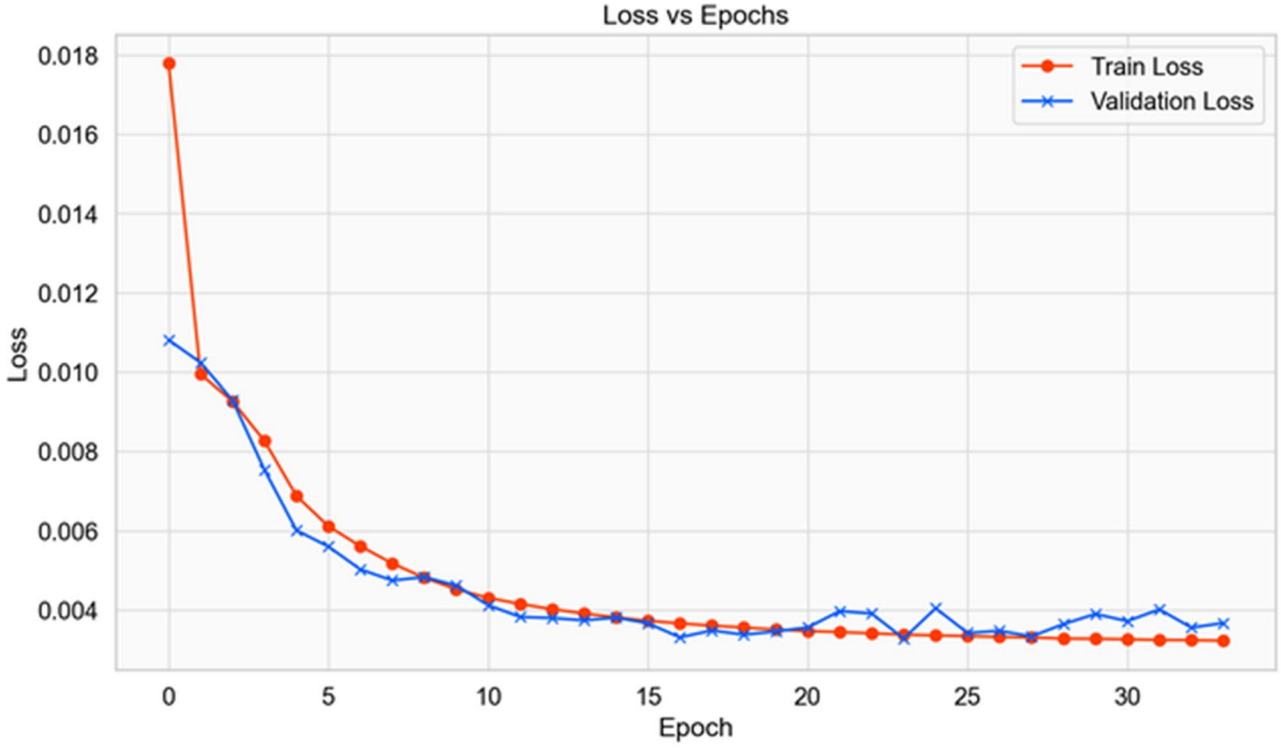


Fig. 4 An example of the training and validation loss curves, demonstrating model convergence and the point at which early stopping might be triggered

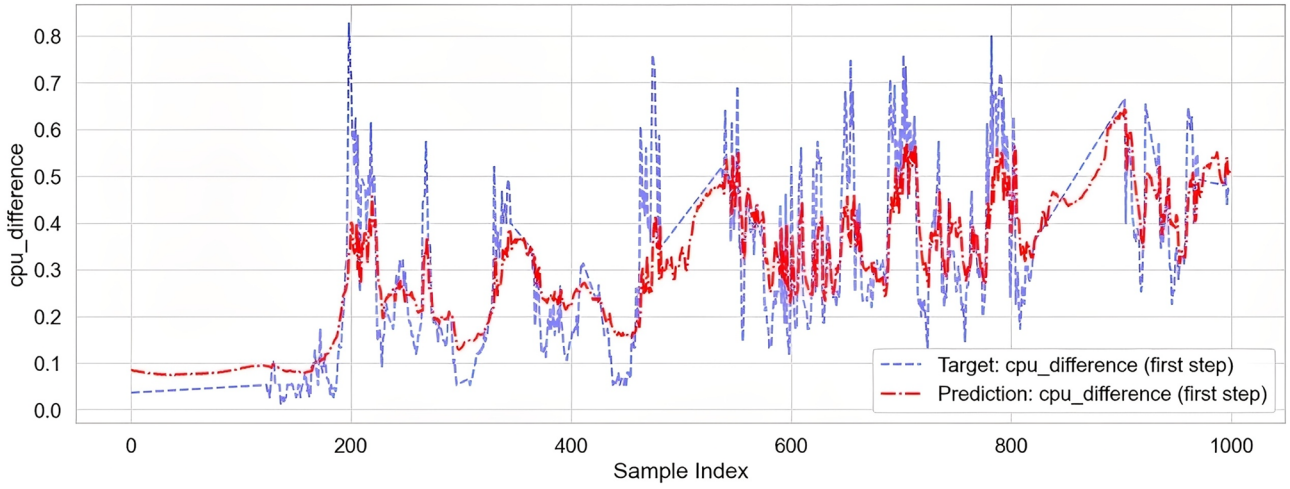


Fig. 5 A sample prediction from the trained model versus the actual ground truth values for a forecast horizon of H = 10

Table 3 CNN and SENet hyperparameters

Layer	Filters	Kernel Size	Stride	Padding / Ratio
Conv2D-1	64	(3, 3)	(1, 1)	same
Conv2D-2	128	(1, 1)	(1, 1)	same
SENet-1	—	—	—	16
SENet-2	—	—	—	16

Table 4 LSTM and DNN Hyperparameters

Layer	Input Size	Hidden Units	Special Settings
LSTM-1	128	64	batch_first = True
LSTM-2	64	32 × 2	bidirectional = True
LSTM-3	64	64	dropout = 0.2
LSTM-4	64	128	—
LSTM-5	128	64	dropout = 0.2
LSTM-6	64	32	dropout = 0.2
Fully Connected	32 → 32 → 30/10/5/1	ReLU activation	—

Implementation details

The entire experimental pipeline—including data preprocessing, feature engineering, and model training—was implemented in Python using `pandas`, `NumPy`, `scikit-learn`, and `PyTorch`, with visualization performed using `seaborn` and `matplotlib` [52]. All experiments were conducted on a workstation equipped with an `Intel Core i7 CPU`, `16 GB RAM`, and an `NVIDIA RTX 3060 GPU`, running `PyTorch 2.2.0` with `CUDA 12.2`.

Data preprocessing scripts automatically exported processed datasets into separate CSV files to ensure reproducibility across runs. Hyperparameters, random seeds, and data splits were fixed throughout all experiments to maintain consistency.

The proposed hybrid model integrates Convolutional (CNN), Squeeze-and-Excitation (SE), and LSTM layers to jointly capture local temporal dependencies and global frequency-domain relationships. Architectural configurations are summarized in Tables 3 and 4.

All inference time measurements were performed under identical hardware and software conditions using the same batch size and computational device. Minor variations across forecast horizons (e.g., $t = 1$, $t = 5$, $t = 10$) are attributed to CUDA kernel caching and initialization overhead amortization.

Results and discussion

This section evaluates the performance of the proposed **SpectraNet** model against recent deep learning baselines on the **Alibaba Cluster Trace (2018)** [48] and **vmCloud** [49] dataset, focusing on cloud workload forecasting. Models are compared using Mean Absolute Error (MAE) and Mean Squared Error (MSE) for predictive accuracy across forecast horizons ($t = 1$, 5, 10, 30). Additionally, model complexity (trainable parameters) and inference time are analyzed to assess computational efficiency,

Table 5 Model complexity and per-sample inference time comparison

Model	Params	Inference Time (ms)			
		t = 1	t = 5	t = 10	t = 30
SpectraNet	89,805	0.0279	0.0250	0.0148	**0.13**
MCDFN	1.76 M	0.0113	0.0151	0.1300	0.68
FreTS	2.0 M	0.0067	0.0095	0.0068	0.38
C LSTM	64,958	0.0070	0.0054	0.0077	0.7700
C LSTM AE	257,152	0.0084	0.0069	0.0095	0.8300
DLinear	36,000	**0.0032**	0.0035	0.0480	0.2130
TimeMixer	48,417	0.0248	0.0281	0.0247	0.2570
CNN 1D	207,155	0.0091	0.0074	0.0106	0.8000
LSTM DNN	**26,366**	0.0038	**0.0027**	**0.0044**	0.7900
T CNN LSTM	343,578	0.1000	0.1300	0.0245	0.6800

Table 6 Model performance across forecast horizons

Model	Forecast Horizon							
	t = 1		t = 5		t = 10		t = 30	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
SpectraNet	0.1200	0.0200	0.1500	0.0300	**0.0549**	**0.0058**	0.0447	0.0050
MCDFN	**0.0362**	0.0032	0.0542	**0.0054**	0.0584	0.0069	**0.0272**	**0.0020**
FreTS	0.0370	0.0031	0.0518	**0.0054**	0.0593	0.0070	0.0704	0.0095
C LSTM	0.0401	0.0034	0.0539	0.0057	0.0607	0.0067	0.0640	0.0078
C LSTM AE	0.0364	**0.0030**	**0.0495**	0.0055	0.0565	0.0064	0.0645	0.0076
DLinear	0.0369	0.0033	0.0508	0.0057	0.0592	0.0071	0.0696	0.0093
TimeMixer	0.0368	0.0032	0.0504	0.0056	0.0585	0.0068	0.0680	0.0092
CNN 1D	0.0430	0.0034	0.0705	0.0077	0.0586	0.0067	0.0704	0.0085
LSTM DNN	0.0376	0.0031	0.0508	0.0055	0.0597	0.0066	0.0664	0.0083
T CNN LSTM	0.1000	0.0100	0.1300	0.0250	0.0651	0.0082	0.0683	0.0088

critical for real-time resource allocation. The subsequent sections present a comprehensive evaluation of the proposed approach through quantitative comparisons, visual analysis, and detailed error analysis to validate its effectiveness.

Quantitative comparison

Table 5 highlights the model complexity and average per-sample inference times across different forecast horizons. All inference time measurements were conducted on a consistent hardware and software configuration: Intel Core i7, NVIDIA RTX 3060 GPU (12 GB VRAM), 16 GB RAM, and PyTorch 2.2.0 with CUDA 12.2. Each model was evaluated using a batch size of 1, with inference times averaged over 100 forward passes. We observed slightly shorter inference times at longer horizons (e.g., $t = 10$ vs. $t = 1$), which can occur due to GPU kernel scheduling and caching effects, where larger batched operations amortize overhead more efficiently than smaller ones. Ultra-lightweight models such as **LSTM DNN** [54, 55], containing only 26,366 parameters, achieve the fastest inference speeds, reaching 0.0044 ms at $t = 10$. However, this efficiency comes at the expense of predictive accuracy, with an MAE of 0.0664 at $t = 30$. In contrast, larger architectures, including **FreTS** [57] with 2.0 million parameters and **T CNN LSTM** [54, 55] with 343,578 parameters, exhibit substantially longer inference times ranging from 0.38 to 0.83 ms at $t = 30$, making them less suited for real-time deployment. DLinear [37], with its low parameter count, achieves the fastest inference at $t = 1$ yet fails to make considerable success at model performance. **SpectraNet**, however, provides a balanced profile, delivering near-best accuracy at $t = 30$, leading performance at $t = 10$, and among the fastest inference speeds for longer horizons, making it a strong candidate for resource-constrained environments.

Table 6 presents a detailed performance breakdown across forecast horizons of $t = 1$, $t = 5$, $t = 10$, and $t = 30$. **SpectraNet** demonstrates robust adaptability, with MAE decreasing from 0.1200 at $t = 1$ to 0.0447 at $t = 30$, and corresponding MSE values from 0.0200 to 0.0050. This trend indicates that the model's predictive capability strengthens over longer horizons, particularly where temporal and spectral fusion plays a significant role.

At $t = 10$, **SpectraNet** achieves the best overall results, with an MAE of 0.0549 and MSE of 0.0058, outperforming all baselines. At $t = 30$, it ranks second, with an MAE of 0.0447 and MSE of 0.0050, surpassed only by **MCDFN** [53], a model with 1.76 million parameters, in contrast to

SpectraNet's compact 89,805 parameters as reported in Table 5. These results demonstrate that even when not leading, **SpectraNet** remains highly competitive, achieving comparable accuracy while requiring a fraction of the computational resources of models more than 19 times its size.

Visual analysis

Figure 6 visualizes the trade-off between model complexity and accuracy. The bar plot compares normalized MSE and parameter counts, showing **SpectraNet** and **LSTM DNN** as efficient, with moderate sizes and competitive errors. Larger models like **FreTS** (2.0 M parameters) show diminishing returns, with marginal accuracy gains.

Figure 7, presents a bubble chart comparing the models across three dimensions: mean squared error (MSE), inference time, and model parameter size. The results highlight **SpectraNet** as a balanced solution, achieving low MSE and reduced inference time while maintaining a modest parameter size, making it well-suited for cloud workload forecasting.

- **X-axis (Inference Time)**: Models like **LSTM DNN** (0.0044 ms at $t = 10$) and **SpectraNet** (0.13 ms at $t = 30$) cluster on the left, indicating suitability for real-time applications.
- **Y-axis (Test MSE Loss)**: Lower values (e.g., **MCDFN**, **SpectraNet** at $t = 30$) reflect higher accuracy.
- **Bubble Size (Parameters)**: Smaller bubbles (e.g., **LSTM DNN**, **SpectraNet**) denote leaner models with less number of trainable parameter, while larger bubbles (e.g., **FreTS**, **MCDFN**) indicate higher complexity.

Error analysis

The primary evaluation metric, Mean Squared Error (MSE), highlights **SpectraNet**'s strength, particularly at longer forecast horizons. At $t = 10$ and $t = 30$, the model achieves MSE values of 0.0058 and 0.0050 respectively shown in Table 6, outperforming most baseline models, with the exception of **MCDFN** at $t = 30$. Notably, despite this slight gap at the longest horizon, **SpectraNet** maintains a significant advantage in computational efficiency: it is approximately 19 times lighter than **MCDFN**, with only 89,805 parameters compared to 1,760,000, and demonstrates around fivefold faster inference at $t = 30$, with per-sample inference time reaching 0.13 ms for **SpectraNet** and 0.68 ms for **MCDFN**, making it highly suitable for real-time deployment scenarios.

While the overall results indicate that SpectraNet achieves a strong balance between efficiency and forecasting accuracy, the trade-off between model size, inference time, and predictive performance is noteworthy. For shorter horizons (e.g., $t = 1$, $t = 5$), SpectraNet exhibits slightly higher inference latency compared to simpler models such as LSTM DNN or DLinear. This is primarily due to the additional computational overhead introduced by its multi-branch spectral-temporal fusion process. However, as the forecast horizon increases, this overhead becomes amortized across longer sequences, resulting in improved inference stability and superior accuracy. In contrast, larger models such as MCDFN or FreTS, despite achieving competitive or marginally better accuracy in isolated cases, incur significant computational costs and are less practical for real-time or resource-constrained settings. This nuanced trade-off analysis reinforces that SpectraNet offers an optimal middle ground—achieving a compact architectural design that sustains high accuracy and stable inference efficiency across multiple horizons.

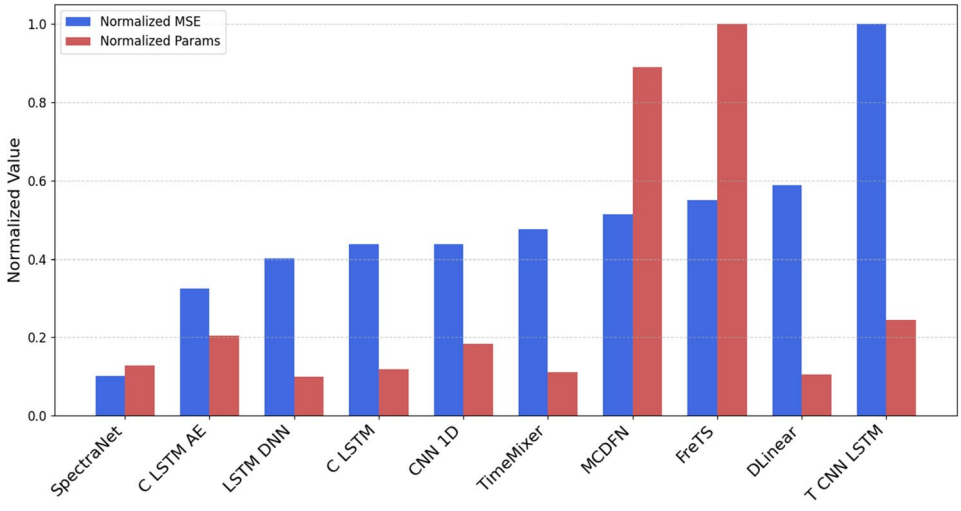


Fig. 6 Relationship between model size and prediction accuracy (Test MSE Loss) on normalized scale

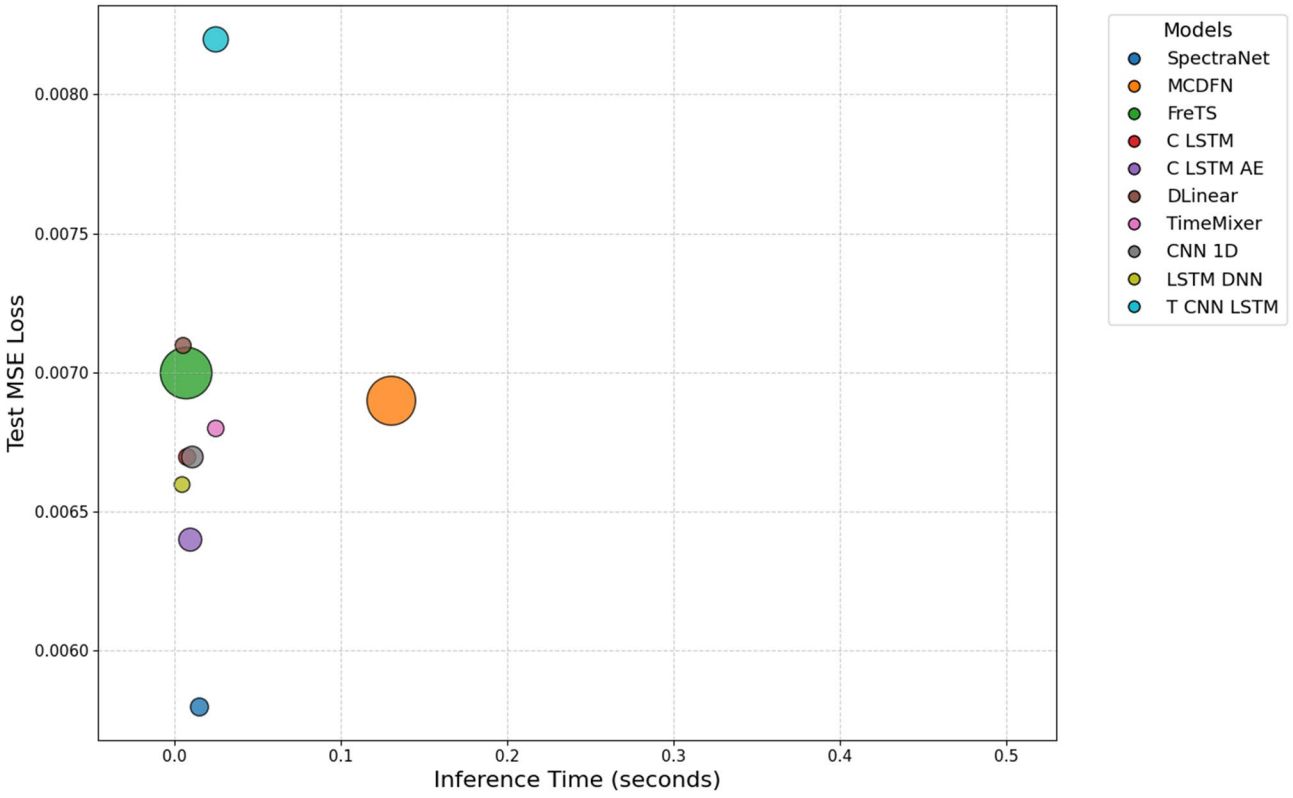


Fig. 7 Trade-off between model accuracy, speed, and size. This bubble chart visualizes the relationship between normalized MSE loss and normalized inference time for various models. The size of each bubble corresponds to the model's parameter count, illustrating the classic trade-off between accuracy, speed, and model complexity

Discussion

Table 6 and Table 5 together highlight the performance and efficiency of different models across forecast horizons. **SpectraNet** demonstrates robust adaptability, with MAE decreasing from 0.1200 at $t = 1$ to 0.0447 at $t = 30$, and corresponding MSE values from 0.0200 to 0.0050, indicating that its predictive capability strengthens over longer horizons where temporal and spectral fusion plays a critical role. At $t = 10$, it achieves the best overall results, with an MAE of 0.0549 and MSE of 0.0058, outperforming all baselines. At $t = 30$, it ranks second, with an MAE of 0.0447 and MSE of 0.0050, surpassed only by **MCDFN** [53], a model with 1.76 million parameters, in contrast to **SpectraNet**'s compact 89,805 parameters.

In terms of efficiency, ultra-light models such as **LSTM DNN** [54, 55] (26,366 parameters) achieve the fastest inference speeds (0.0044 ms at $t = 10$) but at the cost of a 48.55% reduction in accuracy (MAE: 0.0664 at $t = 30$) compared to **SpectraNet**(MAE: 0.0447 at $t = 30$). Larger architectures, including **FreTS** [57] with 2.0 million parameters and **T CNN LSTM** [54, 55] with 343,578 parameters, require substantially longer inference times (0.38–0.83 ms at $t = 30$) as well as TimeMixer (0.0248 ms at $t = 1$), limiting their suitability for real-time applications. In contrast, **SpectraNet** maintains near-leading accuracy while delivering fast inference speeds across all horizons, offering a well-balanced profile that combines high predictive performance with computational efficiency, making it an optimal choice for resource-constrained environments. SpectraNet is resource-agnostic and not limited to CPU metrics. It can naturally extend in several directions, including the development of multi-resource forecasting frameworks for joint prediction of CPU, memory, I/O, and network utilization. The proposed SpectraNet model can be integrated into modern cloud orchestration frameworks for proactive resource management. In a **Kubernetes** [58] setup, metrics from **Prometheus** [59] can feed into a REST API served via **PredictKube** [60], enabling near-real-time workload forecasts that drive auto-scaling (e.g., with the Horizontal Pod Autoscaler). This pipeline can be deployed on local (Minikube) or multi-node (kubeadm) clusters for evaluation using realistic workloads from tools like stress-ng or scalable web apps. Accurate forecasts help prevent over- and under-provisioning, reducing usage costs, energy use, and carbon footprint while minimizing cold-start delays and maintaining SLA compliance—effects well supported by prior studies

linking predictive accuracy to efficiency and sustainability improvements [61–63].

Conclusion and future work

This study presents **SpectraNet**, a lightweight hybrid deep learning model that unifies time- and frequency-domain analysis for accurate CPU usage forecasting in cloud environments. By integrating attention-driven spectral refinement with temporal feature fusion, **SpectraNet** achieves state-of-the-art prediction accuracy while being significantly smaller and faster than conventional models, making it highly suitable for deployment in energy-aware data centers. Its efficiency enables precise resource allocation, reducing over-provisioning, operational costs, and carbon emissions without compromising service quality. Beyond CPU prediction, **SpectraNet**'s hybrid-domain design demonstrates strong potential for broader applications, including IoT systems, edge analytics, and smart infrastructure, where lightweight and adaptive forecasting is critical.

Future research in this direction could explore transformer-inspired hybrid architectures to better capture long-range dependencies, multi-resource forecasting to handle correlated metrics such as memory and network utilization, and cross-domain generalization to enhance robustness across diverse workloads. Moreover, integrating such models with intelligent orchestration and scheduling frameworks may further enable closed-loop optimization in large-scale cloud systems. Overall, this work establishes a practical and extensible foundation for building sustainable, scalable, and energy-efficient AI systems across diverse computing environments.

Acknowledgements
Nahin F. Siddiqui: Writing - review & editing - original draft, Methodology, Investigation, Formal analysis. Zarif Safwan Hoque: Writing - review & editing, Methodology, Data curation. Md. Ehsanul Haque: Writing - review & editing, Methodology, Data curation. Abu Raihan Mostofa Kamal: Rreview & editing, Investigation, Project administration. AKM Azad: Writing - review & editing, Project administration, Investigation, Conceptualization. Salem Ali Alyami: Investigation, Project administration, Data curation. Md Azam Hossain: Writing - review & editing, Project administration, Methodology, Investigation, Formal analysis, Conceptualization.

Authors contributions
Nahin F. Siddiqui: Writing – review & editing– original draft, Methodology, Investigation, Formal analysis. Zarif Safwan Hoque: Writing – review & editing, Methodology, Data curation. Md. Ehsanul Haque: Writing – review & editing, Methodology, Data curation. Abu Raihan Mostofa Kamal: Rreview & editing, Investigation, Project administration. AKM Azad: Writing – review & editing, Project administration, Investigation, Conceptualization. Salem Ali Alyami: Investigation, Project administration, Data curation. Md Azam Hossain: Writing – review & editing, Project administration, Methodology, Investigation, Formal analysis, Conceptualization

Funding
This work was supported and funded by the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) (grant number IMSIU-DDRSP2502).

Data availability
No datasets were generated or analysed during the current study.

Declarations

Supplementary information
Our source code is available at https://github.com/lingdaizon-crypto/SpectraNet.git (accessed on November 2, 2025).

Competing interests
The authors declare no competing interests.

Received: 14 September 2025 / Accepted: 16 November 2025
Published online: 11 December 2025

References

1. Rossi A, Visentin A, Prestwich S, Brown KN (2022) Bayesian uncertainty modelling for cloud workload prediction. In 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), IEEE, Barcelona, Spain, 19–29
2. Cozzolino V, Tonetto L, Mohan N, Ding AY, Ott J (2022) Nimbus: towards latency-energy efficient task offloading for ar services. IEEE Trans on Cloud Comput 11(2):1530–1545
3. Kumar J, Saxena D, Singh AK, Vasilakos (2023) A.V.: a quantum controlled-not neural network-based load forecast and management model for smart grid. IEEE Syst J 17(4):5714–5725
4. Xu M, Buyya R (2019) Brownoutcon: a software system based on brownout and containers for energy-efficient cloud computing. J Syst And Softw 155:91–103
5. Bharany S, Sharma S, Khalaf OI, Abdulsahib GM, Al Humaimeedy AS, Aldhyani TH, Maashi M, Alkahtani (2022) H.: a systematic survey on energyefficient techniques in sustainable cloud computing. Sustainability 14(10):6256
6. Saxena D, Singh (2024) A.K.: a comprehensive survey on sustainable resource management in cloud computing environments. Authorea Preprints
7. Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw: Pract And Exper 41(1):23–50. https://doi.org/10.1002/spe.995
8. Lee YC, Zomaya AY (2012) Energy efficient utilization of resources in cloud computing systems. The J Supercomput 60:268–280
9. Wang H, Mathews KJ, Golec M, Gill SS, Uhlig S (2025) Amazonaicloud: proactive resource allocation using amazon chronos based time series model for sustainable cloud computing. Computing 107(3):77
10. Sarikaa S, Niranjana S, Sri Vishnu Deepika K (2021) Time series forecasting of cloud resource usage. In 2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA), IEEE, Arad, Romania, 372–382
11. Pomeroy J (2024) Transforming business intelligence: leveraging generative AI and Predictive analytics in cloud environments. https://doi.org/10.13140/RG.2.2.34053.46560
12. Nelson BK (1998) Time series analysis using autoregressive integrated moving average (arima) models. Academic Emerg Med 5(7):739–744
13. Masdari M, Khoshnevis A (2020) A survey and classification of the workload forecasting methods in cloud computing. Cluster Comput 23(4):2399–2424
14. Sahay MS, Ganta S, Vardhan BNV, Srilakshmi K (2024) A comprehensive study on improving time series forecasting precision. Semin Med Writ And Educ 3 7
15. Casolaro A, Capone V, Iannuzzo G, Camastra F (2023) Deep learning for time series forecasting: advances and open problems. Information 14(11):598
16. Yang Z, Yin M, Liao J, Xie F, Zheng P, Li J, Hua B (2025) Fftnet: fusing frequency and temporal awareness in long-term time series forecasting. Electronics 14(7):1303
17. Yemets K, Izonin I, Dronyuk I (2025) Enhancing the fft-lstm time-series forecasting model via a novel fft-based feature extraction–extension scheme. Big Data Cognit Comput 9(2):35
18. Zhao F, Lin W, Lin S, Zhong H, Li K (2024) Tfegru: time-frequency enhanced gated recurrent unit with attention for cloud workload prediction. In IEEE Transactions on Services Computing
19. Dogani J, Khunjush F, Seydali M (2023) Host load prediction in cloud computing with discrete wavelet transformation (dwt) and bidirectional gated recurrent unit (bigru) network. Comput Commun 198:157–174

20. Bi J, Ma H, Yuan H, Buyya R, Yang J, Zhang J, Zhou M (2024) Multivariate resource usage prediction with frequency-enhanced and attention-assisted transformer in cloud computing systems. In IEEE Internet of Things Journal
21. Zhang Y, Zhou X, Zhang Y, Li S, Liu S (2025) Improving time series forecasting in frequency domain using a multi resolution dual branch mixer with noise insensitive arctanloss. Sci Rep 15(1):12557
22. Yi K, Zhang Q, Fan W, Wang S, Wang P, He H, Lian D, An N, Cao L, Niu Z (2023) Frequency-domain mlps are more effective learners in time series forecasting. https://arxiv.org/abs/2311.06184
23. Pintelon R, Schoukens J (1999) Time series analysis in the frequency domain. IEEE Trans on Signal Process 47(1):206–210. https://doi.org/10.1109/78.738253
24. Reddy R (2024) Sustainable computing: a comprehensive review of energy-efficient algorithms and systems. TechRxiv Preprints
25. Alhartomi M, Salh A, Audah L, Alzahrani S, Alzahmi A (2024) Enhancing sustainable edge computing offloading via renewable prediction for energy harvesting. IEEE Access 12:74011–74023
26. Clemm C, Stobbe L, Wimalawarne K, Druschke J (2024) Towards green ai: current status and future research. In 2024 Electronics Goes Green 2024+(EGG), IEEE, Berlin, Germany, 1–11
27. Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Salt Lake City, UT, USA, 7132–7141
28. Nashold L, Krishnan R (2020) Using lstm and sarima models to forecast cluster cpu usage. arXiv preprint arXiv:2007.08092
29. Chen J, Wang Y (2019) A hybrid method for short-term host utilization prediction in cloud computing. J Electr And Comput Eng 2019(1):2782349
30. Maiyza AI, Korany NO, Banawan K, Hassan HA, Sheta WM (2023) Vtgan: hybrid generative adversarial networks for cloud workload prediction. J Cloud Comput 12(1):97
31. Liu Y, Zhang C, Song J, Chen S, Yin S, Wang Z, Zeng L, Cao Y, Jiao J (2025) Mofe-time: mixture of frequency domain experts for time-series forecasting models. arXiv preprint arXiv:2507.06502
32. Zhao F, Lin W, Lin S, Zhong H, Li K (2025) Tfegru: time-frequency enhanced gated recurrent unit with attention for cloud workload prediction. IEEE Trans On Serv Comput 18(1):467–478
33. Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE (1991) Adaptive mixtures of local experts. Neural Computation 3(1):79–87
34. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, vol 30. Long Beach, California, USA, pp 5998–6008
35. Oppenheim AV, Schafer RW (2009) Discrete-time signal processing, 3rd edn. Prentice Hall, Upper Saddle River, NJ
36. Cen S, Kim DO, Lim CG (2023) A fused cnn-lstm model using fft with application to real-time power quality disturbances recognition. Energy Sciamp Eng 11(7):2267–2280
37. Zeng A, Chen M, Zhang L, Xu Q (2023) Are transformers effective for time series forecasting? In Proceedings of the AAAI Conference on Artificial Intelligence, vol 37. pp 11121–11128
38. Wang S, Wu H, Shi X, Hu T, Luo H, Ma L, Zhang JY, Zhou J (2024) Timemixer: decomposable multiscale mixing for time series forecasting. arXiv preprint arXiv:2405.14616
39. Zhao F, Lin W, Lin S, Tang S, Li K (2025) Mscnet: multi-scale network with convolutions for long-term cloud workload prediction. In IEEE Transactions on Services Computing
40. Lin S, Lin W, Wu W, Chen H, Yang J (2024) Sparsetsf: modeling long-term time series forecasting with 1k parameters. arXiv preprint arXiv:2405.00946
41. Lin S, Lin W, Wu W, Wang S, Wang Y (2024) Petformer: long-term time series forecasting via placeholder-enhanced transformer. In IEEE Transactions on Emerging Topics in Computational Intelligence
42. Hossin M, Sulaiman MN (2015) A review on evaluation metrics for data classification evaluations. Int J Data Min Knowl Manag Process 5(2):1
43. Calzarossa MC, Della Vedova ML, Massari L, Petcu D, Tabash MI, Tessera D (2016) Workloads in the clouds. In: Principles of performance and reliability modeling and evaluation: essays in honor of Kishor Trivedi on His 70th Birthday. Springer, pp 525–550
44. Duhamel P, Vetterli M (1990) Fast fourier transforms: a tutorial review and a state of the art. Signal Process 19(4):259–299
45. Gomede E (2023) The fourier transform and its application in machine learning. Medium, TDS Archive
46. Qiu S (2018) Global weighted average pooling bridges pixel-level localization and image-level classification. arXiv preprint arXiv:1809.08264
47. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2016) Lstm: a search space odyssey. IEEE Trans on Neural Networks And Learn Syst 28(10):2222–2232
48. (2018) Alibaba: Alibaba cluster Trace Data v2018. https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018. Accessed: 2025-05-28
49. (2023) Entony: cloud computing performance metrics. Kaggle. Accessed:2025–05–28. https://www.kaggle.com/dsv/6165137
50. McKinney W et al. (2010) Data structures for statistical computing in python. scipy 445(1):51–56
51. Hastie T (2009) The elements of statistical learning: data mining, inference, and prediction. Springer
52. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V et al. (2011) Scikit-learn: machine learning in python. The J Mach Learn Res 12:2825–2830
53. Jahin MA, Shahriar A, Amin MA (2025) Mcdfn: supply chain demand forecasting via an explainable multi-channel data fusion network model. Evol Intel 18(3):66
54. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
55. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780
56. Lea C, Flynn MD, Vidal R, Reiter A, Hager (2017) G.D. Temporal convolutional networks for action segmentation and detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 156–165
57. Yi K, Zhang Q, Fan W, Wang S, Wang P, He H, An N, Lian D, Cao L, Niu Z (2023) Frequency-domain mlps are more effective learners in time series forecasting. Adv Neural Inf Process Syst 36 76656–76679
58. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, omega, and kubernetes. Commun of The ACM 59(5):50–57. https://doi.org/10.1145/2890784
59. Volz J, Rabenstein B (2015) Prometheus: a next-generation monitoring system. In USENIX SREcon Europe 2015, Talk. https://www.usenix.org/conference/srecon15europe/program/presentation/rabenstein
60. (2022). Dysnix: PredictKube: AI-based Kubernetes Autoscaler for cloud workloads. https://keda.sh/blog/2022-02-09-predictkube-scaler/blog/technical.case.study
61. Vu D-D, Tran M-N, Kim Y (2022) Predictive hybrid autoscaling for containerized applications. IEEE Access 10:109768–109778
62. Ju L, Singh P, Toor S (2021) Proactive autoscaling for edge computing systems with kubernetes. In Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion, pp 1–8
63. Guruge PB, Priyadarshana Y (2025) Time series forecasting-based kubernetes autoscaling using facebook prophet and long short-term memory. Front Comput Sci 7:1509165

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.