frontiers | Frontiers in Computer Science

TYPE Original Research
PUBLISHED 19 February 2025
DOI 10.3389/fcomp.2025.1509165

OPEN ACCESS

EDITED BY
Kevin Lano,
King's College London, United Kingdom

REVIEWED BY
Rizwan Raheem Ahmed,
Indus University, Pakistan
Ersin Aytaç,
Bülent Ecevit University, Türkiye
Alireza Rouhi,
Azarbaijan Shahid Madani University, Iran

*CORRESPONDENCE
Pasan Bhanu Guruge
✉ w1867877@my.westminster.ac.uk

RECEIVED 10 October 2024
ACCEPTED 29 January 2025
PUBLISHED 19 February 2025

CITATION
Guruge PB and Priyadarshana YHPP (2025) Time series forecasting-based Kubernetes autoscaling using Facebook Prophet and Long Short-Term Memory.
Front. Comput. Sci. 7:1509165.
doi: 10.3389/fcomp.2025.1509165

Time series forecasting-based Kubernetes autoscaling using Facebook Prophet and Long Short-Term Memory

Pasan Bhanu Guruge[1]* and Y.H.P.P. Priyadarshana[2]

[1]College of Design, Creative and Digital Industries, University of Westminster, London, United Kingdom, [2]School of Computing, Informatics Institute of Technology, Colombo, Sri Lanka

The advancement of cloud computing technologies has led to increased usage in application deployment in recent years. Kubernetes, a widely used container orchestration platform for deploying applications on cloud systems, provides benefits such as autoscaling to adapt to fluctuating workload while maintaining quality of service and availability. In this research, we designed and evaluated a proactive Kubernetes autoscaling using Facebook Prophet and Long Short-Term Memory (LSTM) hybrid model to predict the HTTP requests and calculate required pod counts based on the Monitor-Analyze-Plan-Execute loop. The proposed model not only captures seasonal data patterns effectively but also proactively predicts the pod requirements for timely and efficient resource allocation to reduce resource wastage while enhancing cloud computing applications. The proposed hybrid model was evaluated using real-world datasets from NASA and the Federation Internationale de Football Association (FIFA) World Cup to benchmark and compare with existing literature. Evaluation results indicate that the proposed novel hybrid model outperforms single-model proactive autoscaling by a maximum margin of 65–90% accuracy when compared to NASA and FIFA World Cup datasets. This study contributes to the fields of cloud computing and container orchestration by providing a refined proactive autoscaling mechanism that enhances application availability, efficient resource usage, and reduced costs and paves the way for further exploration in increased prediction speed, integrated with vertical scaling and implementations using Kubernetes.

KEYWORDS

Kubernetes, proactive autoscaling, LSTM, time-series forecasting, Facebook Prophet, Kubernetes autoscaling

1 Introduction

The advancement of cloud computing has fundamentally transformed the landscape of application development and deployment (Al-Dhuraibi et al., 2018). One core component of cloud computing is virtualization technology—often referred to as containerization and Kubernetes. Virtualization provides an abstraction layer to application deployment without coupling with underlying host devices, allowing application developers and infrastructure engineers to use the most suitable technology for the designed application (Varghese and Buyya, 2018).

The advancement of containerization technology allows for research and development in application development, deployment, and container orchestration platforms. Containerization enables developers to encapsulate the application and its dependencies into a single package that operates in a decoupled manner with the host machine

(Hardikar et al., 2021). The introduction of microservice architecture has significantly improved the ability to use containerization and Kubernetes deployment in cloud environments.

Initially developed as an open-source project, Kubernetes is a technology that abstracts away the underlying infrastructure of the deployment. The advancements in cloud computing allow engineers to focus away from resources and move to application-centric deployments and management. Kubernetes introduced the concepts of state management and autoscaling, making the applications highly available and reliable in large-scale deployments (Shah and Dubaria, 2019). Kubernetes allows the deployment of different workloads and combines multiple deployment targets, making it an indispensable part of modern cloud infrastructure (Megino et al., 2020).

Adopting the advanced forecasting models in Kubernetes autoscaling leads to significantly reducing the wastage of resources while improving the application availability and timely scaling actions (Marie-Magdelaine and Ahmed, 2020). The application of reinforcement learning for dynamic autoscaling allows systems to learn and adapt to change the scaling conditions based on the feedback (Garí et al., 2021). Using hybrid models combining two or more machine learning methods also shows improved forecasting accuracy compared to single model furcating (Podolskiy et al., 2018).

The integration of machine learning models into Kubernetes autoscaling has made significant improvements, but current solutions lack the ability to capture complex seasonal patterns and workload fluctuations accurately. Current research has examined either statistical methods or neural network-based approaches; however, limited studies integrate both methodologies to exploit their advantages in Kubernetes autoscaling. This raises a critical question: How can a hybrid time series forecasting model that combines statistical and neural network-based approaches enhance the accuracy and efficiency of proactive autoscaling in Kubernetes compared to single-model solutions and default Kubernetes HPA? This study aims to develop and assess a hybrid model-based proactive Kubernetes autoscaling framework that enhances cloud computing applications' workload prediction and scaling accuracy for cloud computing applications.

The proposed framework utilizes time-series analysis (TSA), a mechanism that can be used in forecasting future trends and patterns based on historical data to predict future resource requirements. TSA has two significant implementations. Statistical methods such as Autoregressive Integrated Moving Average (ARIMA) model, Seasonal Autoregressive Integrated Moving Average (SARIMA) model, Facebook Prophet (also known as Prophet), and neural network-based methods such as Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (Bi-LSTM). The practical applications of TSA methods have been widely discussed in both workload prediction and economics (Ahmed et al., 2021), weather, healthcare, and logistics. The proposed framework uses a combination of a statistical model, Prophet, and a neural network-based method, LSTM, to build a hybrid forecasting model.

Significant contributions from this study are listed as follows:

1 Using a hybrid model to increase the prediction accuracy of a seasonal data series in TSA, highlighting the accuracy compared to other TSA methods.

2 Hybrid model-based proactive autoscaling architecture for Kubernetes highlights higher accuracy in scaling decisions than default Kubernetes HorizontalPodAutoscaler (HPA) and single-model-based solutions.

The proposed algorithm was tested and benchmarked against real-world datasets to compare with default HPA and single model-based solutions. The proposed algorithm was compared against different dataset types with complex seasonal patterns and different hybrid model combinations to test efficiency and accuracy.

2 Related work

This section provides an overview of current research that has examined the application of TSA techniques in cloud systems and Kubernetes for autoscaling applications. TSA is utilized in various fields, including weather forecasting, earthquake prediction, and mathematical finance. The method uses past and present observed data to predict future values. Table 1 shows a comprehensive overview of significant autoscaling techniques employed in cloud and Kubernetes applications.

2.1 Reactive autoscaling

HPA is the default scaling mechanism in Kubernetes. It employed rule-based reactive scaling based on the pod's CPU or memory consumption (Kubernetes Documentation, 2024). HPA is effective in many scenarios, but its reactive nature can lead to overprovision or performance degradations due to the cold start time of the applications (Pahl and Lee, 2015).

Zhang et al. (2009) presented a framework to optimize cost and resources in hybrid clouds by distributing the workloads between private and public clouds using ARIMA-based TSA. Al-Dhuraibi et al. (2017) proposed an architecture called ELASTICDOCKER, an autonomic vertical elasticity system for Kubernetes that automatically adjusts the resources allocated to containers based on workload demands. Even though vertical scaling comes with default Kubernetes, the limitations of unavailability in Docker deployments, human intervention, host machine capacity issues, and performance degradations were addressed with ELASTICDOCKER. ELASTICDOCKER framework improved resource utilization, optimized cost and enhanced application performance. Yan et al. (2021) developed the HANSEL framework, which is focused on adaptive horizontal scaling for microservices. It uses Bi-LSTM to capture complex patterns in workload data to generate scaling decisions.

However, the reactive approaches have limits in scaling since the system can only react after the actual workload change happens. In contrast, the proactive approach studies past data and predicts the demand.

2.2 Proactive autoscaling

Prachitmutita et al. (2018) present a cost-effective autoscaling framework for an Infrastructure as a Service (IaaS) platform using

TABLE 1 Overview of Kubernetes autoscaling with TSA.

Study/Resource	Target	Metric	Method	Technique
Kubernetes HPA	Pod	CPU, memory	Reactive	Rule based
Al-Dhuraibi et al. (2017)	Pod	CPU, memory	Reactive	Rule based
Zhang et al. (2009)	Node	Request rate	Reactive	ARIMA[1]
Yan et al. (2021)	Pod	CPU, memory	Hybrid	Bi-LSTM[2]
Prachitmutita et al. (2018)	Pod	Request rate	Proactive	ANN[3], RNN[4]
Calheiros et al. (2015)	Node	Request rate	Proactive	ARIMA
Messias et al. (2015)	-	Request rate	Proactive	GA[5]
Borkowski et al. (2016)	-	Task	Proactive	ANN
Fang et al. (2012)	Pod	CPU	Proactive	ARMA[6]
Ciptaningtyas et al. (2017)	Pod	Request rate	Proactive	ARIMA
Imdoukh et al. (2019)	Pod	Request rate	Proactive	LSTM
Tang et al. (2018)	Pod	CPU	Proactive	Bi-LSTM
Toka et al. (2020)	Pod	Request rate	Proactive	AR[7], HTM[8], LSTM
Dang-Quang and Yoo (2021)	Pod	Request rate	Proactive	Bi-LSTM

[1]Autoregressive integrated moving average model.
[2]Bidirectional long short-term memory.
[3]Artificial neural networks.
[4]Recurrent neural networks.
[5]Genetic algorithm.
[6]Autoregressive moving average model.
[7]Autoregressive.
[8]Hierarchical temporal memory.

ANN and RNN for the predictions and a resource scaling optimization algorithm. It showed that the accuracy of the ARIMA model became worse with more predicted steps ahead. Fang et al. (2012) propose a novel Resource Prediction and Provisioning Scheme (RPPS) for cloud data centers for hybrid clouds. RPPS uses CPU utilization to predict future resource requirements through ARMA, which achieves better results than default HPA. Borkowski et al. (2016) used ANN to forecast task duration and resource utilization. This approach focuses on task-based applications only. Ciptaningtyas et al. (2017) proposed an ARIMA-based resource elasticity controller for docker-based web applications. The proposal is limited to predicting resource allocations only. Tang et al. (2018) proposed a workload prediction model, "Fisher" for Docker-based environments using Bi-LSTM. This study only discusses the prediction part and lacks autoscaling.

Calheiros et al. (2015) proposed ARIMA-based workload prediction, which can provision resources proactively based on HTTP requests. The results show that the model provides 91% accuracy with seasonal data. This proposed method lacks accuracy in non-seasonal workloads and comparison with other approaches. Messias et al. (2015) explore using genetic algorithms (GAs) to combine time-series prediction models for autoscaling web applications. However, this proposal lacks the Kubernetes integration. Imdoukh et al. (2019) proposed an LSTM-based autoscaling framework for Kubernetes pods. The authors concluded that the proposed LSTM model has a slightly higher error than ARIMA in 1 step, but LSTM has a higher prediction speed. Toka et al. (2020) proposed a proactive Kubernetes autoscaling using AI-based forecasting. The proposed proactive auto scaler uses the best model from Autoregressive (AR), Hierarchical Temporal Memory (HTM), and LSTM in the given moment. This proposed method can be considered integration-friendly despite lacking forecasting accuracy and validation results against standard datasets.

Dang-Quang and Yoo (2021) also proposed a proactive Kubernetes autoscaling model using Bi-LSTM. The authors discussed the implementation in Kubernetes and validated the results using (Arlitt and Jim, 1998) and (NASA-HTTP, 1995). The proposed implementation was evaluated against an actual application that uses the Resource Removal Strategy (RRS) to optimize resource usage and cost. Even though this study did not discuss the seasonality component, results show that the proposed model performs better than ARIMA and LSTM.

About published studies, the majority of the reactive approaches use rule-based models to perform autoscaling. In the existing implementations, reactive approaches are common and widely used due to implementation simplicity, despite limitations such as correct threshold determination problems and overprovisioning. In contrast, proactive methods focus more on workload predictions based on metrics, such as CPU usage, memory usage, or request rate. The majority of related work focused on Dockized environments, which are not widely used in enterprise-grade systems. Toka et al. (2020) and Dang-Quang and Yoo (2021) published Kubernetes-related proactive autoscaling approaches, which can be categorized as more relevant to the current technology landscape. None of the studies discussed using the hybrid model and seasonality capturing to improve the prediction accuracy in proactive autoscaling.

Based on an analysis of the literature, there is a significant research opportunity for Kubernetes in the field of proactive autoscaling. Therefore, driven by the aforementioned issues, we suggest implementing a proactive Kubernetes autoscaling that utilizes a hybrid model combining Prophet and LSTM. This model aims to enhance forecast accuracy by effectively capturing seasonality patterns and conducting residual analysis.

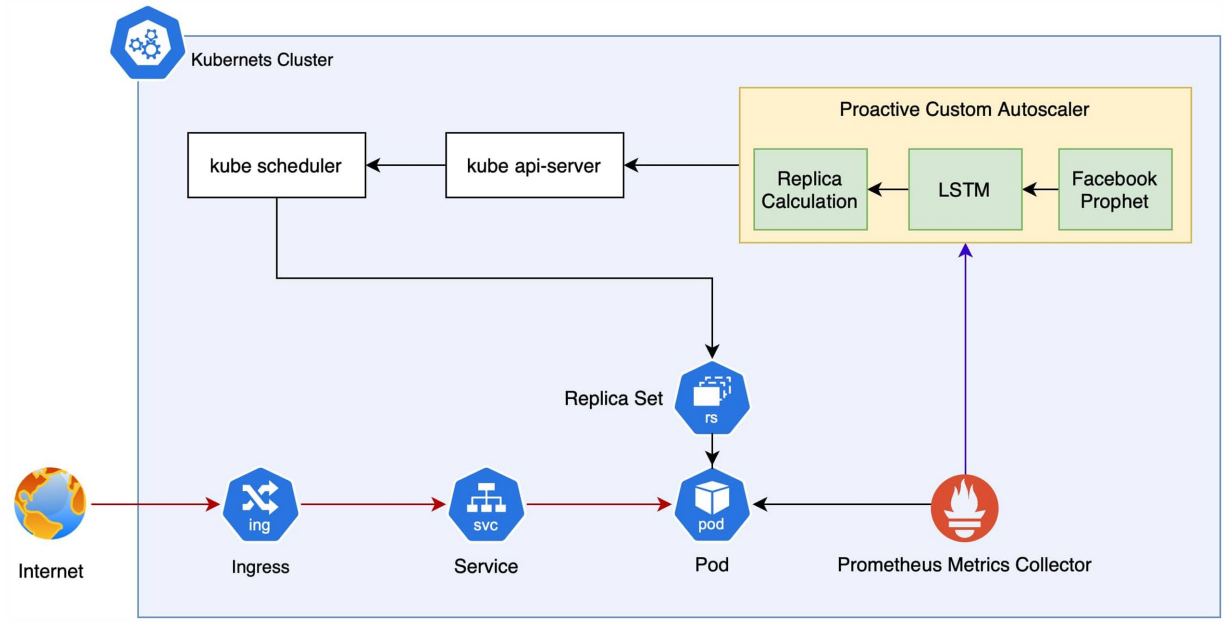

FIGURE 1
System architecture.

3 System architecture

In this section, we discuss the system architecture of the proposed hybrid model and proactive autoscaling, as shown in Figure 1.

3.1 Hybrid model

The proposed hybrid model integrates the Prophet and LSTM models to increase the prediction accuracy needed for web application scaling. The core idea is to leverage Prophet's ability to capture seasonality and combine the LSTM strength in residual analysis. Even though Prophet is a statistical model, it is well-known for its effectiveness in predicting time-series data with multiple seasonality patterns. After data are processed through the Prophet model, the complex, non-linear residual time series is predicted by LSTM. Residuals in time series analysis are the differences between observed and predicted values. They represent the portion of the data that the model cannot explain and are essential for assessing the model's performance and validity. Without a deep learning model, the error component is higher and unreliable for real-world use cases.

3.1.1 Metric selection for forecasting

The metric used by the proposed model is the request rate, which translates to HTTP requests experienced by a pod at a fixed interval. Unlike CPU and Memory usage, the HTTP request rate directly reflects the demand on the application, which can be considered more relevant in scaling decisions. Kubernetes default HPA does not support request rate as a scaling parameter. Still, for reactive scaling, the request rate can be used via the Kubernetes-based event-driven autoscaling component (KEDA) Project, which supports external or application-fed metrics via Prometheus (Bartelucci and Bellavista, 2023). For the proposed model, by focusing on the HTTP request rate, the relevance of proactive scaling can be increased due to its direct relation to demand on the application (Zhang et al., 2009).

3.1.2 Facebook prophet model

The Facebook Prophet model is capable of handling multiple seasonality patterns. For the model evaluation, the daily and weekly seasonality is only used, and the model is capable of supporting yearly and custom seasonality. Also, the model supports configuring holidays and growth patterns for the overall trend (Taylor and Letham, 2018). Prophet can be considered as a nonlinear regression model in the form of Equation 1.

$$y_t = g(t) + s(t) + h(t) + \varepsilon_t \tag{1}$$

where $g(t)$ describes the growth trend, $s(t)$ denotes seasonal patterns, $h(t)$ captures holiday effects and ε_t is the white noise error term. The trend growth patterns are experimented with the appropriate approach for testing datasets in the validation stages.

Facebook Prophet model was used effectively in various forecasting applications such as export quantity forecasting (Aytaç, 2021), cryptocurrency prices (Cheng et al., 2023), stock market predictions (Annapoorna et al., 2024) and environmental studies (Bekkar et al., 2024).

3.1.3 Long short-term memory

LSTM is considered a member of the deep RNN family. Introduced by Hochreiter and Schmidhuber (1996), it has a modified version of RNN architecture. In contrast to conventional neural networks, RNNs use previous and current steps to build the network. Even though RNNs are simple and powerful models, the model has challenges in training and experiencing exploding gradient issues. LSTM was proposed (Pascanu et al., 2012) To address these issues. In the proposed hybrid model, the

LSTM model is used for processing the residuals after seasonality removal. The LSTM model is capable of learning from the temporal dependencies within the residuals. The proposed model has two layers with 50 units and one dense layer. Equations 2–4 can be used to express LSTM model.

$$i_t = \sigma(\omega_i [h_{t-1}, x_t] + b_i \tag{2}$$

$$f_t = \sigma(\omega_f [h_{t-1}, x_t] + b_f \tag{3}$$

$$o_t = \sigma(\omega_o [h_{t-1}, x_t] + b_o \tag{4}$$

where i_t represents input gate, f_t represents forget gate, o_t represents the output gate, σ represents the sigmoid function, ω_i, ω_f, and ω_o are weight for the neurons, and b_i, b_f, and b_o are the bias of gates. h_{t-1} is the output from the previous LSTM block and x_t is the input at the current timestamp.

LSTM can be seen in more diverse forecasting applications, such as Kubernetes workload predictions (Imdoukh et al., 2019), wind speed forecasting (Yang et al., 2024), stock market price predictions (Kothari et al., 2024) and power generation related studies (Abumohsen et al., 2024).

3.1.4 Time complexity analysis

Time complexity analysis of prediction models provides valuable information on prediction latency and computational demand. The time complexity of the Prophet model can be approximated as due to its reliance on Fourier transformation and additive regression modeling as follows in Equation 5.

$$O\left(T \cdot (k + m + n)\right) \tag{5}$$

where k is the order of the Fourier series, m is the number of change points and n is the iterations required for optimization.

The time complexity of the LSTM model can be expressed as follows in Equation 6.

$$O\left(T \cdot n \cdot m\right) \tag{6}$$

where T is the sequence length, n is the number of input features and m is the number of hidden units.

The linear complexity with respect to T ensures scalability for large datasets. Integrating the Prophet model with other forecasting mechanisms, such as LSTM or ensemble models, can significantly increase the computational demands of both prediction latency and resource requirements (Wang and Gu, 2023).

3.2 Proactive autoscaling

The proposed proactive custom autoscaling architecture consists of a Kubernetes cluster, HPA, Metric collector, and Custom autoscaling. The proposed system follows the Monitor-Analyze-Plan-Execute (MAPE) loop for scaling decisions, considering the accuracy and effectiveness shown by Dang-Quang and Yoo (2021).

3.2.1 Kubernetes components

The components indicated in the architecture related to the Kubernetes ecosystem are as follows:

- Ingress exposes the HTTP and HTTPS routes from the internet to the services in the cluster.
- Service is a method for exposing a network application that runs as one or multiple pods in the cluster. Services help to service discovery inside the cluster without modifying the application.
- ReplicaSet (RS) controls how many pods must be deployed in the cluster. Autoscaling adjust the desired pod count in RS to execute scale-up/down commands.
- Pod is a workload or application running in the cluster. This is the target of the autoscaling (Ibryam and Huß, 2023).
- Kubernetes scheduler and Kubernetes API-server are Kubernetes control plane components. The Kubernetes scheduler is designed to assign the new pods to the most appropriate nodes. Kubernetes API-server is exposed to Kubernetes API. These APIs are used to collect and control the Kubernetes cluster (Kubernetes Documentation, 2024).

Prometheus is used as the metric collector in the proposed architecture. It is a robust, widely used open-source monitoring, time-series database. It enables autoscaling to access each scaling target pod required matric request rate (Chen et al., 2020).

3.2.2 Scaling loop

The proposed system architecture uses a MAPE loop to automate the scaling decisions. The MAPE control loop, shown in Figure 2, is not a sequential process but a structural arrangement of its subprocesses (Ashraf et al., 2023).

Monitoring Phase: Prometheus metric collector, an application monitoring service, shown in Figure 1 collects the incoming HTTP request rate for the pods. After collection, the Prometheus server aggregated and added the data to the built-in time-series database for predictions and model training during the phase of the analysis.

Forecasting Phase: The developed Prophet and LSTM hybrid model forecasts the predicted request rate by obtaining the latest collected metric data through Prometheus's APIs. The model will predict the expected workload in the next $t+1$ min and be fed to the planning phase for replica calculation.

Planning Phase: Based on the rated workload of the pod (request rate), the algorithm will calculate the desired pod count needed for the predicted workload from the analysis phase. This will indicate the system to provision or de-provision replicas based on the forecasted workload. Replica calculation utilizes the Adaptation Manager Service algorithm proposed by Dang-Quang and Yoo (2021). Desired pods are calculated as in Equation 7

$$pods_{t+1} = \frac{workload_{t+1}}{workload_{pod}} \tag{7}$$

Execution Phase: Based on the output of the replica calculation, a request to scale commands is sent to Kubernetes API-server. The Kubernetes scheduler executes the Equation 8 or Equation 9 command to achieve the desired replica count for the forecasted period to handle the predicted workload.

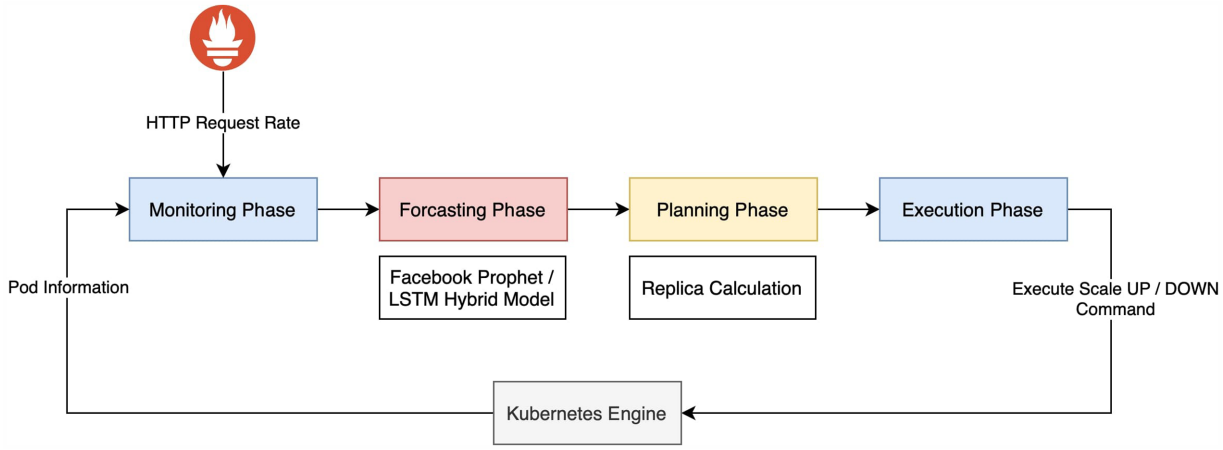

FIGURE 2
MAPE loop.

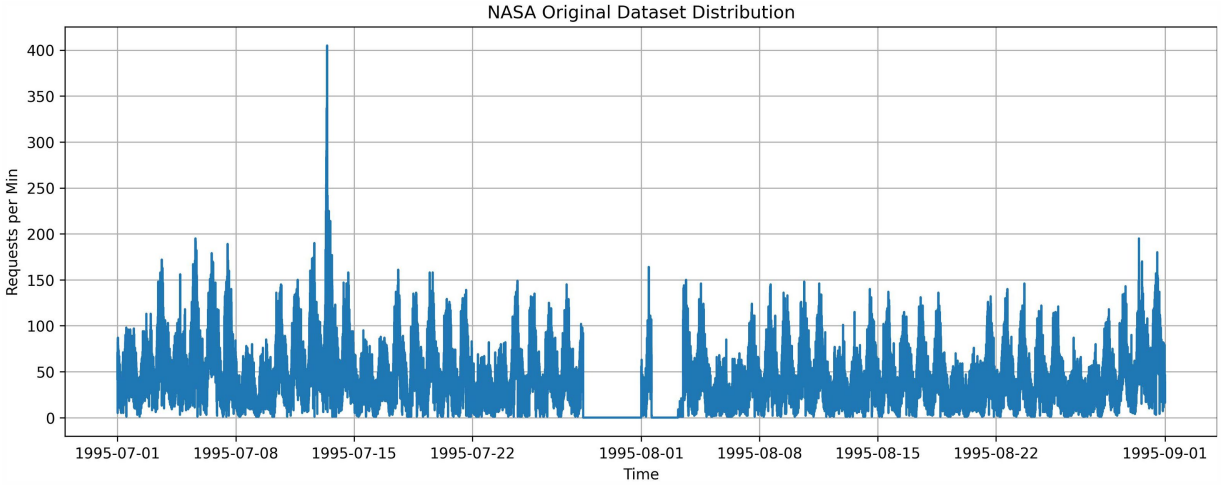

FIGURE 3
Preprocessed NASA dataset.

$$if\ (pods_{t+1} < pods_t) \rightarrow EXECUTE\ \ SCALE\ UP \tag{8}$$

$$if\ (pods_{t+1} < pods_t) \rightarrow EXECUTE\ \ SCALE\ DOWN \tag{9}$$

4 Experiments

This section presents the proposed hybrid model evaluation results and then comparing them with LSTM, Bi-LSTM, and ARIMA models.

4.1 Datasets

4.1.1 NASA dataset

HTTP requests with timestamps collected by NASA Kennedy Space Center over 2 months. The dataset has two subsets ranging from 1 July 1995 to 31 July 1995 and 1 August 1995 to 31 August 1995, containing 3,461,612 requests. The dataset used by Messias et al. (2015), Ye et al. (2017), Aslanpour et al. (2017), and Dang-Quang and Yoo (2021) to evaluate autoscaling. The dataset was preprocessed to aggregate same-minute logs to calculate the HTTP request rate per minute, as shown in Figure 3.

4.1.2 FIFA world cup 1998 dataset

HTTP requests with timestamps collected during FIFA World Cup 1998. The dataset ranges from 30 April 1998 to 26 July 1998, containing 1,352,804,107 requests. The dataset used by Messias et al. (2015), Imdoukh et al. (2019), Roy et al. (2011), and Dang-Quang and Yoo (2021) to evaluate autoscaling. This distribution demonstrates more significant anomalies than the NASA dataset, as shown in Figure 4. The dataset was preprocessed to aggregate logs within the same minute to calculate the HTTP request rate per minute.

4.2 Experimental settings

The proposed hybrid model was evaluated using NASA and FIFA World Cup datasets, which took 70% for training and 30% for evaluation while preserving the time order. The datasets were preprocessed to replace missing data with zero and eliminate duplicate timestamps. The FIFA World Cup dataset was normalized to the 0–1 range to facilitate comparability with other published studies, showcasing the effectiveness of multivariate approaches to data analysis, as highlighted by Ahmed et al. (2024). The hybrid model was then evaluated with Bi-LSTM, LSTM, and ARIMA models. Other hybrid model combinations, such as Prophet with Bi-LSTM and Prophet with GRU, were also assessed. Python programming language and TensorFlow framework were used to implement models.

4.2.1 Model configuration

The model configuration used for training, testing, and evaluation is listed in Table 2. The final parameters were derived following hyperparameter tuning. Other parameters in the model configuration not listed in Table 2 utilized default settings.

A hyperparameter tuning was performed to determine the ideal parameters for both Prophet and LSTM models. The parameters used during the study, together with the tested ranges and values, are presented in Table 3.

4.2.2 Evaluation metrics

The proposed hybrid model was evaluated using mean squared error (MSE) (Equation 10), root mean squared error (RMSE) (Equation 11), mean absolute error (MAE) (Equation 12), the coefficient of determination (R^2) (Equation 13), and total prediction

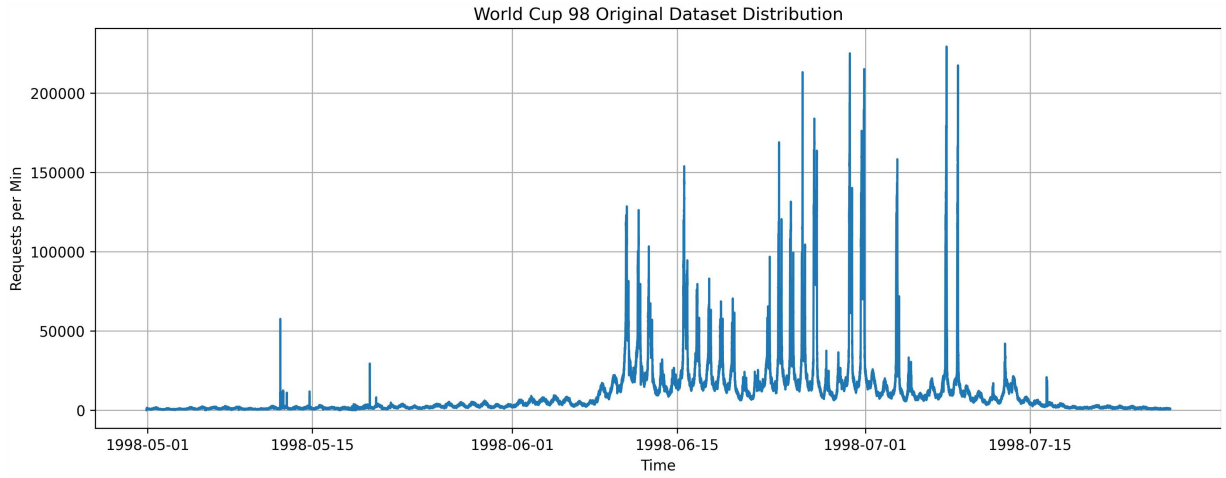

FIGURE 4
Preprocessed FIFA dataset.

TABLE 2 Model configuration.

Model	Parameter	Configured value
Prophet	Growth	Linear
	Changepoint prior scale	5.1
	Yearly seasonality	False
	Weekly seasonality	20
	Weekly seasonality	50
	Seasonality prior scale	30
LSTM	Layers	2 LSTM, 1 dense
	Hidden layers	50
	Loss function	MSE
	Early stopping	5
	Epochs	50
	Batch size	16
	Optimizer	adam
	Learning rate	0.001

GRU and Bi-LSTM models also use the same configuration with GRU and bidirectional layers.

TABLE 3 Hyperparameter tuning study summary.

Model	Parameter	Range/values used for tuning
Prophet	Growth	Linear, flat
	Changepoint prior scale	1–10 (steps of 1) and 5–6 (steps of 0.1)
	Seasonality prior scale	5–50 (steps of 5)
LSTM	Hidden layers	32–512 (steps of 32)
	Optimizer	adam, rmsprop

TABLE 4 Experiment results on NASA dataset.

Model / Metric	Prophet–LSTM hybrid model	Prophet–Bi-LSTM hybrid model	Prophet–GRU hybrid model	ARIMA 1 step	Bi-LSTM 1 step	ARIMA 5 step	Bi-LSTM 5 step
MSE	**63.836**	67.995	75.74	196.288	183.642	237.604	207.313
RMSE	**7.990**	8.246	8.703	14.010	13.551	15.414	14.39
MAE	**6.164**	6.312	6.64	10.572	10.280	11.628	10.592
R^2	**0.900**	0.894	0.882	0.692	0.712	0.628	0.675
TPT (ms)	3,492	4,647	4,124	2,300	**4.3**	2,488	45.1

*Bold values indicate the best results.

time (TPT) (Equation 14). MSE, RMSE, and MAE were used to evaluate the model's prediction error, where the smaller the value, the greater the precision. R^2 shows how well the dataset fits the model—the higher, the better fit. TPT shows the prediction latency of the model where lower TPT means faster results. Evaluation metrics can be expressed as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(a_i - f_i)^2 \tag{10}$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - f_i)^2} \tag{11}$$

$$MAE = \frac{1}{n}\sum_{i-1}^{n}|a_i - f_i| \tag{12}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(a_i - f_i)^2}{\sum_{i=1}^{n}(a_i - \overline{f_i})^2} \tag{13}$$

$$TPT = Prophet\ Prediction\ Time + LSTM\ Prediction\ Time \tag{14}$$

Where a_i represents actual value, f_i represents forecasted value and $\overline{f_i}$ represents the mean of f_i.

4.3 Results

4.3.1 NASA dataset

The proposed hybrid model is compared with ARIMA single and multistep, Bi-LSTM single-step and multistep, hybrid model combinations with Bi-LSTM and GRU. As indicated in Table 4, the proposed model achieves more minor prediction errors on MSE, RMSE, MAE, and R^2. Since the model consists of two models, the prediction latency is more significant than single models where Bi-LSTM single-step records the smallest TPT. All the hybrid models show higher R^2, indicating a better fit for the dataset due to the seasonality capturing since it was not present in single model tests. Finally, the proposed Prophet–LSTM hybrid model overperforms both single and hybrid models in predictions.

Figure 5 shows the trend captured by the Prophet model. The residual input and the prediction from LSTM are shown in Figure 6 which denotes the LSTM model's capability of capturing the residual pattern. The final combined output is shown in Figure 7.

4.3.2 FIFA world cup 1998 dataset

Table 5 shows the experimental results obtained for the FIFA World Cup 1998 dataset. The proposed hybrid model outperforms all the compared models as ARIMA, LSTM, and Bi-LSTM in single-step and multistep modes. The proposed hybrid model shows significant improvement in MSE, RMSE, and MAE, but has higher TPT than all the other models.

Figures 8–10 shows the results obtained for the FIFA World Cup 1998 testing dataset. Considering the pattern of the dataset, trend growth is neglected, and base seasonality detection from the Prophet model is indicated in Figure 8. In the graphs, the difference between the actual and predicted is negligible and can only be seen with the three most significant spikes in the testing dataset.

5 Discussion

Based on the testing conducted, it was observed that hybrid models outperform TSA of HTTP requests compared to ARIMA and Bi-LSTM in both single and multistep. In comparison with hybrid models shown in Table 4, Prophet with LSTM shows 6.1% higher accuracy in prediction with Bi-LSTM and 15.7% with GRU in MSE metric. The Prophet–LSTM model shows 65.2% higher accuracy in

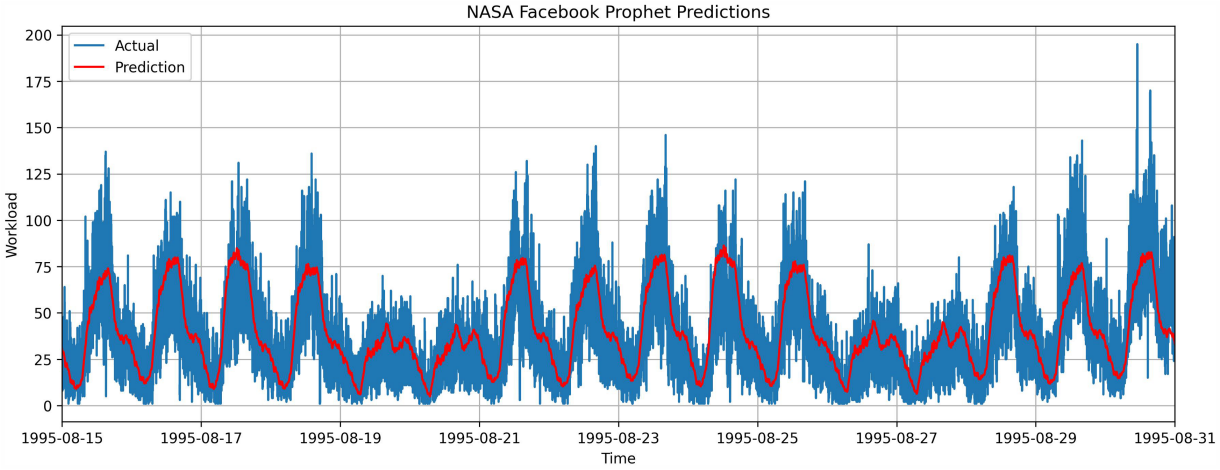

FIGURE 5
Prophet model prediction output vs. original distribution.

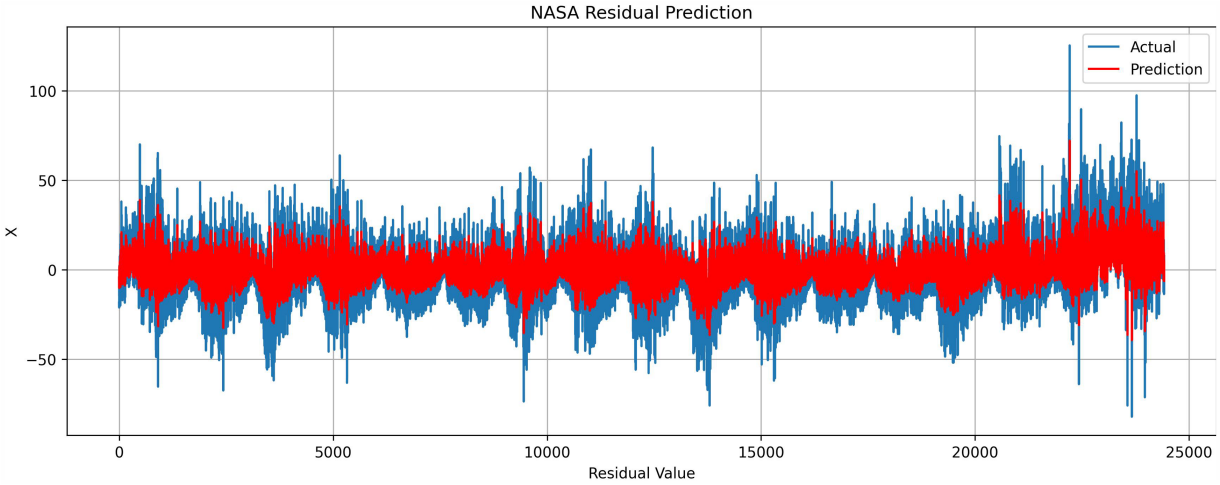

FIGURE 6
Residual prediction from LSTM vs. actual residual distribution.

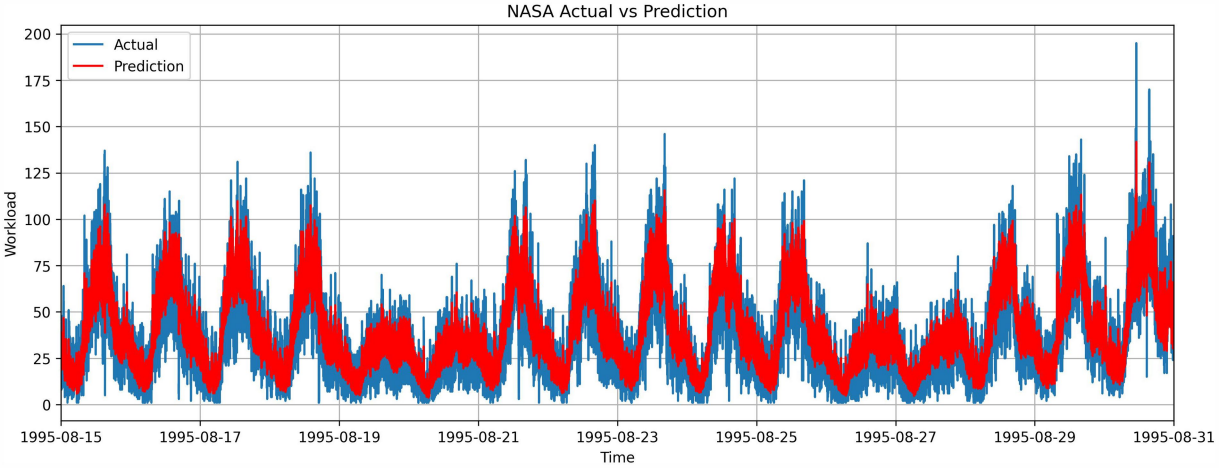

FIGURE 7
Combined prediction results from prophet and LSTM.

TABLE 5 Experiment results on FIFA World Cup 1998 dataset.

Model / Metric	Prophet—LSTM Hybrid Model	ARIMA 1 step	LSTM 1 step	Bi-LSTM 1 step	ARIMA 5 step	Bi-LSTM 5 step
MSE	**0.000011**	0.000040	0.000043	0.000036	0.000172	0.000120
RMSE	**0.003358**	0.006350	0.006523	0.006015	0.0131	0.010
MAE	**0.000675**	0.003302	0.003958	0.003127	0.0049	0.00428
R^2	0.998318	0.998496	0.998397	**0.998637**	0.9930	0.9954
TPT (ms)	3,971	37,00	**5.1**	5.8	4,076	51.2

*Bold values indicate the best results. *Results are normalized.

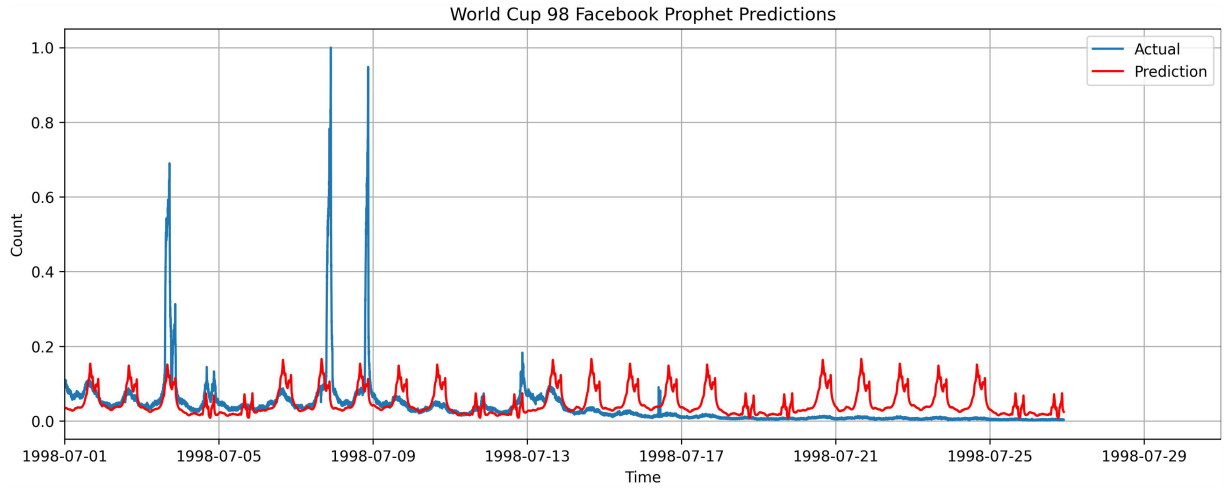

FIGURE 8
Prophet model prediction output vs. original distribution.

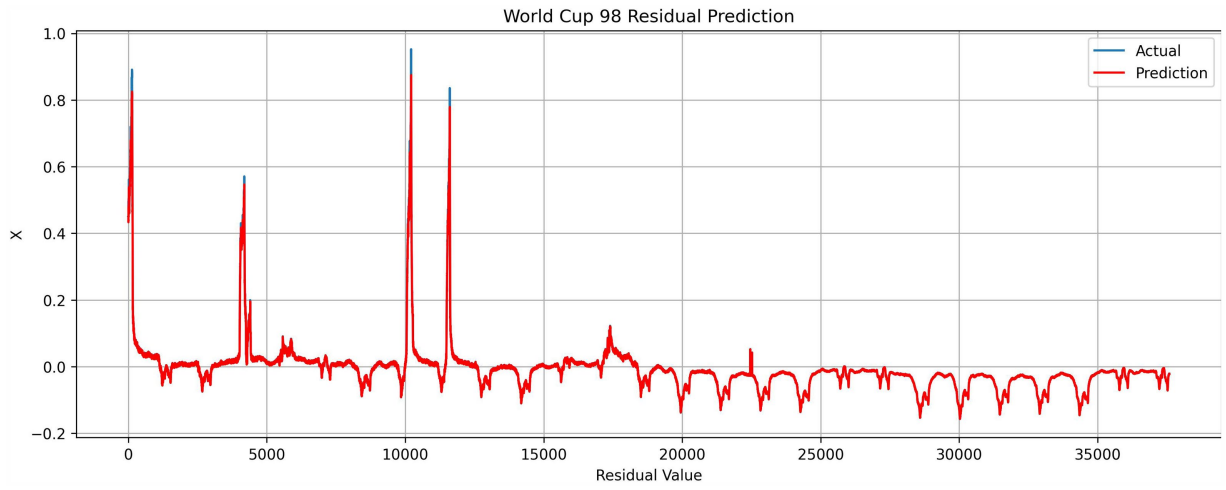

FIGURE 9
Residual prediction from LSTM vs. actual residual distribution.

prediction with MSE metric compared to the best single model Bi-LSTM 1 step.

All the hybrid models showed higher latency compared to single models, as indicated in Tables 4, 5. Since the predictions are made using two separate models, the total time for the final prediction given by Equation 14 combines the sum of Prophet and LSTM prediction times. As presented in section 3.1.4, hybrid models have higher complexity and require more computational demand for the prediction tasks. Using a single model can give computational efficiency with reduced accuracy. Hence, the usage of an appropriate

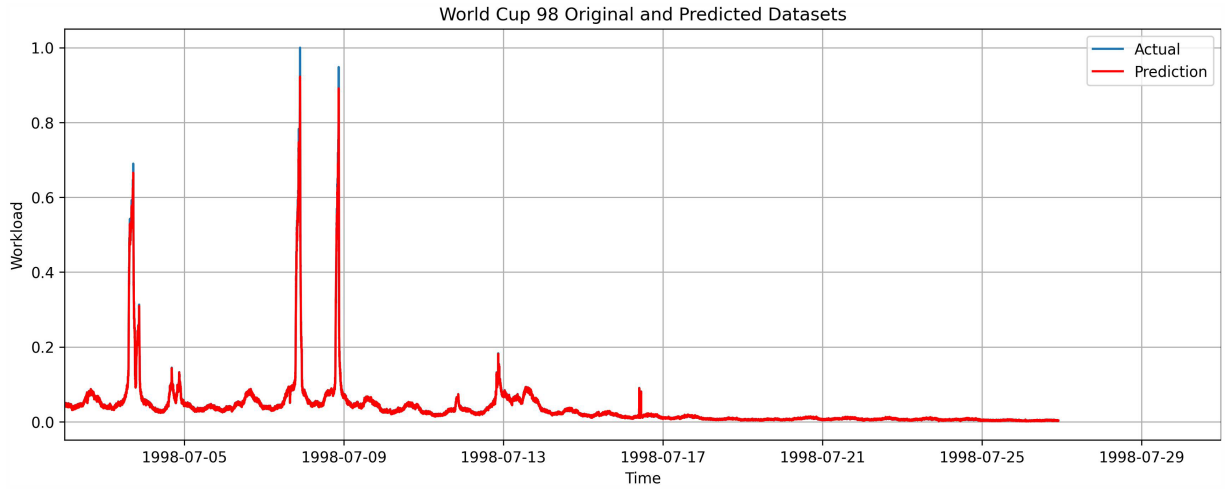

FIGURE 10
Combined prediction results from Prophet and LSTM.

prediction mechanism depends on the requirement of the scaling strategy of the target system.

The exact configuration was used and tested in the World Cup 1998 dataset, which shows the same observations as with the NASA dataset, as indicated in Table 5. Prophet–LSTM model shows 90.8% higher accuracy than the best single model Bi-LSTM 5 steps. Prophet–LSTM hybrid model shows higher latency than the fastest single model, LSTM.

The R^2 in the hybrid model is higher in the NASA dataset with a regular seasonality pattern. This higher R^2 was achieved from the seasonality capturing done by the Prophet model. In the World Cup 1998 dataset, R^2 is 0.032% lower compared to the Bi-LSTM single-step model. This was caused by the dataset's complex seasonality patterns and abnormalities. This shows that hybrid models can fit the model to the dataset better compared to single models, but it can introduce errors if the dataset does not show a clear pattern or has extensive abnormalities.

The lower R^2 occurred due to seasonality capturing done by the Prophet model. Seasonality capturing can introduce significant variations to the prediction when the dataset contains more abnormalities. NASA dataset, which has fewer abnormalities and demonstrates clear seasonality throughout the period, shows the efficient capture of seasonality patterns by Prophet as shown in Figure 5. In contrast, the World Cup 1998 dataset has three large HTTP request spikes and different HTTP request patterns in time ranges 1998 July 01–05, 1998 July 06–10, 1998 July 10–15, and 1998 July 16–27, as indicated in Figure 4. The effect of the abnormalities can be seen in Figure 8, where, starting July 14, the prediction is higher than the actual and predicted data, not capturing spikes in the prediction period. This caused the residual prediction to correct the induced error, as shown in Figure 9 as spikes. In the NASA dataset, this behavior cannot be seen in residual prediction (Figure 6) since the Prophet model captured the dataset's seasonality, which has fewer abnormalities and a consistent seasonality pattern.

The original NASA dataset has a "No Data" period from 1995 July 28 to 1995 July 31 and from 1995 August 01 to 1995 August 03. For the prediction studies conducted, this section was considered as no traffic days and can be highlighted as an anomaly of the workload. These days, less workload or no workload can be commonly seen in production applications based on customer behavior or external considerations. Even though this period can be filled with dummy data to increase the accuracy of seasonality capturing, we have not generated the dummy data to observe the model behavior under abnormalities and increase the comparison accuracy with existing literature.

The results demonstrate that the proposed model achieves higher prediction accuracy than the existing scaling solutions. This showcases its potential as a reliable approach for predictive scaling. Even though Kubernetes integration of the model has been hypothesized, direct application as a proactive autoscaling requires further elaboration. The model utilizes the autoscaling framework described in Section 3.2.2, which provides the necessary architecture for seamless integration with Kubernetes. By expanding upon the methodology suggested by Dang-Quang and Yoo (2021), the model ensures compatibility with Kubernetes' current scaling mechanisms and highlights the advantages of real-time resource management.

Implementing the approach effectively as a proactive autoscaling requires specific measures to be considered. This includes integrating the proposed hybrid model into either Kubernetes HPA or custom autoscaling, such as the KEDA Project, developing necessary APIs, calibrating it to pod forecasting based on the system parameters, and preemptively initiating scaling triggers. This procedure necessitates comprehensive implementation guidelines, including API specifications, input/output mapping between TSA predictions and Kubernetes API calls, and latency and performance monitoring optimization.

The proposed model uses an HTTP request rate for the scaling, considering it provides a more immediate and holistic view of demand fluctuations (Zhang et al., 2009). Still, properly allocating CPU and memory to the target pod is crucial for proper deployment. Although the HTTP request rate may not precisely correspond to the CPU and memory utilization due to the differing computational and memory requirements, engineers should rate the pod for the maximum request rate allowed. This metric is crucial in proactive autoscaling to calculate the necessary pod counts in scaling up or down scenarios. Adding a fixed CPU and memory allocation to the pod and the maximum replicas in a scaling environment

is essential and recommended to appropriately upscale to underline infrastructure automatically to the upcoming resource demand. Proactive autoscaling decisions can be executed without the Kubernetes server running into resource limitations by ensuring proper CPU and memory settings and maximum replicas.

6 Conclusion

Cloud computing is becoming increasingly popular among large-scale cloud applications and Software as a Service (SaaS) applications, considering its flexibility on elasticity. Kubernetes is one of the widely employed deployment strategies that supports elasticity and adjusts the computing power based on the dynamic workload. Autoscaling is a key feature of Kubernetes, which gives resource elasticity to provision and de-provision resources automatically. This mechanism helps the deployers to maintain high service availability while reducing the cloud cost. This study proposes a proactive autoscaling system to Kubernetes based on Prophet and LSTM-based hybrid models. The proposed autoscaling is based on the MAPE loop to determine scaling decisions. The proposed hybrid model was trained and evaluated using NASA and World Cup 1998 datasets. The results of the experiments demonstrated the efficiency of the proposed model compared to single-model proactive autoscaling. Compared to single models, the hybrid model has a higher prediction time since the data is analyzed through two models.

The hybrid model for proactive autoscaling is far from trivial. As for future works, prediction time should be optimized. Implementing the MAPE loop in a more user-friendly way using Kubernetes agents or integrating it with the KEDA Project is a practical side of the exploration of this study. In this study horizontal scaling is focused on adding or removing pods. Still, proactive scaling can be explored vertically (adding or eliminating CPU/memory to the container) or integrated alongside horizontal scaling.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary material; further inquiries can be directed to the corresponding author.

Author contributions

PG: Methodology, Software, Writing – original draft. YP: Writing – review & editing, Supervision.

Funding

The author(s) declare that no financial support was received for the research, authorship, and/or publication of this article.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Generative AI was used in the creation of this manuscript.

Publisher's note

Supplementary material

The Supplementary material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fcomp.2025.1509165/full#supplementary-material

References

Abumohsen, M., Owda, A., Owda, M., and Abumihsan, A. (2024). Hybrid machine learning model combining of CNN-LSTM-RF for time series forecasting of solar power generation. *e-Prime* 9:100636:100636.

Ahmed, R. R., Streimikiene, D., Streimikis, J., and Siksnelyte-Butkiene, I. (2024). A comparative analysis of multivariate approaches for data analysis in management sciences. *E+M Ekon. Manag.* 27, 192–210. doi: 10.15240/tul/001/2024-5-001

Ahmed, R. R., Streimikiene, D., Ghauri, S. P., and Aqil, M. (2021). Forecasting inflation by using the sub-groups of both CPI and WPI: evidence from auto regression (AR) and ARIMA models. *Roman. J. Econ. Forecast.* 2, 144–161. doi: 10.1016/j.prime.2024.100636

al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2018). Elasticity in cloud computing: state of the art and research challenges. *IEEE Trans. Serv. Comput.* 11, 430–447. doi: 10.1109/TSC.2017.2711009

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2017). "Autonomic vertical elasticity of Docker containers with ELASTICDOCKER." in 2017 IEEE 10th International Conference on CLOUD computing (CLOUD). [Preprint].

Annapoorna, E., Sujil, S. V., Sreepriya, S., Abhishek, S., and Anjali, T. (2024). "Revolutionizing stock Price prediction with automated Facebook prophet analysis." 2022 International Conference on Inventive Computation Technologies (ICICT) [Preprint].

Arlitt, M., and Jim, T. (1998). 1998 World Cup Web Site Access Logs. Available at: http://www.acm.org/sigcomm/ITA/ (Accessed September 1, 2024).

Ashraf, Q. M., Tahir, M., Habaebi, M. H., and Isoaho, J. (2023). Toward autonomic internet of things: recent advances, evaluation criteria, and future research directions. *IEEE Internet Things J.* 10, 14725–14748. doi: 10.1109/JIOT.2023.3285359

Aslanpour, M. S., Ghobaei-Arani, M., and Toosi, A. N. (2017). Auto-scaling web applications in clouds: a cost-aware approach. *J. Netw. Comput. Appl.* 95, 26–41. doi: 10.1016/j.jnca.2017.07.012

Aytaç, E. (2021). Forecasting Turkey's hazelnut export quantities with Facebook's prophet algorithm and box-cox transformation. *Adv. Distribut. Comp. Artif. Intellig. J.* 10, 33–47. doi: 10.14201/adcaij20211013347

Bartelucci, N., and Bellavista, P. (2023). "A practical guide to autoscaling solutions for next generation internet applications." 2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom) [Preprint].

Bekkar, A., Hssina, B., Douzi, S., and Douzi, K. (2024). Forecasting ozone levels in Morocco. A Comparative Study of SARIMA and FB Prophet Models. In: *Adv. Environ.*

Eng. Green Technol. Book Ser. eds. J. Mabrouki & M. Azrour. IGI Global Scientific Publishing. 9–28. doi: 10.4018/979-8-3693-3807-0.ch002

Borkowski, M., Schulte, S., and Hochreiner, C. (2016). Predicting cloud resource utilization, 9th international conference on utility and cloud computing (UCC '16). New York, United States of America: Association for Computing Machinery, 37–42.

Calheiros, R. N., Masoumi, E., Ranjan, R., and Buyya, R. (2015). Workload prediction using ARIMA model and its impact on cloud applications' QOS. *IEEE Trans. Cloud Comp.* 3, 449–458. doi: 10.1109/TCC.2014.2350475

Chen, L., Xian, M., and Liu, J. (2020). "Monitoring system of OpenStack cloud platform based on Prometheus." in 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL) [Preprint].

Cheng, J., Tiwari, S., Khaled, D., Mahendru, M., and Shahzad, U. (2023). Forecasting bitcoin prices using artificial intelligence: combination of ML, SARIMA, and Facebook prophet models. *Technol. Forecast. Soc. Chang.* 198:122938. doi: 10.1016/j.techfore.2023.122938

Ciptaningtyas, H.T., Santoso, B.J., and Razi, M.F. (2017). "Resource elasticity controller for docker-based web applications." 2017 11th international conference on information communication technology and system (ICTS). Surabaya, Indonesia, pp. 193–196.

Dang-Quang, N.-M., and Yoo, M. (2021). Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. *Appl. Sci.* 11:3835. doi: 10.3390/app11093835

Fang, W., Lu, Z., Wu, J., and Cao, Z. (2012). "RPPS: A novel resource prediction and provisioning scheme in cloud data center." in 2012 IEEE ninth international conference on services computing. Honolulu, United States of America. pp. 609–616.

Garí, Y., Monge, D. A., Pacini, E., Mateos, C., and García Garino, C. (2021). Reinforcement learning-based application autoscaling in the cloud: a survey. *Eng. Appl. Artif. Intell.* 102:104288. doi: 10.1016/j.engappai.2021.104288

Hardikar, S., Ahirwar, P., and Rajan, S. (2021). "Containerization: cloud computing based inspiration Technology for Adoption through Docker and Kubernetes." 2021 second international conference on electronics and sustainable communication systems (ICESC) [Preprint].

Hochreiter, S., and Schmidhuber, J. (1996). LSTM can solve hard long time lag problems. *Neural Inform. Proc. Syst.* 9, 473–479.

Ibryam, B., and Huß, R. (2023). Kubernetes patterns: Reusable elements for designing cloud-native applications. California, USA: O'Reilly Media.

Imdoukh, M., Ahmad, I., and Alfailakawi, M. G. (2019). Machine learning-based auto-scaling for containerized applications. *Neural Comput. & Applic.* 32, 9745–9760. doi: 10.1007/s00521-019-04507-z

Kothari, A., Kulkarni, A., Kohade, T., and Pawar, C. (2024). Stock market prediction using LSTM. *Lect. Notes Networks Syst.* 143–164. doi: 10.1007/978-981-97-1326-4_13

Kubernetes Documentation (2024). Available at: https://kubernetes.io/docs/home/ (Accessed August 28, 2024).

Marie-Magdelaine, N., and Ahmed, T. (2020). "Proactive autoscaling for cloud-native applications using machine learning." GLOBECOM 2020–2020 IEEE Global Communications Conference [Preprint].

Megino, F. H. B., Albert, J. R., Berghaus, F., De, K., Lin, F., MacDonell, D., et al. (2020). Using Kubernetes as an ATLAS computing site. *EPJ Web Conf.* 245:07025. doi: 10.1051/epjconf/202024507025

Messias, V. R., Estrella, J. C., Ehlers, R., Santana, M. J., Santana, R. C., and Reiff-Marganiec, S. (2015). Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure. *Neural Comput. & Applic.* 27, 2383–2406. doi: 10.1007/s00521-015-2133-3

NASA-HTTP (1995). Available at: https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html (Accessed September 1, 2024).

Pahl, C., and Lee, B. (2015). "Containers and clusters for edge cloud architectures -- a technology review." in 2015 3rd International Conference on Future Internet of Things and Cloud [Preprint].

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. arXiv (Cornell University) [preprint].

Podolskiy, V., Jindal, A., Gerndt, M., and Oleynik, Y. (2018). Forecasting models for self-adaptive cloud applications: A comparative study: in 2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems [Preprint].

Prachitmutita, I., Aittinonmongkol, W., Pojjanasuksakul, N., Supattatham, M., and Padungweang, P. (2018). "Auto-scaling microservices on IAAS under SLA with cost-effective framework." 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI) [Preprint].

Roy, N., Dubey, A., and Gokhale, A. (2011). "Efficient autoscaling in the cloud using predictive models for workload forecasting." in 2011 IEEE 4th International Conference on Cloud Computing [Preprint].

Shah, J., and Dubaria, D. (2019). "Building modern clouds: Using Docker, Kubernetes & Google Cloud Platform." in 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC) [Preprint].

Tang, X., Liu, Q., Dong, Y., Han, J., and Zhang, Z. (2018). "Fisher: an efficient container load prediction model with deep neural network in clouds." in 2018 IEEE Intl Conf on parallel Distributed processing with applications, ubiquitous Comput- ing communications, big data cloud computing, social computing networking, sustainable computing communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). Melbourne, Australia, pp. 199–206.

Taylor, S. J., and Letham, B. (2018). Forecasting at scale. *Am. Stat.* 72, 37–45. doi: 10.1080/00031305.2017.1380080

Toka, L., Dobreff, G., Fodor, B., and Sonkoly, B.. (2020). "Adaptive AI-based auto-scaling for Kubernetes." in 2020 20th IEEE/ACM international symposium on cluster, Cloud and Internet Computing (CCGRID). pp. 599–608.

Varghese, B., and Buyya, R. (2018). Next generation cloud computing: new trends and research directions. *Futur. Gener. Comput. Syst.* 79, 849–861. doi: 10.1016/j.future.2017.09.020

Wang, Z., and Gu, X. (2023). "A time series prediction algorithm based on BiLSTM and prophet hybrid model." in 2023 4th International Conference on Computer Engineering and Application (ICCEA). pp. 128–132.

Yan, M., Liang, X. M., Lu, Z. H., Wu, J., and Zhang, W. (2021). HANSEL: adaptive horizontal scaling of microservices using bi-LSTM. *Appl. Soft Comput.* 105:107216. doi: 10.1016/j.asoc.2021.107216

Yang, D., Li, M., Guo, J. E., and du, P. (2024). An attention-based multi-input LSTM with sliding window-based two-stage decomposition for wind speed forecasting. *Appl. Energy* 375:124057. doi: 10.1016/j.apenergy.2024.124057

Ye, T., Guangtao, X., Shiyou, Q., and Minglu, L. (2017). "An auto-scaling framework for containerized elastic applications." in N Proceedings of the 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM), pp. 422–430.

Zhang, H., Jiang, G., Yoshihira, K., Chen, H., and Saxena, A. (2009). "Intelligent workload factoring for a hybrid cloud computing model." in 2009 Congress on Services - I[Preprint].