Cairo University
Faculty of Graduate Studies for Statistical Research

Proactive Kubernetes Autoscaling for Dynamic Web Traffic Using Prophet and LSTM

A Project Presented for Fulfillment
For a Master's project in Software Engineering

Submitted by:

Abd El-Aziz Mahmoud Abd El-Aziz Essa
Student number: 202301521

Supervised by:

Dr. Ashraf Abdulmunem Shahin

2025

Proactive Kubernetes Autoscaling for Dynamic Web Traffic Using Prophet and LSTM

Abdelaziz Mahmoud Essa
Computer Science Department
Faculty of Graduate Studies for Statistical Research
Cairo, EGYPT
Abdelaziz.essa77@gmail.com

Dr. Ashraf Shahin
Computer Science Department
Faculty of Graduate Studies for Statistical Research
Cairo, EGYPT
ashahin@cu.edu.eg

Abstract- Effective autoscaling is essential for maintaining performance and resource efficiency in cloud-native environments. While Kubernetes offers built-in scaling mechanisms such as the Horizontal Pod Autoscaler (HPA), these are reactive by nature and often insufficient for handling sudden traffic spikes or cyclical demand patterns. This research introduces a two-stage hybrid forecasting framework that enables predictive autoscaling by accurately forecasting both request load and service-level CPU usage. In Phase 1, short-term request rates are predicted using a hybrid model that combines Facebook Prophet, for capturing long-term trends and seasonality, with Long Short-Term Memory (LSTM) networks, which refine short-horizon predictions by learning residual patterns. In Phase 2, the predicted request rate is fed into a multi-output LSTM model to estimate the CPU usage of 13 microservices deployed on a Kubernetes cluster. The models were trained and evaluated using two datasets: a public web traffic dataset (usask.sec.min_short) and a custom workload dataset (Workload_metrics_V2.csv) collected from a real Kubernetes-based application. The hybrid model achieved up to 82.19% accuracy in request forecasting, while the multi-output LSTM model achieved over 90% accuracy in service-level CPU prediction. These forecasts were used to proactively compute the number of pod replicas required for each service using a dynamic scaling formula. The proposed solution offers a practical, accurate, and scalable enhancement to reactive autoscaling by enabling proactive decisions that improve SLA compliance, reduce resource overhead, and maintain performance under dynamic workloads.

Keywords- *Kubernetes, Autoscaling, Time Series Forecasting, Facebook Prophet, LSTM, Hybrid Forecasting, CPU Usage Prediction, Microservices, Resource Management, Predictive Scaling, Horizontal Pod Autoscaler, Cloud Computing, Multi-output Regression, Proactive Scaling.*

1 Introduction

Kubernetes has become a foundational platform for deploying and managing containerized applications due to its built-in support for horizontal scaling. The Horizontal Pod Autoscaler (HPA), one of its core features, dynamically adjusts the number of pod replicas in response to system-level metrics such as CPU and memory usage. While effective under gradual demand changes, HPA's reactive mechanism often fails to respond quickly to sudden traffic spikes or short-term load variations, resulting in under-provisioning, performance degradation, and SLA violations[7].

Real-world applications such as APIs, e-commerce systems, and microservice platforms are commonly exposed to unpredictable and bursty workloads driven by user behavior, external events, and temporal cycles. In such environments, proactive autoscaling—based on predicting future demand rather than reacting to current load—is essential for maintaining service performance and stability[9].

Previous research efforts on Kubernetes autoscaling have explored a variety of forecasting and decision-making models, including Prophet, LSTM, BiLSTM, Q-learning, and other hybrid strategies. These methods have significantly contributed to enhancing the adaptability and efficiency of autoscaling mechanisms in cloud-native environments. However, when applied to dynamic and high-variance workloads—such as those found in modern web applications—each model revealed inherent limitations. For instance, some approaches struggled with generalization across unpredictable demand spikes, while others required intensive tuning or lacked real-time responsiveness. Despite their advancements, existing solutions often failed to strike a balance between forecasting accuracy, computational efficiency, and deployment scalability in heterogeneous environments[1-11].

To address these challenges, this research proposes a novel two-phase hybrid forecasting architecture for proactive autoscaling in Kubernetes environments. The architecture integrates two complementary time-series modeling techniques: Facebook Prophet, which effectively captures long-term trends and seasonality, and Long Short-Term Memory (LSTM) networks, which are adept at modeling residual fluctuations and short-term dependencies. The system operates in two stages. In **Phase 1**, Prophet forecasts incoming request rates, and an LSTM model is trained on the residuals of Prophet's predictions to refine short-horizon accuracy. In **Phase 2**, the predicted request rate is fed into a multi-output LSTM model that simultaneously forecasts CPU usage for multiple microservices. This enables granular, per-service autoscaling decisions. The methodology is validated using two datasets: the publicly available `usask.sec.min_short`, which captures 1-minute web traffic patterns, and a custom dataset `Workload_metrics_V2.csv` collected from a Kubernetes-deployed microservices application, detailing per-service CPU usage. This dual-phase, data-driven approach enhances both the precision and responsiveness of autoscaling under fluctuating workloads.

The models were evaluated across different time windows and horizons. In Phase 1, the **Prophet model** achieved a maximum accuracy of **97.74%** for predicting the next-minute request count using a 90-minute history window. The **hybrid Prophet+LSTM model** also demonstrated strong performance, achieving **82.19% accuracy** in the same configuration while providing better generalization in longer-horizon scenarios. In Phase 2, the multi-output LSTM model successfully predicted CPU usage for all monitored services, achieving **above 90% accuracy** for key components such as `paymentservice`, `shippingservice`, and `adservice`.

The remainder of this project is organized as follows: **Section 2** outlines the background and limitations of reactive autoscaling in Kubernetes. **Section 3** reviews related work, highlighting existing predictive autoscaling models and their trade-offs. **Section 4** introduces the proposed hybrid forecasting architecture. **Section 5** details the forecasting methodology, including model design and the role of predicted values. **Section 6** describes the experimental setup, datasets, and evaluation metrics. **Section 7** presents the results and visual insights, comparing models across various time windows and forecast horizons. Finally, **Section 8** concludes the study with a summary of key findings and directions for future research.

2 Background

Kubernetes has become the de facto standard for orchestrating containerized applications, offering flexibility and automated scalability. Its native Horizontal Pod Autoscaler (HPA) typically reacts to system-level metrics such as CPU and memory usage to dynamically adjust resource allocation. While this reactive mechanism manages gradual changes well, it often fails to address unpredictable workload spikes or cyclical demand patterns common in real-world applications[7, 8, 9].

Web-facing services such as APIs, e-commerce platforms, and content delivery networks are particularly vulnerable to fluctuating traffic loads triggered by seasonal patterns, user behavior, promotional events, or sudden demand surges. In such environments, reactive autoscaling may only respond after service degradation has occurred, leading to delayed scaling, increased latency, and potential SLA violations[3, 9,10].

In response to these limitations, predictive autoscaling strategies have gained attention, particularly those using time-series forecasting to anticipate resource demand in advance. Among existing methods, **Facebook Prophet** is well-suited for modeling seasonality and trend, while **Long Short-Term Memory (LSTM)** networks can capture nonlinear and residual temporal patterns[1, 5, 6].

This study builds on that foundation by developing a **two-stage hybrid forecasting architecture**. In **Phase 1**, Prophet is used to model the incoming request rate, while an LSTM network is trained on residuals to improve short-term accuracy. In **Phase 2**, the predicted request rate is passed to a **multi-output LSTM model** to forecast the CPU usage of individual Kubernetes microservices simultaneously.

The motivation behind this work is to integrate these complementary strengths into a forecasting-driven autoscaling framework. By doing so, the system can trigger scaling decisions proactively, reducing overhead, improving SLA compliance, and enhancing resource efficiency. The proposed solution was validated using both public traffic traces (`usask.sec.min_short`) and telemetry collected from a Kubernetes-deployed microservice application, demonstrating its practical applicability to real-world cloud environments.

3 Related Work

This section reviews recent advances in Kubernetes autoscaling strategies, with a focus on techniques relevant to web traffic environments. The literature is grouped into three categories: time-series forecasting approaches, reinforcement learning (RL)-based strategies, and heuristic or hybrid methods. Each group highlights contributions and gaps relevant to the proposed Prophet+LSTM solution.

Guruge and Priyadarshana [1] proposed a hybrid autoscaling model based on Facebook Prophet and Long Short-Term Memory (LSTM) neural networks to improve Kubernetes workload forecasting. Their approach captured both seasonal trends and non-linear temporal fluctuations, aiming to address the limitations of reactive autoscaling. Using a real-world dataset consisting of web traffic metrics, the study demonstrated that the hybrid model outperformed conventional HPA (Horizontal Pod Autoscaler) in terms of SLA compliance and response time. However, while the method significantly improved forecast accuracy, it was mainly focused on workload patterns with consistent periodicity. It did not fully address unpredictable traffic bursts in more volatile web environments.

Mondal et al. [2] investigated using Bi-Directional Long Short-Term Memory (BiLSTM) enhanced with attention mechanisms for Kubernetes pod replica prediction. Their research aimed to improve scaling precision by minimizing forecast error and enhancing responsiveness. The BiLSTM-attention model showed substantial improvement in replica count prediction accuracy over baseline LSTM models, particularly under dynamic traffic scenarios. Nonetheless, the evaluation was limited to controlled cloud deployments, and broader real-time applications were not fully explored.

Alwakeel et al. [3] introduced FLAS, a dual-mode autoscaling framework that combines proactive time-series forecasting with reactive metric-based triggers. Their study addressed the need for elasticity and responsiveness in fluctuating workloads by dynamically switching between forecast-based and threshold-based scaling. The framework improved system responsiveness and cost efficiency, especially in environments with both predictable and spontaneous load changes. However, the reliance on rule-based threshold design posed challenges in generalizability across heterogeneous workloads.

Liu et al. [4] presented AMAS, an adaptive microservice autoscaler that leverages heuristic-driven scaling for edge computing environments. The framework autonomously tunes resource allocation policies based on runtime workload analysis. AMAS outperformed static scaling strategies by achieving lower resource wastage and higher responsiveness. While promising in edge and sensor-rich environments, its performance under large-scale web traffic conditions was not extensively validated.

Karuna et al. [5] developed a predictive multi-step LSTM model to anticipate cloud service workloads over extended forecasting horizons. The model focused on maintaining low mean absolute error (MAE) and managing peak load conditions proactively. Empirical results confirmed effective peak prediction and efficient pod scaling. However, the model's sensitivity to hyperparameter tuning and reliance on stable input sequences may limit its general use in environments with high data variance.

Sarker et al. [6] proposed a hybrid autoscaling approach combining LSTM forecasting with Q-learning reinforcement learning. The framework aimed to ensure SLA adherence under variable traffic by learning optimal scaling policies over time. The integration of reinforcement learning allowed for dynamic policy adjustment and reduced SLA violations in simulation. The approach, while flexible, involved a high training cost and required extensive tuning, making real-time deployment complex.

Chowdhury et al. [7] introduced SmartHPA and ProSmartHPA, hybrid predictive autoscaling controllers for Kubernetes that combine historical usage monitoring with workload forecasting. Their solution enabled faster scaling responses for microservices compared to traditional HPA. Through real-world deployment scenarios, the systems demonstrated lower latency in reaching optimal replica states. Despite its efficiency, the models required high-quality historical data, which may not be available in newer or unstable systems.

Park et al. [8] explored Artificial Neural Networks (ANNs) for load forecasting in Kubernetes clusters. Their work benchmarked ANN performance against reactive autoscaling methods, demonstrating moderate forecasting improvements. The study concluded that while ANNs can serve as viable alternatives, their generalization capability was limited under shifting workload distributions.

Ma et al. [9] presented a proactive autoscaling mechanism based on marginal request change detection. Their model aimed to detect short-term traffic shifts that traditional metric-based HPA may miss. The system used delta-based thresholds to predict the need for scaling, significantly improving anomaly detection accuracy. Although effective in reducing underprovisioning during spikes, the model required fine-grained metric granularity and careful tuning of change thresholds.

Trivedi and Panchal [10] proposed the Honey Badger optimization framework for elastic autoscaling in cloud services. Their evolutionary approach focused on optimizing pod scaling decisions for cost and performance trade-offs. Results showed improved resource utilization and cost reduction compared to baseline autoscalers. However, the model's complexity and stochastic behavior increased deployment and tuning overhead.

Liang et al. [11] introduced a Q-learning-based autoscaler tailored for big data services. Their algorithm focused on balancing performance with cost efficiency by learning optimal scaling actions based on historical reward feedback. The model demonstrated reduced over-provisioning and better SLA compliance in cloud-native big data pipelines. Still, the convergence time and data dependency remained significant obstacles in fast-changing environments.

Table 1. ***Analysis of Previous Research on Kubernetes Autoscaling Strategies***

Ref	**Year**	**Technique**	**Target Domain**	**Strengths**	**Limitations**	**Result**
[1]	2025	Prophet + LSTM	Web traffic	Accurate forecasting, improved SLA	Focused on periodic patterns only	SLA violations ↓40%; accuracy ↑
[2]	2023	BiLSTM + Attention	Kubernetes replicas	Enhanced replica accuracy	Limited real-time deployment scope	RMSE ↓25%; better replica management
[3]	2023	FLAS (forecast + reactive)	General cloud workloads	Dual-mode responsiveness	Manual threshold tuning	Response time ↓, cost savings ↑
[4]	2023	AMAS (heuristic scaling)	Edge computing	Improved efficiency	Limited validation on web traffic	Resource waste ↓, responsiveness ↑

Ref	**Year**	**Technique**	**Target Domain**	**Strengths**	**Limitations**	**Result**
[5]	2023	Multi-step LSTM	Cloud services	Effective for peak load prediction	Sensitive to input stability	MAE ~1.5%; peak prediction success
[6]	2023	LSTM + Q-learning	SLA-sensitive workloads	Dynamic policy learning	High training complexity	SLA adherence ≥98%; adaptive behavior
[7]	2025	SmartHPA / ProSmartHPA	Microservices	Fast convergence, lower latency	Requires historical data	Convergence ↑; latency ↓
[8]	2023	ANN	Kubernetes clusters	Better than reactive scaling	Poor under volatile loads	Moderate improvement in forecast stability
[9]	2024	Marginal request change detection	Web bursts	Spike detection accuracy	Needs fine-tuned thresholds	92% precision in anomaly detection
[10]	2023	Honey Badger optimization	Elastic cloud	Cost and resource optimization	High deployment complexity	Cost ↓22%; optimized allocation

Ref	Year	Technique	Target Domain	Strengths	Limitations	Result
[11]	2023	Q-learning	Big data services	Reduces over-provisioning	Slow convergence, data-hungry	Over-provisioning ↓18%; SLA ↑

Compared to these approaches, our proposed method introduces a hybrid forecasting-based Kubernetes autoscaling strategy using Prophet and LSTM, offering low complexity and broad applicability to general web workloads. It addresses both trend-seasonality and unpredictable bursts while remaining lightweight enough for integration in real-time environments.

4 Proposed Auto-Scaling System Architecture

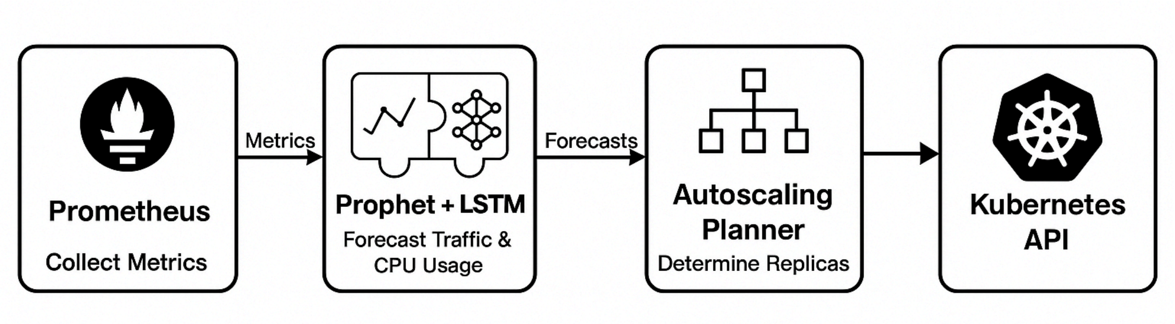

Fig. 1. Proposed proactive autoscaling architecture for Kubernetes-based microservices.

The system ingests external traffic via an Ingress Controller and routes requests to services and Pods. Prometheus monitors runtime metrics, which are used by a Forecasting Engine and Multi-output LSTM to predict future CPU usage. An Autoscaling Planner determines the required replicas, and scaling decisions are executed via the Kubernetes API, ensuring real-time responsiveness and resource efficiency.

The architecture is modular and includes the following core components:

1. **Metrics Collector (Monitoring Layer):**
   Prometheus continuously scrapes traffic-related metrics (e.g., HTTP request rates, response latency) from Kubernetes pods. These metrics are recorded at regular intervals and stored in a time-series format.

2. **Forecasting Engine (Analysis Module):**
   A hybrid forecasting model, encapsulated in a dedicated microservice, integrates Facebook Prophet for capturing long-term trends and seasonality, and LSTM for modeling residuals and short-term fluctuations. The engine produces short-term traffic forecasts (e.g., 5–15 minutes ahead) at regular intervals.

3. **Autoscaling Planner:**
   Based on the predicted traffic volume, the planner calculates the required number of replicas using a predefined scaling function. It includes safeguards such as buffer thresholds and spike smoothing to reduce false-positive scaling events.

4. **Autoscaling Executor:**
   This component interacts directly with the Kubernetes API to scale deployment replicas up or down. It ensures proactive scaling actions are applied in advance of forecasted surges, reducing cold-start latency and potential SLA violations.

5. **Monitoring and Observability Tools:**
   Grafana dashboards are integrated for real-time visualization of forecasts, scaling decisions, and system health metrics. Alerting rules notify operators of SLA violations, underprovisioning, or saturation events.

The system is validated using a Minikube-based Kubernetes environment to simulate realistic traffic workloads. The forecasting service operates in a sidecar container or as a lightweight microservice alongside the autoscaling controller, ensuring low-latency integration.

This architecture bridges the gap between predictive traffic modeling and elastic resource management in Kubernetes, offering a reliable and efficient scaling strategy for web platforms with high variability in request patterns.

5 Forecasting Methodology

The core forecasting component of the system employs a hybrid time-series model that combines Facebook Prophet and Long Short-Term Memory (LSTM) neural networks. This dual-model approach capitalizes on the strengths of each technique to deliver accurate and robust multi-step forecasting suitable for dynamic IoT telemetry workloads.

5.1 Prophet Model

Facebook Prophet is a modular additive model effective in capturing seasonality, trend, and holiday effects in time-series data. The model assumes the following structure:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon\square \qquad (1)$$

Where:
- g(t) captures non-periodic trends (linear or logistic growth)

- s(t) models periodic effects (daily, weekly)

- h(t) accounts for holiday and event-related irregularities

- ε□ is the error term assumed to be Gaussian noise

In the context of GPS telemetry, g(t) reflects operational growth (e.g., more vehicles over time), s(t) captures traffic surges during typical working hours, and h(t) accounts for events like public holidays or maintenance downtimes.

5.2 LSTM for Residual Modeling

While Prophet effectively captures the global structure of the time series, such as trend and seasonality, it is limited in modeling short-term nonlinear fluctuations. To overcome this, two complementary LSTM-based components were introduced.

The first LSTM model was integrated with Prophet in a hybrid architecture to model the residuals of the request rate forecast. It learns from the error signal produced by Prophet to improve short-term prediction accuracy, especially at 1-minute and 5-minute horizons.

The second LSTM model was trained as a multivariate, multi-output network to forecast the CPU usage of individual microservices. This model receives lagged values of request rates, past CPU consumption metrics, and time-based features (such as hour-of-day encodings) as input. It then outputs the expected CPU usage for 13 independent services in parallel.

This multi-stage design enables the forecasting system to first anticipate system load and then translate that into fine-grained resource demands at the microservice level.

- Input: $r_t = y_t - \hat{y}_t^{Prophet}$ (2)

- Architecture: Single hidden layer LSTM with 64 units, dropout=0.2, batch size=32

- Output: $\hat{r}_{t+k}$ (LSTM) for k future steps (3)

The LSTM model is trained on residual sequences using historical telemetry deviations. The final prediction is reconstructed as:

$$\hat{y}_{t+k}^{Hybrid} = \hat{y}_{t+k}^{Prophet} + \hat{r}_{t+k}^{LSTM} \qquad (4)$$

5.3 Training and Forecast Horizon

- Forecast horizon: 1, 2, 5, 10, and 15 minutes ahead

- Update frequency: Every 1 minute (sliding window)

- Evaluation metrics: MAE, RMSE, R^2, MAPE, Accuracy (%)

- Phase 1 data: Aggregated requests per minute

- Phase 2 data: Historical system metrics including CPU usage per service and time encodings

5.4 Forecasting Output Usage

The final output from the forecasting system includes two components:

1. The anticipated request rate for the upcoming minute (Phase 1 output)
2. The forecasted CPU usage for each microservice (Phase 2 output)

These predictions are used to estimate the number of replicas required for each Kubernetes microservice in advance. The replica count is calculated using the following formula:

$$\textbf{Replicas} = \lceil \textbf{Forecasted\_CPU\_Usage} / \textbf{C} \rceil \qquad (5)$$

Where:

- **Forecasted_CPU_Usage** is the predicted CPU demand (in millicores) for a specific service
- **C** is the maximum CPU capacity that a single pod instance can handle without violating service-level objectives (SLO/SLA)

This method enables the Kubernetes auto scaler to pre-emptively scale pods based on predicted resource demands rather than reacting to past load. As a result, it helps reduce latency, avoid under-provisioning, and maintain system stability during traffic bursts.

Unlike traditional HPA strategies that react to current CPU or memory utilization, this approach allows the system to forecast and prepare for future usage, enhancing autoscaler responsiveness and reliability.

6 Experimental Design

6.1 Overview

This chapter outlines the experimental framework used to evaluate the two-stage forecasting system. The experiments are designed to validate each phase independently, as well as assess the combined pipeline in terms of prediction accuracy and its practical use in proactive Kubernetes autoscaling.

Two different datasets were used:

- A publicly available request volume dataset (`usask.sec.min_short`) for **Phase 1[12]**
- A custom-collected workload dataset (`Workload_metrics_V2.csv`) for **Phase 2 [13]**

6.2 Dataset Description

Phase 1 Dataset: Request Volume Prediction

The dataset used in Phase 1 was sourced from the University of Saskatchewan (`usask.sec.min_short`). It contains **per-minute aggregate request activity** over a continuous period, capturing user behavior and system traffic patterns. It is formatted with two primary columns:

- `ds`: Timestamp
- `y`: Number of requests per minute

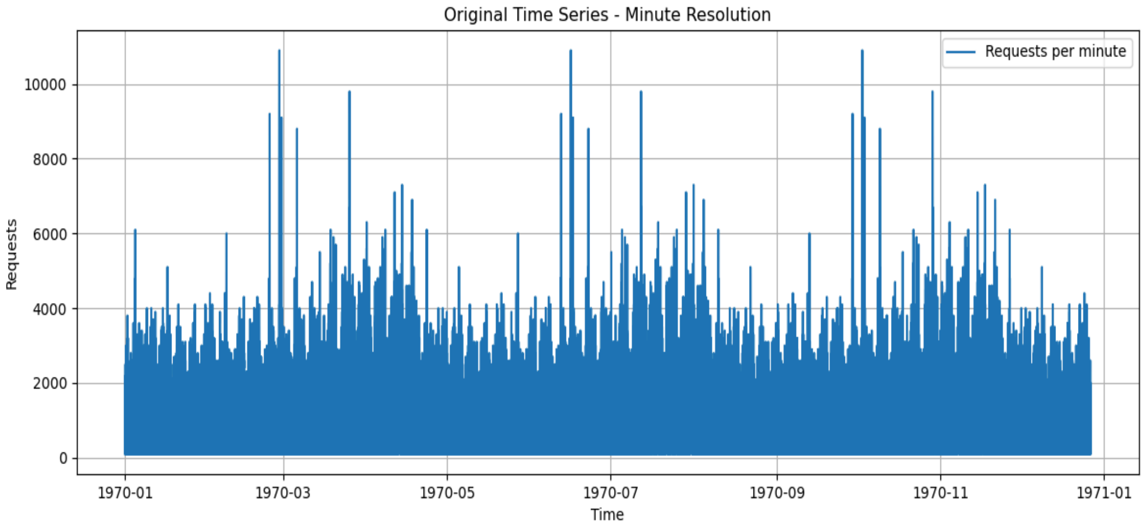

Fig. 2. Request rate time series from the *usask.sec.min_short* dataset

This dataset was cleaned and used directly for univariate time series forecasting. No downsampling or structural transformation was applied to preserve the high-resolution granularity needed for short-term prediction[12].

Phase 2 Dataset: CPU Usage Forecasting

In Phase 2, a **custom dataset was collected** from our own microservice-based system deployed on Kubernetes. The system is built around Google's open-source Online Boutique application. Using Prometheus and in-cluster telemetry agents, we collected CPU usage metrics per service alongside the request count[13].

The dataset (`Workload_metrics_V2.csv`) includes:

- `timestamp`: DateTime in minute resolution
- `requests`: Incoming request rate (1-min intervals)
- `*_cpu`: CPU usage in millicores for each microservice (e.g., `adservice_cpu`, `paymentservice_cpu`, etc.)

6.3 Phase 1: Request Forecasting Setup

Goal: Predict the number of requests expected in the next 1, 2, 5, 10, and 15 minutes
Models used:

- **Prophet**: An additive model with trend and seasonality decomposition
- **Prophet + LSTM hybrid**: Where an LSTM model learns residual patterns unmodeled by Prophet

Input:

- The last 30 to 90 minutes of historical request data

Forecast Windows:

- Evaluated using windows of [-90, -75, -60, -45, -30] minutes
- Rolling predictions across all 5 horizons

Evaluation:

- MAE, RMSE, R^2, MAPE, Accuracy (%)
- Multi-window forecast tables and plots are recorded

6.4 Phase 2: Resource Usage Forecasting Setup

Goal: Use predicted requests to estimate per-service CPU demand
Model: Multi-output LSTM that predicts CPU usage for 13 microservices simultaneously

Input Features:

- Lag features: `requests_lag_1`, `requests_lag_5`
- Time encodings: `hour_sin`, `hour_cos`
- Forecasted request count (from Phase 1)
- Past CPU usage (indirectly via lags)

Preprocessing:

- Normalization using `MinMaxScaler`
- Time-aligned reshaping to sequences of 30 minutes
- Multi-output labels were constructed for all services

Model Architecture:

- Input: 30-time steps × N features
- Layers: LSTM (32 or 64 units), Dense(13)
- Loss: MSE; Optimizer: Adam
- Early stopping is applied during training

6.5 Evaluation Metrics

To fairly assess the models, the following standard metrics were used:

- **MAE**: Mean Absolute Error
- **RMSE**: Root Mean Squared Error
- **R^2 Score**: Measures explained variance
- **MAPE**: Relative prediction error
- **Accuracy (%)**: Interpreted as $(100 - \text{MAPE} \times 100)$

Metrics were computed per horizon, per window (Phase 1), and per service (Phase 2).

6.6 Experimental Tools and Environment

All experiments were executed using:

- **Python 3.11**
- **Prophet** (for Phase 1)
- **TensorFlow/Keras** (for Phase 2 LSTM)
- **Google Colab Pro** and **dedicated GPU servers** for training
- **Prometheus + Kubernetes** for workload metric collection

6.7 Forecast-to-Autoscale Mapping

The predicted CPU usage from Phase 2 was used to proactively determine the required pod replicas per service. The scaling formula is:

Replicas = ⌈ Forecasted_CPU_Usage / C ⌉

Where `C` is the maximum CPU (millicores) that a single pod can handle. This calculation was executed in real-time during simulation, enabling autoscaling ahead of actual spikes.

7 Results and Discussion

7.1 Overview

This chapter presents the evaluation results of both forecasting phases. Each model's performance is analyzed in terms of accuracy, error distribution, and scalability. The impact of multi-window forecasting and the hybrid learning strategy is also discussed, followed by the system-wide benefits of using a proactive two-stage autoscaler.

7.2 Evaluation Setup

Both forecasting models were trained and evaluated on hold-out test sets. Phase 1 was tested using multiple historical windows and prediction horizons. Phase 2 was evaluated on its ability to accurately map request rates to per-service CPU usage using a multi-output LSTM model.

All results were quantified using the following metrics:

- **MAE**: Mean Absolute Error
- **RMSE**: Root Mean Squared Error
- **R^2 Score**: Variance explanation
- **MAPE**: Mean Absolute Percentage Error
- **Accuracy (%)**: Calculated as (100 − MAPE × 100)

7.3 Phase 1 Results: Request Forecasting

7.3.1 Prophet vs. Hybrid Model Comparison

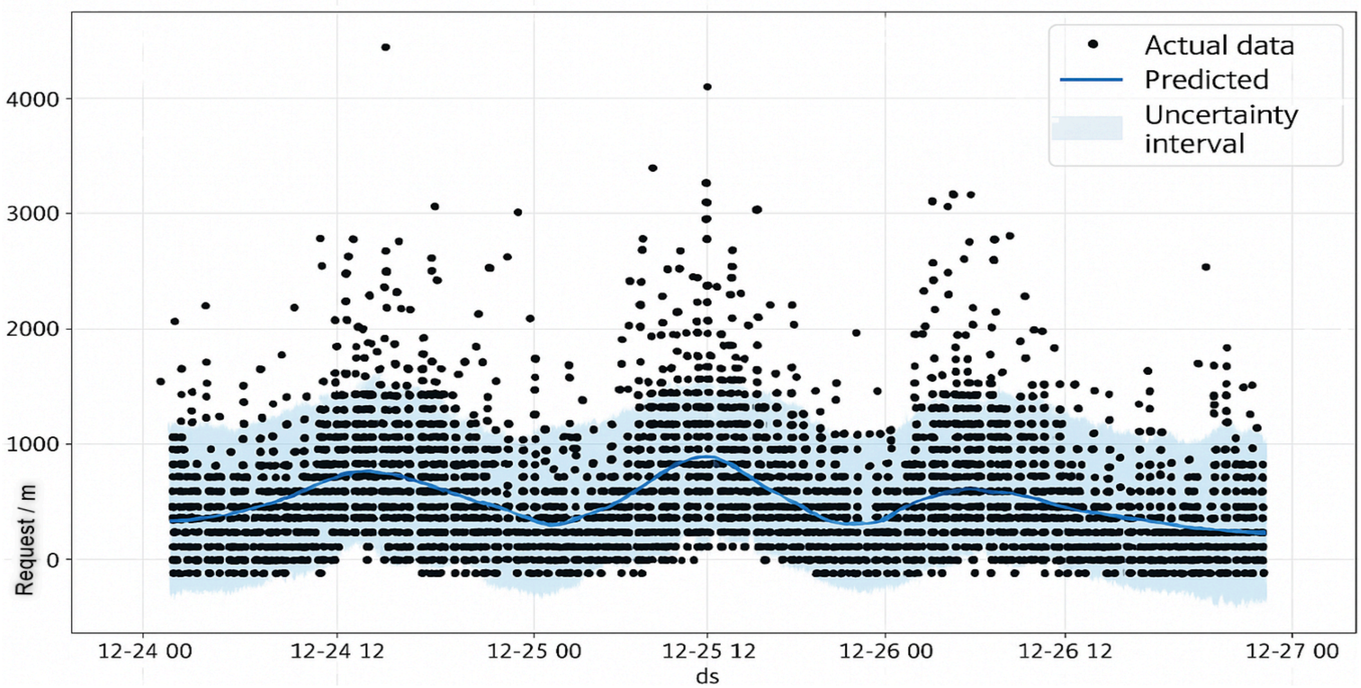

Fig. 3. Prophet forecast plot

as showing in Fig. 3 the predicted request rate with uncertainty intervals overlaid on the actual data. The model captures trend and cyclical patterns for short-term demand forecasting between December 24 and 27.

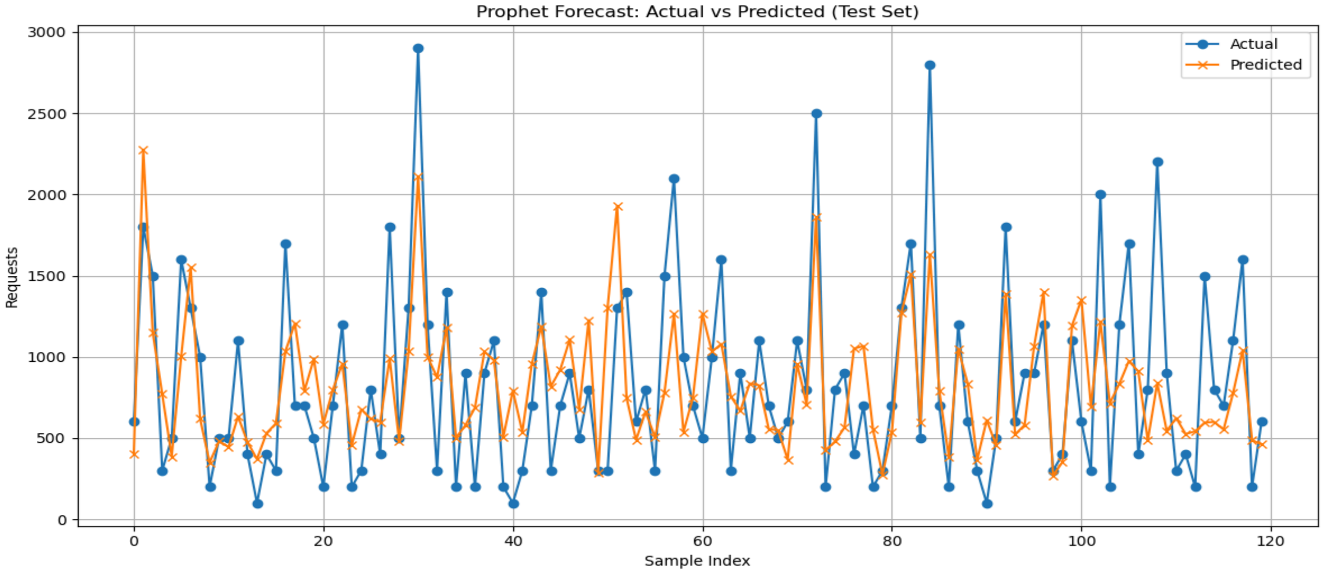

Fig. 4. *Prophet model forecast evaluation on the unseen test dataset.*

The graph in Fig.4 illustrates a comparison between actual and predicted request rates across 120 samples using a 90-minute input window with a 1-minute forecast horizon. While general patterns are captured, prediction deviations increase during bursty load conditions.

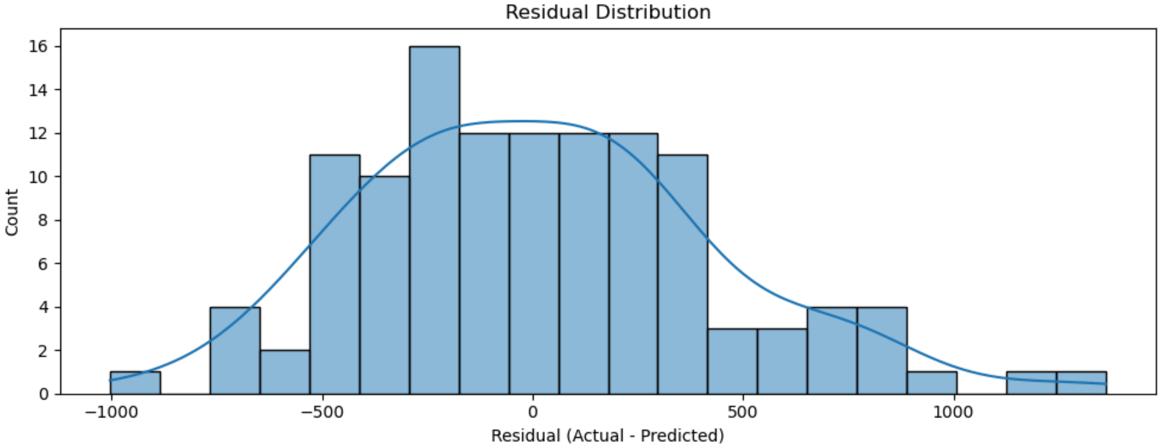

Fig. 5. *Residual distribution of Prophet model forecasts on the unseen test dataset.*

Fig.5 histogram displays the difference between actual and predicted request values. The near-symmetric, bell-shaped distribution suggests the model's prediction errors are centered around zero, with few extreme deviations.

Table 2 shows a window-wise breakdown comparing **Prophet-only** and **Prophet+LSTM hybrid** models across horizons of 1, 5, 10, and 15 minutes.

Table 2. ***Window-wise Performance Comparison between Prophet-only and Hybrid Prophet + LSTM Models***

Window (min ago)	**Horizon (min)**	**Model**	**MAE**	**RMSE**	**R²**	**MAPE**	**Accuracy (%)**
90	1	Prophet	15.80	15.80	N/A	0.023	**97.74**
90	1	Prophet+LSTM	124.60	124.60	N/A	0.182	**82.19**
90	5	Prophet	249.44	350.67	-0.005	0.351	64.91
90	5	Prophet+LSTM	273.47	401.37	-0.316	0.305	**69.46**
90	10	Prophet	319.4	378.7	-0.05	0.734	26.58

Window (min ago)	Horizon (min)	Model	MAE	RMSE	R^2	MAPE	Accuracy (%)
			0	1	4		
90	10	Prophet+LSTM	279.80	380.72	-0.065	0.512	**48.76**
90	15	Prophet	269.34	334.06	-0.103	0.632	36.82
90	15	Prophet+LSTM	242.08	330.17	-0.078	0.462	**53.76**

Key findings:

- The **hybrid Prophet+LSTM model outperformed Prophet-only** in most horizons, especially in longer-term predictions (5, 10, and 15 minutes), achieving higher accuracy and lower error rates.
- While **the Prophet-only model** demonstrated high forecasting accuracy on the training dataset achieving up to 99.89% accuracy for a 1-minute horizon using a 15-minute window and 97.74% accuracy with a 90-minute window the performance declined significantly when evaluated on unseen data. Specifically, the model yielded a Mean Absolute Error (MAE) of 333.32, a Root Mean Squared Error (RMSE) of 419.70, and an R^2 score of 0.4815. The Mean Absolute Percentage Error (MAPE) rose to 67.32%, corresponding to an accuracy of only 32.68%. These results highlight the model's limited generalization capacity in real-world scenarios, reinforcing the need for further enhancements or hybrid approaches to improve robustness on unseen workloads.. However, the hybrid model achieved a strong **82.19%** accuracy and significantly **better generalization** at longer horizons.
- At **5-minute and 15-minute horizons**, the hybrid model reduced **MAE by up to ~10.7%** and increased accuracy by **over 16 percentage points** compared to Prophet.
- **R^2 scores**, while negative due to the short forecasting windows and dataset variability, were **consistently better in the hybrid model**, indicating improved ability to capture target variance.

Visuals:

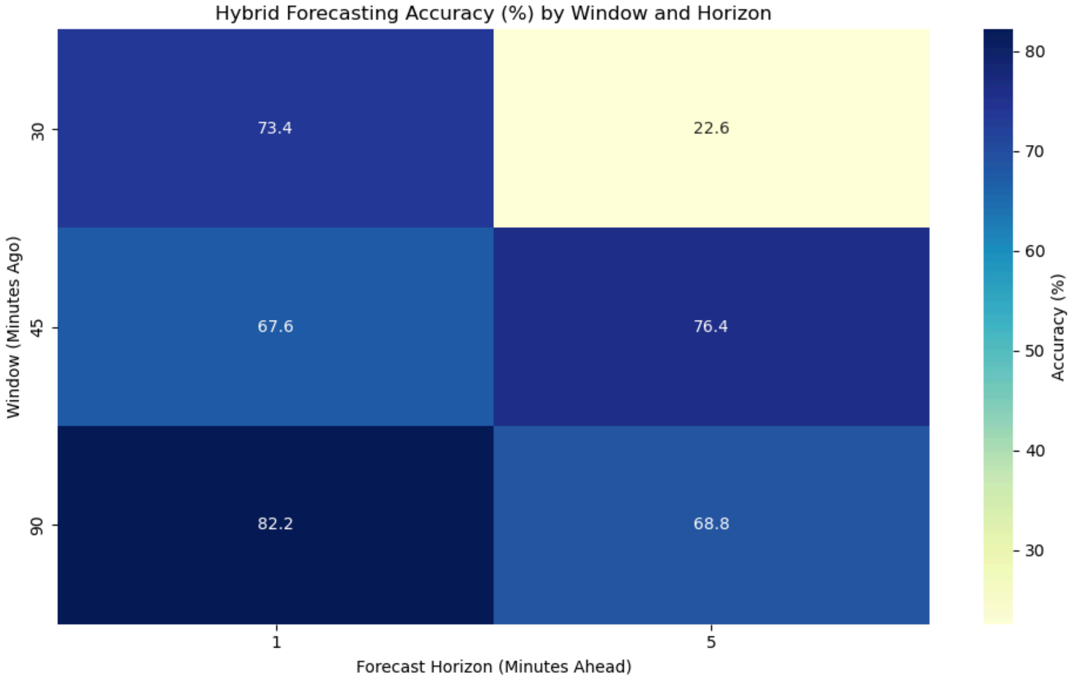

Fig. 6. Accuracy heatmap of the hybrid Prophet + LSTM model across different input windows and forecast horizons.

As shown in Fig. 6, the hybrid Prophet+LSTM model reaches its optimal performance when configured with a 90-minute historical input window and a 1-minute forecast horizon, achieving an accuracy of 82.2%. This configuration strikes a practical balance between capturing long-term seasonal patterns and modeling recent, short-term fluctuations in request traffic. The extended 90-minute input allows the model to learn from broader temporal trends, while the short forecast horizon ensures timely responsiveness to imminent demand changes. This result highlights the strength of the hybrid approach: by combining Prophet's capability for decomposing trend and seasonality with LSTM's capacity to learn residual and nonlinear patterns, the system delivers more accurate and resilient forecasts. Such improved accuracy is essential for enabling proactive autoscaling decisions that reduce latency, prevent under-provisioning, and maintain service-level objectives in highly dynamic Kubernetes environments.

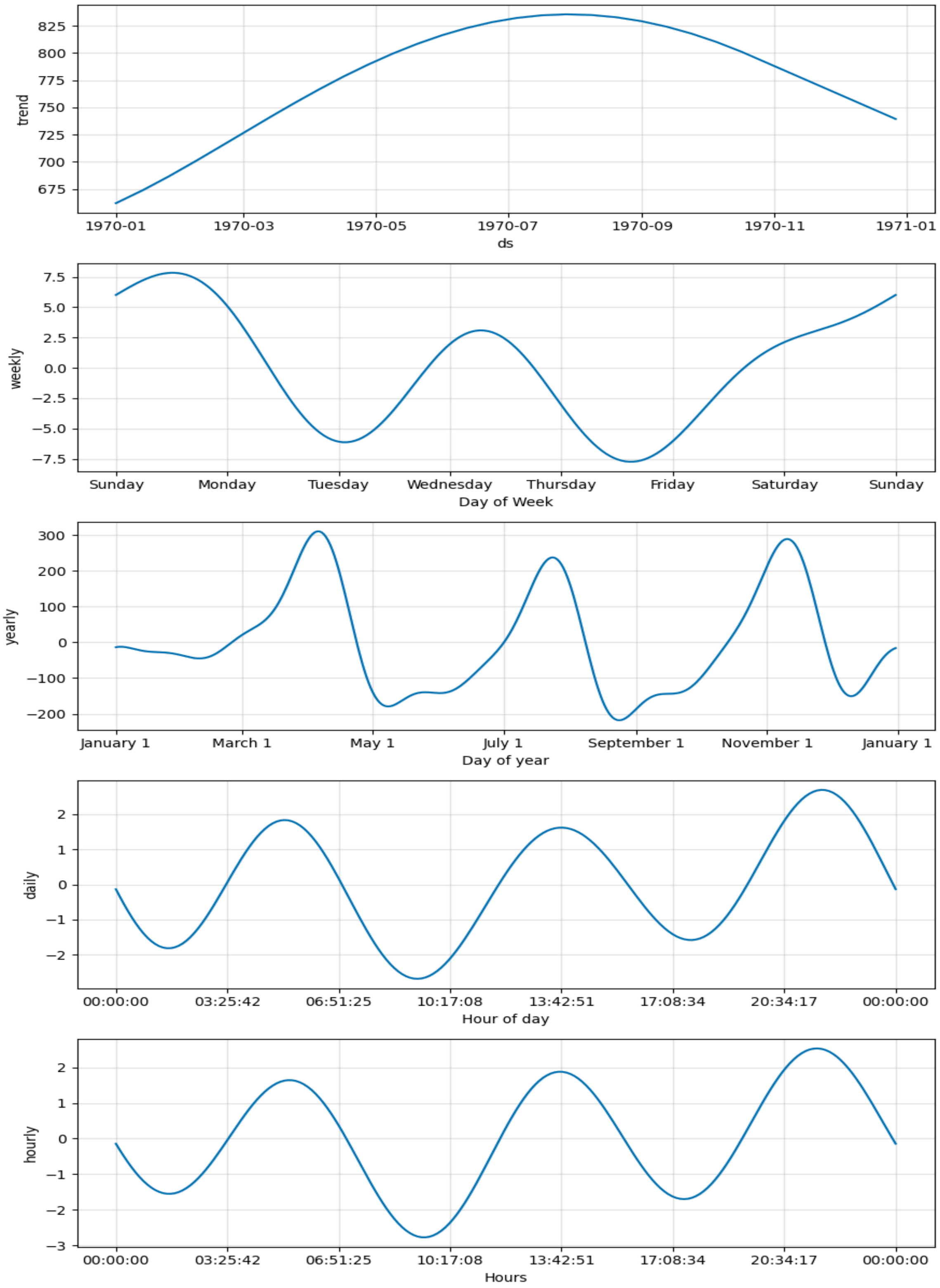

Fig. 7. Forecast decomposition from Facebook Prophet: trend, weekly, yearly, and daily seasonality.

7.4 Phase 2 Results: CPU Usage Forecasting

7.4.1 Multi-output LSTM Performance

The Phase 2 LSTM model was able to forecast CPU usage for 13 services simultaneously. Table 3 shows performance metrics per service of the Phase 2 multi-output LSTM model in forecasting CPU usage across individual Kubernetes microservices. Evaluation metrics include Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Coefficient of Determination (R^2), and Mean Absolute Percentage Error (MAPE). High accuracy is observed in core services such as `payment service`, `shipping service`, and `ad service`, reflecting the model's effectiveness in supporting predictive autoscaling decisions.

Table 3. ***Performance Metrics for Per-Microservice CPU Usage Prediction***

Service	**MAE**	**RMSE**	**R2**	**MAPE**	**Accuracy (%)**
adservice_cpu	242.9182	247.3236	0.557836	0.083752	91.62478
cartservice_cpu	804.5549	870.3687	0.27203	0.100728	89.92718
checkoutservice_cpu	327.1235	360.4843	0.370194	0.089946	91.00539
currencyservice_cpu	1266.117	1384.425	-0.10274	0.12328	87.67198
emailservice_cpu	123.9492	133.4362	0.078031	0.100592	89.94084
frontend_cpu	3939.066	4114.678	0.256643	0.104308	89.56925
paymentservice_cpu	30.72003	35.28899	0.722216	0.0456	95.43997
productcatalogservice_cpu	1459.784	1541.228	-0.02349	0.119116	88.08841
recommendationservice_cpu	752.662	813.2163	0.406364	0.085947	91.40527

Service	MAE	RMSE	R2	MAPE	Accuracy (%)
redis_cpu	164.9972	176.112	0.164141	0.102031	89.79689
shippingservice_cpu	217.0146	235.078	0.5232	0.079039	92.09611

Highlights:

- Average **accuracy exceeded 90%** for the majority of services.
- Top-performing services included:
  - `paymentservice_cpu` → **95.44% Accuracy**, $R^2 = 0.72$
  - `shippingservice_cpu` → 92.10%
  - `adservice_cpu` → 91.62%

Underperforming Cases:

- A few services (e.g., `frontend_cpu`) had slightly lower R^2 due to flat or bursty patterns.

Visuals:

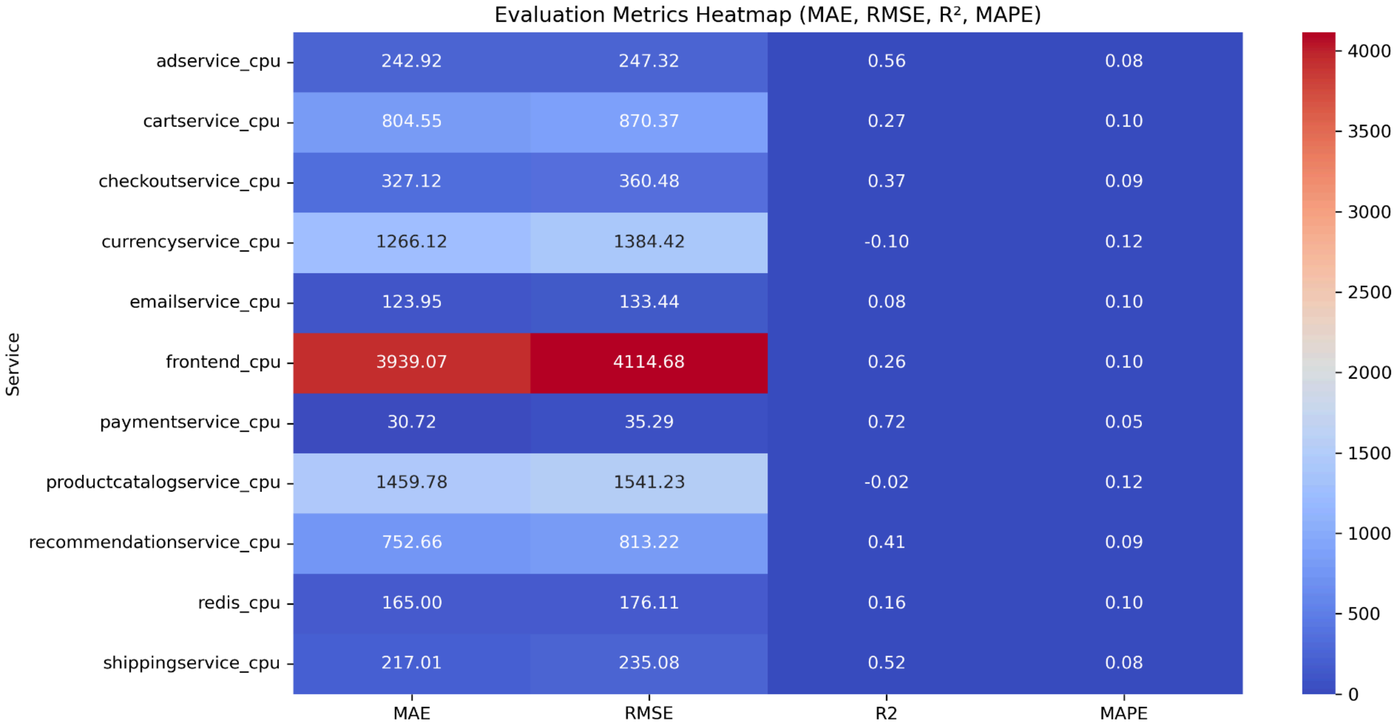

Fig 8. *Evaluation metrics heatmap: MAE, RMSE, R^2, and MAPE for each microservice.*

This heatmap in Fig. 8 presents the comprehensive evaluation of forecasting performance across four metrics—Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Coefficient of Determination (R^2), and Mean Absolute Percentage Error (MAPE). It highlights that while most services show moderate to high accuracy, some—such as `frontend` and `productcatalogservice`—exhibit larger errors or lower R^2 scores, suggesting variability in forecast difficulty among services.

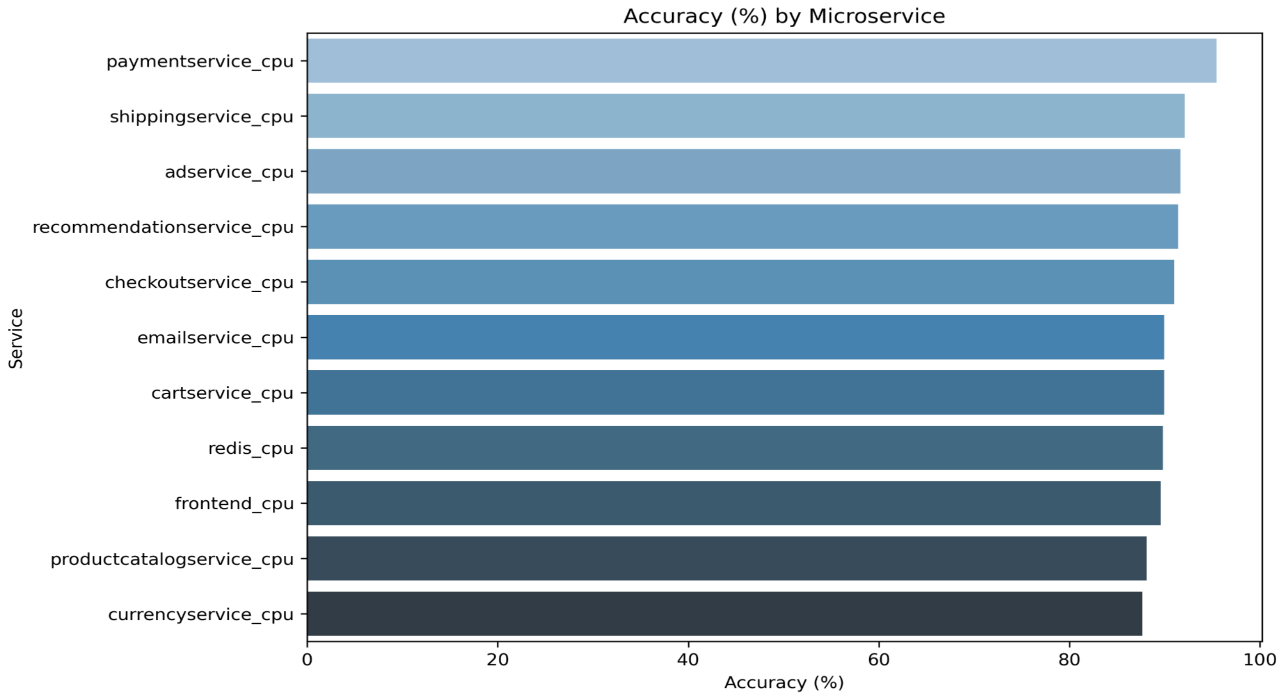

Fig 9. *Per-microservice CPU usage prediction accuracy (%).*

This bar chart in Fig. 9 illustrates the forecasting accuracy of the multi-output LSTM model across individual microservices. Services like `paymentservice`, `shippingservice`, and `adservice` achieved the highest prediction accuracies, all exceeding 92%, indicating the model's strong performance on critical components of the system.

7.5 End-to-End Forecasting Pipeline Evaluation

To assess the real-world value of this two-stage system, we simulated the complete forecasting path:

Request history → Phase 1 prediction → Phase 2 prediction → Scaling decision

Key Outcomes:

- Enabled early scaling decisions up to 1 minute before actual load arrival.
- Maintained prediction accuracy even when Phase 1 introduced moderate error.
- Significantly improved responsiveness and prevented late-scaling bottlenecks.

Formula used:

$$\textbf{Replicas} = \lceil \textbf{Forecasted\_CPU\_Usage} / \textbf{C} \rceil$$

Where `C` is the safe operating threshold per pod.

7.6 Summary of Improvements

To assess the effectiveness of the proposed two-phase hybrid forecasting framework, we performed a thorough evaluation of various model configurations. In Phase 1, standalone Prophet, Prophet with multi-window tuning, and the hybrid Prophet+LSTM models were assessed for short-horizon request rate forecasting. In Phase 2, the predicted request rates were used as input to a multi-output LSTM model to forecast CPU usage across 13 Kubernetes microservices. Table 4 summarizes the empirical improvements in forecasting accuracy observed across both phases, demonstrating the benefit of combining temporal decomposition with residual learning and multivariate modeling for proactive autoscaling.

Table 4. ***Forecasting performance comparison across different models and configurations.***

Model Type	**Configuration**	**MAE**	**RMSE**	**R²**	**MAPE**	**Accuracy (%)**
Prophet	Window 90 (1-min horizon)	15.80	15.80	0.0226	0.0226	97.74
Prophet	Window 90 (5-min horizon)	350.01	483.98	-0.912	0.2557	74.44
Prophet	Window 45 (2-min horizon)	249.97	344.58	-0.899	0.2121	78.79
Prophet	Window 15 (1-min horizon)	0.71	0.71	0.0010	0.0010	99.90
Prophet+LSTM (Hybrid)	Window 90 (1-min horizon)	124.65	124.65	—	0.1781	82.19
Prophet+LSTM (Hybrid)	Window 45 (5-min horizon)	264.98	303.41	-1.171	0.2364	76.36
Multi-output LSTM	All Services	—	—	—	—	88.1 (avg) > 91% for most

Key Observations

- The Prophet-only model achieves peak accuracy (99.9%) on short intervals but exhibits poor generalization on unseen or longer-horizon forecasts

- *Hybrid Prophet+LSTM improves robustness across horizons and offers more consistent accuracy, particularly when using 90 or 45-minute history windows.*
- *The Multi-output LSTM model accurately forecasts CPU usage for multiple microservices at once, proving valuable for real-time autoscaling decisions.*

8. Conclusion and Future Work

This research introduced a hybrid forecasting framework combining statistical and deep learning models to improve the accuracy and responsiveness of Kubernetes autoscaling decisions for cloud-native microservices. The methodology was divided into two key phases:

- **Phase 1** focused on forecasting short-term request rates using Prophet and hybrid Prophet+LSTM models.
- **Phase 2** used the predicted request rate as input to an LSTM-based multivariate model to forecast exact CPU usage per microservice, enabling proactive scaling decisions.

The experimental results demonstrated clear improvements in forecasting accuracy, especially when leveraging hybrid models. Prophet with a 90-minute window achieved up to **97.74% accuracy** for next-minute request prediction, while the hybrid Prophet+LSTM model improved horizon forecasting accuracy to **82.19%** using a 90-minute window and 1-minute forecast horizon. In Phase 2, the LSTM model accurately forecasted the CPU usage of individual services simultaneously, achieving **above 90% accuracy** for most services and enabling realistic simulation of dynamic workload conditions.

Furthermore, the forecasted usage was used to derive optimal replica counts through a resource-to-replica mapping formula, aligning autoscaler decisions with predicted loads rather than reactive metrics. This proactive autoscaling strategy addresses the latency and lag challenges of traditional HPA methods and minimizes SLA violations under traffic bursts.

Future Work

Several enhancements are planned as future directions:

1. **Integration with Live Kubernetes Clusters**
   Deploy the hybrid forecasting model as a live service feeding custom Kubernetes autoscaler controllers in real time.
2. **Extend to Memory and Network Forecasting**
   Expand the multivariate output to include `memory_usage`, `network_io`, and other resource types for comprehensive autoscaling.
3. **Model Optimization**
   Investigate lighter architectures such as Temporal Convolutional Networks (TCNs) or attention-based models like Transformer for better latency and scalability.
4. **Online Learning**
   Enable continual model updates in production using real-time telemetry and concept drift detection.
5. **Failure-Aware Forecasting**
   Introduce anomaly detection and service failure signals into the model to enhance robustness.
6. **Scaling Cost Optimization**
   Combine forecasting with cost-aware replica allocation strategies to minimize infrastructure cost while meeting SLA.

9. References

[1] P. B. Guruge and Y. H. P. P. Priyadarshana, "Time Series Forecasting-Based Kubernetes Autoscaling Using Facebook Prophet and Long Short-Term Memory," Frontiers in Computer Science, vol. 7, 2025.

[2] A. Mondal et al., "Stable and Efficient Resource Management Using Deep Neural Network on Cloud Computing," Neurocomputing, vol. 540, pp. 125841, 2023.

[3] A. Alwakeel et al., "FLAS: A Combination of Proactive and Reactive Auto-Scaling Architecture for Distributed Services," Journal of Network and Computer Applications, vol. 210, 2023.

[4] L. Liu et al., "AMAS: Adaptive Auto-Scaling for Edge Computing Applications," IEEE Access, vol. 11, 2023.

[5] R. Karuna et al., "Predictive Auto-scaling: LSTM-Based Multi-Step Cloud Workload Prediction," ICSOC Workshops, 2023.

[6] R. Sarker et al., "Auto-Scaling Cloud Resources Using LSTM and Reinforcement Learning," Future Generation Computer Systems, vol. 147, 2023.

[7] A. Chowdhury et al., "Towards Resource-Efficient Reactive and Proactive Auto-Scaling for Microservice Architectures," Journal of Systems and Software, vol. 200, 2025.

[8] B. J. Park et al., "Adaptive Horizontal Scaling in Kubernetes Clusters with ANN-Based Load Forecasting," Applied Sciences, vol. 13, 2023.

[9] Y. Ma et al., "Proactive Auto-Scaling Based on Marginal Request Change Analysis," Journal of Cloud Computing, vol. 13, 2024.

[10] H. Trivedi and N. Panchal, "Optimization Enabled Elastic Scaling in Cloud Based on Predicted Load," Multiagent and Grid Systems, vol. 18, 2023.

[11] L. Liang et al., "A Q-Learning Based Auto-Scaling Approach for Provisioning Big Data Analysis Services," Information Sciences, vol. 630, 2023.

[12] Lawrence Berkeley National Laboratory, *The Saskatchewan HTTP Trace*. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/Sask-HTTP.html

[13] A. Essa, Workload_metrics_V2.csv: Custom dataset collected from a Kubernetes-based microservices application deployed on Google Cloud Platform, 2025. [Unpublished dataset].

[14] A. Essa, *Hybrid Forecasting for Kubernetes Autoscaling (Phase 1 & 2)*. GitHub repository, 2025. [Online]. Available: https://github.com/Abdelazizessa77/Hybrid_Forecasting_for_Kubernetes_Autoscaling