PredictScale: Proactive Autoscaling via Workload Forecasting for Edge Microservices Application

Bablu Kumar[1], Anshul Verma[1,2], Pradeepika Verma[3], Rajkumar Buyya[2]
[1]*Department of Computer Science*, Banaras Hindu University, Varanasi, India
[2]*Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory,*
School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
[3]*School of Computer Science Engineering & Technology*, Bennett University, Greater Noida, India
bablu.kumar@bhu.ac.in, anshul.verma@unimelb.edu.au, anshul.verma@bhu.ac.in,
pradeepika.verma@bennett.edu.in, rbuyya@unimelb.edu.au

***Abstract*—Edge application platforms increasingly host latency-sensitive microservices under highly dynamic and bursty workloads. Reactive autoscaling mechanisms, such as the Kubernetes Horizontal Pod Autoscaler (HPA), respond only after resource saturation occurs, which can lead to performance degradation and inefficient resource utilization. This paper presents PredictScale, a Transformer-based proactive autoscaling framework that anticipates workload fluctuations and provisions resources ahead of demand. PredictScale integrates workload forecasting with Kubernetes-native autoscaling control to reduce scaling delay, improve resource utilization, and mitigate latency without modifying the underlying orchestration platform. Experimental evaluation using real workload traces from the Bitbrains dataset compares PredictScale with reactive HPA and recurrent neural network-based predictors, including LSTM, BiLSTM, and CNN-LSTM. Results show that PredictScale achieves higher forecasting accuracy and more balanced resource provisioning, leading to improved pod utilization, reduced operational cost, and lower latency under bursty workloads. These findings demonstrate that Transformer-based temporal modeling can effectively support proactive autoscaling and improve resource management in dynamic cloud–edge environments.**

***Index Terms*—Proactive Autoscaling, Kubernetes, Transformer, Edge Computing, Microservices Application**

I. Introduction

Edge computing has emerged as a fundamental paradigm for supporting modern latency-sensitive and data-intensive applications, including Internet of Things (IoT) analytics, real-time monitoring, online inference, and interactive services [1]–[5]. These applications are increasingly deployed as containerized microservices orchestrated by platforms such as Kubernetes, which provide scalability, fault tolerance, and portability across heterogeneous edge infrastructures [6], [7]. However, managing resources efficiently in such environments remains a critical challenge due to highly dynamic, bursty, and unpredictable workload patterns.

Autoscaling plays a central role in maintaining service performance while controlling operational cost. Kubernetes Horizontal Pod Autoscaler (HPA) is the most widely adopted autoscaling mechanism, adjusting the number of pods based on instantaneous resource utilization metrics such as CPU or memory usage [6], [8]. While effective for relatively stable workloads, HPA is inherently reactive, as scaling decisions are triggered only after resource saturation occurs [1]. This often leads to delayed responses under sudden workload spikes, resulting in transient performance degradation, increased tail latency, and inefficient resource utilization, particularly in edge environments where network latency and resource constraints amplify these effects [7], [9].

To overcome the limitations of reactive scaling, proactive autoscaling approaches have been proposed [10], [11]. These methods aim to predict future workload demand and provision resources in advance, thereby reducing scaling delays and improving system responsiveness. Early proactive solutions relied on statistical or short-horizon forecasting techniques, while more recent approaches leverage machine learning models such as recurrent neural networks (RNNs) and long short-term memory (LSTM) networks to capture temporal workload patterns [10]. Although these models improve short-term prediction accuracy, they often struggle to model long-range temporal dependencies and complex workload dynamics commonly observed in real-world edge deployments [12].

Transformer architectures, originally introduced for natural language processing, have demonstrated strong capability in modeling long-term dependencies through self-attention mechanisms. Recent advances such as Informer, Autoformer, and FEDformer have shown that Transformer-based models significantly outperform recurrent approaches in long-horizon time-series forecasting tasks [13]–[15]. These characteristics make Transformers particularly attractive for proactive autoscaling, where understanding long-range workload trends is essential for reliable resource provisioning. Despite this promise, the integration of Transformer-based forecasting into Kubernetes-native autoscaling pipelines remains underexplored, especially in edge scenarios characterized by heterogeneous resources and bursty demand [16], [17].

In this paper, we present **PredictScale**, an intelligent proactive autoscaling framework designed for edge microservices. PredictScale leverages long-range workload forecasting to anticipate future demand and dynamically adjust pod allocation before resource contention occurs. Unlike standalone predictors evaluated in isolation, PredictScale is tightly integrated with Kubernetes autoscaling control loops, enabling seamless deployment in real-world environments. By combining

979-8-3315-7047-7/26/$31.00 ©2026 IEEE
DOI 10.1109/CCGridW69005.2026.00037

accurate workload prediction with proactive scaling actions, PredictScale aims to reduce tail latency [3], improve pod utilization, and lower operational cost under dynamic workloads [12]. The contributions of this work are threefold.

- First, we design a Transformer-based workload forecasting model specifically tailored for autoscaling scenarios, which is capable of capturing both short-term workload fluctuations and long-term temporal trends in dynamic edge environments using self-attention mechanisms.
- Second, We integrate the forecasting model with Kubernetes-native autoscaling to enable proactive pod provisioning without modifying the underlying orchestration platform, ensuring compatibility with edge microservice deployments.
- Third, We perform extensive experiments under realistic bursty workloads, comparing PredictScale with reactive Kubernetes HPA and learning-based baselines (LSTM, BiLSTM, and CNN–LSTM) using both prediction accuracy and system-level metrics such as pod utilization, operational cost, and tail latency.

The results demonstrate that PredictScale consistently improves forecasting accuracy, reduces scaling delay, and achieves superior overall system performance in edge environments.

II. Related Work

Autoscaling has been extensively studied as a mechanism to dynamically manage resources in edge computing environments. Existing approaches can be broadly categorized into reactive autoscaling, proactive autoscaling based on workload prediction, and learning-based autoscaling strategies (Table I).

A. Reactive Autoscaling in Kubernetes

Reactive autoscaling mechanisms are the default choice in most container orchestration platforms. Kubernetes Horizontal Pod Autoscaler (HPA) dynamically scales application replicas based on instantaneous resource utilization metrics such as CPU and memory usage [6], [18]. While HPA is simple to deploy and effective for relatively stable workloads, it reacts only after resource saturation occurs, often leading to delayed scaling under sudden workload spikes [19]. Prior studies have shown that purely reactive autoscaling can cause transient performance degradation, increased tail latency, and inefficient resource utilization, particularly in edge environments where resource constraints and network latency amplify these effects [9], [11]. Although several variants attempt to improve responsiveness through threshold tuning or multi-metric policies, these approaches remain fundamentally reactive and lack the ability to anticipate future demand.

B. Proactive Autoscaling and Workload Forecasting

To overcome the limitations of reactive scaling, proactive autoscaling approaches incorporate workload forecasting to provision resources ahead of time [8], [10]. Early proactive solutions relied on statistical models such as moving averages, ARIMA, and seasonal decomposition, which are computationally efficient but often fail to capture nonlinear patterns and long-term dependencies in real-world workloads. More recent studies have applied machine learning models, including recurrent neural networks (RNNs), long short-term memory (LSTM), and gated recurrent unit (GRU) networks, to predict CPU utilization, request rates, or response times for autoscaling decisions [11]. While these methods improve short-term prediction accuracy, they often struggle with long-range dependencies and abrupt workload changes. Hybrid architectures combining convolutional and recurrent layers have also been explored to capture both local and temporal features, but they introduce additional complexity and are typically evaluated in isolated prediction settings rather than within full Kubernetes autoscaling pipelines [12], [17], [18].

C. Transformer-Based Autoscaling Approaches

Transformer architectures, based on self-attention mechanisms, have demonstrated superior performance in modeling long-term temporal dependencies in time-series forecasting tasks [13]–[15]. Their ability to capture global workload patterns without recurrent structures makes them particularly suitable for autoscaling scenarios with complex and bursty demand. Recent work has explored Transformer-based forecasting for resource management, showing improved accuracy compared to LSTM- and GRU-based approaches [16], [17]. However, most existing studies focus primarily on prediction accuracy and treat Transformers as standalone predictors, without tight integration into Kubernetes-native autoscaling control loops [17]. Moreover, evaluations are often limited to centralized edge settings, overlooking edge challenges such as heterogeneous resources, network variability, and scaling overhead.

While prior work such as the MAPE-Transformer autoscaling framework and InformerAutoScale focused primarily on large-scale cloud workloads and long-sequence forecasting, PredictScale targets proactive autoscaling in cloud–edge environments characterized by bursty and short-term workload fluctuations (Table I). The proposed framework integrates a lightweight Transformer forecasting model with Kubernetes-native autoscaling mechanisms, enabling responsive and resource-efficient scaling decisions suitable for edge-oriented microservices.

Despite significant progress, existing autoscaling solutions exhibit notable gaps. Reactive approaches lack foresight, while many proactive methods emphasize prediction accuracy without addressing deployment realism or orchestration integration. Transformer-based models show strong promise, yet their adoption in Kubernetes-native autoscaling pipelines remains limited [19]. This work addresses these gaps by proposing a Transformer-driven proactive autoscaling framework that is tightly integrated with Kubernetes orchestration. Unlike prior approaches, our method jointly considers workload forecasting and system-level performance metrics, including pod utilization, scaling latency, and operational cost, under realistic edge conditions.

TABLE I
COMPREHENSIVE COMPARISON OF RELATED AUTOSCALING FRAMEWORKS

Reference	Prediction Model	Control Strategy	Workload Type	Key Contribution
[7], [10]	Predictive model-based autoscaling	Threshold + predictive scaling	VM workloads	Introduces predictive autoscaling framework for cloud resource provisioning using workload forecasting.
[17]	Multivariate Transformer	MAPE-based autoscaling	Multivariate workload metrics	Proposes Transformer-based autoscaling integrated with the MAPE control loop for cloud resource management.
[12]	Informer model	Proactive autoscaling controller	Long sequence workloads	InformerAutoScale leverages Informer architecture to capture long-range workload dependencies for autoscaling.
LSTM-based scaling	LSTM	Predictive scaling	Time-series CPU workload	Uses LSTM forecasting for proactive autoscaling decisions in containerized environments.
CNN-LSTM scaling	CNN-LSTM	Predictive scaling	Bursty workloads	Combines convolutional feature extraction with LSTM for improved workload prediction.
Reactive HPA	None (threshold-based)	Reactive threshold control	Real-time CPU metrics	Default Kubernetes Horizontal Pod Autoscaler based on resource utilization thresholds.
PredictScale (Proposed)	Transformer Encoder	Proactive scaling with forecasting	Bursty edge microservices	Lightweight Transformer-based workload forecasting integrated with Kubernetes-native autoscaling to support proactive resource allocation in cloud–edge environments.

III. SYSTEM ARCHITECTURE

The proposed system architecture is designed to enable proactive and efficient autoscaling for Edge applications deployed in Kubernetes environments [19]. The architecture integrates workload monitoring, Transformer-based forecasting, and Kubernetes-native scaling mechanisms in a closed-loop control framework. Figure 1 illustrates the high-level architecture and interactions among system components. The architecture follows a modular and layered design, consisting of five main components: (i) application pods, (ii) monitoring and metric collection, (iii) workload forecasting engine, (iv) autoscaling decision module, and (v) Kubernetes orchestration layer. This separation of concerns allows the system to remain extensible, deployment-ready, and compatible with existing Kubernetes infrastructures [19]. At runtime, applications are deployed as containerized pods on edge worker nodes. Resource utilization and workload metrics are continuously collected and analyzed to predict future demand and proactively adjust resource allocation before performance degradation occurs [12], [17].

The application layer consists of containerized services deployed as Kubernetes pods representing workloads such as web services, data processing pipelines, or machine learning applications with dynamic resource demands. Each pod is configured with CPU and memory requests and limits that guide autoscaling decisions. The architecture supports heterogeneous edge deployments where worker nodes may differ in computational capacity, network bandwidth, and latency characteristics, motivating proactive scaling to avoid underprovisioning and excessive overprovisioning.

Resource monitoring is performed using Kubernetes-native tools such as `cAdvisor` and the Metrics Server. Key metrics, including CPU utilization, memory usage, and pod-level workload indicators, are collected at fixed intervals and aggregated to form time-series inputs for the forecasting module. Unlike reactive autoscaling mechanisms that rely only on instantaneous metrics, the proposed system maintains a historical

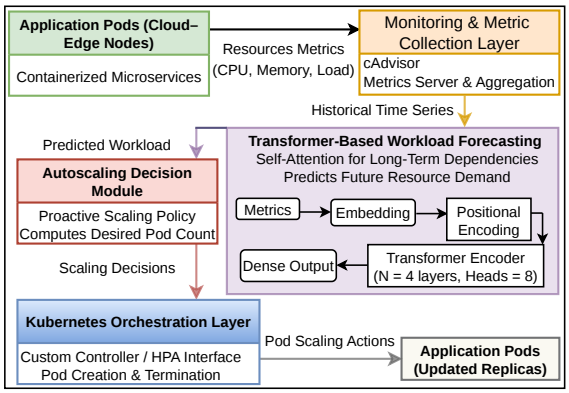

Fig. 1. System architecture of PredictScale showing the closed-loop integration of monitoring, Transformer-based workload forecasting, proactive autoscaling decision module, and Kubernetes orchestration for edge microservices.

observation window to capture workload trends and temporal dependencies.

Workload prediction is performed using a stacked Transformer encoder architecture designed for time-series forecasting. The model contains four encoder layers with multi-head self-attention (eight heads), an embedding dimension of $d_{model} = 512$, and a feedforward dimension of $d_{ff} = 2048$. Sinusoidal positional encoding preserves temporal ordering, while residual connections and layer normalization stabilize training. Historical CPU utilization values are provided as input using a sliding window of length $W = 20$, and the encoded representation is passed through fully connected layers to predict the next-step workload. A dropout rate of 0.2 is applied to reduce overfitting. The forecasting engine operates independently of application logic and can be trained offline or periodically updated with newly observed data. The autoscaling decision module converts predicted workload demand into scaling actions by computing the required number

of pods according to a predefined scaling policy. Unlike traditional reactive autoscalers, scaling actions are triggered proactively based on predicted demand, thereby reducing scaling latency and mitigating performance degradation during workload surges.

Scaling decisions are enforced by the Kubernetes orchestration layer through the control plane, which creates or terminates pods using Kubernetes APIs. The architecture remains fully compatible with existing Horizontal Pod Autoscaler (HPA) mechanisms or custom controllers, ensuring seamless integration with Kubernetes-based deployments while providing fault tolerance, load balancing, and resource isolation.

Overall, the system operates as a continuous control loop in which monitoring components collect metrics, the forecasting model predicts future workload, the autoscaling module computes scaling decisions, and Kubernetes enforces resource adjustments. This closed-loop mechanism enables adaptive and proactive autoscaling under dynamic and unpredictable workloads. Through this integrated architecture, the proposed system bridges the gap between workload forecasting and autoscaling mechanisms, providing a scalable Kubernetes-native framework for proactive resource management in edge environments.

IV. Methodology

This section presents the proposed proactive autoscaling methodology for Kubernetes-based edge environments. The approach integrates a Transformer-based workload forecasting model with a prediction-driven adaptation service to enable proactive pod scaling decisions. The design follows the *Monitor-Analyze-Plan-Execute* (MAPE) control loop and aims to minimize latency, resource waste, and scaling instability under dynamic workloads.

Let $\{x_t\}_{t=1}^{T}$ represent a time series of observed workload metrics collected from running application pods at fixed intervals, where x_t denotes CPU utilization at time step t. Given historical observations, the objective is to predict future workload values and proactively determine the required number of pods. Formally, the forecasting problem is defined as:

$$\hat{x}_{t+1}, \hat{x}_{t+2}, \ldots, \hat{x}_{t+H} = f(x_{t-W+1}, \ldots, x_t), \tag{1}$$

where W is the historical window size and H is the prediction horizon. Unlike reactive autoscaling mechanisms such as the Kubernetes Horizontal Pod Autoscaler (HPA), which respond only after utilization thresholds are violated, the proposed method anticipates workload variations and initiates scaling actions ahead of time. CPU utilization is selected as the primary scaling signal because it directly reflects computational demand and is natively supported by Kubernetes autoscaling mechanisms. Metrics are collected using Kubernetes monitoring services (e.g., cAdvisor). A sliding window of length W is used to construct model inputs:

$$\mathbf{X}_t = [x_{t-W+1}, x_{t-W+2}, \ldots, x_t] \tag{2}$$

Each input sequence is normalized to ensure numerical stability and consistent prediction across heterogeneous edge nodes. The workload forecasting module is implemented using a Transformer encoder architecture due to its ability to capture long-range temporal dependencies via self-attention. Unlike recurrent models, the Transformer enables parallel computation and avoids vanishing-gradient issues. Given an input sequence $\mathbf{X}_t$, the Transformer predicts the next workload value as:

$$\hat{x}_{t+1} = f_{\text{Transformer}}(\mathbf{X}_t) \tag{3}$$

The model is trained by minimizing the Mean Squared Error (MSE) loss:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{4}$$

RMSE provides an error metric in the same unit as the workload signal:

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{5}$$

MAE measures the average absolute deviation between predictions and observations:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{6}$$

To evaluate autoscaling efficiency, the following metrics are used in the analysis of under-provisioning and over-provisioning:

$$\Theta_U(\%) = \frac{100}{T}\sum_{t=1}^{T}\frac{\max(r_t - p_t, 0)}{r_t} \tag{7}$$

$$\Theta_O(\%) = \frac{100}{T}\sum_{t=1}^{T}\frac{\max(p_t - r_t, 0)}{r_t} \tag{8}$$

where r_t and p_t denote required and allocated resources at time t, respectively. Lower values indicate better provisioning accuracy.

To translate the predicted workload into proactive scaling actions, the output of the Transformer-based forecasting model is integrated with a prediction-driven autoscaling policy. At each control interval, the forecasted CPU demand $\hat{x}_{t+1}$ is mapped to the required number of pods based on the per-pod resource capacity. Specifically, the desired pod count is computed as:

$$p_{t+1} = \left\lceil \frac{\hat{x}_{t+1}}{C_{\text{pod}} \times \rho} \right\rceil, \tag{9}$$

where C_{pod} denotes the CPU capacity allocated to a single pod and $\rho \in (0, 1]$ is a safety margin introduced to account for prediction uncertainty and workload variability.

Unlike reactive autoscaling mechanisms, this proactive policy initiates scale-up or scale-down actions before resource saturation occurs, thereby reducing scaling latency and preventing transient performance degradation. The computed pod count is communicated to Kubernetes through custom metrics or controller interfaces, enabling seamless enforcement via the control plane. This process completes a closed-loop

MAPE (Monitor-Analyze-Plan-Execute) cycle that continuously adapts resource allocation to workload dynamics while minimizing under-provisioning and over-provisioning.

A. End-to-End Workflow

The proposed proactive autoscaling framework operates as a continuous closed-loop control process aligned with the Monitor–Analyze–Plan–Execute (MAPE) paradigm [12], [17]. In the monitoring phase, real-time workload and resource utilization metrics, primarily CPU usage, are continuously collected from running application pods using Kubernetes monitoring services such as `cAdvisor`. These metrics capture the current computational demand of edge applications. The collected measurements are organized into historical input sequences using a fixed-length sliding window. Each sequence represents recent workload patterns and is normalized to ensure robustness across heterogeneous nodes. These sequences are then provided as input to the Transformer-based forecasting model, which predicts the upcoming workload for the next time step. By leveraging self-attention mechanisms, the model captures long-range temporal dependencies and anticipates future workload variations.

The predicted workload is subsequently translated into the required number of pods using the prediction-driven estimation logic described in Algorithm 1. This step determines the appropriate pod count while enforcing minimum availability constraints. The computed scaling decisions are then executed through Kubernetes APIs using the proactive autoscaling strategy defined in Algorithm 2. Scale-out operations are triggered ahead of workload surges, whereas scale-in actions are applied gradually using the Resource Removal Strategy (RRS) and Container Deployment Target (CDT) to prevent oscillations and maintain system stability.

By continuously repeating this control loop, the framework enables anticipatory and adaptive autoscaling that aligns resource provisioning with predicted workload demand, thereby improving responsiveness, reducing under- and over-provisioning, and maintaining stable performance under dynamic workloads.

B. Autoscaling and Adaptation Service

The proactive adaptation service operates in the *Planning* phase of the MAPE loop. Based on predicted workload values, it computes the optimal number of pods and executes scaling decisions during the *Execution* phase.

Algorithm 1 estimates the required number of pods based on the predicted workload and a target utilization threshold. It converts the forecasted workload into the required pod count while enforcing a minimum pod threshold to maintain baseline service availability during low-demand periods and prevent aggressive scale-in actions. Algorithm 2 implements proactive scaling using a Resource Removal Strategy (RRS) and a Container Deployment Target (CDT) to prevent oscillatory behavior caused by prediction uncertainty or short-term workload fluctuations. Instead of removing all surplus pods at once, only a fraction controlled by RRS is removed, improving system stability during low-demand phases while maintaining responsiveness to workload spikes. In our implementation, $RRS = 0.6$ and $CDT = 60$ seconds.

Algorithm 1 Prediction-Driven Pod Estimation

Require: Predicted workload $\hat{x}_{t+1}$, target utilization U_{target}, minimum pod count $Pods_{\min}$
Ensure: Estimated pod count $Pods_{t+1}$
1: $Pods_{t+1} \leftarrow \left\lceil \frac{\hat{x}_{t+1}}{U_{\text{target}}} \right\rceil$
2: $Pods_{t+1} \leftarrow \max(Pods_{t+1}, Pods_{\min})$
3: **return** $Pods_{t+1}$

Together, the two algorithms realize a prediction-driven autoscaling policy that balances responsiveness and stability. While scale-out actions are applied immediately to prevent under-provisioning and performance violations, scale-in decisions are applied conservatively. The RRS limits the fraction of surplus pods removed in a single control interval, and the CDT introduces a cooldown period between scaling actions to allow system stabilization.

Algorithm 2 Autoscaling with Resource Removal Strategy

Require: Current pod count $Pods_t$, estimated pod count $Pods_{t+1}$, RRS $\in [0, 1]$, CDT
Ensure: Updated pod allocation
1: **if** $Pods_{t+1} > Pods_t$ **then**
2: Scale out $(Pods_{t+1} - Pods_t)$ pods
3: **else if** $Pods_{t+1} < Pods_t$ **then**
4: $Pods_{\text{surplus}} \leftarrow (Pods_t - Pods_{t+1}) \times RRS$
5: Scale in $Pods_{\text{surplus}}$ pods
6: **end if**
7: Wait for CDT interval
8: **return** Updated pod count

By separating aggressive scale-out from gradual scale-in, the proposed adaptation mechanism ensures low latency during workload surges while avoiding excessive resource churn during declining demand. This design is particularly important in edge environments where provisioning delays, network variability, and resource heterogeneity can amplify instability. Overall, the proactive autoscaling strategy enables Kubernetes-native, forecast-aware resource management that reduces under-provisioning, limits over-provisioning, and improves system efficiency..

V. Dataset and Experimental Environment

Experiments were conducted on a Kubernetes-based edge testbed consisting of one master node and multiple worker nodes. The master node managed cluster coordination, scheduling, and autoscaling control, while worker nodes executed containerized application workloads. Each node was equipped with Intel Xeon processors, 32 vCPUs, and 64 GB RAM. To emulate realistic edge conditions, the cluster operated under a heterogeneous network environment with variable latency and bandwidth. Network variability was introduced using Linux traffic control (`tc`). Artificial latency between nodes ranged from 10–120 ms, while bandwidth limits between 10–100 Mbps simulated constrained edge-to-cloud links. In addition, selected worker nodes were configured with reduced

CPU quotas and limited pod capacities to represent resource-constrained edge devices. These settings enable the cluster to approximate practical edge deployment scenarios while maintaining experimental reproducibility. Kubernetes version 1.27 was used, and resource monitoring was performed using `cAdvisor`. The system was implemented using Python, Go, and Kubernetes `.yaml` configuration files. Docker was used for container management, and Kubernetes APIs enabled dynamic pod scaling. Forecasting models were implemented using TensorFlow 2.11 and PyTorch 1.12. Supporting libraries included NumPy and Pandas for preprocessing, the Kubernetes Python client and Prometheus API client for metric collection, APScheduler for periodic execution, and FastAPI for exposing forecasting models as RESTful services for real-time prediction and scaling decisions.

A. Workload Dataset Description

To evaluate proactive autoscaling under realistic conditions, we used the Bitbrains workload dataset from the Grid Workloads Archive (GWA-T-12) [20]. The dataset contains performance traces from 1,750 virtual machines (VMs) deployed in an enterprise datacenter operated by Bitbrains IT Services Inc. The traces are divided into two categories:

- **fastStorage**: Includes 1,250 VMs connected to high-performance Storage Area Network (SAN) devices, mainly hosting compute-intensive workloads.
- **Rnd**: Contains 500 VMs connected to either fast SAN or slower Network Attached Storage (NAS) devices, primarily supporting auxiliary and management services.

Each VM trace is organized as timestamped observations where columns are separated by ";\t". The metrics include CPU cores provisioned, CPU capacity (MHz), CPU usage (MHz and %), memory provisioned and used (KB), disk read/write throughput (KB/s), and network received/transmitted throughput (KB/s). Timestamps are recorded as milliseconds since the Unix epoch. These traces exhibit diverse workload characteristics such as bursty CPU demand, idle periods, and sudden workload spikes, making them suitable for evaluating forecasting accuracy and autoscaling robustness.

- **Workload Generation:** VM-level CPU utilization traces were transformed into container-level workloads by normalizing CPU usage values and replaying them as container demand patterns at fixed monitoring intervals. Each VM trace segment was mapped to a group of containerized tasks representing microservice workloads deployed on Kubernetes pods. The workload generator issued requests according to the temporal patterns observed in the traces, enabling realistic reproduction of bursty demand behavior. During experiments, workloads were replayed using the same monitoring interval employed by the autoscaling controller. The Kubernetes deployment initially started with a baseline number of pods, and the autoscaling module dynamically adjusted the number of replicas based on predicted CPU demand. This setup translates VM-level workload traces into container-level resource demand for evaluating proactive autoscaling strategies.
- **Model Training and Configuration:** The Transformer-based forecasting model was trained using historical CPU utilization traces. Input sequences were constructed using a sliding window of length $W = 20$. The forecasting model adopts a stacked Transformer encoder architecture with four encoder layers, each using multi-head self-attention with eight attention heads. The embedding dimension is $d_{model} = 512$, and the feedforward network dimension is $d_{ff} = 2048$. Sinusoidal positional encoding is applied to preserve temporal ordering, while residual connections and layer normalization stabilize training.
  The final encoded representation is passed through two fully connected layers to predict CPU workload for the next time step. Training used the Adam optimizer with a learning rate of 10^{-4}, batch size of 64, and up to 100 epochs with early stopping. A dropout rate of 0.2 was applied to improve generalization.
  For fair comparison, baseline models including LSTM, BiLSTM, and CNN-LSTM were trained using the same dataset, batch size, and optimization settings. Each model was evaluated across multiple runs with different random seeds, and the reported results correspond to averaged metrics.

Overall, this experimental setup enables a realistic and reproducible evaluation of the proposed proactive autoscaling framework under dynamic edge conditions. By combining real workload traces, Kubernetes-native deployment, and consistent training configurations, the experiments provide a fair assessment of both forecasting accuracy and system-level autoscaling performance. Although the Bitbrains dataset originates from enterprise cloud workloads, its bursty utilization patterns and workload variability make it well suited for evaluating autoscaling strategies in dynamic microservice environments.

VI. Results and Discussion

This section evaluates the effectiveness of **PredictScale**, the proposed Transformer-based proactive autoscaling framework, using real workload traces from the Bitbrains dataset. The evaluation considers both forecasting accuracy and system-level autoscaling performance under bursty edge workloads.

A. Workload Forecasting Accuracy

We first evaluate the accuracy of workload prediction models using standard regression metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). Accurate workload forecasting is critical for proactive autoscaling, as prediction errors directly affect pod provisioning decisions. Table II compares the performance of LSTM, BiLSTM, CNN-LSTM, and the proposed Transformer-based model. The results show that PredictScale consistently achieves the lowest error across all metrics, indicating its superior ability to capture both short-term workload fluctuations and long-term temporal trends.

TABLE II
WORKLOAD FORECASTING ACCURACY COMPARISON

Model	MSE	RMSE	MAE
LSTM	0.0541	0.2326	0.1913
BiLSTM	0.0467	0.2161	0.1785
CNN-LSTM	0.0395	0.1987	0.1632
PredictScale	**0.0293**	**0.1711**	**0.1460**

B. Provisioning Efficiency

Accurate forecasting must translate into effective resource provisioning (Table III). We therefore evaluate autoscaling performance using under-provisioning (Θ_U) and over-provisioning (Θ_O) metrics, which measure resource shortages and excess allocation, respectively. Lower Θ_U indicates fewer SLA violations, while lower Θ_O reflects reduced resource waste.

TABLE III
PROVISIONING EFFICIENCY COMPARISON

Autoscaling Method	Θ_U (%)	Θ_O (%)
Static Provisioning	18.6	34.2
Reactive HPA	11.4	22.8
LSTM-based Scaling	9.3	18.6
CNN-LSTM-based Scaling	7.8	15.4
PredictScale	**4.9**	**11.2**

C. Pod Utilization Analysis

Figure 2 illustrates pod utilization over time under bursty workloads for different autoscaling strategies. The horizontal axis represents sequential workload sampling intervals derived from the Bitbrains traces, where each time step corresponds to the fixed monitoring interval (30 seconds) used by the autoscaling controller during workload replay. The plotted curves correspond to representative workload segments extracted from the traces to illustrate autoscaling behavior under dynamic demand conditions. Static provisioning results in consistently low utilization due to fixed resource allocation, leading to significant idle capacity. Reactive HPA improves utilization by scaling based on observed CPU thresholds; however, delayed responses to workload spikes still cause short-term inefficiencies. In contrast, **PredictScale** maintains higher utilization by anticipating workload changes and provisioning resources in advance. This proactive behavior allows pods to be allocated closer to actual demand, reducing both underutilization and excessive scale-out events. Although the difference between PredictScale and reactive HPA appears moderate, the trend aligns with the reductions in under-provisioning (Θ_U) and over-provisioning (Θ_O) shown in Table III, indicating more balanced resource allocation.

D. Operational Cost Analysis

Figure 3 compares the cumulative operational cost incurred by different autoscaling approaches. The horizontal axis represents sequential workload sampling intervals derived from the Bitbrains traces, where each time step corresponds to the fixed monitoring interval (30 seconds) used by the autoscaling controller during workload replay. Static provisioning results in the highest cost due to persistent overprovisioning. Reactive HPA reduces cost through dynamic scaling, but frequent scaling actions introduce inefficiencies under rapidly changing workloads. PredictScale achieves the lowest overall cost by proactively aligning resource allocation with predicted demand, reducing unnecessary pod deployments while maintaining performance.

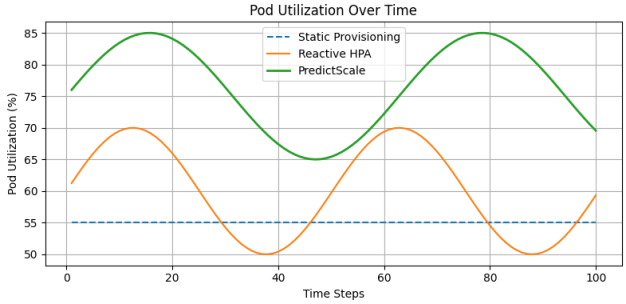

Fig. 2. Pod utilization over time (sampling interval: 30 seconds) comparing static provisioning, reactive HPA, and PredictScale under representative bursty workload conditions derived from the Bitbrains traces.

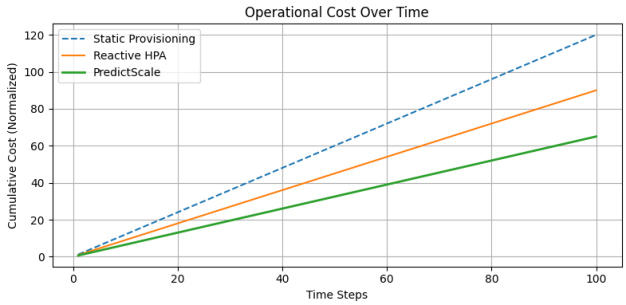

Fig. 3. Cumulative operational cost over time (sampling interval: 30 seconds) comparing different autoscaling strategies under representative workload conditions derived from the Bitbrains traces.

E. Proactive Autoscaling Behavior and Latency Impact

Figure 4 shows the end-to-end autoscaling behavior and its impact on application latency. The horizontal axis represents sequential workload sampling intervals derived from the Bitbrains traces, where each time step corresponds to the fixed monitoring interval (30 seconds) used by the autoscaling controller during workload replay. Reactive HPA triggers scaling only after utilization exceeds predefined thresholds, which leads to delayed responses and temporary performance degradation during workload spikes. PredictScale mitigates this issue by initiating scaling actions ahead of demand surges, resulting in smoother pod transitions and lower latency. This proactive behavior reduces scaling oscillations and improves application responsiveness under bursty workloads.

Overall, the results demonstrate that PredictScale consistently outperforms reactive and recurrent-model-based autoscaling approaches. Improved forecasting accuracy enables

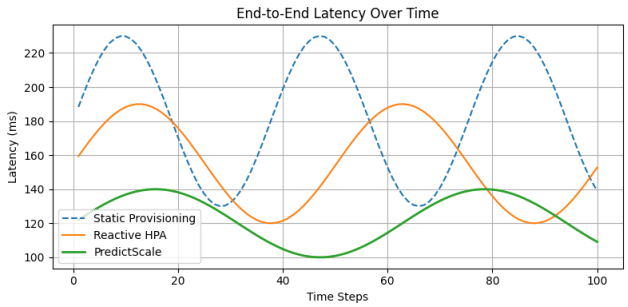

Fig. 4. End-to-end latency over time (sampling interval: 30 seconds) comparing reactive and proactive autoscaling under representative workload conditions derived from the Bitbrains traces.

proactive resource provisioning, leading to higher pod utilization, lower operational cost, and reduced latency. These findings suggest that integrating Transformer-based workload prediction with Kubernetes-native autoscaling mechanisms can significantly improve resource management in dynamic edge-oriented workloads without modifying the underlying orchestration platform.

Recent time-series architectures such as Informer, Autoformer, and FEDformer have shown strong performance for long-sequence prediction tasks. However, these models typically introduce higher computational complexity and are mainly evaluated on large-scale sequence datasets. In contrast, this work focuses on proactive autoscaling in Kubernetes-based cloud–edge environments, where forecasting models must operate efficiently within lightweight control loops. Therefore, we evaluate widely used forecasting models in autoscaling systems (LSTM, BiLSTM, CNN-LSTM) together with the reactive Kubernetes HPA baseline. Exploring advanced Transformer variants within autoscaling frameworks remains an interesting direction for future work.

VII. Conclusion

This paper presented **PredictScale**, a Transformer-based proactive autoscaling framework for Kubernetes-based cloud–edge microservice environments. By forecasting future workload demand and capturing temporal dependencies in resource utilization patterns, the proposed approach enables proactive resource provisioning instead of reactive scaling. Experimental results show that PredictScale achieves higher forecasting accuracy than recurrent baselines, resulting in more balanced resource provisioning, improved pod utilization, lower operational cost, and reduced latency compared with static provisioning and reactive HPA. These findings demonstrate that integrating Transformer-based workload prediction with Kubernetes-native autoscaling mechanisms can significantly enhance resource management in dynamic edge environments. Future work will extend the framework to incorporate multivariate workload metrics, larger-scale distributed deployments, and advanced time-series forecasting architectures such as Informer, Autoformer, and related Transformer variants.

Funding: This work is supported by the Institutions of Eminence (IoE) Scheme of Banaras Hindu University, Varanasi, India under Dev. Scheme No. 6031.

References

[1] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–33, 2018.

[2] M. Goudarzi, M. Palaniswami, and R. Buyya, "Scheduling iot applications in edge and fog computing environments: A taxonomy and future directions," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–41, 2022.

[3] N. Dhameliya, B. Patel, S. Maddula, and K. Mullangi, "Edge computing in network-based systems: enhancing latency-sensitive applications," *American Digits: Journal of Computing and Digital Technologies*, vol. 2, no. 1, pp. 1–21, 2024.

[4] B. Kumar, M. Singh, A. Verma, and P. Verma, "Optimal cloudlet selection in edge computing for resource allocation," *SN Computer Science*, vol. 4, no. 6, p. 745, 2023.

[5] V. R. Verma, Pushkar, B. kumar, A. Verma, V. Sharma, and P. K. Tripathi, "An extensive investigation on lyapunov optimization-based task offloading techniques in multi-access edge computing," *SN Computer Science*, vol. 6, no. 6, p. 603, 2025.

[6] Q. Zhao, F. Chen, and K. Li, "Autoscaling microservices in kubernetes: A survey," *ACM Computing Surveys*, vol. 54, no. 8, pp. 1–36, 2021.

[7] B. Jeong and Y.-S. Jeong, "Autoscaling techniques in cloud-native computing: A comprehensive survey," *Computer Science Review*, vol. 58, p. 100791, 2025.

[8] B. Kumar, A. Verma, and P. Verma, "Critical insights into runtime scheduling, image, storage, and networking challenges in modern kubernetes environments," *Computer Science Review*, vol. 59, p. 100851, 2026.

[9] A. A. Khaleq and I. Ra, "Intelligent autoscaling of microservices in the cloud for real-time applications," *IEEE access*, vol. 9, pp. 35464–35476, 2021.

[10] N. Wu and Y. Xie, "A survey of machine learning for computer architecture and systems," *ACM Computing Surveys (CSUR)*, vol. 55, no. 3, pp. 1–39, 2022.

[11] Z. Peng, B. Tang, W. Xu, and et al., "Microservice auto-scaling algorithm based on workload prediction in cloud-edge collaboration environment," in *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom)*, pp. 608–615, IEEE, 2023.

[12] B. Kumar, A. Verma, P. Verma, and A. Bennour, "Optimizing resource allocation in cloud-native applications through proactive autoscaling with the informerautoscale model.," *The Journal of Supercomputing*, vol. 81, no. 9, p. 1077, 2025.

[13] H. Zhou, S. Zhang, and J. Peng, "Informer: Beyond efficient transformer for long sequence time-series forecasting," *Proceedings of AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11106–11115, 2021.

[14] H. Wu, J. Xu, and J. Wang, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22419–22430, 2021.

[15] T. Zhou, Z. Ma, and Q. Wen, "Fedformer: Frequency enhanced decomposed transformer for long-term time-series forecasting," *International Conference on Machine Learning (ICML)*, pp. 27268–27286, 2022.

[16] B. J. Gort, G. M. Kibalya, M. A. Serrano, and A. Antonopoulos, "Forecasting trends in cloud-edge computing: Unleashing the power of attention mechanisms," *IEEE Communications Magazine*, 2024.

[17] B. Kumar, A. Verma, and P. Verma, "A multivariate transformer-based monitor-analyze-plan-execute (mape) autoscaling framework for dynamic resource allocation in cloud environment," *Computing*, vol. 107, no. 3, p. 69, 2025.

[18] B. Kumar, A. Verma, and P. Verma, "Optimizing resource allocation using proactive scaling with predictive models and custom resources," *Computers and Electrical Engineering*, vol. 118, p. 109419, 2024.

[19] B. Kumar, A. Verma, and P. Verma, *Modern Kubernetes: From Core Concepts to Intelligent Autoscaling for Cloud Applications*. Springer Nature, 2026.

[20] "gwa-bitbrains — kaggle.com." https://www.kaggle.com/datasets/gauravdhamane/gwa-bitbrains. [Accessed 25-12-2025].