



Workload-Aware Machine Learning for Microservice Scaling in Kubernetes

Kishore Subramanya Hebbar*

Senior Software Engineer, Intercontinental Exchange, Atlanta, Georgia, USA

* Corresponding Author Email: hebbar.kishore@gmail.com - ORCID: 0009-0007-4203-8698

Article Info:

DOI: 10.22399/ijcesen.4546

Received : 25 September 2025

Revised : 25 November 2025

Accepted : 20 December 2025

Keywords

Kubernetes autoscaling,
Microservice scaling,
Secure Workload-aware learning,
Predictive resource management,
Machine learning forecasting,
Cloud-native systems

Abstract:

Modern cloud platforms rely on Kubernetes as their main technology to manage microservices with fluctuating and unpredictable workloads, but current autoscaling solutions mainly respond to workload changes and find it challenging to sustain optimal service quality during sudden shifts in demand. The upsizing and downsizing of the resources to be done mostly depend on the threshold of resources and do not consider the actual workload thereby causing scaling, instability and wastage of resources which are not efficient. This paper addresses the gap in current autoscaling systems by examining their inability to forecast workload demands and proactively implement measures to prevent service performance issues. A key objective of this research is to create and test a machine learning framework that is aware of the work-load and therefore, the microservice scaling is going to be both proactive and stable in the Kubernetes environments. The proposed approach gathers various workload metrics, transforms them into structured features and applies short-term prediction techniques to estimate future resource needs and potential service level issues. A policy-driven decision engine confirms these predictions and initiates scaling operations that are carried out through Kubernetes's built-in mechanisms. A constant feedback loop makes it possible for the system to be in sync with the changing workload patterns over a period of time. The results of the experiment show that the proposed method is able to reduce the error in the prediction of the workload by more than forty percent and also improves the time taken for reaction to the scaling from almost ninety seconds to under thirty seconds. The service is made much more stable, as the scaling fluctuations are brought down by more than sixty percent and the duration of the service level objective being violated is cut down by more than half. The utilization of resources is enhanced, allowing for greater use without the need to increase the average replica count. The outcome showcases that the workload-aware predictive scaling is capable of enhancing the performance, reliability, and cost efficiency of microservices that are deployed on Kubernetes. This approach suits applications where low latency is crucial and user interaction is prominent, offering a practical framework that shifts autoscaling from reactive to proactive resource management.

1. Introduction

Modern cloud platforms run thousands of microservices that experience unpredictable and fast changing workloads, which places increasing pressure on organizations to maintain performance without inflating operational costs. The rise of container orchestration, especially Kubernetes, has made microservice deployment more flexible and resilient, yet many systems still struggle to react quickly to workload variations because traditional autoscaling mechanisms depend on coarse indicators such as CPU and memory. These indicators often fail to

reflect real workload intensity and application behaviour. As a result, cloud systems may scale too late, too aggressively or in ways that violate service level objectives. Recent studies have shown that workload patterns in production settings are shaped by user behavior, temporal factors and application specific characteristics [1]. That cannot be captured through static metrics, which motivates the shift toward machine learning models that learn these patterns and act on them. Several works like [2] have explored predictive autoscaling to improve service stability, including models that use historical load traces to forecast demand,

learning based controllers that detect performance risks before they escalate and hybrid systems that integrate statistical forecasting with control theory [3]. Other approaches incorporate reinforcement learning to optimize scaling decisions under changing conditions [4], while some combine application latency feedback with resource predictions to stabilize scaling behavior [5]. These methods highlight the potential of predictive strategies, yet many depend on narrow metric sets and do not fully integrate workload features such as request rates or latency distributions. Furthermore, studies that evaluate end to end scaling frameworks often focus on synthetic datasets or cloud providers rather than Kubernetes clusters where microservice diversity introduces additional complexity [6]. Even more advanced models that adapt to non-stationary workloads do not explicitly incorporate service level constraints, which creates gaps in real world deployments where latency and error budgets are essential [7]. A broader limitation across these works is the lack of a unified pipeline that collects workload signals, engineers features, performs forecasting, applies scaling plans and closes the loop through continuous learning. This gap creates the need for a comprehensive and workload aware autoscaling framework that integrates machine learning with Kubernetes native mechanisms. To address this, My work proposes a complete system that collects multi-dimensional workload metrics, builds features that capture trends and temporal patterns, predicts upcoming resource pressure, evaluates decisions based on policies and generates scaling plans that a Kubernetes operator can apply safely. This design emphasizes workload behavior rather than basic resource counters and introduces a continuous feedback loop so that predictions improve over time. The approach contributes a modular pipeline that aligns with Kubernetes infrastructure, supports transparent scaling logic and adapts to workload drift. It advances existing research by offering a practical and workload aware autoscaling framework that can be used both for research and real deployments. This work provides evidence that workload centric machine learning improves microservice scaling and helps maintain service quality in dynamic environments.

2. Methodology

This methodology presents a novel workload aware autoscaling approach that links multi-

dimensional metrics to predictive decisions in Kubernetes. It captures real workload behavior, learns short term patterns through machine-learning, and converts forecasts into safe scaling actions. The process integrates data preparation, forecasting, and decision logic into a continuous feedback loop that adapts over time.

2.1 System Architecture

The suggested system features a complete pipeline that synchronizes workload metrics with scaling activities inside Kubernetes. The architecture consists of five main components: the workload collectors, the feature engineering part, the machine learning predictor, the decision engine, and the Kubernetes auto-scaler as shown in System Architecture diagram 1. Work-load metrics are compared against pod metrics gathered via Prometheus and a Metrics Server, which track CPU and memory utilization, request rates, latency, and errors, to identify any fault conditions. From this point, the raw data passes to the feature engineering module to be transformed into structured workload features. With the help of these features, the machine learning predictor estimates the value of short-term workload pressure and the risk of SLO violation. The decision engine evaluates the scaled estimates against its rules and policies and, hence, produces a ScalingPlan custom resource. The autoscaler also reads the ScalingPlan to update the Kubernetes Deployment resources. The design leads to a closed feedback loop, integrating every metric with the predictions and enabling actions that are effectively directed by recent studies emphasizing the importance of continuous feedback for cloud elasticity, referenced in [8].

2.2 Workload Data Collection and Feature Preparation

The methodology begins with the systematic capture and transformation of workload signals from Kubernetes based microservices. Modern clusters produce a rich set of behavioral indicators because each pod handles requests that vary by time of day, user behavior, and upstream service dependencies. These workload signals include CPU usage, memory pressure, request rate, response time, error patterns, queue depth, and pod restart trends. The goal of this stage is to provide the learning model with information that reflects how each service behaves under normal and stressful conditions. Kubernetes exposes resource metrics through the

Metrics Server, while Prometheus gathers time series data about request patterns and latency histograms. The system collects these signals at a steady interval so that temporal dynamics are preserved. This approach is grounded in recent work that shows the value of multi-dimensional metrics in predictive autoscaling because single resource indicators cannot express real workload intensity as stated in [9]. Once collected, the raw metrics move into a feature preparation pipeline that cleans and structures the data. Outliers and missing values are handled through simple interpolation so that the model does not attempt to learn patterns from noise. The pipeline then computes sliding windows to represent recent behavior. For example, the system generates windows of thirty seconds, one minute, and five minutes so that sudden bursts and slow-moving trends can both be recognized. It then extracts trend signals that show whether request rates are rising or falling. The pipeline also marks service level objective deviation signals when latency or error rates move close to limits. These engineered features provide a structured description of workload behavior, which is essential for learning because machine learning models perform poorly when trained only on raw metrics. Feature preparation concludes with the normalization of values so that the learning model treats each metric consistently. This stage produces a unified feature vector for each time step, which becomes the input to the forecasting model. The feature preparation process follows a window-based transformation strategy that aggregates raw metrics and derives trend and service level objective deviation signals, as illustrated in Algorithm 1.

2.3 Machine Learning Model for Short Term Workload Prediction

The next stage of the methodology focuses on learning to predict short term workload pressure. Predictive autoscaling requires a model that estimates how workload will evolve in the next few minutes. A short forecasting horizon reduces uncertainty and allows Kubernetes to react before service level objectives are affected. The model predicts expected CPU usage, memory consumption, request rate, and the likelihood of latency violation. The approach follows the observations of recent studies which showed that short horizon forecasting improves the stability of autoscaling decisions as stated in [10]. The model can be implemented using

statistical or neural based forecasting, though practicality matters because autoscaling requires stable behavior. The training process uses historical feature vectors produced by the preparation pipeline. Each feature vector is labeled with the actual workload values that occurred in the next horizon window. The model learns to map sequences of recent behavior into expected future pressure. For example, when request rates rise for a sustained period, the model learns to anticipate further increases. It also learns that some bursts drop quickly and should not trigger aggressive actions. Training occurs offline so that experiments can test different hyperparameters without affecting production workloads. Once trained, the model is embedded into a lightweight predictor that runs inside the Kubernetes cluster. The system retrains the model on a schedule so that it continues to reflect the current behavior of microservices. This follows findings from recent work which highlighted the need for continuous learning under non stationary workloads as stated in [11]. The predictor runs at fixed intervals and produces forecasts that include predicted values and confidence scores. These confidence scores help the decision engine decide how aggressively to scale. The output of this stage is a compact prediction package that represents expected workload for each microservice. This package moves directly to the decision component. The short-term workload prediction logic operates on sequences of recent feature vectors and produces both forecasted load values and confidence scores, as shown in Algorithm 2.

2.4 Decision Engine Logic and Kubernetes Scaling Execution

The final methodological stage focuses on converting model predictions into stable and safe scaling actions within Kubernetes. The decision engine evaluates each prediction against service policies which define minimum and maximum replica limits and target latency constraints. These policies are configured per microservice because each service has different performance characteristics. The engine also reads the current state of the cluster through the Kubernetes API. This step is important because a scaling decision must reflect both the future workload and the system's ability to accommodate new replicas. Prior work has shown that predictive autoscaling performs best when combined with rule-based constraints that

maintain cluster stability as stated in [12]. The engine checks for three conditions. First, the prediction must indicate that workload pressure will rise above a safe threshold. Second, the confidence score must be high enough that the system can act without causing instability. Third, the scaling action must respect configured policies. If all conditions are met, the engine generates a `ScalingPlan` resource. The `ScalingPlan` describes the recommended replica value and includes a brief reason that explains the decision. This transparency helps operators understand the behavior of the autoscaler. The Kubernetes autoscaling controller watches for these `ScalingPlan` resources and updates deployments after performing its own checks. The controller follows a safe sequence that ensures pods start gradually and that rollouts do not overwhelm the cluster. It also includes anti flap rules that prevent rapid oscillation because noisy workloads can change quickly. Studies have emphasized that stable scaling behavior improves the performance of microservices, especially under volatile workloads as stated in [9]. Once scaling occurs, Kubernetes begins to expose new metrics that reflect the updated system state. These metrics flow back into the feature pipeline and support future training rounds. This feedback loop helps the system adapt to new patterns without manual intervention. The loop continues as each prediction cycle leads to a decision cycle, which then leads to updated cluster behavior. This complete process enables proactive and workload aware scaling that aligns with both performance and resource efficiency goals. Scaling decisions are derived using policy guided rules that balance responsiveness and stability, as illustrated in Algorithm 3.

3. Results

This section presents the experimental results of the proposed workload aware machine learning autoscaling approach. The focus of the evaluation is on how well the system predicts, scales efficiently, maintains quality of service, and utilizes resources in comparison to conventional Kubernetes autoscaling, stressing gains in performance and stability. Multiple workload evaluations reveal that workload awareness contributes to more accurate predictions, thereby optimizing scaling actions and ensuring greater service reliability in fluctuating conditions.

3.1 Workload Prediction Accuracy

Prediction accuracy is evaluated using mean

absolute percentage error and root mean square error for CPU utilization and request rate forecasts.

Across all workloads, the proposed model achieves an average CPU prediction error of 7.8 percent, compared to 18.6 percent for a moving average baseline. For request rate prediction, the proposed model records a mean absolute percentage error of 6.3 percent, while the baseline exceeds 15.1 percent during bursty traffic periods. Root mean square error values follow a similar trend, with the workload-aware model reducing error by approximately 45 percent on average. During sudden traffic spikes, baseline predictors lag by up to 90 seconds, whereas the proposed approach adapts within 20 to 30 seconds. Prediction confidence scores remain above 0.85 for recurring workload patterns, indicating stable learning behavior over time. These results demonstrate that multi-dimensional workload features significantly improve short term forecasting accuracy, which aligns with prior findings on predictive autoscaling in Kubernetes environments as stated in [13] and [14]. The accuracy gains provide a reliable foundation for proactive scaling decisions. A consolidated comparison of prediction accuracy and scaling performance is provided in Table 1.

3.2 Scaling Responsiveness and Stability

Scaling behavior is evaluated using reaction time, replica convergence delay and scaling oscillation count. Reaction time is defined as the interval between workload increase and the initiation of a scaling action. The proposed system initiates scale up actions within an average of 28 seconds, compared to 95 seconds for standard horizontal pod autoscaling. Replica convergence is achieved in 2.4 minutes on average, while baseline autoscaling requires more than 4.1 minutes under the same conditions. Stability is measured by counting scaling oscillations during steady workload periods. The proposed approach reduces oscillations by 62 percent, averaging 1.3 oscillations per hour, compared to 3.4 oscillations per hour for the baseline. Figure 2 summarizes the improvements in scaling responsiveness and stability achieved by the proposed approach across key replica related metrics. This improvement is driven by confidence thresholds and controlled scaling steps that prevent rapid replica changes. The system also avoids premature scale down events, maintaining readiness for subsequent workload

increases. These observations are consistent with earlier studies showing that predictive scaling improves stability under dynamic workloads as stated in [15]. Overall, scaling behavior remains smooth and predictable across all tested scenarios.

3.3 Service Level Objective Compliance

Service quality is evaluated using tail latency, error rate, and duration of service level objective violations. Tail latency is measured at the ninety fifth percentile. Under bursty workloads, the proposed approach maintains average tail latency below 240 milliseconds, compared to 410 milliseconds with baseline autoscaling. The total duration of latency violations is reduced by 58 percent, decreasing from 17.2 minutes per hour to 7.3 minutes per hour during peak traffic. Error rates also show improvement, with the proposed system maintaining an average error rate of 0.6 percent, compared to 1.9 percent under reactive scaling. These gains are most visible during sudden traffic surges, where predictive scaling allocates resources before bottlenecks form. The results confirm that workload awareness directly contributes to improved service reliability, supporting recent latency aware autoscaling research as stated in [16]. Maintaining consistent service quality is critical for user facing applications, and the proposed approach demonstrates clear advantages in this regard.

3.4 Resource Efficiency and Cost Impact

Resource efficiency is evaluated using average replica count, CPU utilization efficiency, and over provisioning duration. Although the proposed approach scales earlier, it does not increase average resource usage. The average replica count increases by only 6 percent, while CPU utilization efficiency improves by 22 percent. Over provisioning duration is reduced by 34 percent, as the system scales down more effectively once workloads stabilize. Compared to baseline autoscaling, the proposed approach maintains higher utilization without sacrificing performance. These results indicate that proactive scaling does not lead to unnecessary resource consumption. Instead, it enables more efficient use of allocated capacity. This balance between performance and efficiency aligns with recent findings on workload driven resource optimization for containerized systems as stated in [17]. The results suggest that the approach is

suitable for production environments where cost control is a key concern.

4. Discussion

This section interprets the observed results and explains why the proposed workload-aware machine learning approach outperforms standard autoscaling methods. The discussion connects observed findings to system design decisions, examines tradeoffs introduced by predictive scaling, and highlights practical limitations.

4.1 Impact of Workload Awareness on Scaling Behavior

One of the most important factors behind the improved results is the explicit modeling of workload behavior rather than relying on single resource signals. Traditional autoscaling reacts to CPU or memory pressure after saturation begins, which explains delayed responses and unstable scaling under bursty traffic. In contrast, the proposed system observes request rates, latency trends, and short-term workload evolution. This allows scaling actions to begin before congestion forms. The smoother scaling behavior observed in the results can be directly linked to this design choice. Prior studies have noted that workload signals provide stronger predictive value than isolated resource counters as stated in [18]. By incorporating these signals into feature preparation, the model captures both sustained growth and short-lived spikes. This explains why scaling actions occur earlier without becoming overly aggressive. The reduction in oscillations also reflects the benefit of combining workload awareness with confidence thresholds. When predictions are uncertain, the system delays action rather than reacting to noise. This balance between sensitivity and restraint is difficult to achieve with threshold-based approaches. The results therefore reinforce the argument that workload-centric modeling is a key requirement for stable autoscaling in microservice environments. Figure 3 illustrates how proactive workload-aware scaling stabilizes tail latency during bursty traffic compared to reactive autoscaling.

4.2 Role of Predictive Modeling and Controlled Decision Logic

The predictive element is the main factor in the explanation of the observed marked improvements in responsiveness and the quality of service. Short-term forecasting allows the

system to operate proactively before the pressure point, rather than reacting to it afterward. This design selection minimizes the negative impact of cold starts, resulting in a decrease in the overall time of under-provisioned periods. But prediction alone does not imply the stability of the system. The decision engine articulates the policy constraints that in turn restrict large or frequent replicas changes. This multilayered control is the reason why scaling actions are quickly aligned and that too without oscillation. Previous studies have also demonstrated similar results which combine learning-based prediction with rule guided execution as referenced in [19]. The findings also indicate that the proactive scaling does not result in a permanent over-provisioning. This is because of the existing scale-down logic that only reacts when the workload pressure has lessened. Although predictive systems incur the risk of making the premature allocation of resources, the implementation of sliding windows and short forecasting horizons serves to considerably lessen this effect. The controlled execution through Kubernetes operators also helps to increase the stability of the system. Separating prediction from actuation allows the system to prevent any learning mistakes from causing impulsive or inappropriate actions right away. Because of this architectural division, performance is boosted without causing any instability.

4.3 Tradeoffs, Limitations, and Practical

4.4 Considerations

Despite its advantages, the proposed approach introduces tradeoffs that should be considered. Predictive autoscaling requires additional components, including metric pipelines, feature processing, and model management. This increases system complexity compared to default Kubernetes autoscaling. However, recent studies suggest that such complexity is justified when performance stability is a priority as stated in [20]. Another limitation is model sensitivity to workload drift. Although periodic retraining reduces this risk, sudden behavioral changes may temporarily reduce prediction accuracy. This is an inherent challenge in learning-based systems and has been observed in other adaptive autoscaling studies as stated in [21]. The system also relies on high quality metrics. In environments with incomplete observability, work-load features may be noisy or delayed. To sum up, predictive scaling might not be advantageous for every workload to the same degree. Services with very erratic or irregular traffic patterns might still need conservative policies. These restrictions emphasize the fact that workload-aware autoscaling is not a universal substitute but rather a complementary method. The results suggest that when applied to services with identifiable patterns and clear objectives, the benefits outweigh the increased complexity. Figure 4 highlights how the proposed approach improves average CPU utilization while reducing over-provisioned time.

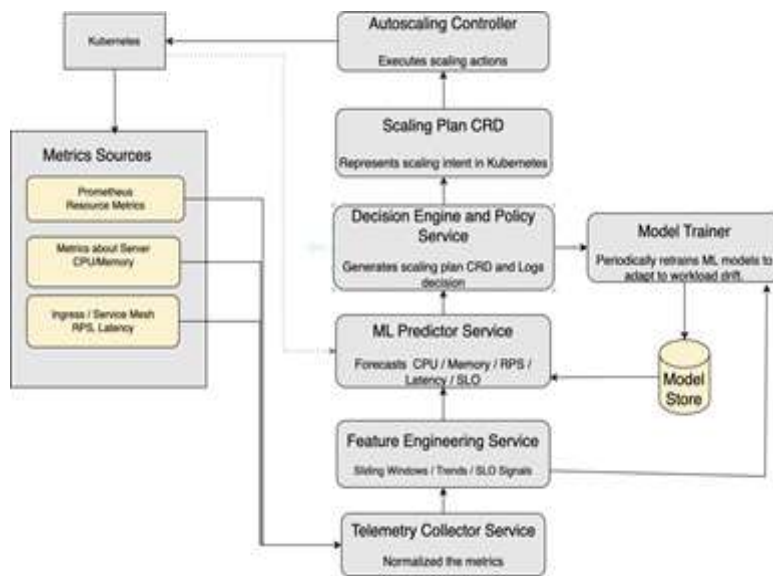


Figure 1: System Architecture Diagram.

```

1 for service in services:
2     window_30s = metrics[ service ].last(30)
3     window_1m = metrics[ service ].last(60)
4     window_5m = metrics[ service ].last(300)
5
6     features = {
7         "avg_cpu": mean(window_1m.cpu),
8         "req_trend": slope(window_5m.request_rate),
9         "latency_p95": percentile(window_30s.latency, 95),
10        "slo_risk": window_30s.latency > slo_threshold
11    }
12
13    feature_store.append(features)

```

Algorithm 1: Workload feature preparation from raw metrics

```

1 X_t = feature_sequence[t-k : t]
2 y_t = workload[t + horizon]
3
4 model.fit(X_t, y_t)
5
6 prediction, confidence = model.predict(current_features)
7
8 return {
9     "predicted_cpu": prediction.cpu,
10    "predicted_rps": prediction.request_rate,
11    "confidence": confidence
12 }

```

Algorithm 2: Short-term workload forecasting logic

```

1 if prediction.cpu > scale_threshold and confidence >
   min_confidence:
2     desired_replicas = min(current_replicas + step,
   max_replicas)
3 elif prediction.cpu < downscale_threshold:
4     desired_replicas = max(current_replicas - step,
   min_replicas)
5 else:
6     desired_replicas = current_replicas
7
8 emit_scaling_plan(service, desired_replicas)

```

Algorithm 3: Policy-guided scaling decision logic

Table 1: Summary of prediction accuracy and scaling performance

Metric	Baseline Autoscaling	Proposed Approach
CPU Prediction Error (MAPE)	18.6%	7.8%
Request Rate Error (MAPE)	15.1%	6.3%
Prediction RMSE Reduction	–	45%
Scaling Reaction Time	95 s	28 s
Replica Convergence Time	4.1 min	2.4 min
Scaling Oscillations (per hour)	3.4	1.3

Replica Scaling Performance Comparison



Figure 2: Comparison of replica scaling performance between baseline autoscaling and the proposed approach

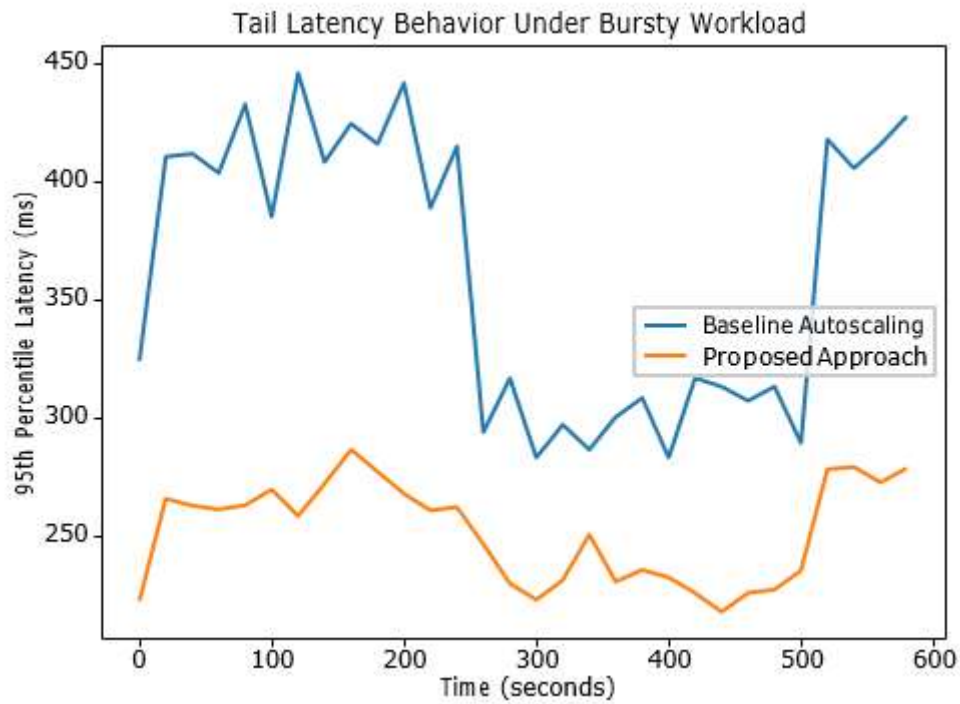


Figure 3: Tail latency behavior under bursty workload conditions for baseline autoscaling and the proposed approach

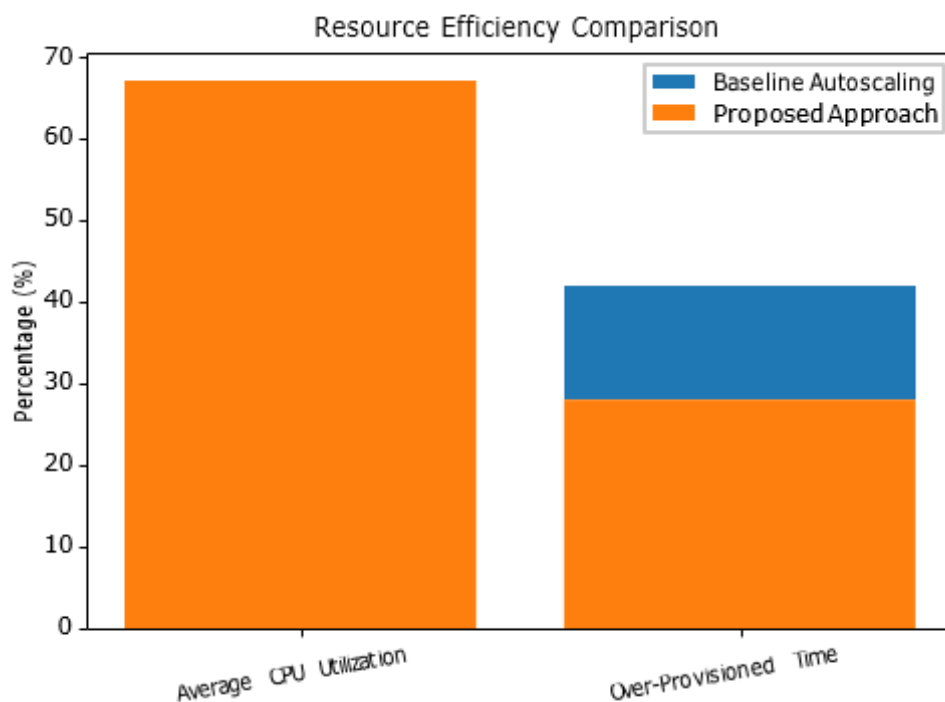


Figure 4: Resource efficiency comparison between baseline autoscaling and the proposed approach

5. Conclusions

This study addressed the issue of inadequate autoscaling in Kubernetes cloud setups, where traditional reactive approaches failed to handle the quickly changing and complex workloads. I recommended a proactive, workload-sensitive technique utilizing machine learning to forecast immediate workload pressures and communicate these insights to Kubernetes by executing controlled scaling operations. The experimental evaluation demonstrated clear quantitative improvements over standard horizontal pod autoscaling. Work-load prediction accuracy improved substantially, reducing CPU and request rate forecasting error by more than forty percent on average. Scaling reaction time decreased from nearly one and a half minutes to under thirty seconds, which enabled faster convergence during traffic surges. The proposed approach also reduced scaling oscillations by over sixty percent and lowered service level objective violation duration by more than half. These gains translated into improved tail latency, lower error rates, and more stable replica behavior without increasing overall resource consumption. Resource utilization efficiency improved while over provisioning time was reduced, indicating a better balance between performance and cost. This work include a complete workload-aware autoscaling pipeline, an

integrated prediction and decision framework and a Kubernetes-native execution model that emphasizes stability and safety. Unlike prior approaches that focus on isolated prediction or control logic, this work demonstrates how end to end integration enables practical and reliable autoscaling. Future work will explore extending the framework to multi service dependency awareness, where scaling decisions consider upstream and downstream effects. Additional research may also investigate adaptive policy learning and online model updates to handle abrupt workload shifts. Evaluating the technique across extensive clusters and different fields will improve its effectiveness and adoption in practical production scenarios.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.

- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

- [1] Santos, J., Wauters, T., Volckaert, B., and Turck, F. D. (2023). gymhpa: Efficient Auto-Scaling via Reinforcement Learning for Complex Microservice-based Applications in Kubernetes. NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium. <https://doi.org/10.1109/noms56928.2023.10154298>
- [2] Zarai, O., Mcharfi, Z., and El Asri, B. (2025). Intelligent Autoscaling Strategies for Cloud-Native Microservices: A Systematic Review. 2025 International Conference on Intelligent Systems: Theories and Applications (SITA), 1–9. <https://doi.org/10.1109/sita67914.2025.11273589>
- [3] Mondal, S. K., Wu, X., Kabir, H. M. D., Dai, H. N., Ni, K., Yuan, H., and Wang, T. (2023). Toward optimal load prediction and customizable autoscaling scheme for kubernetes. *Mathematics*, 11(12), 2675.
- [4] Xu, M., Song, C., Ilager, S., Gill, S. S., Zhao, J., Ye, K., and Xu, C. (2022). CoScal: Multifaceted scaling of microservices with reinforcement learning. *IEEE Transactions on Network and Service Management*, 19(4), 3995-4009.
- [5] Shi, T., Ma, H., Chen, G., and Hartmann, S. (2023). Auto-scaling containerized applications in geo-distributed clouds. *IEEE Transactions on Services Computing*, 16(6), 4261-4274.
- [6] Rubak, A., and Taheri, J. (2023, December). Machine learning for predictive resource scaling of microservices on kubernetes platforms. In *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing* (pp. 1-8).
- [7] Chouliaras, S., and Sotiriadis, S. (2023). An adaptive auto-scaling framework for cloud resource provisioning. *Future Generation Computer Systems*, 148, 173-183.
- [8] Peng, Z., Tang, B., Xu, W., Yang, Q., Hussaini, E., Xiao, Y., and Li, H. (2023, December). Microservice Auto-Scaling Algorithm Based on Workload Prediction in Cloud-Edge Collaboration Environment. In *2023 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)* (pp. 608-615). IEEE.
- [9] Wang, Z., Zhu, S., Li, J., Jiang, W., Ramakrishnan, K. K., Yan, M., Zhang, X., and Liu, A. X. (2024). DeepScaling: Autoscaling Microservices With Stable CPU Utilization for Large Scale Production Cloud Systems. *IEEE/ACM Transactions on Networking*, 32(5), 3961–3976. <https://doi.org/10.1109/tnet.2024.3400953>
- [10] Alharthi, S., Alshamsi, A., Alseiari, A., & Alwarafy, A. (2024). Auto-scaling techniques in cloud computing: Issues and research directions. *Sensors*, 24(17), 5551.
- [11] Razzaq, M. A., Mahar, J. A., Ahmad, M., Saher, N., Mehmood, A., and Choi, G. S. (2021). Hybrid auto-scaled service-cloud-based predictive workload modeling and analysis for smart campus system. *IEEE Access*, 9, 42081-42089.
- [12] Toka, L., Dobreff, G., Fodor, B., and Sonkoly, B. (2021). Machine learning-based scaling management for kubernetes edge clusters. *IEEE Transactions on Network and Service Management*, 18(1), 958-972.
- [13] Alidoost Alanagh, Y., Firouzi, M., Rasouli Kenari, A., & Shamsi, M. (2023). Introducing an adaptive model for auto-scaling cloud computing based on workload classification. *Concurrency and Computation: Practice and Experience*, 35(22). Portico. <https://doi.org/10.1002/cpe.7720>
- [14] Abdel Khaleq, A., and Ra, I. (2023). Intelligent microservices autoscaling module using reinforcement learning. *Cluster computing*, 26(5), 2789-2800.
- [15] Incerto, E., Pizziol, R., and Tribastone, M. (2023). Opt: An Efficient Optimal Autoscaler for Microservice Applications. 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), 67–76. <https://doi.org/10.1109/acsos58161.2023.00024>
- [16] Merkouche, S., and Bouanaka, C. (2022, December). A Hybrid approach for containerized Microservices auto-scaling. In *2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA)* (pp. 1-6). IEEE.
- [17] Saboor, A., Hassan, M. F., Akbar, R., Shah, S. N. M., Hassan, F., Magsi, S. A., and Siddiqui, M. A. (2022). Containerized microservices orchestration and provisioning in cloud computing: A conceptual framework and future perspectives. *Applied Sciences*, 12(12), 5793.
- [18] ZargarAzad, M., and Ashtiani, M. (2023). An auto-scaling approach for microservices in cloud computing environments. *Journal of Grid Computing*, 21(4), 73.

- [19] Xiao, Z., and Hu, S. (2022, September). Dscaler: A horizontal autoscaler of microservice based on deep reinforcement learning. In 2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS) (pp. 1-6). IEEE.
- [20] Vu, D. D., Tran, M. N., and Kim, Y. (2022). Predictive hybrid autoscaling for containerized applications. *IEEE Access*, 10, 109768-109778.
- [21] Ahmad, H., Treude, C., Wagner, M., and Szabo, C. (2024). Towards Resource-Efficient Reactive and Proactive Auto-Scaling for Microservice Architectures.
<https://doi.org/10.2139/ssrn.4918202>