Review

Kubernetes Autoscaling: A Comprehensive Review on Machine Learning Techniques

Ratana Soth *, Sokleng LY , Leangsiv Sok

Posted Date: 15 June 2026

doi: 10.20944/preprints202606.1094.v1

Keywords: kubernetes; autoscaling; machine learning; supervised learning; unsupervised learning; reinforcement learning; semi-supervised learning; HPA; VPA; KEDA; cloud-native applications; microservices; resource optimization

 doi:10.20944/preprints202606.1094.v1

Review

Kubernetes Autoscaling: A Comprehensive Review on Machine Learning Techniques

Ratana Soth *, Sokleng LY and Leangsiv Sok

Computer Science Department, Paragon International University, Phnom Penh, Cambodia
* Correspondence: rsoth@paragoniu.edu.kh

Abstract

Kubernetes has become one of the most widely adopted orchestration platforms for deploying and managing cloud-native applications. Its native autoscaling mechanisms, including the Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA), Cluster Autoscaler (CA), and event-driven autoscaling frameworks such as KEDA, provide elasticity for containerized workloads. However, most default autoscaling mechanisms remain reactive because scaling actions are triggered after resource utilization or external metrics exceed predefined thresholds. This reactive behavior can cause cold-start delay, over-provisioning, under-provisioning, service-level objective (SLO) violations, and inefficient resource utilization, especially in dynamic microservice and multi-tenant environments. Machine learning (ML) has therefore emerged as an important research direction for proactive and adaptive Kubernetes autoscaling. This paper presents a comprehensive review of ML-based Kubernetes autoscaling techniques using four major learning categories: supervised learning, unsupervised learning, reinforcement learning, and semi-supervised learning. Each category is further classified into Level-1 subcategories, including regression, classification, time-series forecasting, clustering, dimensionality reduction, association/pattern mining, value-based reinforcement learning, policy-based reinforcement learning, actor-critic methods, generative/consistency-based learning, graph-based learning, and pseudo-labeling/self-training. The review analyzes how these techniques support workload prediction, resource demand estimation, workload classification, anomaly detection, dependency modeling, adaptive scaling policies, and knowledge transfer. The findings show that supervised time-series forecasting is the most mature direction for proactive HPA, graph-based learning is increasingly important for dependency-aware microservice autoscaling, and reinforcement learning is promising for adaptive closed-loop resource optimization. Semi-supervised learning remains an emerging but important direction for environments where labeled Kubernetes telemetry is limited.

Keywords: kubernetes; autoscaling; machine learning; supervised learning; unsupervised learning; reinforcement learning; semi-supervised learning; HPA; VPA; KEDA; cloud-native applications; microservices; resource optimization

1. Introduction

Cloud-native computing has transformed the way modern software systems are designed, deployed, and operated. Instead of deploying monolithic applications on fixed infrastructure, organizations increasingly deploy applications as containerized microservices managed by orchestration platforms. Kubernetes has become a dominant platform in this ecosystem because it provides automated deployment, scheduling, self-healing, service discovery, rolling updates, and autoscaling support [1]. These capabilities make Kubernetes suitable for dynamic cloud environments, where application workloads may vary according to user demand, business cycles, and external events.

Autoscaling is a critical function in Kubernetes because resource demand is rarely constant. If resources are statically over-provisioned, infrastructure cost and energy consumption increase. If resources are under-provisioned, applications may suffer from latency, request failures, and SLO

 doi:10.20944/preprints202606.1094.v1

violations. Kubernetes provides multiple autoscaling mechanisms. HPA adjusts the number of pod replicas based on metrics such as CPU utilization, memory consumption, or custom application metrics [2]. VPA adjusts CPU and memory requests for containers. CA increases or decreases the number of worker nodes when pods cannot be scheduled or when nodes are underutilized. KEDA extends Kubernetes by enabling event-driven autoscaling from external sources such as message queues, stream processors, and serverless triggers [3].

Although these mechanisms are widely used, they are commonly reactive. A reactive autoscaler observes current metrics and performs scaling only after a threshold violation occurs. This creates several problems. First, scaling action may arrive too late because metric collection, scheduling, container startup, and application warm-up require time. Second, static threshold configuration is difficult because different applications have different performance behavior. Third, microservice dependency chains complicate scaling decisions because the overloaded component may not always be the component with the highest CPU utilization. Fourth, reactive autoscaling may overreact to short-term spikes or fail to anticipate recurring workload patterns.

Machine learning offers a promising solution by enabling predictive, adaptive, and context-aware autoscaling. Supervised models can predict future resource demand. Unsupervised models can discover anomalous workload behavior and hidden patterns. Reinforcement learning agents can learn scaling policies through feedback. Semi-supervised learning can reduce dependency on labeled training data by exploiting abundant unlabeled telemetry. Prior studies have explored predictive scaling, deep learning forecasting, graph-based resource allocation, and reinforcement learning for container and cloud-native environments [4,5]. However, the literature remains fragmented across different ML paradigms, Kubernetes autoscaling mechanisms, and evaluation goals.

This paper reviews ML-based Kubernetes autoscaling through a four-category taxonomy: supervised learning, unsupervised learning, reinforcement learning, and semi-supervised learning. The purpose is not only to summarize individual studies but also to clarify how each ML category contributes to Kubernetes autoscaling. The contributions of this paper are as follows:

- It presents a taxonomy of ML techniques for Kubernetes autoscaling based on four major learning categories and twelve subcategories.
- It analyzes how each ML category supports workload prediction, resource demand estimation, anomaly detection, adaptive scaling, and dependency-aware autoscaling.
- It compares representative studies according to ML technique, autoscaler target, scaling behavior, and contribution.
- It identifies open research challenges in massive multi-tenant Kubernetes clusters, cross-workload generalization, safe reinforcement learning, explainability, energy-aware optimization, and limited-label learning.

The remainder of this paper is organized as follows. Section II presents the review methodology and taxonomy. Section III explains Kubernetes autoscaling background and the ML-based autoscaling workflow. Sections IV–VII review supervised learning, unsupervised learning, reinforcement learning, and semi-supervised learning. Section VIII provides cross-category analysis. Section IX discusses open challenges and future research directions. Section X concludes the paper.

2. Review Methodology and Taxonomy

This paper follows a structured literature review approach. The selected references were collected from official documentation, ACM Digital Library, IEEE Xplore, Springer, MDPI, Frontiers, AAAI proceedings, arXiv, and recognized academic repositories. The search terms included “Kubernetes autoscaling machine learning,” “predictive autoscaling Kubernetes,” “horizontal pod autoscaler deep learning,” “vertical autoscaling Kubernetes,” “reinforcement learning autoscaling,” “graph neural network microservice autoscaling,” “Kubernetes anomaly detection,” and “semi-supervised microservice resource allocation.”

A reference was included if it satisfied at least one of the following criteria:

 doi:10.20944/preprints202606.1094.v1

1. It proposes or evaluates autoscaling for Kubernetes, containers, microservices, serverless workloads, or cloud-native systems.
2. It uses ML, deep learning, graph learning, reinforcement learning, or semi-supervised learning for resource prediction, resource allocation, anomaly detection, or scaling decision-making.
3. It is traceable through a publisher page, DOI, arXiv, official documentation, or recognized academic repository.
4. It contributes directly to the four-category ML taxonomy used in this review.

References with placeholder DOI values, incomplete bibliographic metadata, duplicated entries, anonymous authorship, or unverifiable venue information were excluded. The resulting reference set contains official technical documentation and selected research works that provide a traceable foundation for the taxonomy and analysis.

Table 1 presents the taxonomy used in this review. The taxonomy classifies ML-based Kubernetes autoscaling into four main categories and twelve subcategories.

Table 1. ML Category Taxonomy Applied in This Review

Main Category	Level-1 Subcategories	Focus in Autoscaling
Supervised Learning	Regression; Classification; Time-Series Forecasting	Workload prediction, resource demand estimation, workload classification
Unsupervised Learning	Clustering; Dimensionality Reduction; Association/Pattern Mining	Anomaly detection, workload characterization, pattern identification
Reinforcement Learning	Value-Based (DQN); Policy-Based (PPO/PG); Actor-Critic	Adaptive scaling policies, closed-loop feedback, multi-agent scheduling
Semi-Supervised Learning	Generative/Consistency; Graph-Based; Pseudo-labeling/Self-training	Limited-label scenarios, dependency modeling, knowledge transfer

Figure 1 shows the complete ML taxonomy for Kubernetes autoscaling. It provides the overall framework used to organize the review.

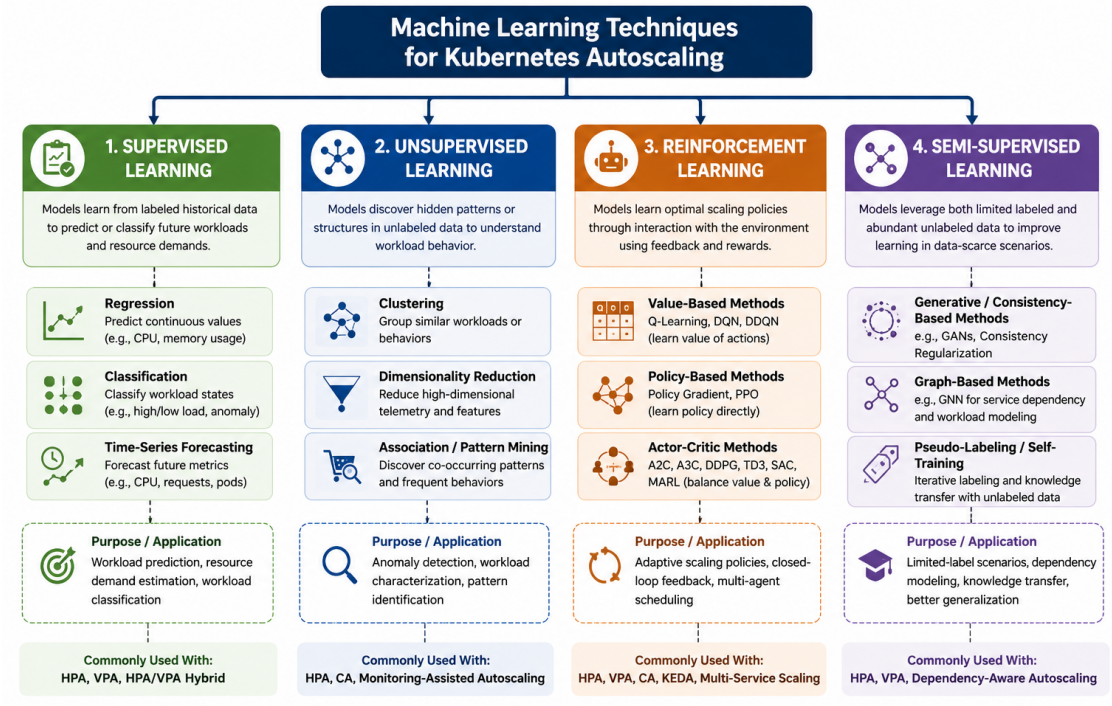

Figure 1. Overall taxonomy of machine learning techniques for Kubernetes autoscaling.

 doi:10.20944/preprints202606.1094.v1

3. Kubernetes Autoscaling and ML-Based Workflow

Kubernetes autoscaling can be viewed at three major layers: pod-level scaling, resource-level scaling, and cluster-level scaling. HPA performs pod-level scaling by increasing or decreasing the number of replicas. VPA performs resource-level scaling by adjusting CPU and memory requests or limits. CA performs cluster-level scaling by adding or removing nodes. KEDA provides event-driven autoscaling by integrating external event sources into Kubernetes scaling decisions.

The traditional HPA algorithm is simple and practical, but it is usually threshold-based and reactive. For example, when CPU utilization exceeds a target value, HPA increases the number of pod replicas. This method works for stable and CPU-bound workloads, but it may not perform well for bursty, latency-sensitive, or dependency-heavy microservices. VPA can improve resource allocation per pod, but traditional vertical scaling may require pod restart. CA provides infrastructure elasticity, but node provisioning is slower than pod-level scaling. Event-driven autoscaling helps asynchronous workloads but may still be reactive if it scales only after queue backlog increases.

ML-based autoscaling extends this process by adding a learning component between telemetry collection and scaling decision-making. As shown in Figure 2, telemetry is collected from the cluster, processed into features, passed into an ML model, and converted into scaling decisions. The scaling decision is then executed through Kubernetes mechanisms such as HPA, VPA, CA, or KEDA. Feedback from the execution result is used to improve future decisions.

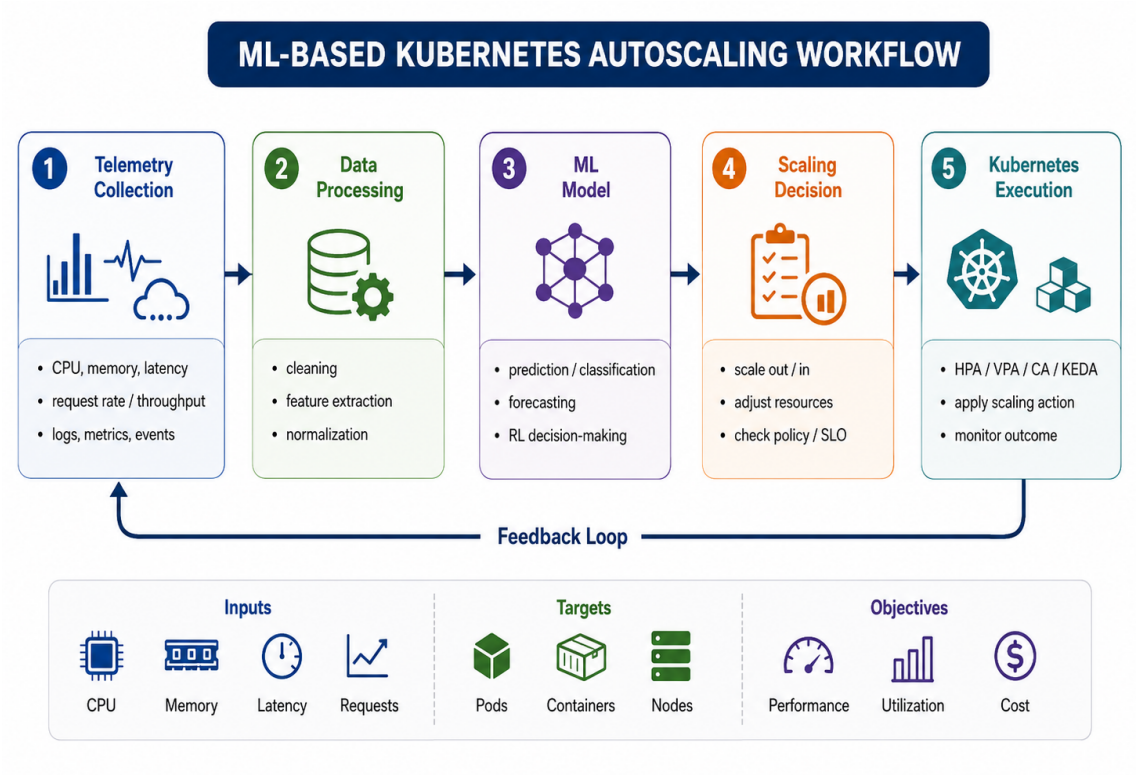

Figure 2. Simplified workflow of ML-based Kubernetes autoscaling.

The input metrics may include CPU utilization, memory usage, request rate, latency, throughput, queue length, pod status, node utilization, and application-specific metrics. The ML model may perform prediction, classification, anomaly detection, policy learning, or graph-based dependency modeling. The scaling decision may involve scaling out, scaling in, adjusting CPU and memory, adding nodes, or triggering event-driven scaling. The optimization objectives often include reducing latency, improving SLO compliance, increasing resource utilization, reducing cost, and improving reliability.

 doi:10.20944/preprints202606.1094.v1

4. Supervised Learning-Based Autoscaling

4.1. Overview

Supervised learning is one of the most mature ML paradigms for Kubernetes autoscaling. A supervised model is trained using historical labeled data. Inputs may include CPU utilization, memory consumption, request rate, pod count, latency, and workload traces. Outputs may include predicted CPU usage, required replicas, workload class, response time, or future resource demand.

In Kubernetes autoscaling, supervised learning supports three major subcategories: regression, classification, and time-series forecasting. Regression predicts continuous resource values. Classification assigns workload or system states into discrete classes. Time-series forecasting predicts future workload or resource demand using historical sequences. Figure 3 summarizes these subcategories.

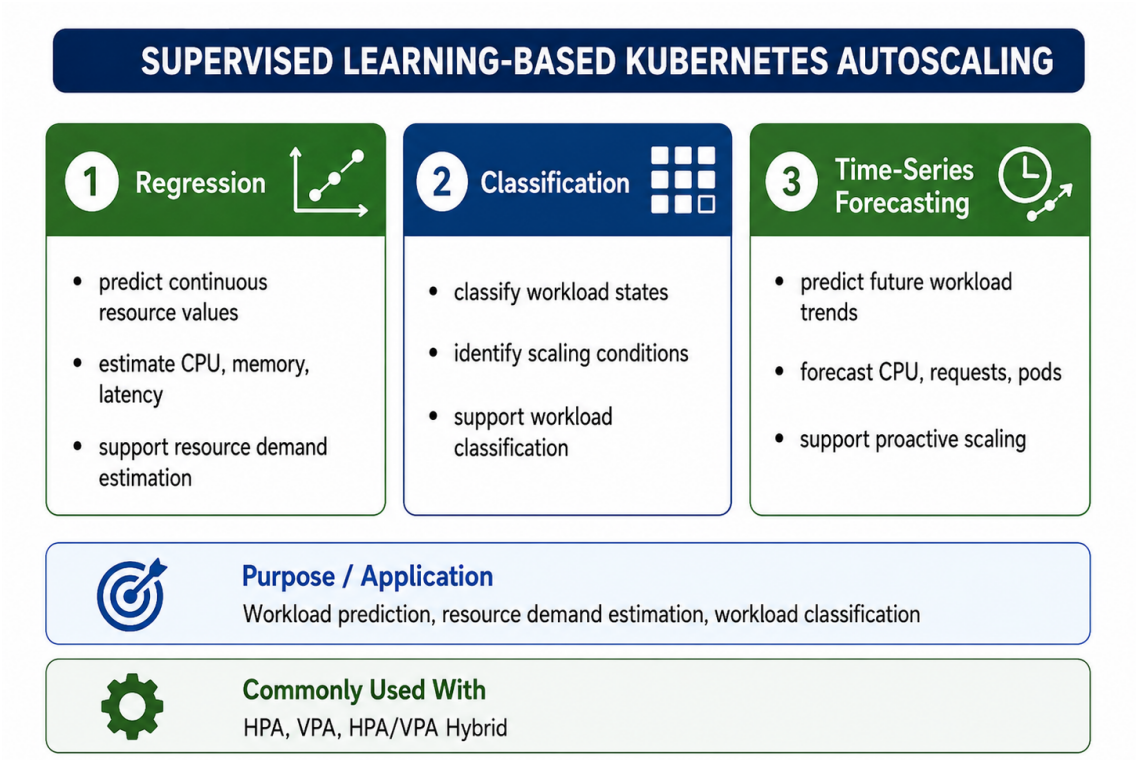

Figure 3. Simplified view of supervised learning-based Kubernetes autoscaling.

4.2. Regression

Regression models predict continuous values such as CPU utilization, memory consumption, response time, or required pod capacity. These models can support proactive autoscaling by estimating the resource demand before the system becomes overloaded. Regression is especially useful when workload history is available and the relationship between telemetry and resource demand is learnable.

Rubak and Taheri proposed predictive resource scaling of microservices on Kubernetes platforms using ML models [6]. This work is directly relevant because it focuses on Kubernetes microservice scaling. Toka et al. proposed machine learning-based scaling management for Kubernetes edge clusters, showing that ML can improve scaling behavior in distributed edge scenarios [7]. Vu et al. proposed predictive hybrid autoscaling for containerized applications, combining prediction with both horizontal and vertical scaling [16]. Regression-oriented resource estimation is also relevant to Transformer-based performance prediction and resource allocation for cloud-native microservices [23].

4.3. Classification

Classification models map workload or cluster states into discrete categories. In autoscaling, classification can identify workload levels, scaling conditions, normal or abnormal behavior, and

 doi:10.20944/preprints202606.1094.v1

system states. Classification is useful when scaling actions are represented as discrete classes such as scale out, scale in, or maintain current configuration.

Zhu et al. proposed a bi-metric autoscaling approach for n-tier web applications on Kubernetes [15]. Although the approach is not a pure ML classifier, it demonstrates the importance of classifying scaling conditions using more than a single metric. Classification-oriented logic is also important in anomaly-aware scaling. Cao et al. studied state-machine learning for monitoring and detecting anomalies in Kubernetes clusters [25]. Nutkumhang et al. proposed SSRFence for detecting SSRF attacks in Kubernetes microservices [26]. Such detection techniques can support autoscaling safety by preventing abnormal traffic from being treated as normal workload demand.

4.4. Time-Series Forecasting

Time-series forecasting is the most mature supervised learning subcategory for proactive Kubernetes autoscaling. Forecasting models learn from historical sequences and predict future workload demand or resource usage. This allows the autoscaler to scale before demand peaks occur.

Dang-Quang and Yoo proposed a BiLSTM-based autoscaling approach for Kubernetes [8]. Their work showed that deep learning can improve proactive autoscaling compared with reactive scaling. Guruge and Yapa proposed a hybrid Prophet and LSTM forecasting approach for Kubernetes autoscaling [9]. Prophet captures seasonality, while LSTM learns sequential workload dynamics. Park and Jeong proposed an autoscaling system that predicts resource demand and responds to prediction failure [10]. Shim et al. compared forecasting models such as Transformer, LSTM, BiLSTM, and ARIMA for Kubernetes cloud autoscaling [11]. Dogani et al. proposed K-AGRUED, an attention-based GRU encoder-decoder technique for container autoscaling [12]. Zhou et al. proposed AHPA, a production-oriented adaptive HPA system for Alibaba Cloud Kubernetes [13].

Table 2 summarizes selected supervised learning studies.

Table 2. Selected Supervised Learning-Based Autoscaling Studies

Ref.	Study	Technique	Target	Action	Contribution
[6]	Predictive scaling of microservices	ML prediction/regression	HPA	Proactive	Predicts resource requirements for Kubernetes microservices
[7]	Scaling for Kubernetes edge clusters	ML forecasting	HPA	Hybrid	Supports dynamic scaling in edge clusters
[8]	BiLSTM autoscaling	BiLSTM	HPA	Proactive	Predicts future workload for proactive scaling
[9]	Prophet-LSTM autoscaling	Prophet + LSTM	HPA	Proactive	Combines seasonal and sequence forecasting
[12]	K-AGRUED	Attention-GRU	HPA	Proactive	Supports multi-step workload prediction
[13]	AHPA	Forecasting + adaptive planning	HPA	Proactive	Demonstrates production-grade adaptive HPA

 doi:10.20944/preprints202606.1094.v1

5. Unsupervised Learning-Based Autoscaling

5.1. Overview

Unsupervised learning is useful when labeled training data is unavailable or expensive to obtain. Kubernetes clusters generate large volumes of unlabeled telemetry from pods, containers, nodes, services, logs, traces, and network flows. Unsupervised learning helps discover hidden structure in this telemetry. In autoscaling, it supports clustering, dimensionality reduction, and association or pattern mining. Figure 4 summarizes this category.

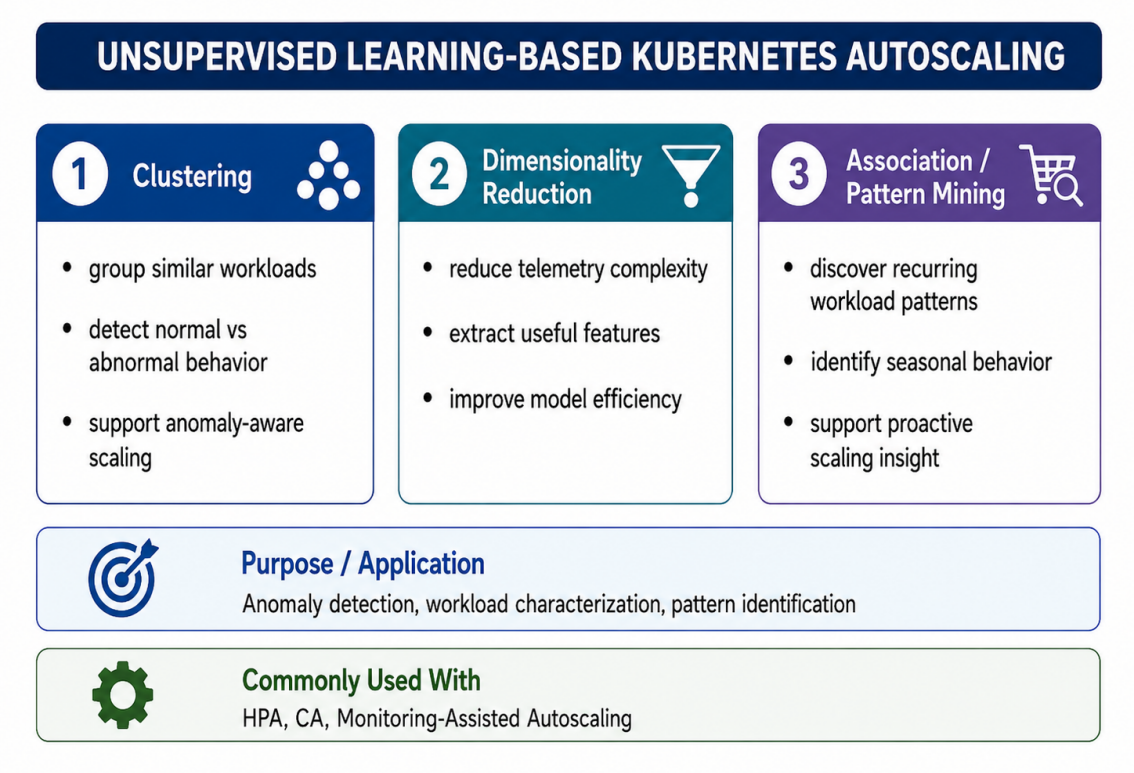

Figure 4. Simplified view of unsupervised learning-based Kubernetes autoscaling.

5.2. Clustering

Clustering groups similar workload states, service behaviors, or telemetry patterns. In Kubernetes, clustering can be used to detect abnormal workload behavior, group similar services, or characterize workload patterns. Clustering is particularly useful for anomaly-aware autoscaling because abnormal traffic or attack behavior should not automatically trigger resource expansion.

Cao et al. proposed learning state machines to monitor and detect anomalies in Kubernetes clusters [25]. The approach learns normal behavior and detects deviations. SSRFence models abnormal inter-service communication patterns in Kubernetes microservices [26]. Marfo studied network anomaly detection in distributed edge computing infrastructure, which is relevant to Kubernetes-supported edge deployments [27].

5.3. Dimensionality Reduction

Dimensionality reduction reduces high-dimensional telemetry into compact representations. Kubernetes environments produce many metrics, and using all metrics directly can increase computational cost and reduce model robustness. Dimensionality reduction improves feature quality and helps downstream learning models.

Graph-based methods often perform representation learning that reduces complex service topology and telemetry into learned embeddings. Graph-PHPA uses LSTM and graph neural networks for dependency-aware proactive HPA [18]. DeepScaler uses spatiotemporal GNNs with adaptive graph learning for holistic microservice autoscaling [19]. GRAF uses graph neural networks for SLO-oriented

 doi:10.20944/preprints202606.1094.v1

resource allocation [20]. HGraphScale applies hierarchical graph learning for autoscaling microservice applications [21].

5.4. Association and Pattern Mining

Association and pattern mining identify recurring relationships, workload cycles, periodic demand, and service-chain patterns. Pattern mining is useful for proactive scaling because future demand can often be inferred from recurring workload behavior.

AHPA is a strong example of pattern-aware scaling because it uses prediction and planning to anticipate workload changes in production Kubernetes environments [13]. Prophet-LSTM also supports pattern-aware forecasting by combining seasonality modeling and sequence learning [9]. ChainsFormer considers chain latency patterns in microservice clusters and uses learning-based resource provisioning [22]. Smart HPA explores resource-efficient scaling behavior for microservice architectures [17].

Table 3 summarizes selected unsupervised learning studies.

Table 3. Selected Unsupervised Learning-Based Autoscaling Studies

Ref.	Study	Technique	Target	Action	Contribution
[25]	Kubernetes anomaly monitoring	State-machine learning	Cluster monitoring	Reactive	Detects anomalous behavior
[26]	SSRFence	Autoencoder/anomaly modeling	Microservices	Reactive	Detects abnormal communication patterns
[18]	Graph-PHPA	LSTM + GNN	HPA	Proactive	Learns dependency-aware representations
[19]	DeepScaler	Spatiotemporal GNN	Microservices	Proactive	Captures dynamic service dependencies
[13]	AHPA	Pattern-aware planning	HPA	Proactive	Uses workload prediction for adaptive planning
[22]	ChainsFormer	Chain pattern analysis	HPA/VPA	Hybrid	Provisions resources based on chain latency

6. Reinforcement Learning-Based Autoscaling

6.1. Overview

Reinforcement learning learns adaptive scaling policies through interaction with the environment. An RL autoscaler observes the system state, selects an action, and receives a reward based on performance, cost, utilization, or SLO satisfaction. Unlike supervised forecasting, RL can optimize long-term behavior through closed-loop feedback. Reinforcement learning is divided into value-based methods, policy-based methods, and actor-critic methods. Figure 5 shows this category.

 doi:10.20944/preprints202606.1094.v1

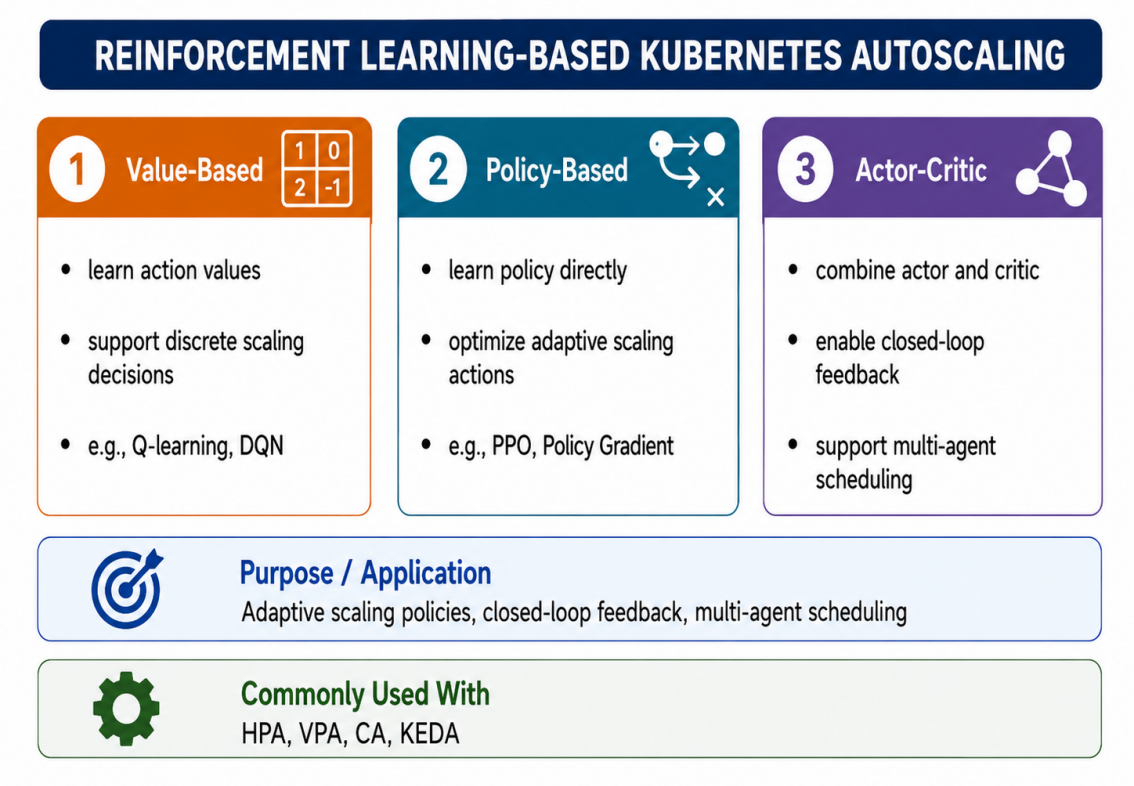

Figure 5. Simplified view of reinforcement learning-based Kubernetes autoscaling.

6.2. Value-Based Methods

Value-based methods, such as Q-learning and Deep Q-Networks (DQN), estimate the expected return of actions in a given state. They are suitable when scaling actions are discrete, such as scale out, scale in, or maintain current configuration. In Kubernetes, value-based methods can be applied to replica adjustment, node scheduling, and resource allocation decisions.

Although pure DQN-based Kubernetes autoscaling studies remain fewer than supervised forecasting studies, value-based methods are important in broader cloud resource management. Gu et al. reviewed DRL algorithms for job scheduling and resource management in cloud computing, including DQN and related methods [30]. ChainsFormer also integrates learning-based resource provisioning for microservice chains [22].

6.3. Policy-Based Methods

Policy-based methods directly learn a policy that maps states to actions. Examples include policy-gradient methods and Proximal Policy Optimization (PPO). These methods are useful when action spaces are continuous or when direct policy optimization is more appropriate than value estimation.

Agarwal et al. proposed a deep recurrent reinforcement learning method for intelligent autoscaling of serverless functions [28]. Although the system focuses on serverless functions, it is relevant to KEDA-style event-driven Kubernetes workloads because both deal with bursty demand and cold-start sensitivity. Majid and Marin reviewed deep reinforcement learning in serverless function scheduling and resource management, showing the increasing importance of policy-based methods for event-driven resource control [29].

6.4. Actor-Critic Methods

Actor-critic methods combine policy learning and value estimation. The actor selects actions, while the critic evaluates them. This structure is useful for autoscaling because the system must balance performance, cost, utilization, and reliability over time.

 doi:10.20944/preprints202606.1094.v1

DInos applies deep reinforcement learning with LSTM and transfer learning for generalizable autoscaling in stateless cloud applications [31]. ChainsFormer uses learning-based provisioning for chain latency-aware microservice clusters [22]. Peng et al. proposed performance prediction and resource adaptive adjustment for cloud-native microservices using prediction and DDPG-style resource adaptation [24]. These works show that actor-critic and hybrid RL methods are promising for adaptive and multi-objective autoscaling.

Table 4 summarizes selected reinforcement learning studies.

Table 4. Selected Reinforcement Learning-Based Autoscaling Studies

Ref.	Study	Technique	Target	Action	Contribution
[22]	ChainsFormer	RL-based provisioning	HPA/VPA	Hybrid	Supports chain latency-aware resource allocation
[30]	DRL cloud review	DQN/DDQN and DRL taxonomy	Cloud resources	Adaptive	Reviews value-based DRL for scheduling and resource management
[28]	Recurrent RL autoscaling	LSTM + PPO	Event-driven/serverless	Proactive	Learns adaptive autoscaling policy
[29]	DRL serverless review	PPO and DRL methods	Serverless workloads	Adaptive	Reviews policy-based RL for scheduling and scaling
[31]	DInos	DRL + LSTM + transfer learning	Cloud applications	Proactive	Improves generalization across workloads
[24]	Adaptive microservice adjustment	Prediction + DDPG	Microservices	Proactive	Supports actor-critic resource adjustment

7. Semi-Supervised Learning-Based Autoscaling

7.1. Overview

Semi-supervised learning uses a small amount of labeled data together with a larger amount of unlabeled data. This is important for Kubernetes because telemetry is abundant, but labeled optimal scaling decisions are difficult and expensive to collect. Semi-supervised learning can reduce labeling cost and improve generalization. In this review, semi-supervised learning is divided into generative/consistency-based methods, graph-based methods, and pseudo-labeling/self-training methods. Figure 6 summarizes this category.

 doi:10.20944/preprints202606.1094.v1

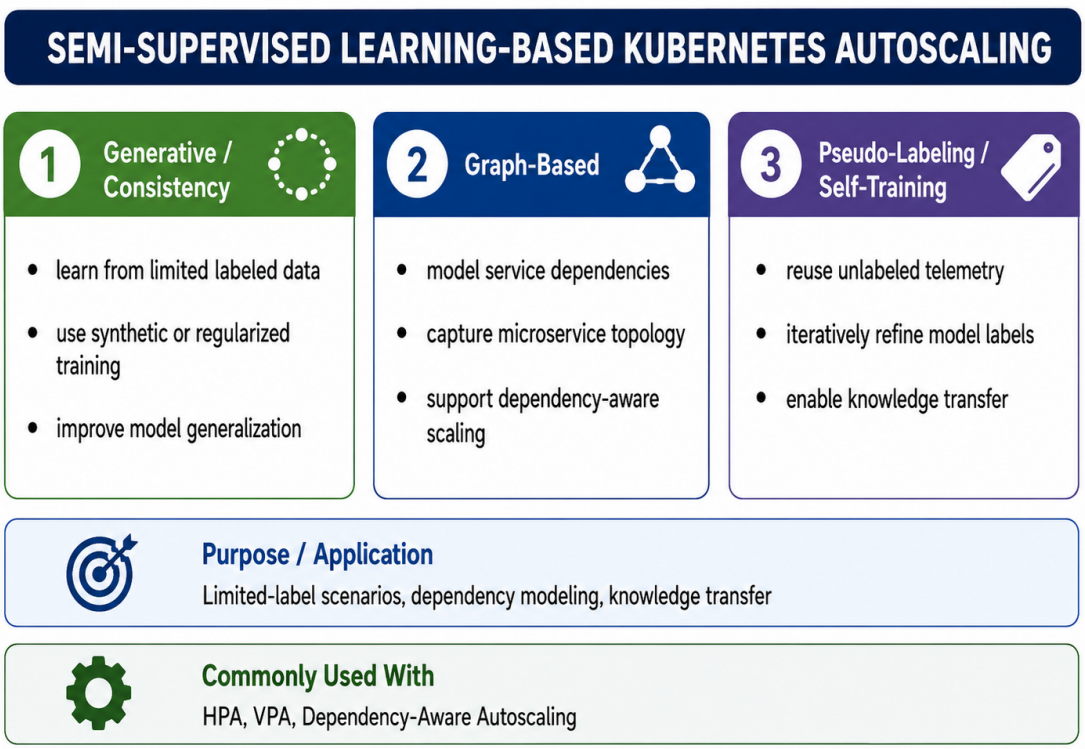

Figure 6. Simplified view of semi-supervised learning-based Kubernetes autoscaling.

7.2. Generative and Consistency-Based Methods

Generative and consistency-based methods can improve learning when labeled samples are limited. For cloud-native microservices, generative models may create synthetic training samples, while consistency regularization may encourage the model to make stable predictions under perturbations. These methods are useful when collecting labeled resource-performance data is expensive.

Cao et al. proposed semi-supervised performance prediction and resource adaptive allocation for cloud-native microservices [32]. Xia et al. explored adaptive resource allocation for cloud-native microservices using meta-learning and hyper-heuristic methods [33]. These studies indicate that limited-label learning is a meaningful future direction for Kubernetes resource optimization.

7.3. Graph-Based Methods

Graph-based methods are especially important for semi-supervised learning because microservice dependency graphs can propagate information across services. Even when only part of the system has labeled performance or scaling data, graph structure can support dependency modeling and knowledge transfer.

Graph-PHPA uses LSTM-GNN for proactive HPA [18]. DeepScaler captures spatiotemporal dependency patterns with adaptive graph learning [19]. GRAF uses graph neural networks for SLO-aware proactive resource allocation [20]. HGraphScale uses hierarchical graph learning to capture pod-level and service-level topology [21]. These works demonstrate the importance of graph-based dependency modeling for Kubernetes autoscaling.

7.4. Pseudo-Labeling and Self-Training

Pseudo-labeling assigns predicted labels to unlabeled data and retrains the model. Self-training extends this idea by iteratively improving model performance as new telemetry arrives. In Kubernetes autoscaling, pseudo-labeling can help create training data for workload states, resource demand classes, or scaling actions. Transfer learning is closely related because knowledge learned from one workload can be reused for another workload.

 doi:10.20944/preprints202606.1094.v1

DInos is relevant to this subcategory because it applies transfer learning to improve generalization across workload traces [31]. Semi-supervised and meta-learning approaches for cloud-native microservices also support this direction [32,33]. These techniques are especially important for massive multi-tenant clusters, where each tenant may have limited labeled data but abundant unlabeled telemetry.

Table 5 summarizes selected semi-supervised learning studies.

Table 5. Selected Semi-Supervised Learning-Based Autoscaling Studies

Ref.	Study	Technique	Target	Action	Contribution
[32]	Semi-supervised performance prediction	Semi-supervised learning	Microservices	Proactive	Supports learning with limited labeled telemetry
[33]	Adaptive resource allocation	Meta-learning + hyper-heuristic	Microservices	Proactive	Improves adaptation with limited performance data
[18]	Graph-PHPA	LSTM + GNN	HPA	Proactive	Uses dependency graph for proactive scaling
[19]	DeepScaler	Spatiotemporal GNN	Microservices	Proactive	Learns dynamic dependency structure
[20]	GRAF	GNN-based resource allocation	Microservices	Proactive	Provides SLO-aware graph-based allocation
[31]	DInos	Transfer learning + DRL	Cloud applications	Proactive	Transfers autoscaling knowledge across workloads

8. Cross-Category Analysis

Table 6 summarizes the reviewed ML categories, subcategories, representative references, and autoscaling roles.

The analysis shows that supervised learning, particularly time-series forecasting, is currently the most mature direction for Kubernetes autoscaling. It is practical, easy to evaluate, and directly supports proactive HPA. However, supervised learning depends on labeled historical data and may not generalize well to unseen workload patterns.

Unsupervised learning is important for anomaly detection, workload characterization, and telemetry representation. It is not always used directly for scaling action, but it can improve autoscaling safety and model efficiency. For example, anomaly detection can prevent an autoscaler from scaling aggressively in response to attack traffic.

Reinforcement learning provides adaptive and policy-driven autoscaling. It is promising for dynamic environments because it can optimize long-term reward rather than only immediate utilization. However, RL introduces challenges in reward design, safe exploration, training stability, and production trustworthiness.

 doi:10.20944/preprints202606.1094.v1

Table 6. Cross-Category Summary of ML-Based Kubernetes Autoscaling

Main Category	Subcategory	Representative References	Main Autoscaling Role
Supervised Learning	Regression	[6,7,16,23,24]	Resource demand estimation
Supervised Learning	Classification	[15,25–27]	Workload and anomaly classification
Supervised Learning	Time-Series Forecasting	[8–13]	Proactive workload prediction
Unsupervised Learning	Clustering	[25–27]	Behavior grouping and anomaly detection
Unsupervised Learning	Dimensionality Reduction	[18–21]	Representation learning for complex telemetry
Unsupervised Learning	Pattern Mining	[9,13,17,22]	Seasonal and chain-pattern identification
Reinforcement Learning	Value-Based	[22,29,30]	Discrete scaling and resource decision learning
Reinforcement Learning	Policy-Based	[28–30]	Direct scaling policy optimization
Reinforcement Learning	Actor-Critic	[22,24,28,31]	Closed-loop adaptive optimization
Semi-Supervised Learning	Generative/Consistency	[23,32,33]	Learning under limited labeled telemetry
Semi-Supervised Learning	Graph-Based	[18–21]	Dependency modeling and graph-based propagation
Semi-Supervised Learning	Pseudo-labeling/Self-training	[31–33]	Knowledge transfer and limited-label adaptation

Semi-supervised learning is the least mature but highly relevant category. Kubernetes telemetry is abundant, but labeled optimal scaling actions are limited. Semi-supervised learning can help models learn from unlabeled telemetry, dependency graphs, and transferred workload knowledge.

9. Open Challenges and Future Directions

9.1. Massive Multi-Tenant Kubernetes Autoscaling

Many existing studies evaluate autoscaling using small testbeds, benchmark applications, or limited workload traces. However, real production clusters may contain hundreds of tenants, thousands of pods, and heterogeneous SLOs. Future research should explicitly address tenant isolation, noisy-neighbor effects, fairness, quotas, and workload diversity in massive multi-tenant clusters.

9.2. Cross-Workload Generalization

ML models often perform well on the workload traces used during training but degrade when deployed to unseen workloads. Transfer learning, continual learning, meta-learning, and self-training are promising directions for improving generalization. DInos is an example of using transfer learning for generalizable autoscaling [31].

9.3. Safe Reinforcement Learning

RL-based autoscaling is powerful but risky. Unsafe exploration can cause SLO violations or service disruption. Production-ready RL autoscalers should include safety constraints, fallback policies, confidence estimation, and rollback mechanisms.

 doi:10.20944/preprints202606.1094.v1

9.4. Explainable Autoscaling

Site reliability engineers need to understand why an autoscaler made a decision. Deep learning and reinforcement learning models are often black boxes. Future autoscalers should integrate explainable AI methods, such as feature attribution, causal explanation, and policy interpretation.

9.5. Energy-Aware and Carbon-Aware Autoscaling

Most autoscaling studies focus on latency, cost, and utilization. Energy efficiency and carbon awareness remain underdeveloped. Future Kubernetes autoscalers should optimize resource allocation while considering energy consumption, carbon intensity, and sustainability objectives.

9.6. Federated and Privacy-Preserving Learning

In multi-organization environments, sharing telemetry may violate privacy or security policies. Federated learning can allow multiple clusters or tenants to train shared models without exposing raw telemetry. This is a promising direction for privacy-preserving autoscaling.

9.7. LLM-Assisted Autoscaling

Large language models may support autoscaling by interpreting logs, recommending scaling policies, summarizing incidents, and assisting SRE decision-making. However, LLMs should not directly control production scaling without verification, guardrails, and integration with reliable telemetry.

10. Conclusion

This paper presented a comprehensive review of ML techniques for Kubernetes autoscaling. The review classified the literature into four main categories: supervised learning, unsupervised learning, reinforcement learning, and semi-supervised learning. Supervised learning supports regression, classification, and time-series forecasting for workload prediction and resource demand estimation. Unsupervised learning supports clustering, dimensionality reduction, and pattern mining for anomaly detection and workload characterization. Reinforcement learning supports value-based, policy-based, and actor-critic methods for adaptive scaling policies and closed-loop optimization. Semi-supervised learning supports generative methods, graph-based dependency modeling, and pseudo-labeling or self-training for limited-label Kubernetes telemetry.

The findings show that supervised time-series forecasting is currently the most mature direction for proactive HPA. Graph-based learning is increasingly important for dependency-aware microservice autoscaling. Reinforcement learning is promising for adaptive and multi-objective resource optimization. Semi-supervised learning remains an emerging but important direction for massive multi-tenant Kubernetes environments where labeled performance and scaling data are limited. Future research should focus on scalable multi-tenant autoscaling, safe reinforcement learning, explainable scaling decisions, cross-workload generalization, energy-aware optimization, and privacy-preserving learning.

References

1. Kubernetes Authors, "Kubernetes documentation," Kubernetes. [Online]. Available: https://kubernetes.io/docs/
2. Kubernetes Authors, "Horizontal Pod Autoscaling," Kubernetes. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
3. KEDA Authors, "KEDA: Kubernetes Event-driven Autoscaling," KEDA. [Online]. Available: https://keda.sh/
4. C. Balla, A. Lal, and M. Anand, "Auto-scaling techniques in cloud computing: Issues and research directions," *Sensors*, vol. 24, no. 17, p. 5551, Aug. 2024, doi: 10.3390/s24175551.
5. Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Computing Surveys*, vol. 55, no. 2, pp. 1–37, Jan. 2022, doi: 10.1145/3510415.

 doi:10.20944/preprints202606.1094.v1

6. A. Rubak and J. Taheri, "Machine learning for predictive resource scaling of microservices on Kubernetes platforms," in *Proc. IEEE/ACM International Conference on Utility and Cloud Computing Companion*, 2023, doi: 10.1145/3603166.3632165.
7. L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for Kubernetes edge clusters," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 958–972, Mar. 2021, doi: 10.1109/TNSM.2021.3052837.
8. N. M. Dang-Quang and M. Yoo, "Deep learning-based autoscaling using bidirectional long short-term memory for Kubernetes," *Applied Sciences*, vol. 11, no. 9, p. 3835, Apr. 2021, doi: 10.3390/app11093835.
9. P. B. Guruge and S. Yapa, "Time series forecasting-based Kubernetes autoscaling using Facebook Prophet and LSTM," *Frontiers in Computer Science*, vol. 7, 2025, doi: 10.3389/fcomp.2025.1509165.
10. J. Park and J. Jeong, "An autoscaling system based on predicting the demand for resources and responding to failure in forecasting," *Sensors*, vol. 23, no. 23, p. 9436, Nov. 2023, doi: 10.3390/s23239436.
11. S. Shim, A. Dhokariya, D. Doshi, S. Upadhye, V. Patwari, and J. Y. Park, "Predictive auto-scaler for Kubernetes cloud," in *Proc. IEEE International Systems Conference*, 2023, doi: 10.1109/SysCon53073.2023.10131106.
12. J. Dogani, F. Khunjush, and M. Seydali, "K-AGRUED: A container autoscaling technique for cloud-based web applications in Kubernetes using attention-based GRU encoder-decoder," *Journal of Grid Computing*, vol. 20, no. 4, p. 40, 2022, doi: 10.1007/s10723-022-09634-x.
13. Z. Zhou *et al.*, "AHPA: Adaptive horizontal pod autoscaling systems on Alibaba Cloud Container Service for Kubernetes," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 37, no. 13, pp. 15621–15629, 2023, doi: 10.1609/aaai.v37i13.26852.
14. L. Baresi, D. Y. X. Hu, G. Quattrocchi, and L. Terracciano, "KOSMOS: Vertical and horizontal resource autoscaling for Kubernetes," in *Service-Oriented Computing*, Cham, Switzerland: Springer, 2021, pp. 821–829, doi: 10.1007/978-3-030-91431-8_59.
15. C. Zhu, B. Han, and Y. Zhao, "A bi-metric autoscaling approach for n-tier web applications on Kubernetes," *Frontiers of Computer Science*, vol. 16, no. 3, 2022, doi: 10.1007/s11704-021-0118-1.
16. D.-D. Vu, M.-N. Tran, and Y. Kim, "Predictive hybrid autoscaling for containerized applications," *IEEE Access*, vol. 10, pp. 109768–109778, 2022, doi: 10.1109/ACCESS.2022.3214985.
17. H. Ahmad, C. Treude, M. Wagner, and C. Szabo, "Smart HPA: A resource-efficient horizontal pod auto-scaler for microservice architectures," arXiv preprint arXiv:2403.07909, 2024.
18. H. X. Nguyen, S. Zhu, and M. Liu, "Graph-PHPA: Graph-based proactive horizontal pod autoscaling for microservices using LSTM-GNN," in *Proc. IEEE CloudNet*, 2022, doi: 10.1109/CLOUDNET55617.2022.9978781.
19. C. Meng, S. Song, H. Tong, M. Pan, and Y. Yu, "DeepScaler: Holistic autoscaling for microservices based on spatiotemporal GNN with adaptive graph learning," in *Proc. IEEE/ACM ASE*, 2023, doi: 10.1109/ASE56229.2023.00038.
20. J. Park *et al.*, "GRAF: A graph neural network based proactive resource allocation framework for SLO-oriented microservices," in *Proc. ACM CoNEXT*, 2021, doi: 10.1145/3485983.3494866.
21. Z. Fang, H. Ma, G. Chen, and R. Buyya, "HGraphScale: Hierarchical graph learning for autoscaling microservice applications in container-based cloud computing," arXiv preprint arXiv:2511.01881, 2025.
22. C. Song, M. Xu, K. Ye, H. Wu, S. S. Gill, R. Buyya, and C. Xu, "ChainsFormer: A chain latency-aware resource provisioning approach for microservices cluster," in *Service-Oriented Computing*, Springer, 2023.
23. Y. Chen, J. Hao, Y. Peng, and H. Xia, "Transformer-based performance prediction and proactive resource allocation for cloud-native microservices," *Cluster Computing*, 2025, doi: 10.1007/s10586-025-05237-9.
24. Y. Peng, J. Hao, and Y. Chen, "Performance prediction and resource adaptive adjustment for cloud-native microservices," *Cluster Computing*, 2025, doi: 10.1007/s10586-025-05437-3.
25. C. Cao, A. Blaise, S. Verwer, and F. Rebecchi, "Learning state machines to monitor and detect anomalies on a Kubernetes cluster," arXiv preprint arXiv:2207.12087, 2022.
26. H. Nutkumhang, W. Huansuriya, J. Thitikawin, and S. Fugkeaw, "SSRFense: A multi-layered SSRF detection and defense system for Kubernetes microservices," in *Proc. International Conference on Knowledge and Smart Technology*, 2026, doi: 10.1109/KST67832.2026.11432077.
27. W. Marfo, "Network anomaly detection in distributed edge computing infrastructure," arXiv preprint arXiv:2503.05700, 2025.
28. S. Agarwal, M. A. Rodriguez, and R. Buyya, "A deep recurrent-reinforcement learning method for intelligent autoscaling of serverless functions," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 1899–1910, 2024, doi: 10.1109/TSC.2024.3384289.

 doi:10.20944/preprints202606.1094.v1

29. A. Y. Majid and A. Marin, "A review of deep reinforcement learning in serverless function scheduling and resource management," arXiv preprint arXiv:2311.12839, 2023.
30. Y. Gu, Z. Liu, S. Dai, C. Liu, Y. Wang, S. Wang, G. Theodoropoulos, and L. Cheng, "Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review," arXiv preprint arXiv:2501.01007, 2025.
31. C. Bitsakos, D. Tsoumakos, and I. Konstantinou, "DInos: A deep reinforcement learning approach to generalizable autoscaling in stateless cloud applications," in *Database and Expert Systems Applications*, Springer, 2025, doi: 10.1007/978-3-032-02049-9_20.
32. R. Cao, J. Hao, and H. Xia, "A semi-supervised performance prediction and resource adaptive allocation for cloud-native microservices," *Journal of Network and Computer Applications*, 2026.
33. H. Xia, J. Hao, and H. Chen, "Adaptive resource allocation for cloud-native microservice via meta-learning and hyper-heuristic algorithms," *Journal of Grid Computing*, 2025, doi: 10.1007/s10723-025-09814-5.
34. M. A. Rodriguez and R. Buyya, "Container-based cluster orchestration systems: A taxonomy and future directions," *Software: Practice and Experience*, vol. 49, no. 5, pp. 698–719, 2019, doi: 10.1002/spe.2660.