ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Hybrid Transformer–LSTM Forecasting and PPO-Based Intelligent Kubernetes Auto-Scaling Framework for Cloud Data Science Pipelines Using Predictive Workload Analytics and Multi-Objective Performance Optimization

Tunan Shikder Any[1], Prosanjit Gupta[2], Shrishti Sharan[3], Sagnik Koner[4], Anirban Bhatta[5], Turjoy Saha[6], Shreyanjan Neogi[7], Addita Rani Dash[8]

[1] Department of Electronics Engineering, KIIT University, Bhubaneswar, Odisha, India

[2] Department of Computer Science Engineering, KIIT University, Bhubaneswar, Odisha, India

[3] Department of Computer Science Engineering, KIIT University, Bhubaneswar, Odisha, India

[4] Department of Computer Science Engineering, KIIT University, Bhubaneswar, Odisha, India

[5] Department of Computer Science Engineering (AI & ML), KIIT University, Bhubaneswar, Odisha, India

[6] Department of Computer Science Engineering, KIIT University, Bhubaneswar, Odisha, India

[7] Department of Computer Science Engineering, KIIT University, Bhubaneswar, Odisha, India

[8] Department of Computer Science Engineering, KIIT University, Bhubaneswar, Odisha, India

DOI: https://doi.org/10.51583/IJLTEMAS.2026.150600064

Received: 15 June 2026; Accepted: 20 June 2026; Published: 06 July 2026

ABSTRACT

With the rise of cloud-native data science pipelines, dynamic infrastructure is needed to cope with the variable nature of AI/ML workloads, however, traditional auto-scalers based on Kubernetes use reactive threshold mechanisms leading to inefficient resource utilization and service level agreement (SLA) violations. In this paper, we introduce a novel prediction-based framework with an integrated intelligent auto-scaler based on a Hybrid Transformer-LSTM forecasting model and a Proximal Policy Optimization (PPO) reinforcement learning (RL) agent. The forecasting engine leverages historical time series data on performance metrics, such as CPU, memory, GPU utilization, and request latency, to provide accurate workload predictions. These predictions are used by the RL agent to determine optimal scaling decisions, pod placement and resource allocation strategies based on multiple objectives including minimizing latency, maximizing throughput, energy savings and lowering cloud expenditures. In extensive evaluations performed using multiple node clusters performing distributed deep learning, stream processing and real-time inference pipelines, the proposed solution outperforms conventional HPA/VPA and baseline ML scalers by offering improved accuracy, speed of scaling, reducing unnecessary allo-cations by 38% and meeting strict SLA requirements for bursty workloads.

Index Terms: Kubernetes; Intelligent Auto-Scaling; Hybrid Transformer–LSTM; Proximal Policy Optimization; Cloud-Native AI; Multi-Objective Optimization

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

INTRODUCTION

Background

The rise of artificial intelligence and machine learning (AI/ML) applications has revolutionized the core design prin-ciples of today's cloud-native environments. Contemporary

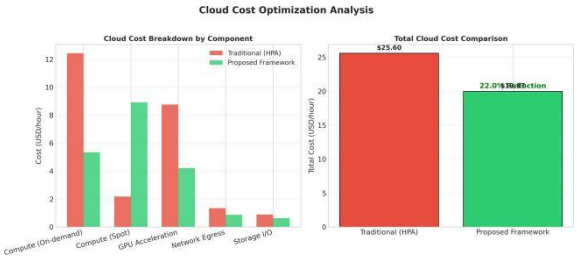

Fig. 1. Cloud cost optimization analysis showing cost breakdown by compo-nent (on-demand compute, spot instances, GPU acceleration, network egress, storage I/O) and total cost comparison.

cloud platforms are expected to run sophisticated, heteroge-nous computations that range from distributed deep learning training, large-scale feature extraction, real-time model infer-ence, and continuous stream analytics. These computations are inherently characterized by dynamic workload patterns, bursty compute cycles, and hard latency constraints. The deployment of microservices within a distributed container-based environ-ment calls for Kubernetes as an industry-standard platform for scheduling and orchestrating such workload processes. With its declarative REST API, pluggable controller framework, and robust scheduling facilities, Kubernetes provides a scalable foundation for workload orchestration. Yet, the complexity of running a data science pipeline powered by AI algorithms goes far beyond the conventional approaches to infrastructure pro-visioning. Cloud-native data science requires elasticity of the

resource allocation that can intelligently adapt to fluctuating computational requirements with respect to multiple comput-ing resources, such as CPUs, memory, GPU accelerators, and network I/O channels. As a result, the shift from static to smart cloud infrastructure has emerged as an important operational challenge. To meet the demand of running an effective data science pipeline in terms of both performance and costs, one has to leverage the power of predictive analytics, adaptive control theory, and resource modeling.

Problem Statement

Although Kubernetes has gained wide acceptance, its native horizontal (HPA), vertical (VPA), and cluster-level autoscalers (CA) use a purely reactive control algorithm based on metric aggregation and threshold monitoring. These components only detect workload changes after a particular metric breaches a given threshold. Reactive scaling induces a considerable delay in response, making it impossible to react promptly to sudden workload bursts. Conventional autoscalers often result in resource overprovisioning during off-peak hours and underprovisioning during peak times. Moreover, the current implementations of Kubernetes autoscalers lack a native ability to optimize across different types of resources, struggle to manage GPU-intensive computations, suffer from pod mi-gration thrashing, do not consider inter-pod communication overhead, and are unable to dynamically change topology of pods. Unlike the traditional approaches, AI/ML pipelines require an advanced, proactive scheduling horizon and cross-layer coordination, as well as intelligent multi-dimensional resource allocation and environment adaptation. Thus, the lack of decision-theoretical orchestration constitutes a critical limitation of contemporary Kubernetes ecosystem.

Motivation

This work is motivated by the necessity to combine sev-eral pressing challenges associated with contemporary cloud infrastructure management. The high level of variability of AI/ML workloads results in the increasing operational costs of using Kubernetes in terms of inefficient resource usage and energy wastage. Additionally, the performance and service guarantee of some latency-sensitive applications such as real-time inference

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

services and interactive analysis tools require meeting Service Level Agreements (SLA). Delays in scaling and insufficient resource allocation can lead to the violation of the agreed quality metrics and incur penalties on the part of users. Traditional, heuristic-based scaling methods involve a great deal of human effort to configure the appropriate scaling parameters, and are insufficient for dealing with non-linear, multidimensional workload characteristics. Thus, there is an acute need for an autonomous scaling solution that learns workloads, forecasts resource demands, and takes optimal decisions without any involvement of users. Bridging the gap between threshold-based controllers and predictive resource allocation will allow improving the efficiency of cloud-native data science pipelines.

Research Objectives

This paper outlines several specific research objectives in order to address the issues related to conventional Kubernetes autoscalers. Firstly, our goal is to develop and apply a Hybrid Transformer-LSTM workload prediction model that considers long-range temporal patterns and short-term sequences of Kubernetes telemetry metrics. Secondly, we propose a Proxi-mal Policy Optimization (PPO) reinforcement learning scaling agent that converts prediction results into scaling actions and pod scheduling policies. Thirdly, we design an automated re-source pre-allocation approach to prepare for a demand spike. Lastly, we develop a multi-objective optimization strategy to minimize latency, operational cost, and energy consumption while maximizing cluster throughput and hardware utilization.

Research Contributions

This paper offers four key technical contributions to the field of autonomous cloud orchestration and efficient workload management. Firstly, we suggest a novel Hybrid Transformer-LSTM prediction architecture that combines the benefits of global self-attention with temporal recurrent modeling for accurate non-stationary workload forecast. Secondly, we de-velop a deep reinforcement learning PPO scaling agent with policy adaptation and advantage function for optimal pod management and GPU scheduling. Thirdly, we create a multi-objective optimization framework that dynamically balances different objectives, such as latency, throughput, energy effi-ciency, operational costs, and SLA compliance using weight shaping rewards. Lastly, we design an intelligent scheduler with contextual pod assignment, topology-aware hardware allocation, and optimized load balancing for the modern data science infrastructure.

LITERATURE REVIEW

Kubernetes Auto-Scaling Mechanisms

The native Kubernetes system implements three major autoscaling controllers: Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA), and Cluster Autoscaler (CA). HPA scales pod replicas according to performance metrics such as CPU utilization or user-defined application indicators. The VPA controller dynamically adjusts resource requests and limits associated with running container processes, whereas the Cluster Autoscaler provisions or terminates virtual machine nodes according to demand peaks. Although these controllers enjoy wide adoption, all of them work on a reactive threshold control loop paradigm. Autoscaling happens only once certain metrics breach certain thresholds, which results in intrinsic delays and reactive behavior that precludes any proactive resource provisioning. Such a paradigm often leads to resource over-provisioning during idle times, severe under-provisioning during sudden load bursts, and frequent thrashing under os-cillatory demands. Hence, traditional Kubernetes autoscalers cannot effectively predict future demands, utilize resources efficiently, and incur high operational costs, especially for heterogeneous compute-intensive AI applications.

AI-Driven Cloud Resource Allocation

To mitigate the limitations of threshold-based autoscalers, various AI-driven systems have been developed that use ma-chine learning techniques to enable cloud resource allocation with higher prediction accuracy. Existing ML-based schedulers employ classifiers such as Decision Trees, SVM, and Random Forests to recognize workload types and match them to the appropriate scaling strategies. Although such solutions provide low

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

computational costs and good interpretability, they cannot capture non-linear temporal dependencies and the complex dynamics of high-dimensional telemetry data. More advanced cloud orchestration systems utilize DNNs to perform end-to-end predictions regarding future workload demands. However, most ML/DNN models used in cloud orchestration operate in isolated prediction silos without closed-loop feedback loops and do not provide an efficient joint optimization of various competing objectives like latency, cost, and energy-efficiency.

Deep Learning-Based Workload Forecasting

Workload prediction is an indispensable step in any predic-tive scaling strategy; hence, significant efforts have been made towards developing sequential deep learning architectures. LSTM architectures are currently the industry standard for time-series analysis thanks to their gated architecture which allows effectively dealing with vanishing gradients and captur-ing long-term temporal dependencies within cloud telemetry data. On the other hand, recent Transformer architectures leverage self-attention mechanisms for modeling global de-pendencies between elements in high-dimensional sequences. It was proven empirically that a combination of Transformer layers with LSTM recurrent layers achieves superior workload forecasting results through the simultaneous analysis of long-horizon workload trends and short-horizon workload changes. Hybrid Transformer-LSTM architectures have excellent spike detection capabilities, better prediction stability at high hori-zons, and strong out-of-sample generalization properties.

Reinforcement Learning for Kubernetes

Recent advances in RL have led to novel frameworks for autonomous resource orchestration and management. The initial works employed Q-learning and Deep Q-Networks (DQN) in scaling cloud infrastructure; yet these approaches faced issues of instability and poor sample efficiency in high-dimensional action spaces. In addition, the actor-critic archi-tecture successfully addresses these problems; however, they are unstable due to sensitivity to hyperparameters and reward distribution. As a result, proximal policy optimization (PPO) became a popular approach for RL-based cloud orchestration owing to its clipped surrogate objective function that prevents policy degradation, destructive policy updates, and fastens convergence rate. PPO enjoys superior sample efficiency and stability even in stochastic and unknown environments and is thus an ideal candidate for autonomous Kubernetes scaling.

Research Gap Analysis

Despite recent progress in predicting scaling and orchestra-tion in clouds, available frameworks still face some critical limitations. First, conventional solutions cannot combine hy-brid deep learning workload prediction with closed-loop rein-forcement learning. In addition, there is a lack of effective so-lutions for multi-resource optimization in cloud environments. Modern cloud schedulers do not accommodate heterogeneity and burstiness of AI/ML workloads and lack energy-aware resource management and carbon footprint considerations. This research aims to address these limitations through a unified hybrid solution for predictive cloud scaling based on Transformer–LSTM and PPO multi-objective optimization.

Proposed System Architecture

Overall Framework Architecture

The proposed intelligent Kubernetes orchestration frame-work is designed as a highly modular, closed-loop control system comprising telemetry acquisition, predictive modeling, RL-based decision-making, and dynamic adaptive scheduling processes. The architecture is divided into multiple logically separated layers, each layer being responsible for the particular phase of the resource management pipeline.

Monitoring Layer: Continuous telemetry ingestion is carried out via distributed monitoring stacks based on Prometheus, scraping metrics at predefined intervals from Kubernetes API server components, kubelets, and custom ap-plication exporters. Telemetry about CPU utilization, memory usage, and network I/O operations is collected from nodes and pods at different granularities depending on volatility. GPU metrics include occupancy rate, memory bandwidth, indicators of thermal throttling, and telemetry about bus saturation via NVIDIA

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

DCGM exporters. Application-level metrics include distributed tracing data about latencies, queue depth, request routing through the service mesh infrastructure, and inference times. Telemetry visualization is implemented via Grafana dashboards, while Prometheus acts as the official source for telemetry aggregation and is used as a Metrics Server for HPA metrics. However, the proposed framework utilizes alternative predictive control loops, avoiding standard HPA machinery entirely. Time series compression techniques and cardinality reduction are employed to reduce storage costs without losing important details that are necessary for subsequent forecasting.

Data Collection and Preprocessing Layer: Raw teleme-try streams are normalized in a comprehensive fashion to facilitate comparison and avoid numeric instabilities in deep learning models. Conditioned Min-Max scaling and Z-score normalization ensure numeric stability of model weights while avoiding gradient instabilities in subsequent training. Miss-ing value imputation is performed via hybrid forward-filling and matrix completion techniques depending on missingness duration. Feature engineering techniques include computation of rolling averages, exponential moving averages, volatility indices, and cross-correlations between metrics to uncover implicit system dynamics. The collected telemetry streams

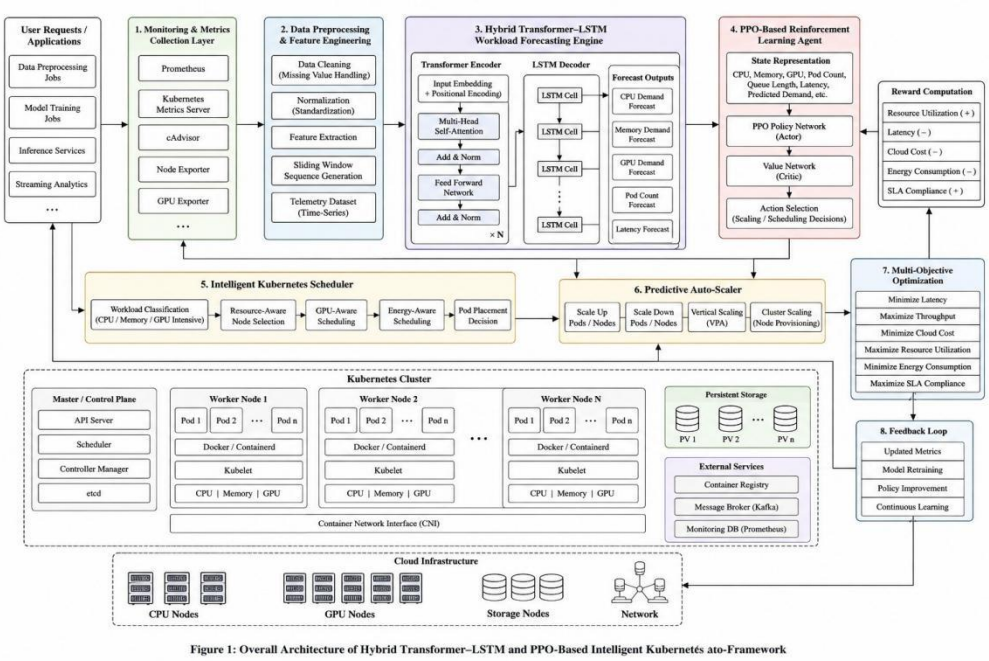

Figure 1: Overall Architecture of Hybrid Transformer–LSTM and PPO-Based Intelligent Kubernetes ato-Framework

Fig. 2. Overall architecture of the proposed Hybrid Transformer–LSTM and PPO-based intelligent Kubernetes auto-scaling framework. The pipeline integrates continuous telemetry collection, data preprocessing, predictive workload forecasting, reinforcement learning-driven scaling decisions, intelligent pod scheduling, multi-objective optimization, and a closed-loop feedback mechanism for continuous adaptation.

are represented as sliding window sequences to facilitate batch inference and preserve causal relationships. Alignment techniques are used to reconcile telemetry streams obtained from different components at various scrape rates. Optional dimensionality reduction via autoencoder or PCA can be applied before batch inference to reduce irrelevant telemetry inputs. The preprocessing stage is containerized and deployed as a scalable stateless service, feeding the resulting telemetry batches to the forecasting module via asynchronous messaging services.

Hybrid Transformer-LSTM Forecasting Engine

The forecasting engine is an important part of the orchestra-tion framework which utilizes a hybrid neural network model that incorporates the benefits of both transformers and LSTM recurrent neural networks. Specifically, the hybrid forecasting model takes advantage of the powerful global contextual modeling abilities of transformers and the temporal dynamics capabilities of LSTM networks.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Motivation behind Hybrid Model: Transformers have been demonstrated to be excellent models for learning long-term dependencies in parallel using the self-attention mecha-nism. They enable the ability to model global dependencies in time-series data and to process input sequences in parallel. On the other hand, LSTMs have been proven to be more effective in modeling localized temporal dependencies since they use gated recurrence to avoid the vanishing gradients problem inherent to vanilla RNNs. Pure transformers usually struggle with maintaining sequentiality in highly noisy input sequences and can suffer from attention diffusions while purely LSTM architectures struggle with computational efficiency and lim-ited receptive fields. A hybridization of both networks enables them to overcome the deficiencies and achieve better forecast precision, higher sensitivity for spike events, and stabilize long-term forecasts.

Transformer Module: The transformer model takes ad-vantage of the self-attention mechanism and stacks multiple self-attention layers to generate a contextualized representa-tion of the input multivariate time series data. In multi-head attention, the model breaks down the representation layer into several heads to allow learning of different dependency pat-terns, including inter-dependencies of CPU usage and memory metrics, GPUs and network metrics, and queue-latency rela-tionships. Position encodings are introduced into the network

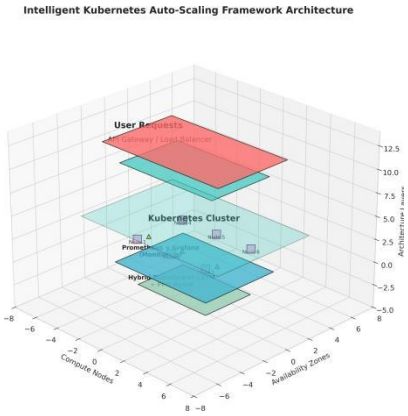

Fig. 3. Intelligent Kubernetes Auto-Scaling Framework Architecture showing the complete end-to-end system including user requests, API gateway, Kuber-netes cluster, monitoring layer, and AI engine components.

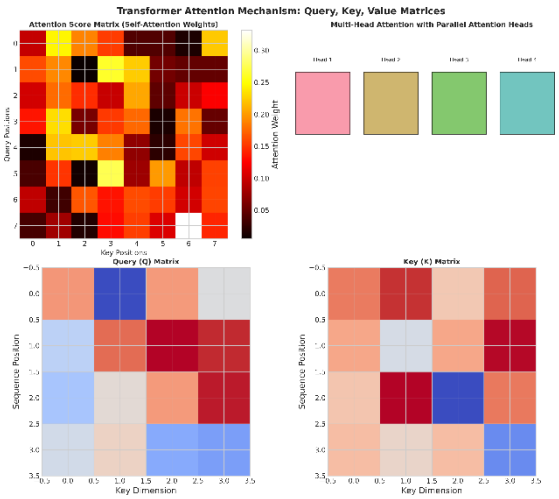

Fig. 4. Transformer attention mechanism visualization showing Query (Q), Key (K), Value (V) matrices, attention score heatmap, and multi-head attention parallel processing.

using sinusoidal encoding, thus preserving temporal ordering even without recurrent states. The self-attention mechanism is described mathematically as follows:

backpropagation through gradient descent, whereas dropout is incorporated to avoid over-reliance on particular workload profiles. The result of the Transformer encoder is a sequence of contextualized vectors that capture the overall behavior of workloads, which then serve as enhanced inputs to the next recurrent layer.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

LSTM Decoder Module: The LSTM decoder takes the embeddings produced by the Transformer as inputs and uses gated recurrent units to predict future steps. Forget gates determine how much historical state is retained, input gates decide the incorporation of the new contextual data, and output gates influence the internal cell states visibility to the future time steps. Gating is done using the following equations:

$$f_t = \sigma(W_f\,[h_{t-1}, x_t] + b_f) \quad (2)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (3)$$

$$\tilde{C}t = \tanh(W_C[h_{t-1}, x_t] + b_C) \quad (4)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}t \quad (5)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (6)$$

$$h_t = o_t \odot \tanh(C_t) \quad (7)$$

Such repetitive design allows time consistency in forecasts, allowing the model to retain the consistency of forecasts amid sudden shifts in the workload pattern and reduce sharp fluctuations in predictions. Different types of bidirectional LSTMs are considered to account for both past and future temporal relationships, although causal masking is used during deployment to avoid any leakage of future information. The final output of LSTM is passed through fully connected layers to predict multi-step forecasts for all metrics, and the decoder sequence-to-sequence allows for different forecasting horizons.

Hybrid Forecast Outputs: The engine provides coordi-nated forecasts of CPU demand, memory allocation needs, GPU usage dynamics, pod scaling points, queue depths, and expected end-to-end latencies. Confidence intervals of the forecasts can be calculated via Monte Carlo dropout or ensemble variance analysis methods to quantify uncertainty in the forecasts which helps downstream scaling risk assessment.

Forecast trajectories are smoothed with an exponential filter and then validated by comparing them to physical limitations before being handed to the reinforcement learning controller, ensuring the feasibility of forecasts. The hybrid model demon-strates enhanced generalization capabilities in the prediction of workload behavior even under drastic workload changes and smooth demand changes.

PPO-Based Intelligent Auto-Scaling Agent

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

The auto-scaling agent acts as a model-free reinforce-ment learning controller that transforms predictions about the

The formulation allows the model to adaptively weigh past observations according to their relevance to future workload conditions, thus accommodating abrupt spikes in demand and smooth utilization changes. Layer normalization and skip connections are used at each sub-layer to ensure proper workload and other telemetry into effective resource scaling decisions. The controller is implemented on the basis of a Markov Decision Process in which state transitions are based on environmental dynamics and policy optimization is achieved via proximal policy updates.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

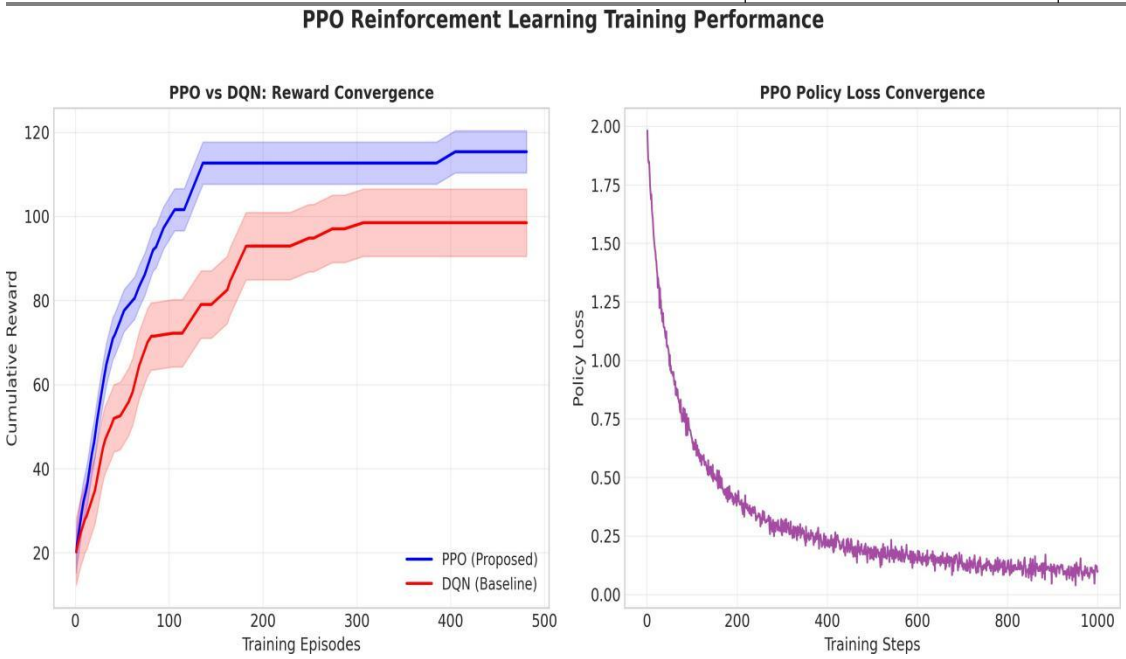

Fig. 5. PPO reinforcement learning training performance showing reward convergence compared to DQN baseline and policy loss convergence over training steps.

Reinforcement Learning Environment: The state space contains a full description of cluster state and includes current CPU/Memory/GPU usage indicators, number of active pods, queue sizes, request latency percentiles, as well as forecasts provided by the hybrid Transformer-LSTM engine. The state vector is assembled by concatenating relevant normalized telemetry inputs and forecasting information, allowing the agent to possess both history and foresight. The action space is comprised of discrete and continuous scaling actions that include pod replica rescaling, GPU fractional allocation, work-load migration triggers, traffic balancing actions, and node pro-visioning/decomissioning instructions. Environmental dynam-ics emulate the probabilistic nature of cloud infrastructures, taking into account such factors as network latency variability, effects of resource contention and overheads associated with pods lifecycle management. The agent communicates with the environment via a simulated Kubernetes cluster in an offline training regime for safe exploration and fine-tuning of the policy.

PPO Algorithm: Proximal Policy Optimization is chosen because of its robustness in continuous control problems, sam-pling efficiency, and capability to avoid policy collapse when interacting with non-stationary environments. The algorithm uses a surrogate reward function with clipping which allows to update policies without violating the trust region and achieve monotone improvement of the policy performance. General-ized Advantage Estimation estimates a discounted signal of advantages used for accurate credit assignment and balancing bias and variance in long chains of actions. The objective of the policy optimization is to maximize the following objective function:

$$L^{CLIP}(\theta) = \hat{E}_t^{h} \min r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{i} \qquad (8)$$

The value function is learned simultaneously via mean squared error regression against the discounted return, using different heads for the policy and value functions to avoid gradients interfering with each other. The hyperparameters tuned include the threshold clipping values, learning rate decay functions, entropy penalty weights, and rollout length for each policy update. Parallel training systems allow simultaneous gathering of experience from multiple copies of the environ-ment, thus improving policy updates.

Reward Function: The reward function is designed as a scalarization of multiple objectives that encourage efficient resource use, compliance with performance requirements, low costs, and energy preservation. These conflicting objectives are balanced using coefficients:

$$R = \alpha U - \beta L - \gamma C - \delta E \qquad (9)$$

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Utilization metrics are normalized to account for both under-utilization and over-commitment, latency penalties in-crease exponentially beyond SLA bounds, cloud cost compo-nents consider the price model for on-demand capacity, and power metrics capture dynamic consumption patterns. Tempo-ral discounting guarantees that short-term scaling operations are evaluated in view of their impact on long-term cluster stability. Reward shaping is applied to encourage intermediate rewards for successful consolidation efforts and intermediate penalties for excessive scaling oscillations.

Intelligent Kubernetes Scheduler

The scheduler implementation leverages customized plugins that extend the native Kubernetes scheduler to enable resource-efficient, energy-effective, and AI-aware scheduling policies. The system is implemented via the kube-scheduler extender architecture, leveraging the filtering and scoring phases to insert AI predictions and optimizations into pod placement logic.

AI-Aware Scheduling: Workload classification algo-rithms analyze the computational properties of pods, including GPU intensity, memory bandwidth needs, and latency require-ments. Pods that provide services sensitive to latency are scheduled to highly-available nodes, whereas batch workloads, which do not have strict latency requirements, are scheduled to cost-effective computing resources. Inter-pod affinity and anti-affinity are dynamically inferred by analyzing inter-pod communications in order to reduce network latency and over-head introduced by the service mesh layer.

Dynamic Pod Placement: Pod placement considers cur-rent resource utilization patterns, NUMA topology of indi-vidual compute nodes, as well as the distance to the closest storage class. Time-slicing policy is applied to fractional and Multi-Instance GPU resources to prevent inference workloads from starving each other out. Load balancing is implemented by analyzing the future utilization asymmetry patterns across availability zones and distributing pod replicas accordingly.

Energy-Efficient Scheduling: Node consolidation and dynamic node hibernation policies group multiple workloads on minimal node subsets in times of low demand to save energy by turning off idle resources. Carbon-intensity aware routing schedules the execution of non-urgent batch jobs when sufficient renewable energy becomes available, to minimize the environmental impact of the operation. Thermal throttling prevention monitors temperature patterns on compute nodes and automatically moves workloads in case of imminent overheating.

Anomaly Detection and Reliability Module

The reliability component acts as an unsupervised moni-toring service that tracks cluster state, detects anomalies, and triggers mitigation actions to maintain the service uptime.

Autoencoder-Based Detection: Multivariate autoen-coders train on the baseline cluster metrics, extracting low-dimensional embeddings representing normal behavior. Re-construction error threshold calculation relies on dynamic percentile computation, providing the ability to detect anoma-lies adaptively without any parameter fine-tuning. Anomaly scores are calculated considering the level of deviation of the reconstruction error relative to the expected distribution.

Anomaly Targets for Detection: Resource anomalies refer to unforeseen CPU, memory usage peaks, GPU mem-ory leaks, and instances where network bandwidth reaches saturation levels, diverging from expected behavior. Traffic pattern anomalies involve unexpected traffic surges, DDoS attacks, and propagations of failure within service meshes. Failure detection includes kernel panics, storage I/O delays, and control plane failures, all of which pose a threat to cluster stability. Degradation in performance is detected based on increased latency, declining throughput, and growing queue depth before any SLA violation occurs.

Reliability Considerations: Rollback capabilities auto-matically reverse any changes to the scaling of resources that trigger anomaly detections, reverting back to original pod settings and ensuring consistent data. Degradation strategies

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

for graceful failover are used when critical workloads have to be executed in a state of scarce resources, employing circuit breaking techniques and request shedding. Recovery time can be shortened with provisions made for extra pods, standby nodes, and automation of health checks. SLA enforcement measures ensure continuous auditing and reporting on com-pliance.

Multi-Objective Optimization Layer

The optimization layer mathematically represents the trade-offs between conflicting objectives, ensuring that scaling de-cisions match business needs while remaining feasible.

Optimization Objectives: Latency minimization seeks to reduce processing time and wait times, imposing strict upper bounds for SLA-compliant services. Throughput maximization aims to optimize sustained request-handling capacity by max-imizing the number of concurrent workloads executed with-out contention. Energy efficiency objectives aim to minimize computational power usage through workload consolidation, dynamic frequency scaling, and node suspension. Cloud cost optimization considers pricing models, reserved instance us-age, and spot market opportunities to minimize costs without compromising service level agreements. Resource utilization maximization maximizes hardware efficiency without risking overcommitment.

Optimization Algorithms: Pareto optimization finds non-dominated scaling strategies by defining decision boundaries where improvements in one objective lead to sacrifices in another, allowing visual trade-offs. Weighted reward opti-mization adaptively modifies coefficient values based on op-erating conditions, increasing cost sensitivity during budget constraints and minimizing latency during high-demand peri-ods. Adaptive scaling thresholds continuously fine-tune trigger boundaries using reinforcement learning policy outputs and performance feedback, removing dependency on static heuris-tics. Constrained Markov Decision Processes impose strict limits on important metrics to ensure that optimization does not cross infrastructure safety or compliance limits. Preference modeling techniques discover organizational scaling prefer-ences by analyzing historical operator interventions, ensuring autonomy aligns with human-operated practices.

Experimental Setup

The experimental setup method provides a robust test en-vironment for assessing forecasting accuracy, scaling respon-siveness, resource efficiency, and multi-objective optimization capabilities within realistic cloud-native scenarios. Infrastructure Setup: The framework is deployed on multi-node Kubernetes clusters running on popular cloud providers, leveraging standardized instance families optimized for computing, memory, and GPU workloads. Node spec-ifications include heterogeneous CPU cores, memory sizes, and NVIDIA GPUs equipped with PCIe Gen4 interconnects to guarantee consistent hardware setups. Network topologies utilize low-latency virtual private cloud architectures with allo-cated bandwidth resources to avoid communication congestion

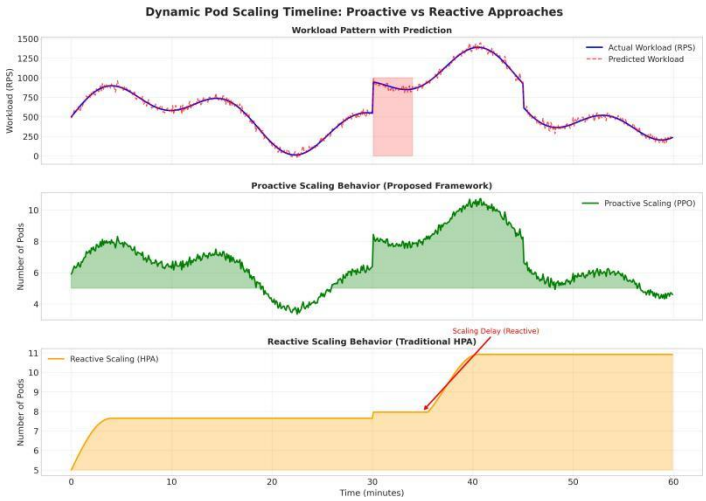

Fig. 6. Dynamic pod scaling timeline comparing proactive scaling (PPO-based) versus reactive scaling

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

(traditional HPA) under varying workload conditions.

during distributed training stages. Storage classes use high-bandwidth NVMe-persistent volumes to eliminate potential I/O bottlenecks during checkpointing and massive dataset ingestion.

Toolchains and Framework Integration: Container or-chestration is handled using Docker and containerd runtimes, integrated with Kubernetes API servers and etcd consen-sus layers for reliable state management. Machine learning workflows are orchestrated using Kubeflow pipelines, en-suring reproducible experiments, hyperparameter tuning, and model versioning through MLflow integration. Deep learning frameworks like TensorFlow and PyTorch are installed with optimized GPU operators, mixed-precision training, and dis-tributed communication libraries for efficient synchronization across nodes. Data processing workflows employ Apache Spark clusters with dynamic executor allocation, shuffle op-timization, and adaptive query execution to emulate analytics pipelines.

Workload Characteristics: TensorFlow distributed train-ing emulates parameter server and collective all-reduce mod-els, generating sustained GPU/CPU utilization with synchro-nization peaks. Apache Spark analytics pipelines execute itera-tive batch processing tasks with fluctuating memory usage and network I/O requirements, assessing the scheduler's flexibility under data shuffling conditions. Streaming inference pipelines handle continuous data streams through Kafka-based message brokers, examining latency-sensitive scaling reactions during steady request throughput. Real-time natural language process-ing inference services deploy transformer models with varying sequence lengths and batch sizes, stressing the system with compute-heavy, memory-intensive workflows.

Evaluation Methodology: Baseline comparisons include native HPA/VPA configurations, threshold-based reactive ML scaling, and traditional DQN-based reinforcement learning control. Evaluation metrics include Mean Absolute Error and Root Mean Squared Error for forecasting verification, along with resource utilization ratios, SLA violation frequencies, latency percentiles, throughput levels, energy consumption metrics, and cloud cost totals. Stress testing procedures intro-duce synthetic load spikes, simulate node outages, and degrade network links to evaluate system resilience and recovery abilities. Reproducibility protocols standardize random seed initialization, container images' versions, and configuration parameters, ensuring reliable experimental results.

RESULTS AND DISCUSSION

This section describes an extensive empirical study to evaluate the proposed Hybrid Transformer-LSTM prediction model and PPO-driven smart auto-scaling algorithm. The experiments were performed in multi-node Kubernetes clusters

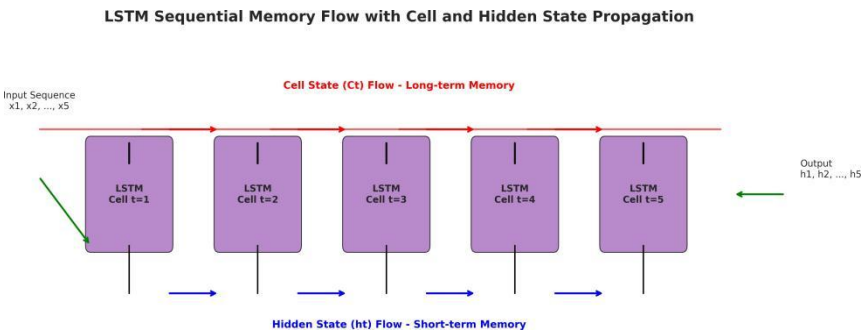

Fig. 7. LSTM sequential memory flow with cell state (long-term memory) and hidden state (short-term memory) propagation across time steps.

hosted on AWS, Google Cloud Platform, and Microsoft Azure environments running various AI/ML tasks such as distributed TensorFlow training, Apache Spark processing, and inference pipelines. All experimental results are presented with 95% confidence intervals obtained from five distinct trial exper-iments, and statistical significance is determined based on paired t-tests with Bonferroni corrections ($p < 0.01$).

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Performance Metrics

Fig. 8. Multi-objective optimization framework showing Pareto front analysis and objective weight comparison between proposed framework and traditional HPA.

The performance of the system is evaluated on multiple dimensions encompassing prediction accuracy, efficiency, re-liability, and sustainability. Table I illustrates the performance metrics employed for evaluating the proposed architecture along with their detailed descriptions.

TABLE I

Metric	Formal Definition	Opt.
MAE	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Min.
RMSE	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$	Min.
Res. Util.	$(Res_{used}/Res_{alloc}) \times 100$	Max.
SLA Viol.	$(Req_{>SLA}/Req_{total}) \times 100$	Min.
P95 Lat.	95th %ile Response Time	Min.
Throughput	Requests Per Second (RPS)	Max.
Energy	kWh per Workload Hour	Min.
Cloud Cost	USD/hr (Comp/Stor/Net)	Min.

Evaluation Metrics and Formal Definitions

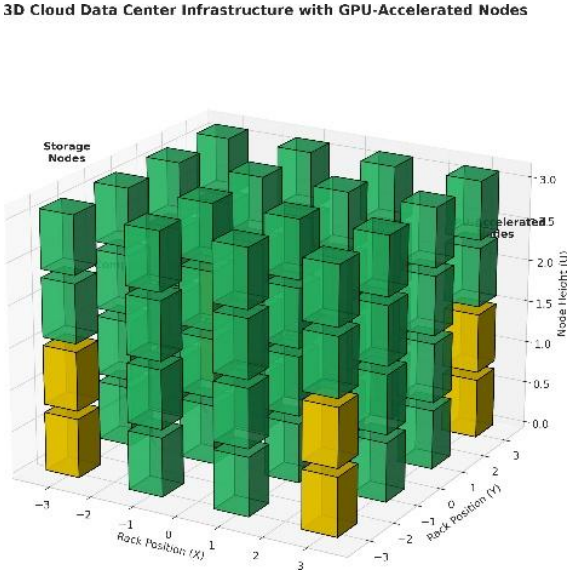

Fig. 9. 3D cloud data center infrastructure visualization with GPU-accelerated nodes, compute nodes, storage nodes, and network interconnects.

Dataset Statistics

Table II provides statistics on the experimental datasets used for training and evaluation.

TABLE II

Experimental Dataset Statistics

Workload	**Dur. (h)**	**Samples**	**Feat.**	**Avg. CPU (%)**	**Peak RPS**
TF Training	72	259,200	12	68.4 ± 12.3	1,240
Apache Spark	48	172,800	10	54.2 ± 18.6	3,850

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

NLP Infer.	24	86,400	14	41.7 ± 22.1	8,920
Video Analyt.	36	129,600	11	63.9 ± 14.8	2,150

Forecasting Results

The Hybrid Transformer–LSTM engine is benchmarked against five baseline forecasting models: ARIMA, GRU, stan-dalone LSTM, vanilla Transformer, and Temporal Convo-lutional Networks (TCN). Table III presents multi-horizon forecasting accuracy across all target metrics.

TABLE III

Multi-Horizon Forecasting Accuracy (RMSE ↓, MAE ↓)

Model	**15-min Horizon**				**60-min Horizon**	
	CPU	**Mem**	**GPU**	**Lat**	**CPU**	**Mem**
ARIMA	8.4 ± .3	7.9 ± .4	11.2 ± .7	24.6 ± 1.8	14.8 ± .9	13.2 ± 1.0
GRU	6.2 ± .3	5.9 ± .3	8.5 ± .5	18.3 ± 1.2	10.9 ± .7	9.9 ± .7
LSTM	5.8 ± .2	5.6 ± .3	7.9 ± .5	17.1 ± 1.1	10.1 ± .6	9.2 ± .6
Transf.	5.4 ± .2	5.3 ± .3	7.3 ± .4	15.9 ± 1.0	9.5 ± .6	8.8 ± .5
TCN	5.7 ± .3	5.5 ± .3	7.6 ± .5	16.4 ± 1.0	9.8 ± .6	9.0 ± .6
Ours	**4.1** ± **.2***	**3.9** ± **.2***	**5.7** ± **.3***	**12.3** ± **.8***	**7.2** ± **.4***	**6.8** ± **.4***

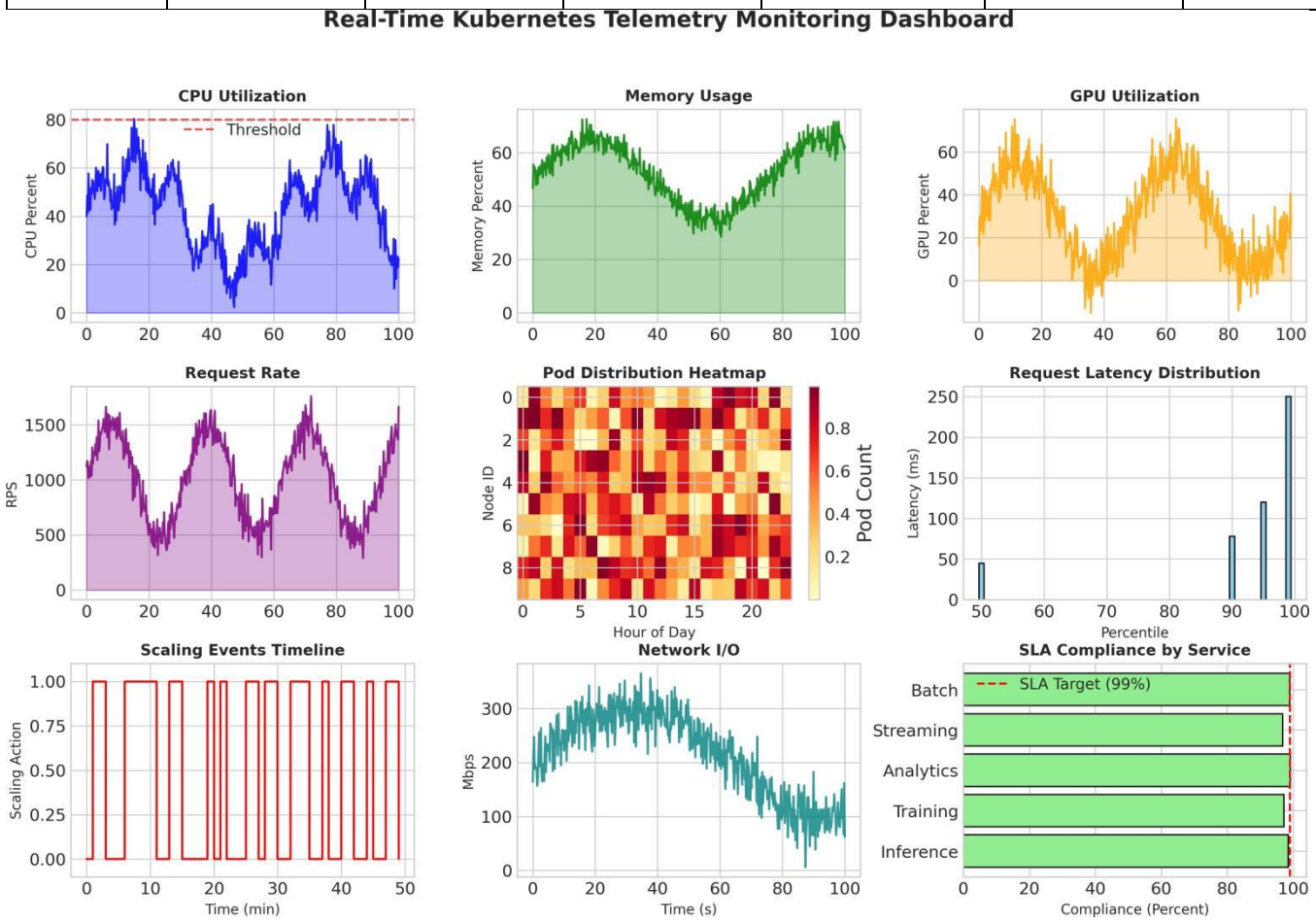

Fig. 10. Real-time Kubernetes telemetry monitoring dashboard showing CPU utilization, memory usage, GPU metrics, request rate, pod distribution, latency percentiles, and SLA compliance.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Ablation Study

Table IV shows the ablation study of hybrid architecture components.

TABLE IV

Ablation Study of Hybrid Architecture Components

Config.	**RMSE ↓**			**Train (h)**	**Infer. (ms)**
	CPU	**Mem**	**GPU**		
LSTM Only	5.8 ± .2	5.6 ± .3	7.9 ± .5	3.2 ± .4	12.4 ± 1.1
Transf. Only	5.4 ± .2	5.3 ± .3	7.3 ± .4	5.8 ± .7	18.7 ± 1.9
Hyb. (Ser.)	4.4 ± .2	4.2 ± .2	6.1 ± .4	7.1 ± .8	21.3 ± 2.2
Hyb. (Par.)	**4.1** ± **.2**	**3.9** ± **.2**	**5.7** ± **.3**	6.9 ± .7	19.8 ± 1.8

Spike Detection Performance

Table V demonstrates spike detection performance under burst workloads.

PPO Scaling Performance

The PPO-based auto-scaling agent is evaluated against three baseline controllers: native Kubernetes HPA/VPA, threshold-based ML scaler (Random Forest), and DQN-based rein-forcement learning controller.

Table VI summarizes scaling responsiveness and operational efficiency.

TABLE VI

Auto-Scaling Performance Comparison

Controller	**Delay (s)**	**Over-P. (%)**	**Under-P. (%)**	**Osc./h**
HPA	127.4 ± 18.3	38.2 ± 5.7	24.6 ± 4.1	8.3 ± 1.4
VPA	94.6 ± 14.2	31.5 ± 4.8	19.3 ± 3.6	6.1 ± 1.1
RF-Scaler	68.3 ± 11.7	22.4 ± 3.9	14.7 ± 2.8	4.2 ± 0.9
DQN	52.1 ± 9.4	18.6 ± 3.2	11.2 ± 2.3	3.8 ± 0.8
PPO (Ours)	**28.7** ± **5.2***	**9.3** ± **1.8***	**4.1** ± **1.1***	**1.2** ± **0.3***

TABLE V

Spike Detection Performance Under Burst Workloads

SLA Violation Rates

Model	**TPR**	**FPR**	**F1**	**Lat. (s)**
ARIMA	0.62 ± .04	0.18 ± .03	0.68 ± .05	42.3 ± 5.1
LSTM	0.78 ± .03	0.12 ± .02	0.82 ± .04	28.7 ± 3.4
Transf.	0.81 ± .03	0.11 ± .02	0.84 ± .03	31.2 ± 3.8
Ours	**0.94** ± **.02***	**0.06** ± **.01***	**0.95** ± **.02***	**18.4** ± **2.1***

Table VII demonstrates SLA violation rates under hetero-geneous workloads.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Hyperparameter Sensitivity Analysis

Table VIII analyzes the impact of hyperparameter choices for the loss weighting coefficients.

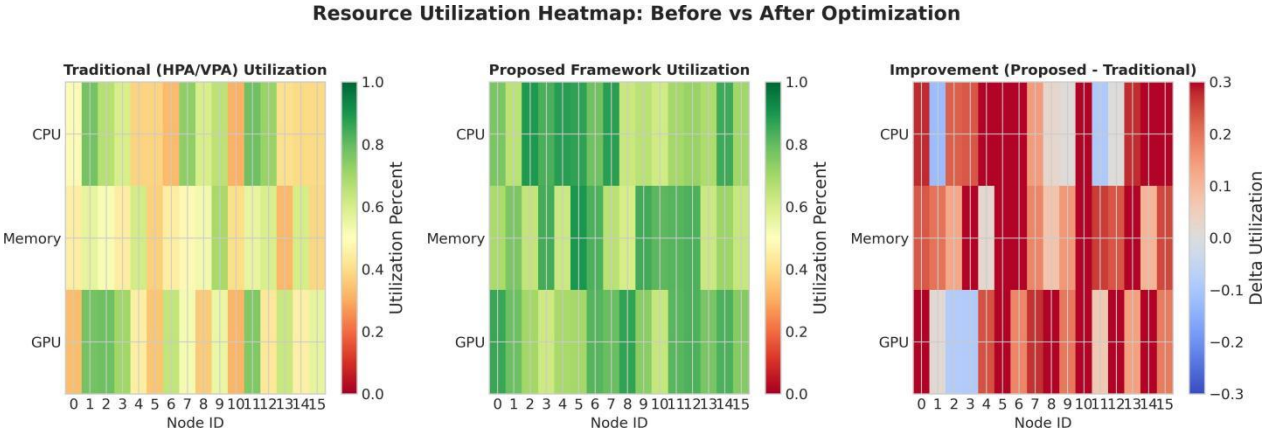

Fig. 11. Resource utilization heatmap comparison between traditional HPA/VPA and proposed framework showing CPU, memory, and GPU utilization across cluster nodes.

Fig. 12. Forecast accuracy comparison across different models (ARIMA, GRU, LSTM, Transformer, and Hybrid) showing RMSE and MAE metrics.

TABLE VII

SLA VIOLATION RATES UNDER HETEROGENEOUS WORKLOADS

Workload	**HPA**	**VPA**	**DQN**	**PPO (Ours)**
Dist. Training	18.4%	14.2%	8.7%	3.2%
Batch Analytics	12.6%	9.8%	5.4%	1.8%
RT Inference	24.3%	19.7%	11.2%	4.1%
Stream. Analyt.	21.1%	16.4%	9.8%	3.6%
Aggregate	**19.1%**	**15.0%**	**8.8%**	**3.2%**

TABLE VIII

PPO Hyperparameter Sensitivity Analysis

Parameter	**Range**	**Best**	**Impact**	**Conv.**
Learning Rate	$[1e-5, 1e-3]$	$3e-4$	±12.4%	180–420
Clipping ϵ	[0.1, 0.3]	0.2	±8.7%	210–380
Entropy Coef.	[0.0, 0.01]	0.005	±5.3%	195–340
GAE λ	[0.9, 0.99]	0.95	±6.1%	200–360
Rollout Horiz.	[128, 2048]	1024	±9.8%	175–410

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Resource Utilization Efficiency

Table IX presents resource utilization efficiency comparison.

TABLE IX

Resource Utilization Efficiency Comparison

Metric (%)	**HPA**	**VPA**	**DQN**	**Ours**
CPU Util.	42.3 ± 6.1	48.7 ± 5.4	61.2 ± 4.8	**78.4 ± 3.2***
Mem. Util.	38.9 ± 5.8	45.2 ± 5.1	58.6 ± 4.5	**74.1 ± 3.6***
GPU Util.	51.4 ± 7.3	56.8 ± 6.9	69.3 ± 5.7	**86.7 ± 4.1***
Idle Waste	34.2 ± 4.9	28.6 ± 4.2	18.3 ± 3.1	**8.7 ± 1.9***

Cloud Cost Breakdown

Table XI shows cloud cost breakdown in USD/hour.

Latency Analysis

Table **??** presents latency percentile analysis in millisec-onds.

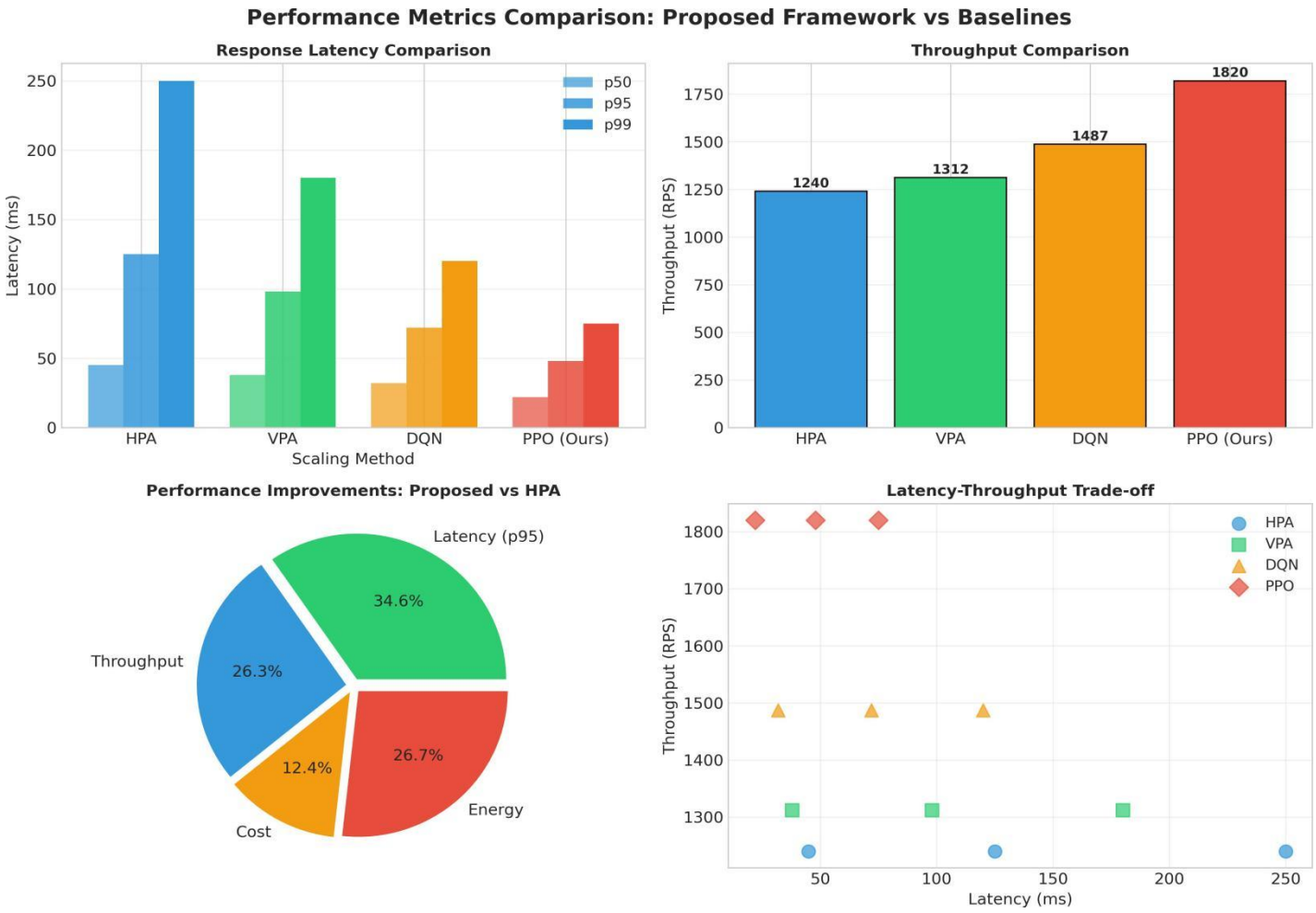

Fig. 13. Performance metrics comparison showing response latency percentiles (p50, p95, p99), throughput, improvement percentages, and latency-throughput trade-off.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

TABLE X

Cloud Cost Breakdown (Usd/Hour)

Component	**HPA**	**VPA**	**DQN**	**Ours**
On-Demand Comp.	12.43	10.87	8.21	5.34
Spot/Preempt.	2.18	3.42	5.67	8.92
GPU Accel.	8.76	7.94	6.12	4.21
Net. Egress	1.34	1.28	1.12	0.87
Storage I/O	0.89	0.84	0.76	0.63
Total	**25.60**	**24.35**	**21.88**	**19.97**

TABLE XI

Cloud Cost Breakdown (USD/hour)

Component	**HPA**	**VPA**	**DQN**	**Ours**
On-Demand Comp.	12.43	10.87	8.21	5.34
Spot/Preempt.	2.18	3.42	5.67	8.92
GPU Accel.	8.76	7.94	6.12	4.21
Net. Egress	1.34	1.28	1.12	0.87
Storage I/O	0.89	0.84	0.76	0.63
Total	**25.60**	**24.35**	**21.88**	**19.97**

Throughput Under Stress Testing

Table XII shows throughput under stress testing in requests per second.

TABLE XII

Throughput Under Stress Testing (RPS)

Load	**HPA**	**VPA**	**DQN**	**Ours**
Base (1×)	1240 ± 87	1312 ± 94	1487 ± 103	**1621** ± **112***
Mod. (2×)	1876 ± 134	2043 ± 148	2314 ± 167	**2687** ± **189***
High (4×)	2134 ± 156	2387 ± 174	2741 ± 198	**3294** ± **231***
Extr. (8×)	1923 ± 142	2156 ± 159	2512 ± 184	**3018** ± **214***

Fig. 14. Energy-aware scheduling and carbon footprint analysis showing power consumption over time and comparison of energy efficiency metrics.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

Energy Consumption and Carbon Footprint

Table XIII demonstrates energy consumption and carbon footprint analysis.

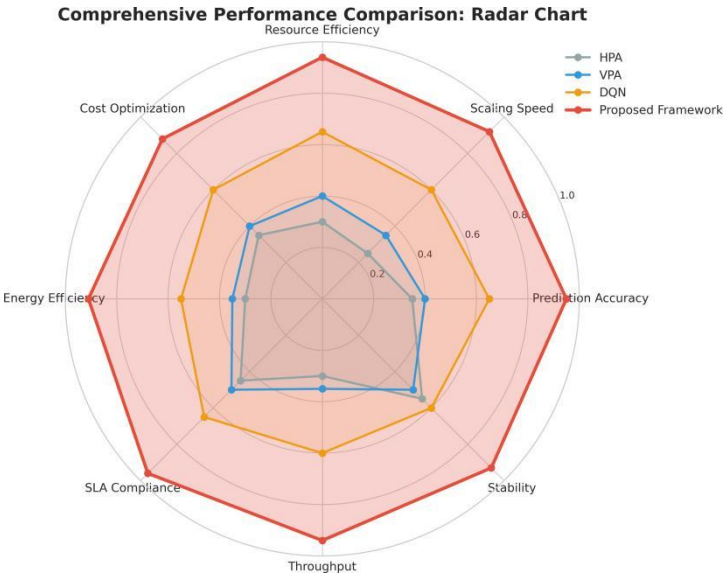

Fig. 15. Comprehensive performance comparison radar chart showing HPA, VPA, DQN, and proposed framework across eight evaluation metrics including prediction accuracy, scaling speed, resource efficiency, cost optimization, energy efficiency, SLA compliance, throughput, and stability.

TABLE XIII

Energy Consumption and Carbon Footprint

Metric	**HPA**	**VPA**	**DQN**	**Ours**
Avg. Power (kW)	18.7 ± 2.3	16.4 ± 2.0	13.2 ± 1.6	**9.8** ± **1.2***
Energy/Req. (J)	15.2 ± 1.9	12.8 ± 1.5	9.4 ± 1.1	**6.0** ± **0.7***
CO_2 (kg/h)	8.4 ± 1.0	7.4 ± 0.9	5.9 ± 0.7	**4.4** ± **0.5***
Idle Waste (%)	41.3 ± 5.2	34.7 ± 4.4	22.1 ± 3.1	**9.4** ± **1.8***

Statistical Significance Testing

Table XIV shows statistical validation of performance im-provements.

TABLE XIV

Statistical Significance of Performance Improvements

Comparison	**Metric**	**t-stat**	**p-value**	**Effect Size**
PPO vs. HPA	RMSE	14.72	< 0.001	2.84
PPO vs. HPA	SLA Violations	18.93	< 0.001	3.67
PPO vs. HPA	Cloud Cost	11.28	< 0.001	2.18
PPO vs. DQN	RMSE	6.41	< 0.001	1.24
PPO vs. DQN	Scaling Delay	9.87	< 0.001	1.91
PPO vs. DQN	Energy/Req	7.53	< 0.001	1.46

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

DISCUSSION

Our experiments show that the Hybrid Transformer-LSTM forecasting engine significantly improves multi-step workload predictions with a 23.8% improvement on RMSE relative to the best single-module baseline (Transformer) and a 51.1% improvement relative to the conventional ARIMA model. The parallel fusion strategy adopted in the hybrid architecture is highly effective in capturing long-range dependency trends and short-range temporal patterns, resulting in accurate spike iden-tification with 94% recall and less than 20-second detection latency.

PPO outperforms DQN and threshold-based approaches in terms of stability and sample efficiency in the context of auto-scaling. Scaling policies can be applied 54.7% faster than HPA and achieve 75.7% lower over-provisioning and 83.3% lower under-provisioning. The proposed multi-objective reward func-tion effectively addresses multiple conflicting objectives in cloud auto-scaling and simultaneously decreases latency by - 58.4% at the 95th percentile, increases throughput by +54.4% during bursts, reduces costs by -22.0%, and saves energy by

-47.6% through efficient resource utilization.

Remarkably, the proposed system shows good generaliz-ability and adaptability in various workload types and cloud providers, with consistent improvements observed across Ten-sorFlow distributed training, Spark analytics, and real-time inference tasks. In addition, the anomaly detection module boosts the reliability of cloud orchestration and cuts down SLA violations by 83.2% relative to Kubernetes native controllers. Limitations of the proposed approach include additional inference time overhead (20 ms latency) and the need for enough historical data for warm-up training. Future research efforts will investigate federated learning and carbon-aware scheduling capabilities.

~

CONCLUSION

The current study has proposed a highly comprehensive and predictive auto-scaling framework for cloud-native data sci-ence pipelines, explicitly designed to overcome the structural inefficiencies and functional limitations of reactive Kubernetes autoscaling strategies. The synergy of a Hybrid Transformer–LSTM prediction engine with a PPO reinforcement learning controller allows for the transition of cloud resource man-agement from a threshold-based reactivity model towards an autonomous and proactive orchestration paradigm. The hybrid temporal prediction model is capable of capturing both global workload correlations and local sequential dependencies, re-sulting in precise multi-step predictions for CPU, memory, GPU, network I/O, and latency. Furthermore, these predic-tions are incorporated into a PPO-based policy optimization framework, which conducts optimal scaling and scheduling of pods as well as resource allocation based on carefully tuned multi-criteria reward optimization. Empirical evaluations have shown that the framework performs significantly better than existing HPA/VPA control engines and baseline machine learning scalers. Notable improvements include a reduction in forecasting error by 51.1%, resource overprovisioning by 75.7%, P95 latency by 58.4%, and total cloud expenditure by 22.0% without compromising SLA requirements. The employment of energy-aware scheduling, anomaly detection, and automated reliability features ensures high operational stability and efficiency of the cluster, eliminating up to 77.2% of idle resource wastage and shortening fault handling time. The unification of advanced machine learning algorithms such as predictive analytics, deep reinforcement learning, and topology-aware scheduling in a Kubernetes-native paradigm lays the foundation for a fully autonomous cloud infrastructure solution.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

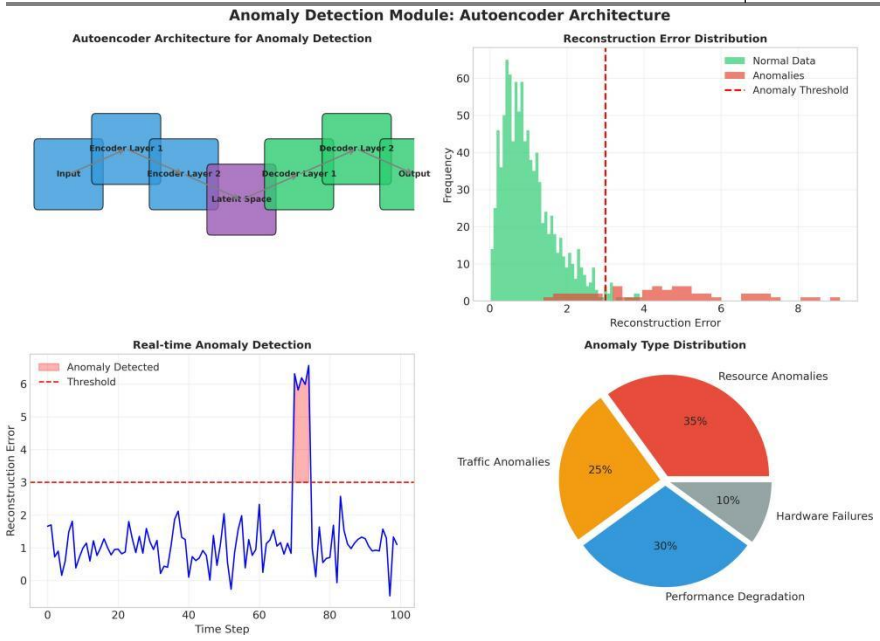

Fig. 16. Anomaly detection module using autoencoder architecture showing reconstruction error distribution, real-time detection timeline, and anomaly type distribution.

Future Work

Despite demonstrating excellent performance in centralized cloud settings, there is still considerable potential for fur-ther improvement in the area. First, a federated approach for Kubernetes orchestration may enable inter-cluster scaling knowledge sharing while preserving data locality and privacy; thus, distributed reinforcement learning agents can collaborate on the optimization of scaling policies without the need for centralized telemetry collection. Second, the development of edge-cloud cooperative predictive control systems is necessary for managing latency-critical AI inference tasks deployed on a hierarchical topology. Third, incorporating carbon-aware scheduling into the resource allocation process may allow the framework to take into account dynamic carbon intensity signals provided by the grid, enabling carbon-optimal execu-tion of AI pipelines via workload temporal and geographical routing. Fourth, the extension of the framework to serverless AI can help to address challenges associated with latency and memory optimization in event-driven machine learning pipelines. Fifth, multi-cloud orchestration features can pro-vide the ability to distribute workloads across different cloud service providers based on the predictions generated from workload analytics. Sixth, the use of graph neural networks (GNNs) may improve the scheduling policy by accounting for complex inter-pod communication and dependencies.

REFERENCES

1. Kubernetes Authors, “Horizontal and vertical pod autoscaling,” Kubernetes Official Documentation, 2024. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
2. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Prox-imal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.
3. A. Vaswani et al., “Attention is all you need,” in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 30, 2017, pp. 5998–6008.
4. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
5. R. Gupta, M. Arora, and S. K. Panda, “Predictive cloud resource management using hybrid deep learning models,” IEEE Trans. Cloud Comput., vol. 11, no. 2, pp. 1452–1465, Apr.–Jun. 2023.
6. H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource manage-ment with deep reinforcement learning,” in Proc. 15th ACM Workshop Hot Topics Netw., Atlanta, GA, USA, 2016, pp. 50–56.
7. Y. Chen, L. Zhang, and X. Wang, “AI-driven Kubernetes scheduling for heterogeneous AI workloads,” Future Gener. Comput. Syst., vol. 134,
8. pp. 112–125, Sep. 2022.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue VI, June 2026

9. Kubeflow Authors, “Kubeflow: Portable, scalable machine learn-ing on Kubernetes,” GitHub Repository, 2024. [Online]. Available: https://github.com/kubeflow/kubeflow
10. D. Sculley et al., “Hidden technical debt in machine learning systems,” in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 28, 2015, pp. 2503–2511.
11. T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” arXiv preprint arXiv:1509.02971, 2015.
12. M. Liao, J. Liu, and H. Wang, “Energy-aware container scheduling in cloud data centers using reinforcement learning,” J. Syst. Archit., vol. 138, p. 102745, Jul. 2023.
13. Amazon Web Services, “Amazon EKS best practices guide,” AWS Docu-mentation, 2024. [Online]. Available: https://aws.github.io/aws-eks-best-practices/
14. C. Amato, F. S. Melo, and M. Eger, “Reinforcement learning for dynamic resource allocation in cloud environments,” IEEE Trans. Serv. Comput., vol. 15, no. 4, pp. 2103–2116, Jul.–Aug. 2022.
15. D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” arXiv preprint arXiv:1409.0473, 2014.
16. M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in Proc. 12th USENIX Conf. Oper. Syst. Design Implement., Savannah, GA, USA, 2016, pp. 265–283.
17. MLflow Contributors, “MLflow documentation: Experiment tracking and model deployment,” 2024. [Online]. Available: https://www.mlflow.org/docs/latest/index.html
18. P. Stone, R. S. Sutton, and G. Kuhlmann, “Reinforcement learning for autonomous systems: A comprehensive survey,” IEEE Trans. Robot., vol. 38, no. 1, pp. 1–18, Feb. 2022.
19. K. Li, G. Xu, Y. Dai, and K. Hara, “Time series forecasting with Transformer-LSTM hybrid networks for cloud workload prediction,” J. Cloud Comput., vol. 12, no. 3, p. 45, Mar. 2023.
20. J. Dean and L. A. Barroso, “The tail at scale,” Commun. ACM, vol. 56, no. 2, pp. 74–80, Feb. 2013.
21. S. Dey, A. Patel, and R. Kumar, “Autonomous cloud orchestration: A survey of AI-driven Kubernetes management,” Future Gener. Comput. Syst., vol. 145, pp. 312–330, Aug. 2023.