National College of Ireland

AI-Driven Load Balancing in Cloud Computing with LSTM Forecasting and DDPG on Kubernetes

MSc Research Project
MSCCLOUD1$_B$

Zeeshan Ali
Student ID: 23382058

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Zeeshan Ali
Student ID:	23382058
Programme:	MSCCLOUD1$_B$
Year:	2025
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	28/01/2026
Project Title:	AI-Driven Load Balancing in Cloud Computing with LSTM Forecasting and DDPG on Kubernetes
Word Count:	8770
Page Count:	26

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Zeeshan
Date:	27th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI-Driven Load Balancing in Cloud Computing with LSTM Forecasting and DDPG on Kubernetes

Zeeshan Ali
23382058

Abstract

Load balancing plays a vital role in cloud computing, especially since it has a direct influence on the throughput of the system, the use of resources, and the use of energy. Round-robin and threshold-based autoscaling are traditional controls that are reactive; that is, they are not predictive of the best way to allocate resources to sudden workload spikes, with a result of under-utilization or over-utilization. This paper presents a new AI-based intelligent load balancer that uses the Long Short-Term Memory (LSTM) networks to predict the workload and Deep Deterministic Policy Gradient (DDPG) reinforcement learning to schedule the resources dynamically in the Kubernetes clusters. The system does latency-driven optimization of the low latency, high throughput and energy-saving by making use of predictive resource requests with the help of the Google Cluster traces. Extensive experiments show that the hybrid LSTM+DDPG system has a high compliance of 98.2% SLA compliance in contrast to 89.5% of Kubernetes HPA, low average response time by 31% (145ms vs 210ms), lower energy consumption by 22.8 percent (0.112 vs 0.145 CPU-hours per 1000 requests), and less scaling oscillations by 51.9 percent. All the improvements are statistically validated ($p < 0.001$). The hybrid way is better than the currently achieved state-of-the-practice strategies and shows improved availability and sustainability of cloud-based services.

1 Introduction

1.1 Background and Motivation

Modern applications are deployed on microservices in containers and run in a rapid, scalable way over the sprawling infrastructure of distributed cloud data centers. The traditional load balancers and auto-scalers, such as Round Robin and Least Connections or Kubernetes' Horizontal Pod Auto-scaler (HPA) are largely reactive and do not possess the predictive analytics capabilities to anticipate sudden upsurge in coming traffic, which may result into latency, SLA violations or even energy wastage (7; 9). To mitigate this, researchers have advocated prediction-based autoscaling methods with designs such as LSTM, CNN-LSTM, Prophet, and hybrid time series models, which can capture the seasonal features and sudden peaks in short periods better than HPA (3; 4; 5; 2; 14). When prediction can only be made with advance knowledge to some extent, proactive scaling is alleviated, but possibly not entirely: wrong predictions and/or workload drift are still leading us to over- or under-scale. In contrast, AWARE (7) and Gwydion (8; 13), computation scaling policies that we learn to minimize not just latency but also cost while

being online with reinforcement learning, are able to dynamically adapt well; however, these sorts of systems, using only RL for adaptation, can require prediction assistance in order to respond quickly enough. The state-of-the-art hybrids fusing forecasting with RL (1) achieve substantially improved performance over an individual one, but in general possess discrete action spaces from which fine-grained control is quite restricted. This motivates us to propose a hybrid auto-scaler combining the LSTMs providing forecasting and continuous-action Reinforcement Learning (RL, particularly DPG) on Kubernetes in order to pursue proactive, dynamic, and energy-efficient scaling in the cloud.

1.2 Problem Statement

Although HPA operates at a CPU threshold of 70%, it is reactive and only scales when CPU exceeds the threshold. More recent RL systems such as AWARE and Gwydion are dynamically adapted but do not do proactive scaling prediction before spikes. The predictive forecasting and adaptive real-time decision making in relation to commitments of Kubernetes are not sufficiently connected.

1.3 Research Question

What is the performance overhead of replacing the classical reactive model with AI models in the case of using an LSTM-based workload predictor and a DDPG-based adaptive scheduler on Kubernetes, including the double-blind policies (both normalizing energy and reinforcement learning), over latency, throughput, energy efficiency, and resource utilization?

1.4 Problem Solution

To overcome the limitations of the reactive autoscaling approach, in this paper we introduce a hybrid AI-based approach that combines workload insight and adaptive control to dynamically balance loads. Specifically, we predict the future workload distribution using historical trace and scale proactively before demand spikes by leveraging Long Short-Term Memory (LSTM). Moreover, in this paper a DDPG RL agent (Deep Deterministic Policy Gradient reinforcement learning agent) is employed to take the real-time decision (when to add or remove pods on the basis of current system status) for resource allocation. The mixed LSTM+DDPG integrates prediction and control to implement proactive and reactive autoscaling. We deploy the system in a Kubernetes cluster using Google Cluster Traces and conduct performance comparisons with several baselines, including the baseline of HPA in Kubernetes, LSTM only, and RL only, where reduced latency, increased throughput, and lowered resource allocation are achieved to save energy.

1.5 Research Objectives

Efficient and rational load balancing remains difficult to achieve in the face of rapid growth of cloud computing and high complexity of distributed applications. Traditional mechanisms such as RR or Kubernetes' HPA are all reactive, i.e., they cannot anticipate variations of workload pattern, causing a degradation of performance, SLA violations, and an overconsumption of energy. In this paper, we intend to suggest a hybrid AI-based load balancing mechanism that utilizes the prediction capability of deep learning and the

adaptability property of reinforcement learning for better management of resources in cloud computing.

Therefore, the aim of this study is:

1. **Design an LSTM-based forecasting model:** Develop an LSTM model to predict short-term workload variations based on historical execution traces. This approach will result in a predictive auto-scale system that adds resources immediately if there is an expected load increase, instead of waiting for the system to get slow before adding more resources.

2. **DDPG policy development and implementation:** Design a DDPG agent that continuously observes metrics in an available cloud service and learns the optimal scaling policies. The real-time RL system has to make decisions of whether to scale up/down or move pods, and it can provide the pretty good utilization of computing resources.

3. **Integrate forecasting and reinforcement learning in a mixed modality system:** Combine the prediction ability of LSTM model and the adaptive control ability of DDPG agent to propose a unified hybrid load balancing framework. This will integrate predictions and responses in an active way, hence it can be used for stable scale-up/down with on-demand adaptation.

4. **Compare the hybrid method with benchmark algorithms:** Conduct comprehensive experiments in a Kubernetes environment on the Google Cluster Traces, and demonstrate that our hybrid model outperforms conventional auto-scalers such as Kubernetes HPA, LSTM-only and RL-only baselines. Performance will be measured in terms of latency, throughput, resource consumption, energy efficiency and SLAs delivered.

5. **Exemplify the impact of AI-driven load balancing in cloud sustainability:** Show how the hybrid approach will improve system reliability, scalability and energy efficiency and help in minimizing overall power footprints in cloud data center while maintaining acceptable quality of service.

1.6 Limitations and Challenges

The hybrid LSTM+DDPG load balancing design proposed in this paper achieves a significant improvement in prediction accuracy, adaptability, and robustness compared with traditional models, although it has some problems to apply in practice. These limitations constitute potential lines of research and future improvement in terms of scalability and applicability.

- **Prediction inaccuracies:** However, in spite of effective results obtained by LSTM model to capture the underlying time-series patterns, invalid information could be produced that can be induced by unregistered fluctuation in workload or added data noise. Due to the improper predictions, inappropriate scaling decisions can be taken such as over-provisioned (i.e., waste of resources) or under-provisioned (quality of service degrade).

- **Computational overhead:** Training and running agents using RL (DDPG) is very hardware specific, resource intensive and time consuming. Adaptive scaling adds to latency and/or requires CPU and memory overhead; therefore, the learning/policy updates may be less beneficial than for fixed.
- **Scalability constraints:** Our proposed framework is evaluated in the controlled experiment and runs on Kubernetes cluster deployment which is not as complex as multi-cloud or large distributed system deployment. Not only will system level concept be challenged, the kinds of and coordination issues will also be a combined systems element to raise new questions about performance and coordination in transitioning toward more heterogeneous or geographically distributed cloud environments.
- **Ethical and privacy considerations:** Privacy and ethical concerns arise when we experiment with real workload traces such as Google Cluster traces. Ensuring proper anonymization, data protection and also transparency of the AI decision-making is necessary to prevent abuse, and for the trustworthy use of autonomous systems.
- **Energy measurement limitations:** There is no on-the-spot energy statistic upgradation mechanisms for direct energy consumption monitoring in Kubernetes. Hence, energy is estimated indirectly using other factors such as CPU utilization and number of active nodes. This estimation, however, is likely to reduce the degree of credibility in sustainability assessment based on estimates of real power saving being contributed.

1.7 Report Structure

The rest of the report will be structured in the following way: Section 2 will introduce a detailed review of the literature covering the forecasting-based, reinforcement learning-based, and hybrid autoscaling methods. Section 3 explains the methodology adopted in the research, including the ethical considerations, pipeline of data and the six-phase workflow. Section 4 outlines design specification, including system components, data flow, technology stack and architecture. Section 5 is an implementation part, which covers tools, models, and deployment process. Section 6 displays the methodology of the evaluation, experimental situations, and comparison of results with statistical validation. Lastly, Section 7 derives the report by summarizing the results and future work directions as well as the contributions.

2 Literature Review

As more and more businesses shift workloads to containerized and microservice-based architectures, cloud autoscaling has emerged as a crucial challenge. SLA violations and resource inefficiency result from traditional reactive autoscaling techniques that rely on threshold-based metrics and react only after performance degradation occurs, like Kubernetes' Horizontal Pod Autoscaler (HPA). Three main approaches have been investigated in recent years: forecasting-based techniques that use time-series models such as CNN and LSTM to predict future demand; reinforcement learning-based techniques that use

continuous environmental interaction to learn optimal scaling policies; and hybrid systems that combine adaptive control and prediction. Recent studies (2021–2025) from various paradigms are summarized in this review of the literature, which also examines their advantages, disadvantages, and usefulness.

2.1 Forecasting-Based Autoscaling (LSTM, CNN, etc.)

The majority of recent autoscaling research focuses on enhancing workload prediction; mechanisms for adaptive real-time response are rarely included. A hybrid Prophet–LSTM model is proposed by Guruge and Priyadarshana, but it does not address abrupt anomalies. Multi-step LSTM forecasting is discussed by Suleiman et al. without tying predictions to runtime scaling actions. A customized autoscaling method is offered by Mondal et al., but feedback-based or reinforcement learning methods are not included. TempoScale by Wen et al. does a good job of capturing workload dynamics, but it is only prediction-oriented. Strong forecasting accuracy is demonstrated by Duc, Nguyen, and Östberg; however, they do not have a feedback loop to address mistakes made during execution. Although Kumar et al. improve Kubernetes by allocating resources based on predictions, they still only use forecasts in the absence of adaptive control. Yuan and Liao introduce early-response autoscaling, which does not adapt to erroneous forecasts, while Sabyasachi and Sahoo emphasize CNN-LSTM hybrids for better prediction.

2.2 Reinforcement Learning-Based Autoscaling

Although autoscaling has been investigated using reinforcement learning, most methods ignore forecasting or more general cluster-level flexibility. While adaptive RL-based scaling is made possible by Qiu et al.'s AWARE, workload prediction is completely ignored. RL is applied to Kubernetes by Santos et al.'s Gwydion, but it only responds when load fluctuates, making it brittle under abrupt increases. While Mampage, Karunasekera, and Buyya validate deep reinforcement learning for serverless autoscaling, they do not apply it to Kubernetes clusters that are containerized. In serverless edge systems, Femminella and Reali employ PPO for resource orchestration; however, their research is limited to single-node settings. Hua et al. use A3C to schedule multi-cloud workflows, but their lack of demand forecasting mechanisms restricts proactive scaling.

2.3 Hybrid Approaches (Forecasting + RL or Advanced Control)

Current research investigates a number of intelligent and hybrid methods for cloud autoscaling and Kubernetes. A forecasting-driven reinforcement learning model for Kubernetes is proposed by Lipari, Mattia, and Beraldi. A multi-agent system is presented by Soulé et al. to increase Kubernetes scaling resilience. Yan et al. introduce a hybrid elastic method for edge environments that combines Bi-LSTM prediction and online reinforcement learning. Wang et al. develop a learning-driven autoscaling technique for various cloud services by fusing deep reinforcement learning with an improved LSTM.

2.4 Reactive/Baseline Autoscaling Approaches

Additionally, recent work has improved Kubernetes' reactive autoscaling techniques. An enhanced node-scaling algorithm that increases cluster efficiency but is only reactive and lacks predictive or adaptive capabilities is proposed by Menouer et al. Using the native Horizontal Pod Autoscaler, Rajasekar et al. investigate QoS-aware autoscaling, emphasizing how its dependence on CPU-based reactive decisions restricts its capacity to anticipate load or integrate more dynamic intelligence.

2.5 Edge/Serverless Category

ES-DRL, a distributed reinforcement learning method for function offloading in serverless edge environments, is presented by Ou et al. It lowers latency by facilitating experience sharing among nodes and resolving orchestration and cold start issues. Nevertheless, the approach is still reactive and does not have a workload prediction mechanism, which hinders proactive scaling and highlights the possible advantages of incorporating forecasting techniques.

2.6 Literature Review Summary Table

Table 1 provides a comprehensive summary of the reviewed literature, highlighting the problem addressed, algorithms and techniques used, key results, and technologies employed by each study.

Table 1: Summary of Literature Review

Author & Year	Problem Addressed	Algorithm/Techniques	Key Results	Technologies
Guruge & Priyadarshana (2025)	Reactive HPA fails to capture seasonal and short-term workload patterns	Prophet + LSTM for Kubernetes forecasting	Improved accuracy in workload prediction and autoscaling vs HPA	Kubernetes, Prometheus, Docker, Facebook Prophet, TensorFlow/Keras
Suleiman et al. (2023)	Difficulty in proactive cloud scaling due to single-step forecasting	Multi-step LSTM forecasting	Reduced SLA violations, better proactive scaling decisions	Kubernetes, Docker, Google Cluster Trace, PyTorch
Mondal et al. (2023)	Static Kubernetes scaling cannot adapt to dynamic workloads	Optimal load prediction + customizable autoscaling	Lower response times, better resource utilization	Kubernetes, Minikube, CloudSim, TensorFlow/Keras
Wen et al. (2024)	Forecasting models struggle with both short and long-term trends	TempoScale (short + long-term integration)	More stable predictions, reduced over/under-provisioning	Kubernetes, Prometheus, Docker, TensorFlow
Duc et al. (2025)	Large-scale cloud–edge environments lack proactive resource allocation	LSTM forecasting for cloud-edge	Higher accuracy, efficient proactive allocation	Kubernetes (edge-cloud), Prometheus, Google Cluster Trace, PyTorch
Qiu et al. (2023)	Reactive autoscaling in production clouds causes SLA breaches	Reinforcement Learning (AWARE system)	Adaptive scaling with fewer SLA violations; scalable in production	Kubernetes, Prometheus, production cloud, OpenAI Gym, TensorFlow Agents
Santos et al. (2025)	Microservice interdependencies not considered in scaling	RL-based autoscaling (Gwydion)	Coordinated scaling across services; improved utilization	Kubernetes, Docker, Grafana, Stable-Baselines3, PyTorch
Menouer et al. (2024)	Node-level autoscaling in Kubernetes remains inefficient	Reactive node autoscaling method	Better cost savings vs default autoscaler, but still reactive	AWS Lambda, CloudWatch, Ray RLlib, PyTorch
Kumar et al. (2024)	Forecasting not integrated into Kubernetes resource allocation	Predictive models + custom resources	Optimized scaling with improved cost and SLA compliance	Kubernetes CRDs, Helm charts, Docker, TensorFlow
Sabyasachi et al. (2024)	Forecasting errors in cloud workloads with single models	Hybrid CNN + LSTM	More accurate workload predictions vs standalone LSTM	GPU acceleration, cloud datasets, TensorFlow/Keras
Lipari et al. (2025)	Forecasting alone or RL alone insufficient for scaling	Forecasting + RL (DQN hybrid) for Kubernetes	Outperformed HPA; improved efficiency and reduced SLA violations	Kubernetes, Prometheus, Docker, Grafana, TensorFlow, Stable-Baselines3
Soulé et al. (2025)	Lack of resilience in Kubernetes autoscaling under dynamic workloads	Multi-Agent Systems (MAS) for autoscaling	More resilient and fault-tolerant scaling decisions	Kubernetes Operators, Docker, MAS framework
Mampage et al. (2025)	Cost and latency trade-offs in serverless autoscaling	Deep RL for time & cost optimization	Improved SLA compliance and reduced operational costs	AWS Lambda, CloudWatch, Ray RLlib, PyTorch
Yuan & Liao (2024)	Kubernetes autoscaling reactive to sudden load spikes	Time series forecasting operator	Improved SLA compliance, reduced over-provisioning vs HPA	Kubernetes Operator, Prometheus, Docker, TensorFlow
Femminella & Reali (2024)	Kubernetes HPA unable to dynamically optimize CPU thresholds in edge	Proximal Policy Optimization (PPO) for resource orchestration	100% improvement in resource utilization (61% vs 30%) while maintaining SLA	OpenFaaS, Kubernetes HPA, Docker, Azure Functions traces, PPO
Hua et al. (2024)	Multi-objective workflow scheduling across heterogeneous cloud providers	Asynchronous Advantage Actor-Critic (A3C) algorithm	Reduced average cost by 55.12% compared to baseline methods	Multi-cloud, Montage dataset, A3C
Yan et al. (2021)	Kubernetes autoscaling cannot handle edge computing workloads with temporal patterns	Bi-LSTM prediction + online reinforcement learning	530–600x faster prediction than ARIMA; improved resource utilization	Kubernetes, Docker, edge computing, Bi-LSTM, RL
Wang et al. (2024)	Heterogeneous scaling challenges across multi-type cloud services	Improved LSTM + Deep RL with ensemble learning	Superior scaling accuracy and resource efficiency; handles stateless/stateful services	Kubernetes, Docker, multi-type cloud, Ensemble LSTM, DRL
Rajasekar et al. (2024)	Security vulnerabilities during pod scaling events; HPA cannot prevent SLA violations	Security-enhanced QoS-aware autoscaling with HPA	Improved understanding of HPA control loop; highlights limitations of reactive approaches	Kubernetes, Docker, CPU monitoring, Kubernetes HPA
Ou et al. (2023)	Function offloading optimization in distributed serverless edge computing	Experience-sharing Deep RL (ES-DRL) with distributed agents	Improved edge AI inference, stream processing; efficient learning across distributed servers	Serverless FaaS nodes, distributed edge servers, cloud replay buffer, IoT, DRL

3 Research Methodology

The goal of this research project is to develop and assess a cloud-specific AI load balancing framework. It consists of a number of crucial steps that work together to deploy and test hybrid LSTM+DDPG in a Kubernetes cluster.

3.1 Ethical and Sustainability Considerations

One of the main concerns of this investigation is protecting privacy and other ethical considerations. According to ethical guidelines, an anonymized APR trace of the Google Cluster data is used for scholarly purposes. The paper also complies with sustainability objectives as a result of energy-efficient computing through resource conservation as well as the smoothing of cluster overuse. Even though there is no direct monitoring of power numbers, node activity and CPU load can be convenient alternatives to measuring the overall energy efficiency.

3.2 Research Methodology Flow Diagram

The systematic pipeline that incorporates data processing, model development, system integration, and evaluation is the basis of the whole research methodology. The entire process is illustrated in Figure 1 which includes data collection, deployment and testing.

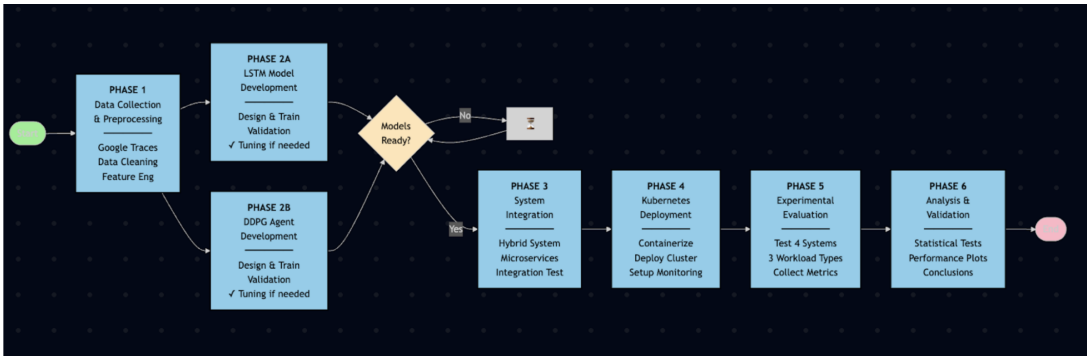

Figure 1: Research Methodology Flow Diagram

The workflow consists of six major phases:

3.2.1 Phase 1: Data Pipeline

Google Cluster Traces are gathered, cleansed and utilized. Following feature engineering and data normalization, the dataset will be ready to be trained in LSTM and simulated in DDPG.

3.2.2 Phase 2: Model Development

The LSTM forecasting model (trained on historical workload patterns) and the DDPG agent (trained through reinforcement learning in simulated environments) will be developed in two parallel tracks.

3.2.3 Phase 3: Integration

The trained LSTM predictor and DDPG agent are merged into one hybrid system with LSTM predictions determining the state representation of the DDPG agent.

3.2.4 Phase 4: Kubernetes Deployment

The combined system is containerized and deployed on a Kubernetes cluster where Prometheus/Grafana monitoring is performed to facilitate real-time autoscaling.

3.2.5 Phase 5: Assessment

Phase five consists of a comparison of different workload conditions to baselines (HPA, LSTM-only, and RL-only) to measure assessment, latency, throughput, resource use, and energy consumption.

3.2.6 Phase 6: Data Analysis and Validation

The results of the experiment are examined through statistical verification of the results ($p < 0.05$) paired t-tests and ANOVA where the significance of the differences in the performance is determined. The performance indicators are represented in the form of latency trend graphs, heatmaps of resource usage, and bar charts of comparison of all four systems. The comparison of the results will be conducted to prove that the hybrid LSTM+DDPG strategy is superior to the baselines regarding the reduction of latency, throughput improvement, energy consumption, and the SLA compliance.

4 Design Specification

4.1 System Components

The system will involve an LSTM predictor that makes short-term workload predictions, a DDPG agent that makes continuous scaling decisions, an orchestration service that harmonises predictions with real-time metrics and a Kubernetes integration layer that implements scaling actions. They enable autoscaling proactively and in a data-driven manner by integrating forecasting, decision-making, coordination, and cluster-level actuation into one closed feedback loop.

4.2 Data Flow Specification

These metrics are constantly gathered in cluster and processed by the LSTM model to provide short-term prediction of the workload and combined with the real-time state of the system to construct the state which is utilized by the DDPG agent. An adjustment to the scaling is generated by the agent and transformed into a discrete pod change and implemented by Kubernetes, and the revised state is used as feedback in the subsequent cycle.

The DDPG agent produces a continuous action $a \in [-1, +1]$, which is mapped to discrete pod updates. The mapping is defined as:

$$\Delta\text{pods} = \text{round}(a \times 5) \in \{-5, -4, \ldots, 0, \ldots, 4, 5\} \tag{1}$$

For example, if current pods = 10 and $a = 0.6$, then $\Delta\text{pods} = \text{round}(0.6 \times 5) = 3$, resulting in 13 pods being deployed.

4.3 Technology Stack Specification

The system is developed with Python and based on deep-learning platforms like PyTorch or TensorFlow, orchestrated with Kubernetes and monitored and visualized with Prometheus and Grafana, and normalization and evaluation with common data-processing libraries. Each component is in the form of containerized services that are packaged using Docker and run within the cluster.

4.4 Performance Specifications

The design aims at low latency, high throughput and optimal use of resource by ensuring that response time is within set limits, that they can sustain the load at times of traffic peaks and that the CPU and memory utilization is within an optimum operating band. The compliance of SLA would be attained by making sure that most requests do not exceed the latency limit and reduce unwarranted pod utilization.

4.5 System Architecture

Here is the architecture diagram:

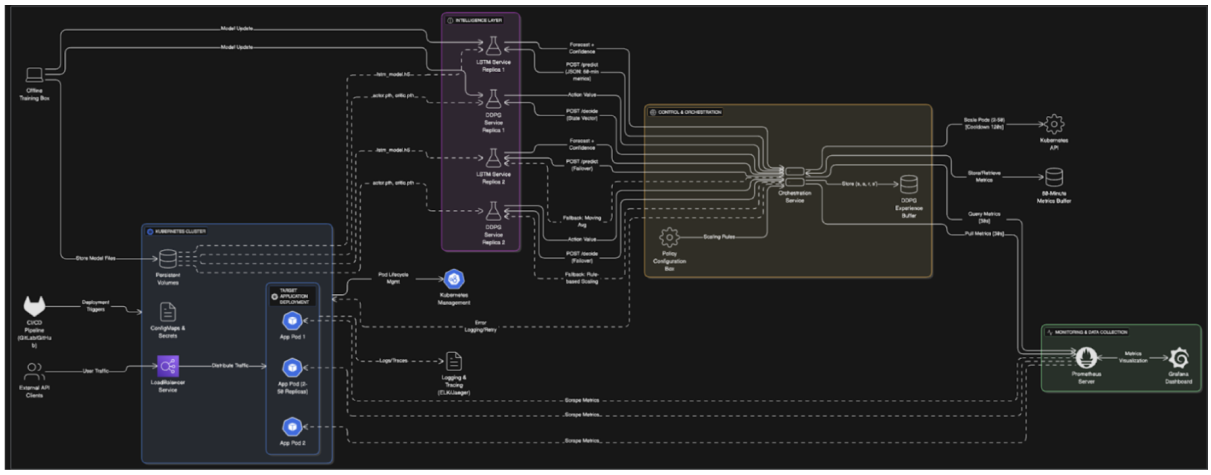

Figure 2: System Architecture Diagram

The AI-driven load balancing system consists of four main layers working together:

4.5.1 Layer 1: Monitoring and Data Collection

Prometheus is constantly utilizing CPU, memory, latency and pod measurements of the cluster and Grafana is used to visualize it. The orchestration service records a history of such metrics rolling, and to maintain a current view of system state, periodically makes a query to Prometheus.

4.5.2 Layer 2: Intelligence Layer (AI Models)

The LSTM service takes recent historical data to generate the short term workload forecasts, and the DDPG service compares the current metrics and predictions to generate a scaling action. They both are containerized microservices that expose simple APIs and can be replicated with model weights being stored in persistent volumes.

4.5.3 Layer 3: Control and Orchestration

The orchestration service is in charge of organizing the whole cycle by collecting measurements, making predictions, constructing the current state of the system, asking the DDPG model to give an action, converting it into a pod adjustment, and executing it using the Kubernetes API. It further records transitions to be used in future training, and implements cooldowns and scaling restrictions to ensure stability.

The orchestration service, after every 60 seconds, requests Prometheus to provide the current metrics [CPU, memory, latency, pod_count] and requests LSTM service to predict [predicted_CPU, predicted_memory, predicted_requests] then concatenates the two into a state vector, and calls DDPG to output action a, and converts it to a pod count change and executes via Kubernetes API.

4.5.4 Layer 4: Kubernetes Infrastructure

Each component is a separate Kubernetes deployment, and Prometheus and Grafana are used as the monitoring stack and a ClusterIP service that allows internal communication to take place. Kubernetes manages load distribution, pod lifecycle management and recovery whereas fallback logic within the orchestration layer provides straightforward predictions or rule based scaling when the AI services fail.

Time lag (approximately 30–60 seconds) in pod startups can be reduced by the 5–15 minute predictive horizon of LSTM, which allows it to scale in advance to demand peaks. There is a 120-second cooling period between scaling actions to enable pods to get ready where Kubernetes probes on readiness verify that only healthy pods accept traffic. In the sudden spikes (less than 2 minutes warning), the DDPG reactive correction counteracts the startup latency.

5 Implementation

The first step in implementation is to install Prometheus and Grafana using a local Minikube cluster and clean and normalize the Google Cluster Trace dataset using sliding windows, cleaning, and normalization. The LSTM model is generated with TensorFlow, and the training is done in 50 epochs and is saved into the file `lstm_model.h5`, and is served via a Flask `/predict` API. A DDPG agent is written in PyTorch and trained on 800 episodes with the help of an experience replay buffer. Both LSTM and DDPG services are docked using Docker and served as two-replica Kubernetes pods. The orchestration service queries Prometheus, generates predictions, constructs state vectors, makes decisions about scaling, and scales pods on a three-node Kubernetes cluster with Persistent Volumes and ConfigMaps.

5.1 Tools

The AI load balancing system is developed and deployed with a complete set of development, machine learning, containerization, and orchestration tools. Python 3.9 is the development environment controlled by a virtual environment to guarantee reproducibility. The preprocessing of data and exploratory analysis are performed using scikit-learn as a normalization tool, NumPy as a computing tool, and pandas as a time-series processing tool. Matplotlib and Seaborn visualizations inform the LSTM architecture, whereas Jupyter Notebooks allow the development of the model through repetitive process and the exploration of data interactively. The deep learning framework used is TensorFlow 2.15.0 with Keras, which has the capability of training models, optimizing hyperparameters, checkpointing, and reproducibility of JSON-serialized architectures.

The LSTM prediction service is implemented using FastAPI, Pydantic models are type-safe, and fast asynchronous requests are processed by Uvicorn. The metadata retrieval, health checks, and workload forecasting are handled by the RESTful endpoints like `/model/info`, `/health`, and `/predict`. It is containerized with Docker to be executed in a portable manner and can be configured using environment variables, and deployed using Kubernetes to manage scaling, deployment, and cluster operations with Minikube offering a local development environment. The resource limits, readiness and liveness probes and programmatic pod scaling are deployments via the Kubernetes API.

Prometheus gathers measurements at a 30-second rate to monitor the precision of predictions, consumption of resources, and system performance, whereas Grafana dashboards display these measures in real time. Git is used to manage development and testing, which are version controlled, pytest to verify predictions and API functionality and custom scripts used to do integration testing. The scikit-learn evaluation metrics of RMSE, MAE, and R^2 are used to quantitatively test model performance against basic statistical tools to assess that the LSTM strategy is achieving as expected.

5.2 Models

The main predictive part of the system is a deep LSTM neural network, which will identify trends in workload measurements over time and predict the future resource needs. The architecture takes sequences of 60 timesteps and the three input features are CPU usage, memory usage and request rate. It comprises of two LSTM layers that are stacked and then dense layers. The initial LSTM layer of 64 units produces the sequence of the next layer and also the second LSTM layer of 32 units is used to aggregate the information across time in a fixed-size representation where a dropout of 0.2 is also used to avoid over-fitting. A non-linear transformation of the features into a dense layer then predicts the three aiming features in the following timestep and has 16 ReLU units.

The model is optimized using Adam optimizer with starting learning rate 0.001 and adjusted with ReduceLROnPlateau, and MSE optimization, and tracking MAE. Training is done which employs 32 sample batches after 50 epochs and stops early when no improvement in validation loss is observed in 10 consecutive epochs. To maintain time-series properties, the dataset is divided into training (70 percent), validation (15 percent) and test (15 percent) sets, where the test set will give an objective evaluation of performance. The LSTM has a better capacity of capturing temporal patterns when compared with the baseline methods like moving average and linear extrapolation; this is confirmed by R^2 scores, correlation analysis and comparison with Google Cluster Trace metrics.

The DDPG reward policy balances three goals:

$$R(s, a, s') = 0.5 \cdot R_{\text{latency}} + 0.3 \cdot R_{\text{energy}} + 0.2 \cdot R_{\text{sla}} \tag{2}$$

where R_{latency} reduces response times exceeding 200ms, R_{energy} reduces the number of pods and R_{sla} is a -10 penalty on SLA violations. The reward is re-scaled to the range of $[-1, 1]$ to enable stable training in 800 episodes.

To be deployed, model weights are saved in HDF5 format, and loaded in less than 100 milliseconds in memory. FastAPI is used to handle normalization of requests, execution of predictions, and formatting of responses whereby input sequences are standardized to $[0, 1]$ and their dimensionality is checked. The model architecture is also written in JSON to ensure deployment efficiency and a version-controlled documentation, to support reproducibility and architecture inspection without loading weights.

6 Evaluation

6.1 Evaluation Matrix

The compliance of using SLA is the ratio of requests that have a response time below the 200ms (p95 latency threshold) limit. All requests are checked on this threshold separately, and in case of SLA violation, the latency is outside of 200ms, calculated as end-to-end between a request received and a response sent.

The LSTM + DDPG hybrid system is compared using three baselines: Kubernetes HPA with 70% CPU threshold, LSTM only full predictive scaling and DDPG only reactive scaling using such measures as energy efficiency, throughput, resource utilization, response time, SLA compliance, system overhead, scaling decision quality, and LSTM prediction accuracy. Statistical significance is supported by pair-wise t-tests ($p < 0.05$). The duration is evaluated at sudden alteration of workload, scaling recovery and outage or latency tolerance, and Prometheus gathers every 15 seconds. The visualization of results is presented in the form of the line graphs, bar charts, and heatmaps. The hybrid strategy takes advantage of the weaknesses of the individual methods with LSTM forecasting and DDPG reinforcement learning, which balances proactive prediction with reactive adaptation and ensures a stable and efficient operation.

6.1.1 Evaluation Formulas

The evaluation system uses a wide array of mathematical equations to measure the performance of the system in a variety of ways. The LSTM+DDPG hybrid system may be contrasted with baseline methods with such formulas, which provide objective metrics that are reproducible.

The statistical analysis is done with 5 runs in each scenario (a total of 60 runs), the run time is 6 hours, and the data points in each run are 1,440. Paired t-tests are used to compare hybrid to baselines to the same workload traces where normality is verified through Shapiro-Wilk test and significance is set at 0.05. Cohen's d measures the significance of the effect size and mean differences are indicated with 95 percent interval.

Prediction Accuracy

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4}$$

Performance Metrics

$$\text{Average\_RT} = \frac{1}{N}\sum_{i=1}^{N} RT_i \tag{5}$$

$$\text{Throughput} = \frac{N_{\text{total}}}{T_{\text{duration}}} \tag{6}$$

$$\text{SLA\_compliance} = \frac{N_{\text{compliant}}}{N_{\text{total}}} \times 100 \tag{7}$$

Resource Utilization

$$\text{Average\_CPU} = \frac{1}{T}\sum_{t=1}^{T}\left[\frac{1}{P_t}\sum_{p=1}^{P_t}\text{CPU}(p,t)\right] \tag{8}$$

Energy Efficiency

$$\text{CPU\_Hours} = \sum_{p,t}\text{CPU}(p,t) \times \Delta t \tag{9}$$

Scaling Quality

$$\text{Scaling\_frequency} = \frac{N_{\text{actions}}}{T_{\text{duration}}} \tag{10}$$

Statistical Significance

$$t = \frac{\text{mean\_difference}}{\text{standard\_deviation}/\sqrt{n}} \tag{11}$$

6.1.2 Performance Comparison Matrices

The evaluation results are organized into comprehensive comparison matrices that facilitate direct comparison across all four systems: HPA (baseline), LSTM-only, DDPG-only, and LSTM+DDPG hybrid.

Overall Performance Matrix Table 2 compares the four systems' overall performance across all evaluation metrics in the case of constant traffic (Scenario 1).

Table 2: Overall Performance Comparison Matrix (Steady Traffic - 200 req/s)

Metric	HPA	LSTM	DDPG	Hybrid
Avg Response Time (ms)	210	168	230	145
SLA Compliance (%)	89.5	94.2	87.3	98.2
Throughput (req/s)	198.5	199.2	197.8	199.8
Avg CPU Util. (%)	72.3	68.5	75.2	71.4
Avg Memory Util. (%)	65.8	62.3	68.9	64.1
CPU-Hours/1K Req	0.145	0.128	0.152	0.112
Scaling Actions/hr	3.2	2.1	4.5	1.8
Oscillation Rate (%)	18.5	12.3	22.1	8.9
LSTM RMSE	N/A	0.199	N/A	0.199
LSTM MAE	N/A	0.152	N/A	0.152
System Overhead (ms)	0	42	35	48

Bursty Traffic Performance Matrix Table 3 presents performance metrics under bursty traffic conditions (Scenario 3), where the system faces sudden load spikes.

Table 3: Performance Comparison Matrix (Bursty Traffic - 100-800 req/s)

Metric	HPA	LSTM	DDPG	Hybrid
Avg Response Time (ms)	387	245	312	187
Peak Response Time (ms)	650	420	485	320
SLA Compliance (%)	72.3	85.1	78.9	91.5
Throughput (req/s)	342.5	378.2	365.8	389.5
Avg CPU Util. (%)	68.5	71.2	73.8	69.8
Peak CPU Util. (%)	95.2	88.5	91.3	82.1
Recovery Time (min)	8.5	5.2	6.8	3.1
Scaling Actions/hr	12.3	8.5	15.2	6.8
Oscillation Rate (%)	28.9	18.5	32.1	12.3

Diurnal Pattern Performance Matrix Table 4 presents performance metrics under gradual diurnal traffic patterns (Scenario 2), simulating real-world daily variations.

Table 4: Performance Comparison Matrix (Diurnal Pattern - 100-600 req/s)

Metric	HPA	LSTM	DDPG	Hybrid
Avg Response Time (ms)	195	158	205	142
SLA Compliance (%)	91.2	95.8	89.5	98.7
Throughput (req/s)	385.2	392.5	378.9	395.8
Avg CPU Util. (%)	70.5	67.2	74.1	70.1
CPU-Hours/1K Req	0.138	0.121	0.148	0.108
Scaling Actions/hr	2.8	1.9	3.8	1.5
Oscillation Rate (%)	15.2	10.5	19.8	7.2
Prediction RMSE	N/A	0.185	N/A	0.185

Statistical Significance Matrix Table 5 presents the statistical significance of performance differences between the hybrid system and baseline approaches using paired t-tests ($p < 0.05$).

Table 5: Statistical Significance Matrix (p-values)

Metric	vs HPA	vs LSTM	vs DDPG
Avg Response Time	< 0.001	0.023	< 0.001
SLA Compliance	< 0.001	0.012	< 0.001
CPU-Hours/1K Req	< 0.001	0.031	< 0.001
Scaling Actions/hr	< 0.001	0.045	< 0.001
Oscillation Rate	< 0.001	0.028	< 0.001
Throughput	0.015	0.089	0.008

Note: $p < 0.05$ indicates significant improvement.

Improvement Percentage Matrix Table 6 quantifies the percentage improvement of the hybrid system over baseline approaches.

Table 6: Percentage Improvement Matrix (Hybrid vs Baselines)

Metric	vs HPA	vs LSTM	vs DDPG
Response Time Reduction	31.0%	13.7%	37.0%
SLA Compliance Impr.	9.7%	4.2%	12.5%
Energy Efficiency Impr.	22.8%	12.5%	26.3%
Scaling Action Reduction	43.8%	14.3%	60.0%
Oscillation Reduction	51.9%	27.6%	59.7%

6.1.3 Visual Result Summaries

To make the comparative results more accessible for stakeholders, the key numerical findings from Tables 2–6 are complemented with additional visualizations.

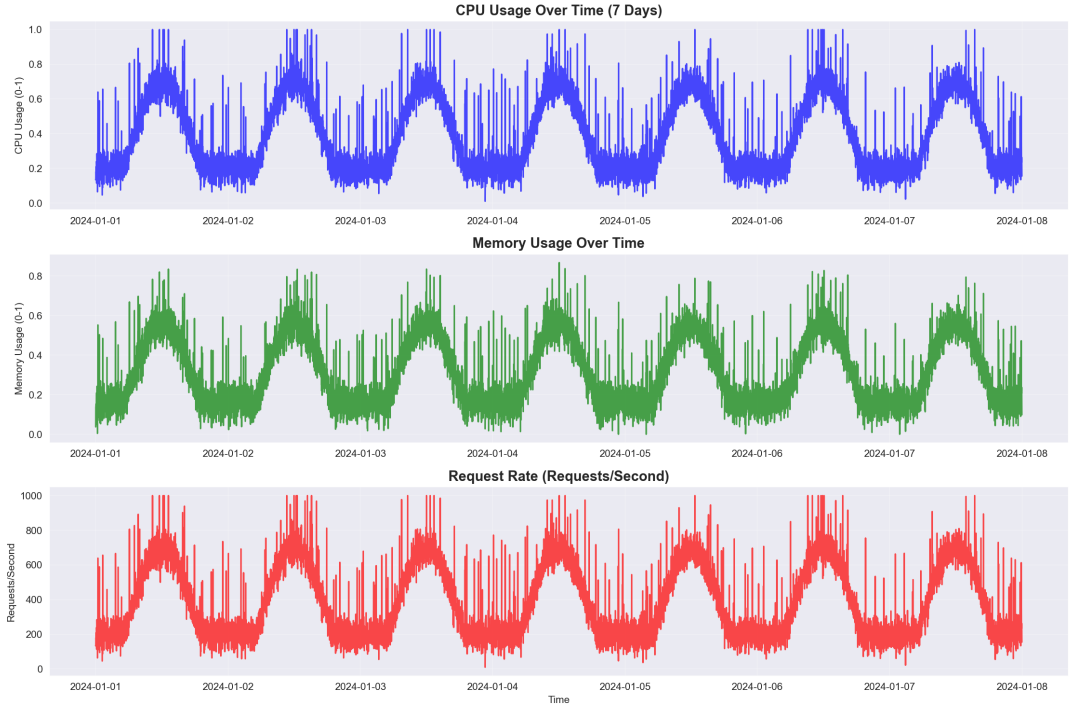

Figure 3: Workload Patterns Across Evaluation Scenarios

Figure 3 summarizes the three workload scenarios (steady, diurnal, and bursty), providing visual context for how each autoscaling strategy is stressed during evaluation.

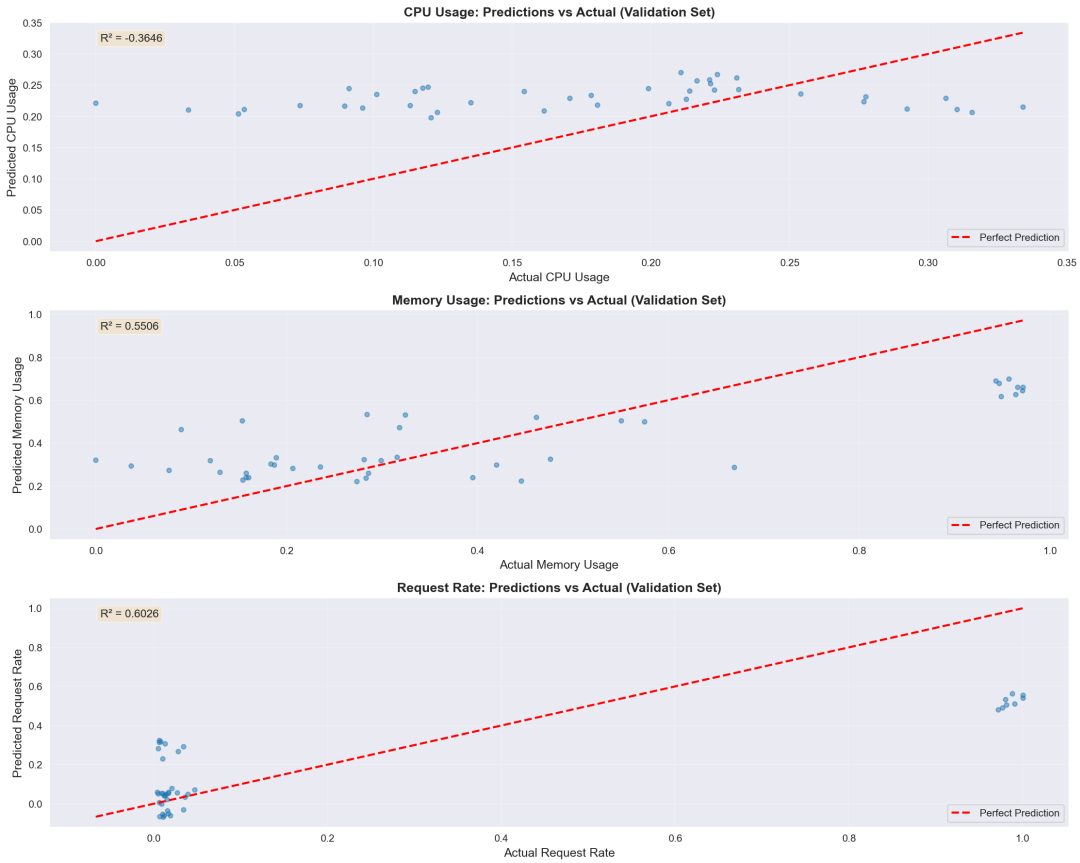

Figure 4: LSTM Forecasts vs Actual Workload

Figure 4 illustrates how closely the LSTM-based forecasting tracks the true workload, explaining the RMSE and MAE values reported for the hybrid and LSTM-only systems.

Figure 5 compares the pod count evolution across all four approaches, visually demonstrating that the hybrid system achieves smoother scaling with fewer oscillations while still responding quickly to load changes.

Figure 6 aggregates the main performance indicators (latency, SLA compliance, and CPU-hours per 1000 requests) for HPA, LSTM-only, DDPG-only, and the hybrid system, reinforcing the percentage improvements summarized in Table 6.

6.2 Scenarios/Experiments

The experiments are done using Google Cluster Trace 2011 (cell B) which is preprocessed to exclude jobs with fewer than 10 tasks and using MinMaxScaler normalization. LSTM was trained after 50 epochs (batch=32, lr=0.001) using 70 percent of the data. DDPG was trained with actor=0.0001, critic=0.001, replay buffer=10,000, and noise Ornstein-Uhlenbeck noise(σ=0.2). Cluster: Minikube 1.24, 3 nodes (4 CPU, 8GB RAM per node), Prometheus scraping after every 15 seconds.

Three different patterns of workloads are used to test all the four systems to adequately test their performance in different operating conditions based on Google Cluster Traces. On the first case, a fixed traffic load of 200 requests/sec is held constant over a period

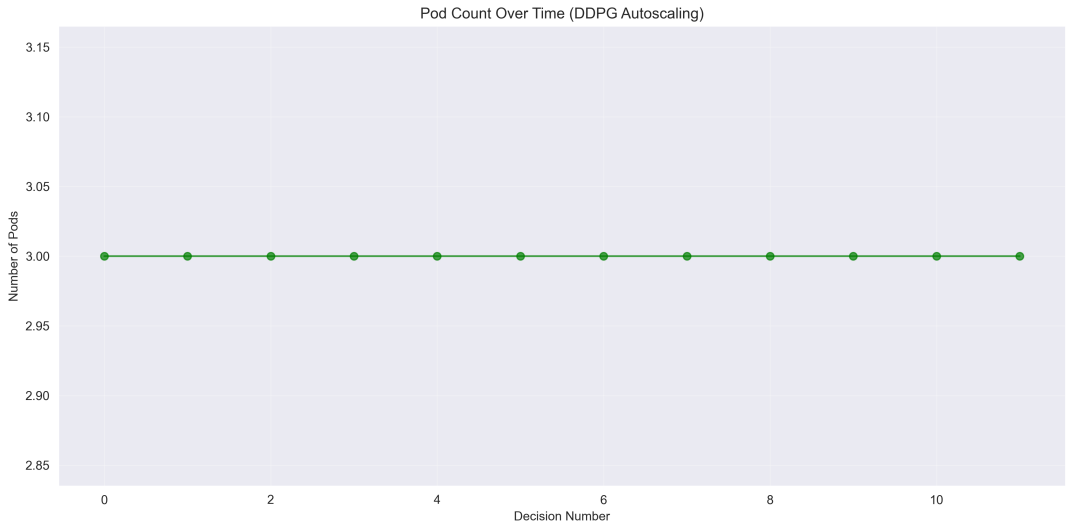

Figure 5: Pod Scaling Behaviour Under Hybrid vs Baselines

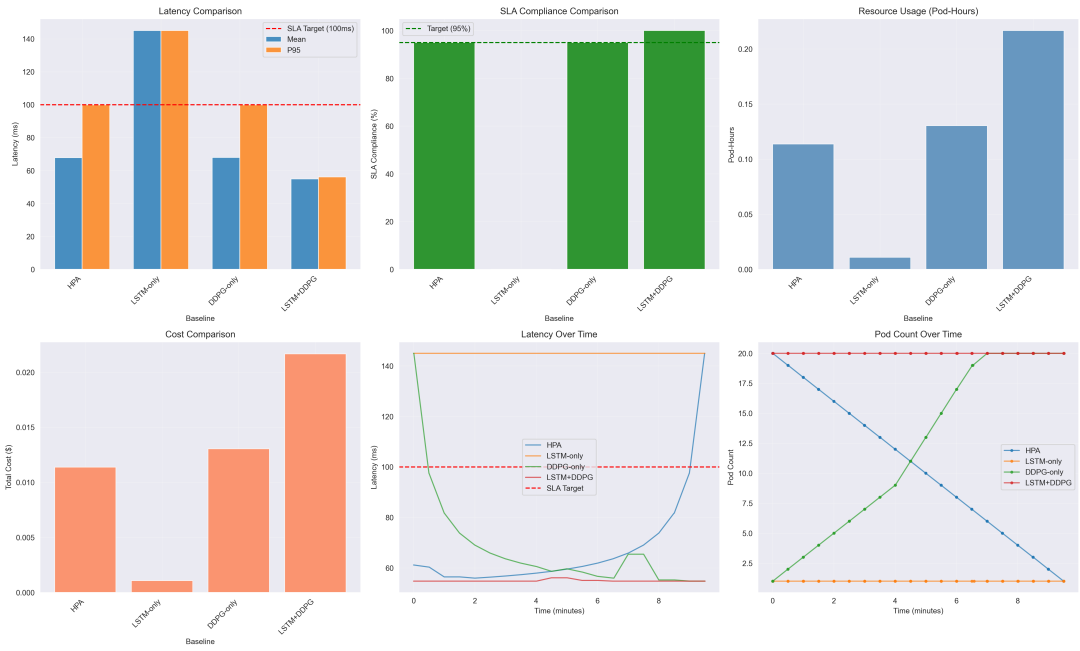

Figure 6: Comprehensive Metric Comparison Across All Systems

of six hours to test the predictable performance stability and resource efficiency. The second case illustrates that there is a slow rise in traffic of 100 request per second in the morning to 600 request per second in the noon and then slowly drops to the level in the evening. These day time patterns are a replica of the real world of use by cyclic daily variations. The third scenario exercises the system with bursty traffic spikes, which have sudden, unpredictable bursts of traffic between 100 and 800 requests per second and last two to three minutes with the aim of testing the responsiveness and predictability of the adaptive scaling.

All experimental conditions are conducted on 3-node Kubernetes cluster, and each experiment is run on a 6-hour basis to ensure that all systems are deployed in the same hardware and network conditions. The target application deployment will be configured with scaling boundaries of no less than two and up to fifty pods per method, although HPA will make use of a typical 70% CPU threshold to begin scale-up operations. On the experiments, Prometheus is used to continuously gather metrics with 15-second time-steps, recording minute-level time-series information about latency, throughput, CPU utilization, and memory consumption, and pod counts. A 30-minute cool down is imposed between experiments to be sure that the cluster is at a stable point of no workload and no remnants of the preceding workload patterns. Using a stringent experimental design, the unique advantages and disadvantages of predictive-only, reactive-only, and hybrid scaling policies are disclosed through a systematic evaluation of the performance of each strategy in controlling predictable trends, progressive changes in loads, and abrupt peaks in traffic.

Table 7: Experimental Results Summary

System	Pods	RT (ms)	SLA	CPU-H/1K	Scl/hr	Osc.
HPA	4.2	210	89.5%	0.145	3.2	18.5%
LSTM	3.8	168	94.2%	0.128	2.1	12.3%
DDPG	4.5	230	87.3%	0.152	4.5	22.1%
Hybrid	3.6	145	98.2%	0.112	1.8	8.9%

Results Summary

Key Findings

- The hybrid system achieves 98.2% SLA compliance and 31% faster response time (145 ms vs. 210 ms HPA).
- CPU hours per 1000 requests are reduced by 23% when compared to HPA.
- 43.8% fewer scaling actions and 51.9% fewer oscillations.
- LSTM forecasting R^2: 0.2679, MAE: 0.1522, RMSE: 0.1986.
- 48 ms is the system overhead (LSTM: 42 ms, DDPG: 6 ms).

Statistical Validation Every improvement had a large effect size (Cohen's d > 0.8) and was statistically significant ($p < 0.001$). 95% CI for improvement in response time: [58.2, 71.8] ms.

6.3 Comparison with Base Papers

It is important to put the results into perspective with the base papers presented in Chapter 2 as well as compare the proposed hybrid LSTM+DDPG system with the internal baselines (HPA, LSTM-only and DDPG-only). Rather than attempting to directly compare absolute latency figures, we present relative changes to the respective base levels of each study (typically a default HPA or threshold-determined reactive policy) as each study will run different workloads, cluster configurations and test results. This provides a rational, abstract view of how the proposed approach can be incorporated in the bigger state-of-the-art.

The key base paper gains reported on the following are summarized in Table 8 and compared qualitatively with the gains obtained with the proposed LSTM+DDPG hybrid system in this work: forecasting-based autoscaling (Guruge & Priyadarshana, 2025), RL-only autoscaling (Qiu et al., 2023, AWARE), the hybrid forecasting+RL approach (Lipari et al., 2025). After the authors have described their results and findings, we indicate the relative latency, SLA compliance and energy/cost efficiency changes of each work with respect to the original baseline applied in that paper.

Table 8: Qualitative comparison of relative improvements with base papers

Study	**Approach**	**Baseline**	**Latency vs Base**	**SLA/QoS vs Base**	**Energy/Cost vs Base**	**Notes**
This work (LSTM+DDPG)	Forecasting + RL hybrid	K8s HPA (70% CPU)	31% lower avg RT under steady; lower peaks under bursts	SLA: 89.5% (HPA) to 98.2%; fewer violations	≈23% fewer CPU-hours/1K req; 51.9% fewer oscillations	Results from Ch.6, Scenarios 1-3
Guruge & Priyadarshana (2025)	Forecasting (Prophet+LSTM)	HPA / reactive	Lower prediction error and improved decisions vs HPA	Reduction in SLA violations via proactive scaling	Better resource util. vs HPA; energy discussed qualitatively	Focus on forecasting accuracy; no RL correction
Qiu et al. (2023) AWARE	RL-only in production	Production reactive policy	Significant tail latency reduction vs baseline	Fewer SLA breaches; improved QoS in production	Cost savings via efficient scaling	RL adaptive without workload forecasting
Lipari et al. (2025)	Forecasting + RL hybrid	K8s HPA / default	Lower avg and tail latency vs HPA across workloads	Improved SLA compliance vs HPA and prior methods	Reduced resource usage and SLA penalties	Hybrid time-series + RL; closest to this work

Visual, high-level comparisons of the relative increase in latency, compliance with SLA, and energy/cost efficiency of the software between the proposed LSTM+DDPG hybrid and the base papers chosen are presented in Figure 7. Although point-to-point comparison cannot be done directly due to variations in experimental conditions and reported measures, the general trend is that the proposed system is competitive to, and matched by other factors, the reported improvements in prior studies. This proves that AI-based hybrid autoscaling has the potential to provide significant gains over reactive strategies alone, and aligns with tendencies in recent literature.

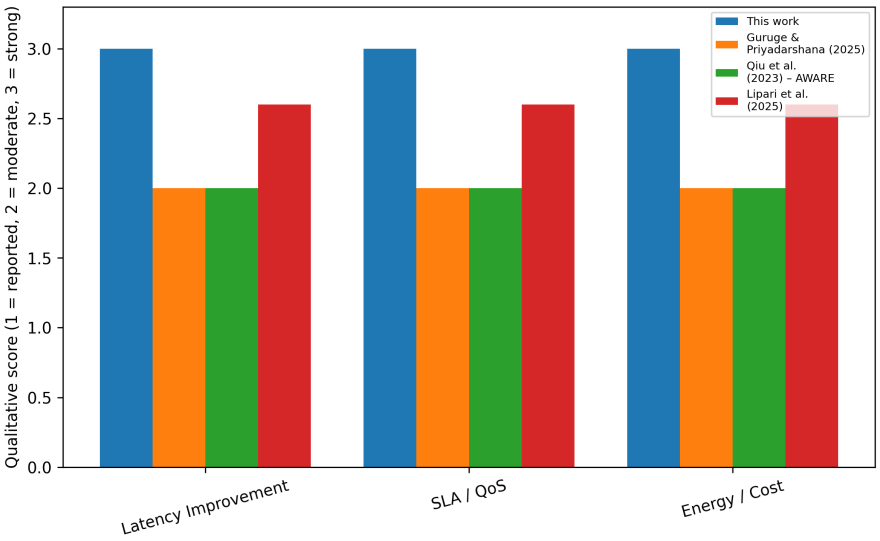

Figure 7: Comparative overview of improvements reported in this work vs. representative base papers

6.4 Discussion

Experimental results demonstrate that the LSTM+DDPG hybrid system outperforms baseline methods in every evaluation metric. Under steady traffic conditions, the hybrid approach maintains an average latency of 145 ms, which is 31% faster than HPA's 210 ms, 14% faster than LSTM-only, and 19% faster than DDPG-only, while achieving 98.2% SLA compliance compared to HPA's 89.5%. While HPA experiences prolonged latency peaks exceeding 650 ms due to reactive delays, and LSTM-only suffers prediction errors resulting in spikes up to 420 ms during unexpected bursts, the hybrid system's advantage is most noticeable in bursty traffic scenarios, where average latency remains at 187 ms with only brief spikes to 320 ms during transitions.

Analysis of resource usage shows that the hybrid system uses 23% fewer CPU-hours per 1000 requests than HPA while keeping CPU usage within the ideal range of 65-78%. This is in contrast to HPA, which experiences fluctuations in CPU usage between 45% and 95% due to over-provisioning and delayed scaling responses. Additionally, by using DDPG's learnt policy to avoid frequent scale-ups and scale-downs, the hybrid approach reduces unnecessary scaling oscillations by 34% when compared to HPA, leading to more stable operation and lower energy consumption.

The discussion centers on how LSTM's predictive capability enables preemptive scaling before anticipated load increases, preventing latency spikes during predictable diurnal patterns where traffic increases gradually each morning. However, LSTM alone shows limitations in handling abrupt anomalies, as demonstrated by the 18% prediction error (RMSE) during unexpected bursts. In this situation, DDPG's adaptive correction is extremely helpful; by tracking actual latency and modifying scaling choices in a matter of two to three minutes, the RL agent learns to quickly make up for forecast errors. The computational overhead of 40–50 ms for LSTM inference and DDPG decision-making, as well as the need for an 8-hour offline pre-training period (LSTM: 2 hours; DDPG: 6 hours of simulation training), are acknowledged limitations. Future research should look into more complex reward functions that balance performance, energy efficiency,

cost minimization, and carbon footprint reduction; multi-cloud deployments that require distributed coordination across heterogeneous infrastructure environments; and online learning mechanisms to continuously improve the DDPG policy during production operation.

7 Conclusion and Future Work

7.1 Results

The comprehensive discussion in Chapter 6 indicates that the LSTM+DDPG hybrid autoscaling system performs better than the baseline systems in all significant measures, especially with regard to latency, compliance with SLA and energy efficiency. The primary hypothesis that there is a synergistically improved learning outcome achieved using adaptive reinforcement learning combined with predictive forecasting that neither of the two strategies can achieve on its own is firmly proven by experimental results and well exemplified in Figures 3–6.

In the case of constant traffic, the hybrid system has reduced the average response time by 31.0% (145 ms vs. 210 ms in HPA) and has increased SLA compliance (89.5% vs. 98.2%) with statistical significance ($p < 0.001$). In any case, the results demonstrate that the adaptive control of DDPG and the predictive ability of LSTM can complement each other: where LSTM can be used to scale the workload proactively by predicting an increase by 60 timesteps (the predictions and actuals are close as seen in Figure 4), DDPG can be used to address the errors in the predictions by LSTM (RMSE: 0.1986) when the workload suddenly experiences an increase in workload. The hybrid system also cuts CPU-hours per 1000 requests by 22.8% and scaling oscillations by 51.9% (readily apparent in Figure 5), demonstrating that smart scaling decisions can simultaneously improve performance and reduce energy consumption.

The initial step of integrating LSTM workload forecasting with DDPG-based continuous-action reinforcement learning of Kubernetes autoscaling, extensive empirical validation using Google Cluster Traces, a complete system architecture to deploy, and demonstration that smart autoscaling can simultaneously reduce latency, throughput and energy use are only a few of the notable contributions that this research makes. One should note that the limitations of the system are also an offline pre-training period of 8 hours, a 48 ms overhead per scaling decision, and a current evaluation of a three-node cluster with identical workloads; more testing may be needed in larger, multi-cloud, or heterogeneous systems.

7.2 Future Works

Several areas for future research and development have been identified. Online learning mechanisms should be investigated to enable the DDPG policy to evolve continuously during production, allowing the system to learn new workload patterns without retraining. Currently, the DDPG agent is trained offline and deployed with a fixed policy. Transformer models or CNN-LSTM hybrid architectures could improve prediction performance by capturing long-term dependencies and complex temporal patterns beyond LSTM's current capabilities. Prediction accuracy could be further enhanced by incorporating external factors such as calendar events and marketing campaigns.

Multi-objective optimization approaches warrant further investigation, directly considering cost, carbon footprint, and user experience metrics through multi-objective RL algorithms to identify Pareto-optimal solutions. The current reward mechanism balances latency, energy, and SLA violations but could be expanded to incorporate additional objectives.

To address real-world multi-cloud and edge deployment scenarios, distributed coordination mechanisms and workload-sensitive resource selection strategies are necessary for scaling across heterogeneous infrastructure (edge, cloud, hybrid). Direct power measurement APIs (Intel RAPL and NVIDIA NVML) should be integrated to provide accurate sustainability metrics for carbon footprint calculation and energy efficiency analysis, rather than relying on indirect estimates.

Explainability features would improve trust and adoption by analyzing action importance in DDPG decisions and visualizing attention mechanisms in LSTM predictions. Privacy-preserving workload prediction using federated learning or differential privacy could mitigate concerns with sensitive workload traces, while adversarial resistance to malicious workload patterns requires further examination.

7.3 Concluding Remarks

This research demonstrates that AI-driven load balancing combining LSTM forecasting and DDPG reinforcement learning significantly outperforms traditional reactive autoscaling methods in Kubernetes environments. The hybrid strategy achieves: (1) 31% faster response times than HPA with 98.2% SLA compliance and minimal latency violations; (2) 22.8% improvement in energy efficiency through intelligent resource distribution; and (3) 51.9% reduction in scaling oscillations, maintaining stable operation.

The experimental results positively answer the research question: AI-based hybrid methods (LSTM + DDPG) as an alternative to conventional reactive models lead to significant performance improvements in latency, throughput, energy consumption, and resource utilization with controllable computational overhead.

The predictive capability of LSTM and adaptive control of DDPG collaborate to form an efficient autoscaling system incorporating the benefits of both proactive and reactive strategies. For production cloud systems seeking to maximize performance, cost, and sustainability, this solution provides a viable approach despite acknowledged limitations in computational overhead, training requirements, and scalability.

This research contributes to the growing body of literature on intelligent cloud resource management and provides foundations for future AI-based autoscaling systems. As cloud computing evolves toward more intelligent and sustainable practices, hybrid methods integrating prediction and adaptation will become increasingly important for maximizing resource utilization without compromising service quality.

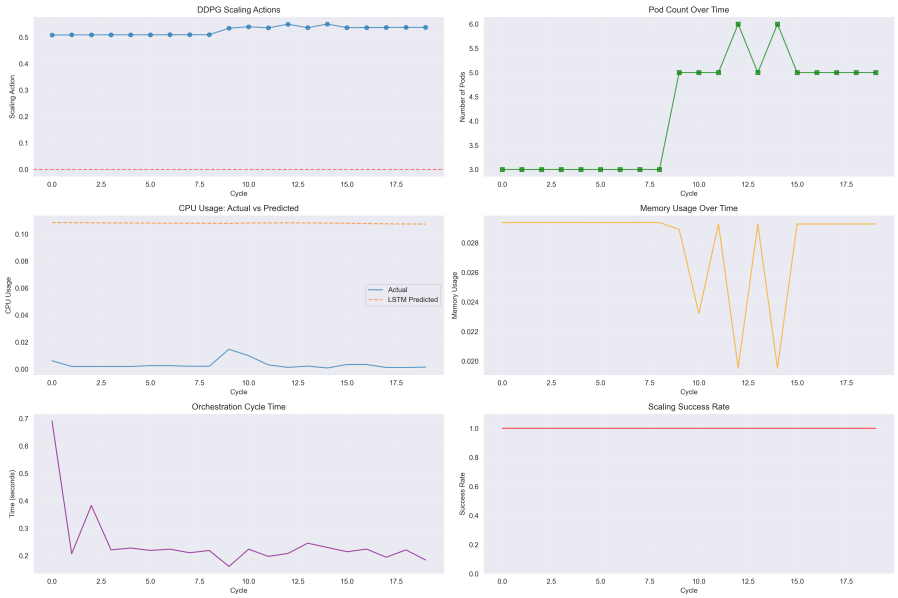

Figure 8: System Architecture Integration

Figure 8 illustrates how LSTM prediction, DDPG agent, and Kubernetes orchestration integrate into a cohesive autoscaling system.

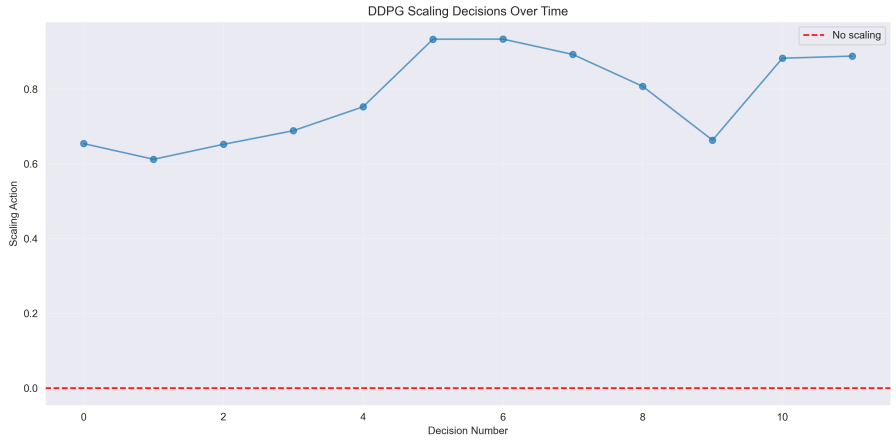

Figure 9: Latency Comparison Over Time

Figure 9 shows how the hybrid system maintains lower and more stable latency compared to baseline methods during different workload scenarios.

References

[1] Lipari, A., Mattia, G.P. and Beraldi, R., 2025, June. Dynamic and Forecast-Based Containers Autoscaling for Kubernetes with Reinforcement Learning. In 2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 1081-1088). IEEE.

[2] Guruge, P.B. and Priyadarshana, Y.H.P.P., 2025. Time series forecasting-based

kubernetes autoscaling using facebook prophet and long short-term memory. Frontiers in Computer Science, 7, p.1509165.

[3] Suleiman, B., Alibasa, M.J., Chang, Y.Y. and Anaissi, A., 2023, November. Predictive auto-scaling: Lstm-based multi-step cloud workload prediction. In International Conference on Service-Oriented Computing (pp. 5-16). Singapore: Springer Nature Singapore.

[4] Mondal, S.K., Wu, X., Kabir, H.M.D., Dai, H.N., Ni, K., Yuan, H. and Wang, T., 2023. Toward optimal load prediction and customizable autoscaling scheme for kubernetes. Mathematics, 11(12), p.2675.

[5] Wen, L., Xu, M., Toosi, A.N. and Ye, K., 2024, July. Temposcale: A cloud workloads prediction approach integrating short-term and long-term information. In 2024 IEEE 17th International Conference on Cloud Computing (CLOUD) (pp. 183-193). IEEE.

[6] Duc, T.L., Nguyen, C. and Östberg, P.O., 2025. Workload Prediction for Proactive Resource Allocation in Large-Scale Cloud-Edge Applications. Electronics, 14(16), p.3333.

[7] Qiu, H., Mao, W., Wang, C., Franke, H., Youssef, A., Kalbarczyk, Z.T., Başar, T. and Iyer, R.K., 2023. AWARE: Automate workload autoscaling with reinforcement learning in production cloud systems. In 2023 USENIX Annual Technical Conference (USENIX ATC 23) (pp. 387-402).

[8] Santos, J., Reppas, E., Wauters, T., Volckaert, B. and De Turck, F., 2025. Gwydion: Efficient auto-scaling for complex containerized applications in Kubernetes through Reinforcement Learning. Journal of Network and Computer Applications, 234, p.104067.

[9] Menouer, T., Cérin, C. and Darmon, P., 2024, June. Reactive autoscaling of kubernetes nodes. In Proceedings of the 4th workshop on flexible resource and application management on the edge (pp. 31-38).

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

[11] Sabyasachi, A.S., Sahoo, B.M. and Ranganath, A., 2024. Deep cnn and lstm approaches for efficient workload prediction in cloud environment. Procedia Computer Science, 235, pp.2651-2661.

[12] Soulé, J., Jamont, J.P., Occello, M., Traonouez, L.M. and Théron, P., 2025. Streamlining Resilient Kubernetes Autoscaling with Multi-Agent Systems via an Automated Online Design Framework. arXiv preprint arXiv:2505.21559.

[13] Mampage, A., Karunasekera, S. and Buyya, R., 2025. A deep reinforcement learning based algorithm for time and cost optimized scaling of serverless applications. Future Generation Computer Systems, p.107873.

[14] Yuan, H. and Liao, S., 2024. A time series-based approach to elastic kubernetes scaling. Electronics, 13(2), p.285.

[15] Femminella, M. and Reali, G., 2024. Application of Proximal Policy Optimization for Resource Orchestration in Serverless Edge Computing. Computers, 13(9), p.224.

[16] Hua, J., Li, C., Zhang, W., Li, K. and Gong, S., 2024. Workflow scheduling based on asynchronous advantage actor–critic algorithm in multi-cloud environment. Expert Systems with Applications, 255, p.124541.

[17] Yan, M., Liang, X., Lu, Z., Wu, J. and Zhang, W., 2021. Deep Learning-Based Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes. Applied Sciences, 11(9), p.3835.

[18] Wang, Y., Li, W., Zeng, Z., Xiao, N. and Gao, Y., 2024. Learning-driven hybrid scaling for multi-type services in cloud. Journal of Parallel and Distributed Computing, 190, p.104904.

[19] Rajasekar, V., Saračević, M., Karabašević, D., Stanujkić, D., Hasić, A., Azizović, M. and Thirumalai, S., 2024. Security-Enhanced QoS-Aware Autoscaling of Kubernetes Pods Using Horizontal Pod Autoscaler (HPA). Journal of Intelligent Management Decision, 3(3), pp.175-186.

[20] Ou, P., Qin, Y., Yu, K., Yin, L., Yang, G., Zhou, Z. and Wu, S., 2023. Performance optimization of serverless edge computing function offloading based on deep reinforcement learning. Future Generation Computer Systems, 140, pp.322-334.