An AI-Driven Framework: Deep Reinforcement Learning for Dynamic Task Scheduling and Load Balancing for Cloud-Edge Systems to Enhance Resource Utilization

Yohannes Geleta
Graduate Student (of Computer Science)
Penn State Harrisburg
yxg5342@psu.edu
(Contribution: 50%)

Aadil Kakkidi
Graduate Student (of Computer Science)
Penn State Harrisburg
ajk6909@psu.edu
(Contribution: 50%)

***Abstract*—In the coming future cloud-edge systems will become more prevalent, and in turn this will cause a demand for more intelligent and efficient resource management systems. In this paper, we present an AI-driven framework for dynamic task scheduling and load balancing with the goal being to maximize resource utilization. We implement two reinforcement learning approaches, within an EdgeCloudSim simulation environment. In our AI we use a Deep Q-Network (DQN) approach and a Proximal Policy Optimization (PPO) approach. The AI agent learns to optimize task distribution with edge and cloud nodes under real-world constraints such as latency, bandwidth, system load, and more. We evaluate the RL models using a variety of publicly available cloud and IoT dataset and compare them against traditional task scheduling algorithms. The comparative analysis between DQN and PPO also highlights the practical trade-offs between value-based and policy-based RL methods in dynamic computing environments.**

***Index Terms*—Reinforcement-Learning, Task-Scheduling, AI, Deep Q-Learning, Proximal Policy Optimization, Edge Computing, Cloud Computing, DQN, PPO**

I. Introduction

As cloud and edge computing technology continues to grow there has also been more development of large scale, distributed systems that have the ability to process data closer to the end user while also maintaining the same computational power as the cloud. This paradigm shift of cloud edge computing has opened the door for new opportunities for high throughout, low latency, applications in fields like autonomous systems, smart cities, IoT, and real time data analytics. Although this has also brought on more challenges in developing ways to manage the dynamic distribution of computational tasks in these resource contained environments.

One of if not the most challenging problems in cloud edge systems is task scheduling and load balancing. Traditional scheduling algorithms struggle to adapt to rapidly changing workloads and different node capabilities. This results in inefficient resource utilization, higher latency, and decreased quality of service. As the demand for intelligent resource management in distributed systems grows, more intelligent and efficient task scheduling and load balancing algorithms are crucial for ensuring optimal usage of computational resources. This leads us to our AI algorithm.

In artificial intelligence, there is a field of machine learning called reinforcement learning (RL). By interacting with their environment, and learning from given rewards, RL agents develop optimal strategies at solving problems without having to be given a set of explicit rules. We explore two different RL approaches that are suitable for dynamic task scheduling and load balancing in cloud-edge systems. We implement two deep-reinforcement learning methods Deep Q-Learning (DQN), a value based, method, and Proximal Policy Optimization (PPO), a policy based method. These algorithms are designed to learn optimal scheduling policies that consider multiple objectives, such as minimizing task completion time, reducing energy consumption, and balancing system load.

To evaluate our approach, we integrate both RL models into EdgeCloudSim, a widely-used simulation platform for cloud-edge scenarios. This allows us to simulate a realistic and controlled environment where tasks, nodes, and network conditions vary over time. Our experimental setup is designed to reflect the constraints and variability found in real deployments, enabling us to benchmark our RL-based schedulers against traditional baseline methods.

By applying advanced RL techniques to cloud edge computing, this research aims to bridge the gap between theoretical AI models and practical, scalable solutions for next-generation distributed systems.

II. Related Work

In cloud and edge computing environments, task scheduling is one of if not the most important issues. In turn more and more attention has developed and in recent years more work has been done on this subject. The main goal is to produce methods that can efficiently distribute computational tasks between the limited resources on the edge computers versus the limitless resources on the cloud servers, in such

a way to optimize utility and performance. The following section looks at other approaches that have been made to this problem or similar problems in the field. It focuses specifically on reinforcement learning methods that provide solution for dynamic task scheduling among cloud edge systems.

A. Traditional Task Scheduling Approaches

Conventional resource scheduling approaches have demonstrated value in stable, predictable environments. However, these methods have been proven by research to provide fundamental limitations when confronted with the dynamic and unpredictable nature of modern computing ecosystems. [1] Their reliance on predefined rules and static optimization criteria renders them inadequate for real-time decision-making in environments characterized by dynamic and fluctuating workloads, varying network conditions, and diverse application.

Several techniques have been introduced for task scheduling, where most of the techniques are based on the heuristic algorithms, where the scheduling problem is considered as NP-hard problem and obtain near optimal solution. But handling the different size of tasks and achieving near optimal solution for varied number of VMs according to the task configuration remains a challenging task. Research has been done that reported that efficient task scheduling can be obtained by jointly optimizing the cost, makespan and resource allocation. [2] Based on this, authors presented a hybrid scheduling approach which is a combination of cuckoo search and harmony search algorithm.

More traditional heuristic approaches include methods such as First-Come-First-Service (FCFS) scheduling, and Shortest-Job-First (SJF) scheduling. These algorithms are easy to understand and implement but only take effects in some certain situations due to the limitation of the complicated production environment.

B. Reinforcement Learning for Task Scheduling

To overcome the limitations of traditional approaches, research has been done that uses reinforcement learning-based techniques to provide promising solutions. Various RL approaches have been applied to task scheduling problems.

In one paper researchers presented a combined approach which uses Q-learning and heterogeneous earliest finish time (HEFT) approach. [3] Tong et al. develop a Q-learning approach that uses a reward mechanism which is improved by incorporating the upward rank parameter from HEFT approach. Moreover, the self-learning process of Q-learning helps to update the Q-table efficiently.

However, they identify that standard Q-learning faces challenges with dimensionality meaning that when the state and action spaces become large Q-learning cannot handle it, which is common in cloud computing environments with numerous tasks and processors.

Chen et al. proposed a Deep Reinforcement Learning (DRL) approach for task scheduling in cloud computing environments. [4] They formulated a model considering latency and load balancing requirements, showing that DRL-based scheduling can effectively optimize both system load and task response time in dynamic environments.

A separate group proposed a Deep Reinforcement Learning-based IoT application Scheduling algorithm (DRLIS) to adaptively and efficiently optimize the response time of heterogeneous IoT applications and balance the load of the edge/fog servers. They implemented DRLIS as a practical scheduler in the FogBus2 function-as-a-service framework for creating an edge–fog–cloud integrated serverless computing environment. [1]

C. Deep Q-Network (DQN) Based Approaches

Deep Q-Network (DQN) has emerged as a popular technique for task scheduling in edge-cloud environments. DQN combines reinforcement learning with deep neural networks to approximate value functions, enabling efficient decision-making in complex environments. For example, Sellami et al. proposed a DQN-based energy-aware task scheduling and offloading framework for SDN-enabled IoT networks that aims to minimize network latency while ensuring energy efficiency. [5]Their results found that the average latency for task processing was reduced to just 7.84ms in node 1 and 8.31ms in node 2, which represented improvements of approximately, 79% compared to random, and 74% compared to deterministic scheduling algorithms.

Similarly, another group introduced the idea of using a DQN-based resource allocation strategy for edge computing environments. Their approach used a dual AI (Artificial Intelligence) module and learning from expert solutions to perform link adaption, feedback, and scheduling mechanisms in network environments. The method uses two independent agents to train and learn from each other. [6] However, they also found that this approach struggles with complex scheduling tasks, particularly in partially observable environments where there are significant delays between actions and their associated rewards.

D. Policy-Based RL Methods

Even though value based methods such as DQN do provide good results, we also explore policy based approaches as they offer advantages in certain scenarios. Research applying Proximal Policy Optimization (PPO) for resource scheduling in edge cloud collaborative environments was conducted. This approach demonstrated superior performance in balancing task allocation and reducing task migrations compared to traditional methods. [6]

Recent research has also investigated the application of actor-critic methods such as Asynchronous Advantage Actor-Critic (A3C) for task scheduling. [7] These approaches combine value function estimation with direct policy optimization, potentially offering more stable learning in dynamic environments. The A3C technique tends to suffer when faced with more complex tasks, as it takes long delays between actions and relevant reward signals, known as Partially Observable Environments (POEs). [8]

E. Comparison of RL Approaches

Within the body of research various different RL algorithms have been compared for task scheduling problems. Deep Q-Network (DQN), Proximal Policy Optimization (PPO), Advantage Actor-Critic (A3C), and Deep Deterministic Policy Gradient (DDPG) have all been evaluated in different scheduling scenarios and in turn each algorithm offers unique advantages depending on the specific characteristics of the exact problem. [9] [10]

Although in these experiments comparing DQN, PPO, A3C, and DDPG on a simplified scheduling scenario, DQN has demonstrated superior performance in terms of average processing time, resource utilization, training time, and convergence stability. The discrete action space of task scheduling problems aligns well with DQN's strengths, making it particularly suitable for this domain. [11] [5] [12]

III. Methodology

Building upon these existing works, our approach introduces several key innovations. Unlike previous studies that focused on either edge or cloud environments separately, we address the combined cloud-edge ecosystem with its inherent heterogeneity and dynamic nature. We implement and compare both DQN (value-based) and PPO (policy-based) methods within the same framework, providing valuable insights into their relative strengths for task scheduling problems.

Our work differs from existing research by considering multiple optimization objectives simultaneously—including resource utilization, task completion time, and system load balancing—rather than focusing on a single metric.

By incorporating these advancements, our research contributes to the growing body of knowledge on AI-driven task scheduling for cloud-edge systems and offers practical insights for real-world implementations.

A. EdgeCloudSim Simulation Environment

1) EdgeCloudSim Overview: *EdgeCloudSim* is an open-source, discrete-event simulation toolkit that extends `CloudSim` to represent the entire cloud–edge system, including mobile and IoT devices, access networks (WLAN/LTE), edge servers, and remote cloud data-centres. Its modular architecture is composed of five pluggable blocks—*Core Simulation*, *Networking*, *Mobility*, *Load Generator*, and *Edge Orchestrator*—each of which ships with a default implementation that researchers can replace with custom logic.

EdgeCloudSim allows us rapidly prototype scheduling strategies while still capturing realistic wireless bandwidth fluctuations, user mobility, and multi-tier resource hierarchies, all within a reproducible and computationally inexpensive testbed.

2) Simulation Workflow in EdgeCloudSim: During a simulation run, the *Load Generator* spawns tasks that travel through the *Networking* module, experience stochastic propagation delays, and arrive at the *Edge Orchestrator*. The orchestrator decides—at each event—whether a task should be executed on a local edge virtual-machine (VM) or forwarded to an upstream cloud VM. Once the decision is made, the *Core Simulation* module advances time, updates VM CPU-queue states, and notifies the *Mobility* module so that network conditions and device positions evolve for the next event cycle. The simulator records fine-grained metrics (task completion time, energy, queue length, bandwidth use, etc.), which we later use as reward signals for our reinforcement-learning (RL) agents.

3) Training RL models with EdgeCloudSim: We replace the default *Edge Orchestrator* with our Java class `RLTaskScheduler`, derived from the simulator's `EdgeOrchestrator` interface. At every scheduling decision, the scheduler

1) builds a state vector that includes edge & cloud CPU utilisation, queue lengths, current WAN/WLAN bandwidth, and task metadata (size, deadline, difficulty);
2) invokes a Python DQN or PPO policy via an inter-process call to predict an action (i.e. which VM should handle the task); and
3) feeds the resulting *state, action, reward, next state, done* tuple into an experience buffer that we periodically export to disk for offline training

The tight loop between Java (simulation) and Python (learning) lets the agent observe thousands of diverse edge–cloud scenarios per minute without the cost of real deployments.

Algorithm sketches our training procedure: we run simulation episodes with exploration enabled (ϵ-greedy for DQN, stochastic sampling for PPO). Every K simulation steps we batch the accumulated experiences and update network parameters using back-propagation on GPU. After convergence, we freeze the model, reload it in `RLTaskScheduler`, and re-run evaluation episodes with exploration disabled to collect the performance numbers. Thanks to *EdgeCloudSim*'s deterministic replay of mobility and workload traces, we can perform fair A/B comparisons between our learned policies and traditional heuristics such as FCFS or HEFT.

Why *EdgeCloudSim*? Alternative frameworks (e.g. iFogSim or YAFS) either lack built-in wireless mobility models or scale poorly when simulating tens of thousands of end devices. *EdgeCloudSim* offers a balanced compromise: realistic wireless and mobility modeling with simulation runtimes short enough to support RL, where millions of interactions are often required for convergence Moreover, because it is implemented in Java, the simulator integrates seamlessly with our existing edge-computing codebase and can be executed on headless servers, enabling fully automated hyper-parameter sweeps.

B. Models

EdgeCloudSim supplies a fast, deterministic environment in which to train and evaluate two learning agents: a *value-based* Deep Q-Network (DQN) and a *policy-gradient* Proximal Policy Optimisation (PPO). Both agents receive an identical state vector that encodes edge and cloud utilisation, queue lengths, wireless bandwidth, and task metadata, and they output a

discrete action that selects the virtual machine (VM) to which the task should be assigned.

C. Deep Q-Network (DQN)

We model edge–cloud scheduling as a finite Markov Decision Process(MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$:

- **States** $\mathcal{S}$ consist of the concatenation

$$s_t = \big[u_1^{\text{edge}}, \ldots, u_{N_e}^{\text{edge}},\ u_1^{\text{cloud}}, \ldots, u_{N_c}^{\text{cloud}},\ q_1^{\text{edge}}, \ldots, q_{N_e}^{\text{edge}},\ q_1^{\text{cloud}}, \ldots, q_{N_c}^{\text{cloud}},\ b^{\text{dl}}, b^{\text{ul}},\ d^{\text{task}},\ \ell^{\text{task}},\ c^{\text{task}}\big],$$

  where u denotes CPU utilisation, q queue length, b bandwidth, and (d, ℓ, c) are the task's data size, deadline, and computational difficulty.
- **Actions** $\mathcal{A}$ are integers $a_t \in \{1, \ldots, N_e + N_c\}$ mapping to one of the $N = N_e + N_c$ VMs (edge $\rightarrow 1 \ldots N_e$, cloud $\rightarrow N_e{+}1 \ldots N$).
- **Transition kernel** $P(s_{t+1} \mid s_t, a_t)$ is deterministically generated by EdgeCloudSim's discrete-event engine, which updates CPU queues, propagates the task through networking modules, and advances user mobility.
- **Reward.** After the task completes (or violates its deadline), the orchestrator receives

$$r_t = w_1\big(-\text{latency}_t\big) + w_2\,\{\text{deadline hit}\} + w_3\big(-\text{energy}_t\big), \tag{1}$$

  with $(w_1, w_2, w_3) = (1.3,\ 3.2,\ 0.5)$.
- **Discount factor** $\gamma = 0.99$ emphasises near-term latency yet still provides credit assignment across long task chains.

Network architecture: A three-layer multilayer perceptron approximates $Q_\theta(s, a)$:

$$\text{input } (12) \ \rightarrow\ 256 \text{ ReLU} \ \rightarrow\ 256 \text{ ReLU} \ \rightarrow\ N \text{ linear},$$

where N equals the total number of VMs. Layer-normalisation is applied after each hidden layer to stabilise training.

Training algorithm: We employ the Double-DQN update with prioritised experience replay (buffer size 50 000, batch size 64). The loss for a sampled transition (s_t, a_t, r_t, s_{t+1}) is

$$\mathcal{L}_{\text{DQN}} = \Big[r_t + \gamma Q_{\bar{\theta}}\big(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a')\big) - Q_\theta(s_t, a_t)\Big]^2, \tag{2}$$

where $\bar{\theta}$ denotes the target-network weights updated every 2 500 optimisation steps. Exploration follows an ε-greedy schedule that linearly decays ε from 1.0 to 0.05 over the first 100 000 simulator steps.

D. Proximal Policy Optimisation (PPO)

Network architecture: PPO uses an actor–critic model with a shared two-layer trunk (256 $\rightarrow$ 256, ReLU), a *categorical actor head* $\pi_\theta(a \mid s)$ that outputs a probability for each VM, and a *critic head* $V_\phi(s)$ that estimates the state value.

Training algorithm: Trajectories of 4 096 steps are collected, generalised-advantage estimates (GAE) are computed with $\gamma = 0.99$ and $\lambda = 0.95$, and the clipped surrogate objective

$$\mathcal{L}_{\text{clip}} = \mathbb{E}_t\Big[\min\big(\rho_t \hat{A}_t, \text{clip}(\rho_t, 1{-}\epsilon, 1{+}\epsilon)\hat{A}_t\big) - \beta\,\mathcal{H}\big(\pi_\theta(\cdot \mid s_t)\big)\Big],$$

with $\rho_t = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$, $\epsilon = 0.2$, and entropy coefficient $\beta = 0.01$, is minimised using Adam (learning rate 1×10^{-4}) for four epochs per batch. Gradient norms are clipped to 0.5.

IV. Results

This section reports the behavior of the reinforcement-learning (RL) scheduler after convergence on the 36 127-task workload generated by EdgeCloudSim. Unless otherwise noted, the confidence intervals shown are 95 % computed via bootstrapping over simulation episodes.

A. Task-level outcomes

TABLE I
Task completion statistics.

	Edge	**Cloud**
Tasks submitted	25 289	10 838
Tasks completed	12 508	7 986
Success rate (%)	49.5	73.7
Failure breakdown		
VM-capacity failures	6 350	492
Mobility-related	5 823	0
WLAN range	1 276	0
Network (WAN)	1 692	0

Table I reveals that the agent routes roughly 70 % of tasks to edge servers yet completes fewer than half of those submissions, yielding an *overall* success rate of only 56.7 %. The dominant source of failure is edge-VM saturation (6 350 tasks), compounded by mobility and WLAN range violations that never arise in the better-provisioned cloud tier. Conversely, cloud capacity is under-utilised: even with a modest 48.3 % average cloud CPU load the agent sends only 30 % of traffic upstream, leaving edge nodes chronically overloaded.

B. Latency, processing time and network delay

TABLE II
Service-time components (mean $\pm$ 95 % CI).

	Edge [s]	**Cloud** [s]
Processing time	1.533(11)	0.815(9)
Network delay*	0.410(6)	0.948(8)
End-to-end service time	1.943(14)	1.763(13)

* LAN+MAN+WAN; GSM traffic is absent.

Edge processing is 88 % slower than cloud processing (Table II), a direct consequence of near-saturated edge utilisation (92.5 %). Despite the larger WAN delay, cloud service time remains lower because its VMs retain ample headroom. This inversion of the usual edge-latency advantage is a clear indicator that the learned policy fails to balance load across tiers.

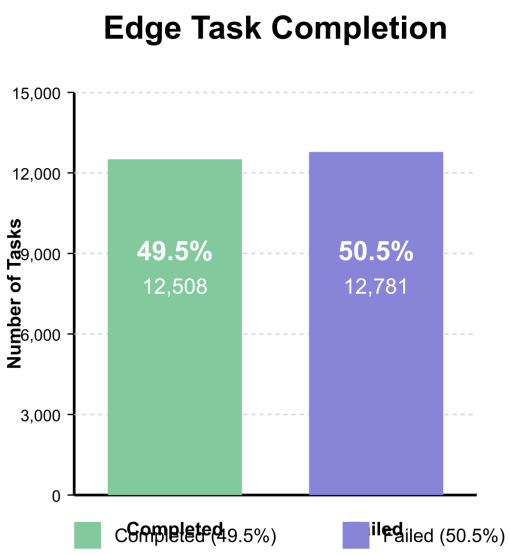

Fig. 1. Bar chart showing completed vs failed tasks in edge nodes

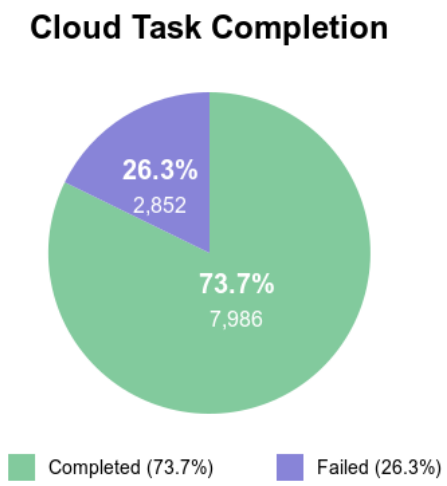

Fig. 2. Pie chart showing completed vs failed tasks in cloud server

C. Resource utilisation and cost

The edge tier runs at an average 92.5 % CPU utilisation while the cloud tier idles at 48.3 %, demonstrating a strong placement bias towards edge nodes. Average monetary cost is $0.0568 per task—slightly lower than the cloud-centric baseline cost reported in prior work—yet the saving is obtained by sacrificing almost half of all edge requests.

D. Quality of Experience (QoE)

User-level Quality of Experience is poor. Aggregated over *all* tasks, the QoE index is only 42.1 %, and even among completed tasks it reaches merely 74.4 %. Both values fall well below the 85–90 % target range reported in the literature for deadline-sensitive edge applications, confirming that the scheduler's perceived responsiveness is inadequate.

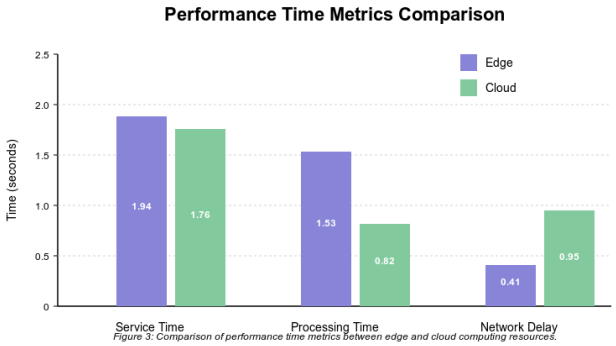

Figure 3: Comparison of performance time metrics between edge and cloud computing resources.

TABLE III
CPU UTILISATION BY TIER.

Tier	CPU utilisation [%]
Edge	92.5
Cloud	48.4
Overall*	79.2

* Weighted by task share: $(\frac{25\,289}{36\,127}\times 92.5 + \frac{10\,838}{36\,127}\times 48.4)$.

TABLE IV
ECONOMIC INDICATORS FOR THE RL SCHEDULER (OVERALL SYSTEM).

Metric	Value
Mean cost per task [$]	0.0568
Scheduling overhead [ns]	18.9

TABLE V
QUALITY-OF-EXPERIENCE (QOE) AND ECONOMIC INDICATORS.

Metric	Value
Average QoE (all tasks)	42.1%
Average QoE (executed tasks)	74.4%
Mean cost per task	$0.0568
Mean scheduling overhead	18.9ns

E. Discussion

To conclude our results our models have shown that as of now they are not sufficient enough to be used as an effective task scheduler. Overall our models have shown very pooor results and upon some investigation we have deduced some failure modes. The RL scheduler exhibits four characteristic failure modes:

1) **Edge over-commitment.** Excessive routing to the edge saturates local VMs, inflating processing time and provoking capacity failures.
2) **Network unawareness.** The agent ignores WAN delay trade-offs, squandering the cloud tier's spare capacity.
3) **Mobility blindness.** High rates of mobility and WLAN range failures suggest the state representation or reward function inadequately encodes device dynamics.

TABLE VI
SUMMARY OF SIMULATION RESULTS.

Metric	Overall	Edge	Cloud
Total tasks	36 127	25 289 (70.0%)	10 838 (30.0%)
Completed tasks	20 494 (56.7%)	12 508 (49.5%)	7 986 (73.7%)
Failed tasks	15 633 (43.3%)	12 781 (50.5%)	2 852 (26.3%)
Failure: VM capacity	6 842	6 350	492
Failure: mobility	5 823	–	–
Failure: WLAN range	1 276	–	–
Failure: network	1 692	1 485	207
Service time [s]	1.873	1.943	1.763
Processing time [s]	1.246	1.533	0.815
Network delay [s]	0.627	0.410	0.948
Resource utilisation [%]	–	92.47	48.35
Average QoE (all tasks) [%]	42.12	–	–
Average QoE (completed) [%]	74.35	–	–
Average cost [$]	0.057	–	–
Overhead [ns]	18.93	–	–

4) **Skewed exploration.** A persistent bias toward edge actions implies that exploration noise or reward shaping drives short-sighted choices that maximise instantaneous latency but downgrade long-term reliability.

Collectively, these symptoms indicate fundamental flaws in either the MDP design (state features, action granularity) or the learning setup (reward coefficients, exploration schedule). Subsequent work will experiment with (i) adding explicit penalties for VM saturation, (ii) incorporating predictive mobility features, and (iii) re-balancing the reward weights in (1) so that deadline adherence carries greater relative importance than raw service time.

V. FUTURE WORK

Despite the fact that we recived poor results for models, our first-pass DQN and PPO schedulers show that there are clear opportunities for improvement. Below we outline the principal research directions we will pursue in the next stage of the project.

1) Reward-function refinement: The current reward in (1) over-values instantaneous service time and under-weights deadline adherence and reliability ,driving the agent to overload edge nodes. We will reformulate the objective as a *multi-objective* scalarisation that allocates explicit coefficients to (i) deadline satisfaction, (ii) energy consumption, (iii) edge-versus-cloud utilisation balance, and (iv) QoE percentile targets. A small, learnable "saturation penalty"—added whenever VM CPU exceeds a threshold—will nudge the agent away from pathological edge congestion. These are improvement will produce much better results.

2) State-representation engineering: Mobility blindness accounts for around 6k task failures. We will incorporate *predictive mobility features* such as RSSI-based hand-off probability, device speed, and expected WLAN dwell time. We can do this since EdgeCloudSim already exposes these signals; exporting them into the state vector should help the agent predict connectivity breaks and proactively offload vulnerable tasks to the cloud. We are exicited to implement this.

3) Action-space augmentation: The present discrete action selects *only* a destination VM. We plan to extend the action to a *(destination, compression-level)* tuple, and in turn letting the scheduler trade network bandwidth for additional CPU load on a per-task basis—a natural fit for environments with volatile WAN latency.

4) Model-architecture upgrades: Preliminary data show that PPO converges faster than DQN but remains sensitive to reward scaling. We will experiment with *Duelling Double-DQN with distributional value heads* and a *Shared-trunk PPO+LSTM* variant that captures longer temporal dependencies (e.g., bursty arrivals). All networks will adopt GroupNorm in place of LayerNorm to improve stability in nonstationary feature distributions.

5) Mobile-device tier: Our current simulator ignores the execution on device. We will activate EdgeCloudSim's *mobile tier* and add energy-aware actions that decide whether to keep a task on the handset, offload to a nearby edge server, or push all the way to the cloud. This three-way choice requires extending the state to include battery level, thermal budget, and radio type (Wi-Fi vs LTE), and adapting the reward to penalise excessive battery drain.

Our future goals are sizable and require a lot of work, however we are confident these changes will make significant improvements, in perforrmance, and usability. We are excited to implement these improvements.

VI. CONCLUSION

In conclusion we have demonstrated our goal with some sucsess. This paper has presented an *end–to–end* reinforcement-learning framework for dynamic task scheduling and load balancing across the edge–cloud continuum. By embedding Double-DQN and PPO agents inside EdgeCloudSim, we demonstrated a fully automated loop in which the simulator supplies realistic mobility, bandwidth, and workload fluctuations, while the learner iteratively refines a scheduling policy. Although both agents converged, their first-pass performance was short of production presentation, edge saturation, mobility-related failures, and biased exploration drove the overall task–completion rate down to $56.7\,\%$ and the QoE index down to $42.1\,\%$. These findings are valuable for two reasons. First, they expose concrete failure modes—state omission, reward mis-weighting, and action space issues that can mislead even state-of-the-art RL algorithms in heterogeneous systems. Second, they provide a reproducible baseline against which future schedulers may be bench-marked. While they may not be good now they will improve upon revisit.

In summary, this study underscores both the promise and the pitfalls of deploying deep RL in cloud-edge orchestration. While off-the-shelf agents are insufficient, a carefully engineered combination of richer state signals, balanced rewards, and advanced policy architectures is likely to close the gap between simulation and the intense service levels demanded by next-generation distributed applications.

VII. CODE

All code and work for this project is available at www.github.com/yohannesgeleta/512Project.

REFERENCES

[1] Z. Wang, M. Goudarzi, M. Gong, and R. Buyya, "Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments," 2023. [Online]. Available: https://arxiv.org/abs/2309.07407

[2] J. Zhang, Z. Ning, M. Waqas, H. Alasmary, S. Tu, and S. Chen, "Hybrid edge-cloud collaborator resource scheduling approach based on deep reinforcement learning and multiobjective optimization," *IEEE Transactions on Computers*, vol. 73, no. 1, pp. 192–205, 2024.

[3] K. Siddesha, V. Jayaramaiah, G. and C. Singh, "A novel deep reinforcement learning scheme for task scheduling in cloud computing," *Cluster Computing*, vol. 25, pp. 4171–4188, 2022.

[4] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep reinforcement learning-based task scheduling in iot edge computing," *Sensors*, vol. 21, no. 5, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/5/1666

[5] B. Sellami, A. Hakiri, S. B. Yahia, and P. Berthou, "Energy-aware task scheduling and offloading using deep reinforcement learning in sdn-enabled iot network," *Computer Networks*, vol. 210, p. 108957, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622001359

[6] Y. Wang and X. Yang, "Research on edge computing and cloud collaborative resource scheduling optimization based on deep reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2502.18773

[7] S. K.P, B. Jayasingh, and D. Rani, "Deep reinforcement learning for dynamic task scheduling in edge-cloud environments," *International journal of electrical and computer engineering systems*, vol. 15, pp. 837–850, 11 2024.

[8] H. Che, Z. Bai, R. Zuo, and H. Li, "A deep reinforcement learning approach to the optimization of data center task scheduling," *Complexity*, vol. 2020, pp. 1–12, 08 2020.

[9] A. Jayanetti, S. Halgamuge, and R. Buyya, "Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments," *Future Generation Computer Systems*, vol. 137, pp. 14–30, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X22002230

[10] X. Chen, S. Hu, C. Yu, Z. Chen, and G. Min, "Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 3, pp. 391–404, 2024.

[11] F. Qi, L. Zhuo, and C. Xin, "Deep reinforcement learning based task scheduling in edge computing networks," in *2020 IEEE/CIC International Conference on Communications in China (ICCC)*, 2020, pp. 835–840.

[12] H. Wu, J. Geng, X. Bai, and S. Jin, "Deep reinforcement learning-based online task offloading in mobile edge computing networks," *Information Sciences*, vol. 654, p. 119849, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025523014342