Adaptive Cloud Resource Allocation Using Attention-Driven Deep Reinforcement Learning

Pandi S. Prabha
Department of Computer Science and Information Technology (CSIT), Jain (Deemed to be University), Bengaluru, India
h4prabha@gmail.com (corresponding author)

A. Rengarajan
Department of Computer Science and Information Technology (CSIT), Jain (Deemed to be University), Bengaluru, India
a.rengarajan@jainuniversity.ac.in

Received: 16 July 2025 | Revised: 28 August 2025 and 6 September 2025 | Accepted: 9 September 2025

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | *DOI: https://doi.org/10.48084/etasr.13443*

ABSTRACT

Efficient resource allocation remains a critical challenge in dynamic Cloud Computing (CC) environments, where maintaining Quality of Service (QoS), minimizing latency, and ensuring fairness are paramount. This study proposes a novel Deep Reinforcement Learning (DRL)-based framework that models resource allocation as a multi-agent Markov Decision Process (MDP), with each Virtual Machine (VM) link treated as an autonomous agent. Leveraging a Deep Q-Network (DQN) architecture enhanced by an attention mechanism, the framework enables agents to refine state observations and coordinate decisions adaptively. A custom reward function balancing throughput, latency, and resource cost guides the learning process, whereas experience replay and temporal annealing strategies promote optimal policy convergence. Experimental results demonstrate significant improvements in energy efficiency, execution time, waiting time, fairness, and throughput when benchmarked against existing Reinforcement Learning (RL)-based, Resource Management Framework–Deep Neural Network (RMF-DNN), and Federated Reinforcement Learning (F-RL) models. The proposed system introduces architectural innovations, including decentralized agent-based learning, attention-guided state refinement, and fairness-aware scheduling, establishing a scalable and intelligent solution for cloud resource management.

Keywords-Cloud Computing (CC); resource allocation; Deep Reinforcement Learning (DRL); Deep Q-Network (DQN); multi-agent model

I. INTRODUCTION

Cloud Computing (CC) has become foundational to modern infrastructure, offering scalable, cost-efficient, and remote services for distributed applications [1-3]. By virtualizing resources above physical hardware, CC enables customizable environments and flexible provisioning with minimal overhead [4]. This shift has transformed service deployment, delivering benefits such as dynamic scaling, reduced capital costs, and optimized resource use [5]. Evolving from distributed, grid, and parallel computing [6], CC now supports service-oriented and hybrid cloud architectures. However, its potential is constrained by resource allocation challenges, particularly in meeting dynamic demands, Service-Level Agreements (SLAs), cost limits, and energy goals [7-9]. Conventional resource allocation methods, often static or heuristic-based, lead to bottlenecks, resource starvation, and inefficiencies in large-scale data centers (DCs) [10, 11].

Addressing this issue requires a robust strategy that can dynamically allocate resources based on real-time observations and predictive intelligence. Reinforcement Learning (RL)-based techniques have gained momentum, enabling agents to learn optimal allocation policies through interaction with their environment. However, RL models often face limitations in scalability, fairness enforcement, and real-time responsiveness [12, 13]. Thus, there is a growing need for frameworks that integrate deep learning with adaptive decision-making to handle the multi-objective nature of cloud workloads.

The objective of this study is to propose a Deep Reinforcement Learning (DRL)-based framework that models resource allocation as a multi-agent Markov Decision Process (MDP), wherein each Virtual Machine (VM) link functions as an autonomous agent. The system incorporates a Deep Q-Network (DQN) enhanced by an attention mechanism to refine state observations and facilitate coordinated decision-making. By implementing a custom reward function that balances throughput, latency, and cost, the proposed framework seeks to achieve high Quality of Service (QoS) while ensuring energy efficiency and fairness.

Existing frameworks have laid valuable groundwork, yet many fall short in adaptability, and multi-objective optimization. Authors in [14] proposed an RL-fuzzy logic framework to improve energy efficiency and reduce SLA violations in cloud DCs. Though effective in lowering ownership costs, it lacks fine-grained workload adaptation and agent-level coordination. Authors in [15] applied an improved DQN with multi-replay memory for resource allocation, boosting convergence and job completion, but the model falls short in multi-agent interpretability and fairness. Authors in [16] introduced Bi-directional Gated Long Short-Term Memory (BG-LSTM) with Savitzky-Golay filtering for workload forecasting, which smooths noisy data well but remains predictive rather than reactive, limiting real-time provisioning. Authors in [17] developed the Resource Management Framework–Deep Neural Network (RMF-DNN) using Gray Wolf Optimization for multi-cloud routing, offering scalability and delivery gains, yet relying on centralized control and lacking autonomous agent support.

Authors in [18] introduced Proactive Hybrid Pod Autoscaling (ProHPA), a Bi-directional Long Short-Term Memory (Bi-LSTM)–attention-based autoscaling method for forecasting CPU and memory usage. It mitigates resource over-allocation and enables fast scaling but is limited to Kubernetes and lacks cross-topology adaptability. Authors in [19] reviewed DRL in cloud scheduling, underscoring its potential, yet noting its underuse in resource management. They advocated for hybrid models and attention-driven learning to address complex scheduling, which is still an underexplored frontier.

The Laxity-based Cost-efficient Task Scheduling (LCTS) method [20] employs laxity-based prioritization to reduce cost and execution time in fog task scheduling but lacks energy metric validation and comparative analysis with advanced schedulers. The VM load balancing model [21] improves resource use and SLA adherence via multi-stage VM-to-Physical Machine (PM) assignment, yet fails to address workload dynamics and the overhead of layered decisions. Enhanced Q-Learning–Secure Q-Learning (EQL-SQL) [22] integrates reinforcement learning with secure, adaptive mechanisms for energy-efficient fog resource allocation, but faces deployment challenges in scalability and the computational burden of dual-learning strategies.

In response to these gaps, our proposed DRL-based framework builds upon these foundations by integrating a decentralized multi-agent system, where each VM link is treated as a learning agent. The model introduces an attention mechanism to refine state representations and coordinate agent behavior, overcoming limitations of static or centralized methods. A fairness-aware reward function enables equitable and adaptive resource distribution, whereas the use of temporal annealing strategies supports optimal policy convergence over time. Thus, this study advances the landscape of cloud resource allocation by offering a scalable, context-sensitive, and performance-balanced solution.

II. PROPOSED METHODOLOGY

A fairness-based RL model ensures effective resource management via a rule-based resource manager. It calculates functions for deadline adherence and mean job delay using fairness values, addressing practical issues and improving efficiency. For a limited set of N users requesting resources, constraints such as limited security, availability, and fairness are taken into consideration. When resources are scarce or conflicting, fair allocation divides resources proportionally or equally, ensuring every user receives guaranteed access. Distribution follows fairness principles, including Pareto efficiency, envy-freeness, incentive sharing, and proofness of strategy.

In a cloud system with an infinitely divisible resource pool, there are n resources represented as $\{1, 2, \ldots n\}$, each corresponding to a specific resource type. Shared users are denoted by $S = \{1, 2, \ldots m\}$, each processing an infinitely divisible set of tasks. Each task for user i is expressed as a vector: $V_i = (V_{i_1}, V_{i_2}, \ldots . V_{i_n})$, where V_{i_r} signifies the quantity of resource r required by user i for a single task. All elements of the vector are assumed to be positive.

For $\Delta_i \in U$ and $n \in N$, the fractional demand vector is defined as:

$$v_{i_r} = \frac{V_{i_r}}{d_r} \quad (1)$$

where r represents the resource index, v_{i_r} denotes the fractional demand (largest component of V_{i_r}), and d_r denotes the total available quantity of resource r.

Let $N = (N_1, N_2, \ldots . N_n)$ represent the number of tasks and N_i signify the number of tasks for user i. During resource allocation, $N_i \cdot V_{i_r}$ denotes the total quantity of resource r required by user i for all its tasks.

Feasible allocations satisfying the capacity constraints are given by:

$$\sum_{i=1}^{m} N_i \cdot V_{i_r} \leq d_r, \quad r = \{1, 2, .. n\} \quad (2)$$

ensuring that the total demand from all users does not exceed the available resources.

To ensure fairness among users, a fairness ratio is applied. For users i and j in a queue, the fairness ratio for resource r is defined as:

$$\tau_r = \frac{N_i}{N_{fi}} \quad (3)$$

where N_i denotes the number of tasks for user i, and N_{fi} represents the minimum number of tasks for that user. If tasks between two users encounter a bottleneck, each receives a fair share based on this ratio.

Fairness between two users i and j, with their respective leading resources r_i and r_j, is expressed as:

$$N_i \cdot V_{i_r} > \max_{n \in N}(N_n \cdot V_{n_r}) \quad (4)$$

Equation (4) ensures that resource allocation among users in different queues is fair, preventing any user from receiving preferential treatment.

In this context, a neural network computes the outputs using an approximate function based on the inputs, which are

then compared with a target function. A first-order iterative algorithm minimizes this function. The inputs are multiplied by weights to compute outputs, with weights representing the relative significance of each input. These weights are adjusted during training and retain learned information throughout the process.

While traditional neural networks use predefined targets, DRL compares the outputs with the Bellman function to compute the results. This is given by (5):

$$Q(s,a) = m(s,a) + \varphi \cdot \max Q(s', a'|s,a) \quad (5)$$

After determining the target from the function, it is used to train the network through Q-learning updates. The reward function evaluates the effectiveness of the agent's action and is defined as:

$$R(s,a) = k_1 \cdot \text{Throughput} - k_2 \cdot \text{VM Usage} - k_3 \cdot \text{Latency} \quad (6)$$

A distributed architecture is employed (Figure 1), with each agent maintaining its own Q-network to optimize its policy.

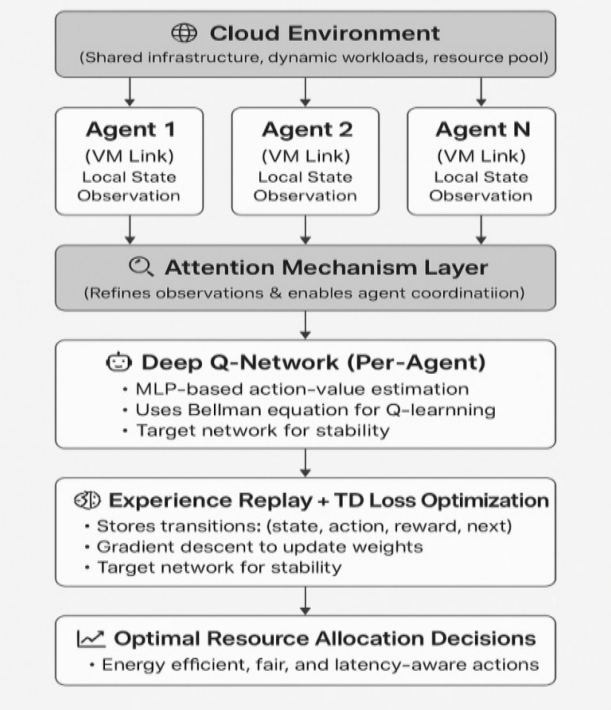

Fig. 1. FairNet-Q architecture.

The DRL-based algorithm is designed to address the resource management problem. Every link among VMs is treated as an agent, and the problem is modeled as an MDP, where all VMs exist within the same cloud environment. The agents interact with the environment to gather local observations and use an attention mechanism to exchange information with other agents, facilitating resource allocation. The framework employs a DQN to maximize link rates while ensuring low latency.

The components of the FairNet-Q architecture are:

- Cloud environment: A dynamic pool of resources and workloads that agents interact with.
- Agents (VM links): Each agent observes its local state and operates independently, forming a decentralized learning environment.
- Attention based co-ordination: The attention mechanism layer emphasizes the importance of context refinement, helping agents focus on the most relevant information in a dynamic environment. This improves decision-making and collaboration among agents.
- DQN: The integration of DQNs per agent illustrates learning via Q-value estimation and Bellman updates.
- Experience replay and Temporal Difference (TD) loss optimization: The experience replay buffer stores agents' past interactions (s, a, r, s′) to stabilize learning through randomized training samples, whereas the TD loss computes errors to update the Q-network weights via gradient descent, ensuring accurate and consistent decision-making.
- Optimal resource allocation decisions: The final layer produces optimal resource allocation decisions, demonstrating tangible outcomes in energy efficiency, latency reduction, and fair and adaptive VM scheduling.

The system accounts for resource allocation over time and integrates an attention mechanism to adapt to environmental changes. This mechanism helps agents focus on relevant information, improving action-value estimation (Q function), which is defined as:

$$Q_n(s,a,\varphi) = f_n\left(\text{add}(s_n^A, s_n)\right) \quad (7)$$

where $\left(\text{add}(s_n^A, s_n)\right) = s_n^A + s_n$ is a 3-layer Multi-Layer Perceptron (MLP), s^A represents the output state of the agent from the attention network, and φ denotes network parameters (Figure 2).

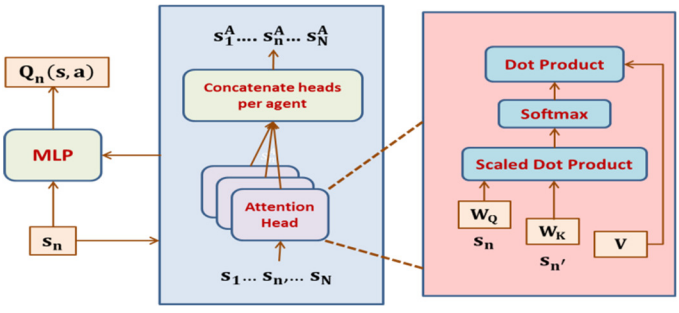

Fig. 2. Computation of Q value.

Algorithm 1 outlines a decentralized learning procedure where each VM link acts as an autonomous agent within a cloud environment. Initially, the agents are equipped with DQN models, attention mechanisms, and experience replay buffers. For each training episode, the system updates environmental parameters and resets payload and latency conditions. Agents observe their states, refine them via

attention networks, and select actions using the ε-greedy policy. After taking actions, they receive a shared reward and update the environment accordingly. The agents then store their transition tuples (state, action, reward, next state) into replay buffers. Following this, each agent samples mini-batches to update its Q-network parameters via TD learning. Periodically, the target network parameters φ^{tar} are synchronized with the current network weights to stabilize learning and reduce estimation bias. This algorithm enables coordinated, fairness-aware resource allocation through attention-driven policy optimization across multiple agents.

To obtain the optimal policy (π), the ideal action-value function is determined by:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad (8)$$

Based on Bellman optimality:

$$Q^*(s_t,a_t) = \mathbb{E}_{s_{t+1}\sim p(.|s_t,a_t)}\left(R_t + \Psi \cdot \max_{a\in A} Q^*(s_{t+1},a)\,|s_t = s, a_t = a\right) \quad (9)$$

where Ψ denotes the discount factor. Using Monte Carlo approximation, (9) is transformed into:

$$Q^*(s_t,a_t) \approx r_t + \Psi \cdot \max_{a\in A} Q^*(s_{t+1},a) \quad (10)$$

Approximating using the network yields:

$$Q(s_t,a_t{:}\varphi) \approx r_t + \Psi \cdot \max_{a\in A} Q^*(s_{t+1},a;\varphi) \quad (11)$$

The Left-Hand Side (LHS) of (9) represents the current prediction of the Q-network, whereas the Right-Hand Side (RHS) shows the TD target, which is the predicted Q value at time t +, represented as $\Psi \cdot \max_{a\in A} Q^*(s_{t+1},a;\varphi)$.

The loss function is given by:

$$\text{loss} = [Q(s_t,a_t;\varphi) - y_t]^2 \quad (12)$$

DQN training consists of two main parts: gathering training data and updating parameters φ:

- Gathering training data: The n-th link interacts with the environment using the behavioral policy π, typically ε-greedy:

$$a = \begin{cases} \underset{a\in A}{\text{argmax}}\, Q_n(o',a;\varphi), & \text{with probability } 1-\varepsilon \\ \text{random action}, & \text{with probability } \varepsilon \end{cases}$$

The link takes an action, which changes the environment, forming a trajectory for the episode, represented by $o_1^n, a_1^n, r_1^n, \ldots, o_t^n, a_t^n, r_t^n$, stored as 4-tuples $(o_t^n, a_t^n, r_t^n, o_{t+1}^n)$ in the experience replay array D.

- Updating parameters: A batch of experiences is randomly selected from the experience replay buffer to update φ using stochastic gradient descent. While TD learning trains the DQN, the maximization step can overestimate the TD target. To resolve this, a target network is employed to compute the TD target:

$$y_t' = r_t + \Psi.\max_{a'} Q_n(o',a';\varphi^{tar}) \quad (13)$$

The loss function is determined by:

$$L_n(\varphi) = \frac{1}{2D_n}\sum_{t\in D_n}[y_t' - Q_n(o,a,\varphi)]^2 \quad (14)$$

where $\delta_t = y_t' - Q_n(o,a,\varphi)$ is the TD error.

Gradient descent updates network parameters as:

$$\varphi = \varphi - \beta\sum_{t\in D_n}\delta_t.\nabla_{\varphi} Q_n^{\pi}(o,a,\varphi) \quad (15)$$

where φ^{tar} is the periodically updated target network parameter to enhance network stability.

```
Algorithm 1: FairQ-AttendDRL
Input: Attention network and DQN models,
       size of payload, maximum tolerant
       latency
Output:Network weights
Set experience replay array, DQN
parameters, and final DQN parameters
For every episode or task i = 1,2, …, do:
  Update environment
  Reset remaining payload L_t^n and remaining
  time U_t^n
  For every step t = 1,2 …, do:
    Observe the state of agents: o_t =
    {o_t^n}_n=1,2,…
    Through the attention network: o_t^n =
    {o_t^A,n}_n=1,2,…
    For every agent n = 1,2 …, do:
      Based on add(o_t^A,n, o_t), select a_t^n using
      the ε-greedy policy
    End for
    All agents take actions and receive
    shared reward R_t
    Update environment
    For every agent n = 1,2 …, do:
      Gain the subsequent observation o_t+1^n
      Store (o_t^n, a_t^n, r_t^n, o_t+1^n) in the
      experience replay array
    End for
  End for
  For every agent n = 1,2 …, do:
    Sample mini-batch experiences from D_n
    experience replay array
    Update DQN parameters using (14)
    Synchronize target DQN every k steps:
    φ^tar = φ
  End for
End for
```

III. RESULTS AND DISCUSSION

A. Simulation Methodology

To validate the proposed FairNet-Q architecture, a comprehensive simulation environment was designed using Python, tailored to reflect realistic cloud infrastructure dynamics. The simulation was structured to evaluate

decentralized, attention-driven DRL agents under varying workload conditions and resource constraints. The simulated environment is summarized in Table I.

TABLE I. SIMULATED ENVIRONMENT SETUP FOR THE PROPOSED FRAMEWORK

Component	Configuration
DCs	2 geographically distributed (DC1-east, DC2-west)
Hosts per DC	25 hosts per DC, each with multiple processing elements
VMs	100 total VM links (distributed across both DCs)
VM configuration	2–4 cores per VM, 2 GB RAM, 1000 MIPS
Cloudlets (tasks)	500–1000 cloudlets simulating diverse workloads
Workload distribution	Mix of latency-sensitive and throughput-oriented tasks
Scheduling policy	Time-shared or space-shared (configurable per host)
Network bandwidth	100 Mbps (shared with prioritization queues)
Execution environment	Python libraries
Simulation time horizon	1000 simulation steps (episodes)
Reward weights	k_1 = 0.4 (throughput), k_2 = 0.3 (VM usage), k_3 = 0.3 (latency)

1) Agent Behavior and Learning

The following describes the key aspects of agent modeling and learning in the proposed framework:

- Agent description: Each VM link is modeled as an autonomous agent within a multi-agent MDP.
- State space: Includes current VM load, task queue length, latency tolerance, and remaining payload.
- Action space: Involves VM-task mappings, prioritization decisions, and scheduling actions.
- Attention mechanism: Agents refine local observations using a shared attention layer to emphasize relevant environmental features.
- Reward function: The reward function is defined in (6):
  - Experience replay: Each agent maintains a buffer of transitions (s_t, a_t, r_t, s_{t+1}) for stable learning.
  - TD learning: The Q-values are updated using the Bellman approximation and gradient descent.

2) Simulation Workflow

The simulation follows these steps:

- Initialization: Agents are instantiated with DQN and attention modules. Replay buffers are initialized.
- Episode loop:
  1. The environment is reset with randomized cloudlet distribution.
  2. Agents observe states and refine them via the attention mechanism.
  3. Actions are selected using the ε-greedy policy.
  4. The environment updates based on the agents' actions, and the shared reward is distributed.
  5. Transitions are stored in the replay buffers.
- Training loop:
  1. Mini-batches are sampled from the replay buffers.
  2. The TD error is computed, and Q-network weights are updated.
  3. Target networks are synchronized every k steps.

B. Performance Evaluation

Table II provides the performance of the proposed DRL-based method compared with the RL-based [14], RMF-DNN [17], and Federated Reinforcement Learning (F-RL) [18] models in terms of energy consumption, execution time, waiting time, throughput, and fairness. To analyze the effectiveness of the proposed framework, each performance metric is discussed in detail below:

- Energy consumption: Energy consumption is typically measured as the total amount of electrical energy (in J or kWh) used by active VMs and infrastructure components during task execution. The proposed framework achieves the lowest energy usage (173.06 J), indicating efficient task execution with minimal computational overhead. This efficiency stems from intelligent VM activation and deep Q-learning policies that avoid unnecessary resource engagement, unlike the energy-intensive logic in RL-based [14] and RMF-DNN [17] methods.
- Execution time: Execution time refers to the total duration required to process and complete a cloudlet (task) once it is assigned to a VM. It includes the time spent on resource allocation, CPU cycles, and actual task execution, excluding queuing or network delays. By employing a decentralized multi-agent system enhanced with attention mechanisms, the framework ensures rapid state interpretation and efficient action selection. Each agent learns optimal task–VM mappings through deep Q-learning, minimizing computational delays. Compared to existing models such as RL-based [14] and RMF-DNN [17], the system achieves a lower execution time of 22.75 ms, demonstrating its effectiveness in accelerating resource scheduling and enhancing real-time responsiveness in dynamic cloud environments.
- Waiting time: Waiting time refers to the average duration a task spends in the queue before being assigned to an available VM. It reflects the system's responsiveness, with the proposed DRL model achieving the shortest delay (10.41 ms) by enabling faster scheduling through agent collaboration and attention-guided decisions. The attention mechanism ensures prompt identification of ready VMs, preventing backlog accumulation and reducing queue latency.
- Throughput: Throughput refers to the number of tasks (or cloudlets) successfully processed per unit time by the resource allocation framework. It measures system

efficiency, showing how well the proposed DRL-based method maximizes task completion under dynamic workloads through optimal agent coordination and scheduling. With a peak throughput of 73.11 units/sec, the system processes more tasks per unit time by leveraging real-time learning and parallel execution, outperforming RMF-DNN and RL-based systems in overall service efficiency.

- Fairness index: The fairness index quantifies how evenly cloud resources are distributed among competing tasks or agents. It ensures that no single agent monopolizes resources while others are starved, promoting equitable access across the system. Achieving a fairness index of 0.911, the framework ensures fair task distribution even under resource contention by integrating Pareto-aware decision rules and attention-guided collaboration among agents, offering a balance between individual performance and global scheduling equity.

TABLE II. PERFORMANCE COMPARISON OF RESOURCE ALLOCATION MODELS

Metric	RL-based [14]	RMF-DNN [17]	F-RL [18]	Proposed framework
Energy consumption (J)	218.37	231.25	194.72	173.06
Execution time (ms)	34.67	32.82	25.92	22.75
Waiting time (ms)	21.86	19.47	12.84	10.41
Throughput (units/s)	63.59	67.42	71.83	73.11
Fairness index	0.802	0.827	0.884	0.911

From Table II, it is observed that the proposed DRL-based method consumes 10.5%, 4.7%, and 8.1% less energy compared to the RL-based, RMF-DNN, and F-RL models, respectively. The proposed scheme achieves 50%, 20%, and 16% lower execution time compared to the standard mechanisms considered in this study. Similarly, the proposed DRL-based method achieves 66.7%, 33.3%, and 11.1% lower waiting time compared to RL-based, RMF-DNN, and F-RL models. Furthermore, it offers 27.1%, 18.7%, and 11.3% higher throughput, and 12.9%, 6.7%, and 2.1% higher fairness compared to the corresponding baseline schemes.

IV. CONCLUSION

The proposed framework represents a significant advancement in cloud resource allocation methodologies by integrating Deep Reinforcement Learning (DRL), attention-guided coordination, and fairness modeling in a decentralized multi-agent framework. The results demonstrate that the proposed scheme achieves superior performance in terms of energy consumption, execution time, waiting time, throughput, and fairness compared to the Reinforcement Learning (RL)-based method, Resource Management Framework with Deep Neural Networks (RMF-DNN), and Federated Reinforcement Learning (F-RL) models.

Furthermore, the incorporation of temporal annealing, experience replay, and target network stabilization contributes to faster convergence and reduced decision volatility, enhancing learning stability. These improvements make the system particularly well-suited for heterogeneous, real-time cloud platforms, including serverless functions and edge-compute clusters.

Future work may focus on integrating real-world workload traces, extending the framework to hybrid cloud ecosystems, and conducting comparative studies against actor-critic or transformer-based DRL variants to further validate and broaden the model's applicability.

REFERENCES

[1] M. V. Fard, A. Sahafi, A. M. Rahmani, and P. S. Mashhadi, "Resource allocation mechanisms in cloud computing: a systematic literature review," *IET Software*, vol. 14, no. 6, pp. 638–653, Dec. 2020, https://doi.org/10.1049/iet-sen.2019.0338.

[2] S. Malhotra, F. Yashu, M. Saqib, D. Mehta, J. Jangid, and S. Dixit, "Deep Reinforcement Learning for Dynamic Resource Allocation in Wireless Networks." arXiv, Mar. 13, 2025, https://doi.org/10.48550/arXiv.2502.01129.

[3] A. Mani, "Resource Allocation and Scheduling Techniques in Cloud Computing: A Comprehensive Review," *International Journal For Multidisciplinary Research*, vol. 7, no. 1, Feb. 2025, Art. no. IJFMR250138013, https://doi.org/10.36948/ijfmr.2025.v07i01.38013.

[4] M. C. Ho and S. Cho, "Deep Reinforcement Learning-Assisted Resource Allocation for Fluid Antenna System: Overview, Research Challenges and Future Trends," in *2025 International Conference on Artificial Intelligence in Information and Communication*, Fukuoka, Japan, 2025, pp. 148–150, https://doi.org/10.1109/ICAIIC64266.2025.10920759.

[5] T. Ma, Y. Chu, L. Zhao, and O. Ankhbayar, "Resource Allocation and Scheduling in Cloud Computing: Policy and Algorithm," *IETE Technical Review*, vol. 31, no. 1, pp. 4–16, Jan. 2014, https://doi.org/10.1080/02564602.2014.890837.

[6] A. Mahida, "A Comprehensive Review on Ethical Considerations in Cloud Computing-Privacy, Data Sovereignty and Compliance," *Journal of Artificial Intelligence & Cloud Computing*, vol. 1, no. 4, Dec. 2022, Art. no. SRC/JAICC-248, https://doi.org/10.47363/JAICC/2022(1)231.

[7] A. Kumar and W. Yeoh, "DECAF: Learning to be Fair in Multi-agent Resource Allocation," in *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, Detroit, MI, USA, 2025, pp. 2591–2593.

[8] K. H. Ang *et al.*, "Classification of Wafer Defects with Optimized Deep Learning Model," *Proceedings of International Conference on Artificial Life and Robotics*, vol. 28, pp. 609–614, Feb. 2023, https://doi.org/10.5954/ICAROB.2023.OS25-4.

[9] D. Lim and I. Joe, "MAARS: Multiagent Actor–Critic Approach for Resource Allocation and Network Slicing in Multiaccess Edge Computing," *Sensors*, vol. 24, no. 23, Dec. 2024, Art. no. 7760, https://doi.org/10.3390/s24237760.

[10] X. Wang, A. Li, and G. Han, "A Deep-Learning-Based Fault Diagnosis Method of Industrial Bearings Using Multi-Source Information," *Applied Sciences*, vol. 13, no. 2, Jan. 2023, Art. no. 933, https://doi.org/10.3390/app13020933.

[11] N. Khan, A. Abdallah, A. Celik, A. M. Eltawil, and S. Coleri, "Explainable AI-Aided Feature Selection and Model Reduction for DRL-Based V2X Resource Allocation," *IEEE Transactions on Communications*, vol. 73, no. 9, pp. 7633–7649, Sep. 2025, https://doi.org/10.1109/TCOMM.2025.3554655.

[12] G. K. Shyam and S. S. Manvi, "Resource allocation in cloud computing using agents," in *2015 IEEE International Advance Computing Conference*, Banglore, India, 2015, pp. 458–463, https://doi.org/10.1109/IADCC.2015.7154750.

[13] J. J. Aloor, S. N. Nayak, S. Dolan, and H. Balakrishnan, "Cooperation and Fairness in Multi-Agent Reinforcement Learning," *Journal on Autonomous Transportation Systems*, vol. 2, no. 2, Dec. 2024, Art. no. 8, https://doi.org/10.1145/3702012.

[14] T. Thein, M. M. Myo, S. Parvin, and A. Gawanmeh, "Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 10, pp. 1127–1139, Dec. 2020, https://doi.org/10.1016/j.jksuci.2018.11.005.

[15] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, Jun. 2020, https://doi.org/10.1109/JSAC.2020.2986615.

[16] J. Bi, S. Li, H. Yuan, and M. Zhou, "Integrated deep learning method for workload and resource prediction in cloud systems," *Neurocomputing*, vol. 424, pp. 35–48, Feb. 2021, https://doi.org/10.1016/j.neucom.2020.11.011.

[17] S. B. Sangeetha, R. Sabitha, B. Dhiyanesh, G. Kiruthiga, N. Yuvaraj, and R. A. Raja, "Resource Management Framework Using Deep Neural Networks in Multi-Cloud Environment," in *Operationalizing Multi-Cloud Environments: Technologies, Tools and Use Cases*, R. Nagarajan, P. Raj, and R. Thirunavukarasu, Eds. Cham: Springer International Publishing, 2022, pp. 89–104, https://doi.org/10.1007/978-3-030-74402-1_5.

[18] B. Jeong, S. Baek, S. Park, J. Jeon, and Y.-S. Jeong, "Stable and efficient resource management using deep neural network on cloud computing," *Neurocomputing*, vol. 521, pp. 99–112, Feb. 2023, https://doi.org/10.1016/j.neucom.2022.11.089.

[19] G. Zhou, W. Tian, R. Buyya, R. Xue, and L. Song, "Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions," *Artificial Intelligence Review*, vol. 57, no. 5, Apr. 2024, Art. no. 124, https://doi.org/10.1007/s10462-024-10756-9.

[20] P. K. Mishra and A. K. Chaturvedi, "An Improved Laxity based Cost Efficient Task Scheduling Approach for Cloud-Fog Environment," *Engineering, Technology & Applied Science Research*, vol. 15, no. 1, pp. 19037–19044, Feb. 2025, https://doi.org/10.48084/etasr.8595.

[21] E. Suganthi and F. K. M. Selvi, "Virtual Machine Load Balancing Model Framework for Cloud Computing," *Engineering, Technology & Applied Science Research*, vol. 15, no. 3, pp. 22553–22558, Jun. 2025, https://doi.org/10.48084/etasr.10556.

[22] K. G. S and M. Devi, "Optimized Resource Management and Security Enhancement in Fog Computing using Advanced Q-Learning Approaches," *Engineering, Technology & Applied Science Research*, vol. 15, no. 3, pp. 23965–23971, Jun. 2025, https://doi.org/10.48084/etasr.10995.