Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 33 No. 1s (2026)

Adaptive Hierarchical Reinforcement Learning for Decoupled Service Caching and Computation Offloading in Mobile Edge-Cloud Network

1. **Mr. T. Sai Lalith Prasad**

Assistant Professor

Artificial Intelligence and Data Science Department Vignan Institute of Technology and Science Hyderabad, India

lalithresearch15@gmail.com

2. **Kanneti Venkata Yeshwanth Reddy**

Artificial Intelligence and Data Science Department
Vignan Institute of Technology and Science
India

yeshwanth.reddy.kv@gmail.com

3. **Kummari Deepna**

Artificial Intelligence and Data Science Department
Vignan Institute of Technology and Science
India

kummarik.deepna@gmail.com

4. **Yanala Deepika**

Artificial Intelligence and Data Science Department
Vignan Institute of Technology and Science
India

yanaladeepika2005@gmail.com

Article History:
Received: 04-02-2026
Revised: 20-03-2026
Accepted: 10-04-2026

Abstract:

In modern mobile edge-cloud computing environments, efficient management of computation and caching resources essential to meet the increasing demands of latency sensitive applications Traditional deep reinforcement learning-based offloading frameworks focus mainly on single agent optimization, often leading to high system overhead and poor scalability. The growing demand for faster and smarter mobile applications requires efficient management of computation offloading and service caching in mobile edge cloud networks. Existing reinforcement learning models often rely on single agent systems, which struggle to handle complex decisions and dynamic network conditions. To address this issue, the proposed project introduces an Adaptive Hierarchical Reinforcement Learning (HRL) framework that separates decision making into two layers. The high-level agent coordinates global resource

allocation between edge and cloud servers, while the low-level agents manage local caching and computation offloading tasks. Using advanced multi-agent reinforcement learning algorithms such as MAPPO and MADDPG the system enables cooperative and adaptive learning among distributed agents. This adaptive approach improves learning stability reduces latency, and enhance energy efficiency, making it a more intelligent and scalable solution for modem mobile edge-cloud networks.

Index Terms— Mobile Edge Computing, Hierarchical Reinforcement Learning, Computation Offloading, Service Caching, MAPPO, MADDPG, Edge–Cloud Networks.

I. INTRODUCTION

Close to where data is created, processing power now reaches further than before - thanks to a shift toward edge-based systems paired with distant clouds. Instead of relying only on faraway centers, tasks get handled nearer to users, cutting delays while easing traffic loads across main pathways. Still, small capacity at local hubs makes it tough when demand jumps suddenly or people keep moving around. Unpredictable shifts in how much work arrives, along with tight limits on space and speed at these nearby nodes, create ongoing strain for keeping services running smoothly within this hybrid setup.

Picking what to store on edge devices shapes how tasks get processed nearby or sent farther away. When choices about storage shift, so do the options for where work happens. Where a job runs often depends on what data already sits close by. Decisions around running code here or there ripple back into what stays saved locally. One choice tug at the other, constantly. Holding things closer can speed up responses, yet that only matters if routing aligns well. Past research shows tuning both moves together cuts delays and power use across mobile networks linking edge and cloud [1], [2].

Old ways of handling cache and data transfer usually stick to fixed rules or basic math models built on shaky guesses [12]. These ideas help explain things in theory but fall short when networks shift - like changing user demand, spotty signal strength, or mixed hardware at the edge. The task gets tougher because balancing both choices together turns into a puzzle that grows too fast to solve quickly across big systems. Solving it live becomes nearly impossible just from how many possibilities pile up.

One way past those limits? Reinforcement learning - often explored lately - for smarter resource handling on edge devices. Instead of fixed rules, methods built on RL pick up smart moves through trial and error in real settings, needing no precise blueprints of the system [5], [6]. In much the same vein, deep versions of it help shape how services and data get stored at network edges, adjusting on the fly when users shift their needs [10], [11]. Yet here's the catch: many current approaches bundle cache choices and task shifting too closely, relying on

basic solo or group agent setups that tend to crawl toward answers while struggling to grow smoothly across larger setups.

Lately, people have started noticing a method called hierarchical reinforcement learning because it breaks tough choices into simpler layers [7], [9]. Instead of mixing big-picture plans with instant actions, splitting them helps models learn faster and behave more consistently. When used in mobile edge computing, this approach guides tasks toward nearby servers - top tiers aim at wide goals, bottom ones decide exactly when and where to act locally [14]. Still, most current designs center only on shifting workloads, rarely treating cache management as its own separate challenge apart from moving computations, which holds back responsiveness amid rapidly changing cloud-edge conditions.

Spurred by these insights, this study introduces an Adaptive Hierarchical Reinforcement Learning setup tailored for separating service caching from computation offloading in mobile edge–cloud systems. At the core lies a top-tier agent stationed in the cloud, refining broad strategies over time through summarized views of network conditions. Meanwhile, scattered across edge nodes, several bottom-layer agents act on their own, shaping task routing choices using nearby data alone. Because it splits decision-making between levels - caching up high, offloading down low - the method sidesteps overwhelming complexity that comes with solving both together. As a result, handling bigger, more intricate networks becomes far more manageable.

Starting from how tasks are organized, the new AHRL setup connects big-picture control with on-the-ground actions through layered teamwork and smart adjustments. Because it learns in stages, mistakes stay small, progress speeds up, yet outcomes still beat older single-level methods built on standard reinforcement learning. Ending at real-world needs, this approach fits well into future mobile cloud systems where response time matters and conditions keep changing.

II. Related Work

Lately, work on mobile edge and cloud systems has zeroed in on balancing task delegation with efficient resource handling - especially for apps that demand quick responses. At first, researchers leaned heavily on math-driven models plus competitive frameworks to figure out when and how tasks should shift between devices, all while managing power use and speed limits. Take Mao and team’s approach - they used random process methods to study shifting computations dynamically, showing clear drops in energy drain during changing load times [1]. In a similar vein, scientists looked at multiplayer-style solutions where each user acts independently, revealing how rivalry and shared resource fights shape outcomes in wireless clouds [2].

As edge–cloud setups grow more complex, reinforcement learning steps into the spotlight for smarter task shifting and resource handling. Instead of relying on fixed rules, some methods

let machines adjust by experiencing changes in real time. These smart systems pick up how to act best through trial, skipping the need for exact blueprints of the environment. Under fast-changing network situations, they tend to do better than older rule-based strategies. While one path focuses on moving tasks around, another uses similar ideas to decide what data stays close to users. Learning from repeated use, these agents tweak where things are stored based on who wants what and when. Patterns shift, decisions evolve - no manual tuning needed. Over time, choices get sharper simply by watching what happens next.

Instead of sticking to basic setups, researchers turned to layered learning systems to tackle slow progress and instability when dealing with complex choices. Early studies revealed that breaking down decisions into different time scales and levels made training faster and policies more reliable [7], [9]. Following this idea, structured models were later used in mobile edge networks - where top-tier controllers set broad goals while lower ones take care of immediate tasks [14]. Results showed better performance than standard single-agent techniques, especially as network density increased.

Some research has tried combining service caching with task offloading by using Markov models along with deep learning techniques. One approach used MDP frameworks to cut down delay and power use when edge capacity is limited [12]. Even if results get better, such solutions tend to bundle decisions too closely while depending on centralized training. That setup tends to struggle when networks grow big or mix different types of devices.

Even with progress, most studies handle caching and task transfer as a single combined problem - or they zero in on just one part while ignoring details of the other. Learning methods that rely on flat structures or central control tend to stall or waver during training when faced with fast-changing conditions across edge and cloud systems serving varied needs. Another snag: today's layered strategies put effort into where tasks are run, yet leave storage choices tangled with routing logic, which limits how freely resources can be managed overall.

With these shortcomings in mind, few studies have tackled how to build layered learning systems that split service storage from task delegation - yet still let cloud and edge units work together smoothly. Solving this problem matters because future mobile networks linking edge and cloud need resource handling that scales well, stays steady, and runs without waste.

III. METHODOLOGY

A. An Introduction to the Proposed Framework

This work proposes an Adaptive Hierarchical Reinforcement Learning (AHRL) framework for decoupled service caching and computation offloading in mobile edge–cloud networks. The framework integrates global caching control and local offloading decisions into a unified hierarchical learning pipeline, enabling scalable, adaptive, and interpretable resource management under dynamic network conditions. Unlike conventional flat reinforcement learning approaches that jointly optimize caching and offloading in a single decision space,

the proposed framework explicitly separates these two tightly coupled problems across hierarchical layers while maintaining cooperative interaction between agents.

The overall methodology is organized into three major stages in temporal order: system state observation and modelling**,** hierarchical decision-making using cooperative reinforcement learning**,** and policy evaluation and adaptation through feedback-driven learning. A high-level learning layer focuses on long-term service caching coordination across edge servers, while a low-level learning layer performs fine-grained computation offloading decisions for individual user tasks. This hierarchical decomposition significantly reduces action-space complexity and improves learning stability in large-scale edge–cloud environments.

B. System State Observation and Environment Modelling

To enable effective decision-making, the framework continuously observes the system state across user, edge, and cloud layers. The observed state includes task arrival rates, task sizes, latency constraints, available bandwidth, edge server CPU utilization, cache occupancy, queue lengths, and device energy levels. These parameters collectively describe the dynamic operating environment of the mobile edge–cloud system.

The system is modelled as a Markov Decision Process (MDP) defined by the tuple,

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$$

where "$\mathcal{S}$" denotes the state space, $\mathcal{A}$the action space, $\mathcal{P}$the state transition probability, $\mathcal{R}$the reward function, and $\gamma \in (0,1)$the discount factor. State normalization and temporal aggregation are applied to stabilize learning under stochastic workloads and time-varying network conditions.

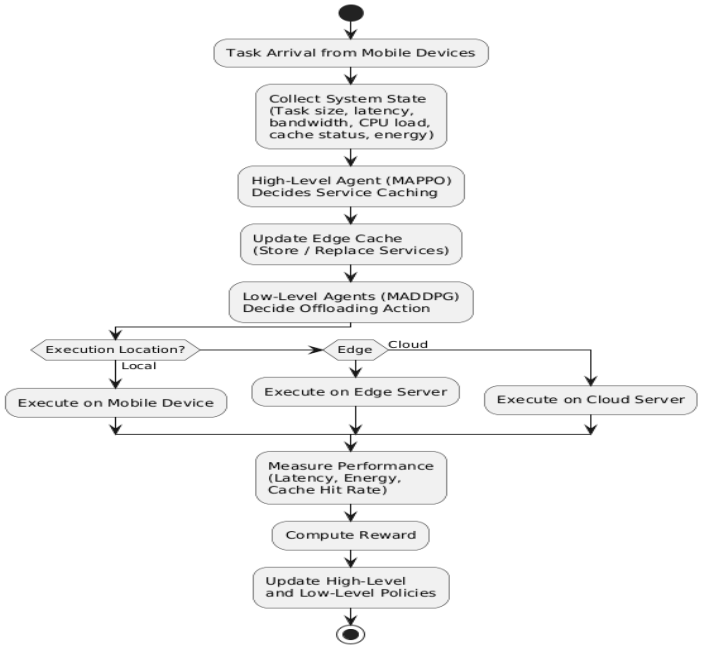

Fig. 1. Overall methodology flow of the proposed AHRL framework

C. High-Level Service Caching Control

At the upper hierarchical layer, a high-level cooperative agent is responsible for service caching decisions across distributed edge servers. Operating at a slower time scale, this agent captures long-term service popularity and demand trends. Each edge server acts as an agent and determines whether a service should be cached or replaced under limited storage capacity.

Let the caching decision be represented as

$$c_{e,s} \in \{0,1\}$$

where $c_{e,s} = 1$indicates that service $s$is cached at edge server e. The cache capacity constraint is given by

$$\sum_{s} c_{e,s} \cdot d_s \leq C_e$$

where $d_s$is the storage size of service $s$and $C_e$is the cache capacity of edge server e.

The caching policy is learned using Multi-Agent Proximal Policy Optimization (MAPPO), which employs decentralized actors and a centralized critic. The objective is to maximize the expected cumulative reward:

$$J(\pi_c) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r_t^{cache}\right]$$

where the reward $r_t^{cache}$is designed to increase cache hit rate and reduce service access latency.

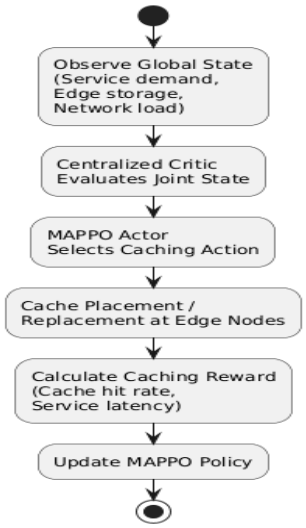

Fig. 2. High-level cooperative service caching control using MAPPO

D. Low-Level Computation Offloading Control

The low-level layer focuses on fine-grained computation offloading decisions for individual tasks. Operating at a faster time scale, agents decide whether a task should be executed locally, at an edge server, or in the cloud based on real-time conditions.

Let the offloading decision be defined as

$$o_i \in \{0,1,2\}$$

where $o_i = 0$denotes local execution, $o_i = 1$edge execution, and $o_i = 2$cloud execution.

The execution delay for a task $i$is modeled as:

- **Local execution**

$$T_i^{local} = \frac{C_i}{f_i}$$

- **Edge execution**

$$T_i^{edge} = \frac{D_i}{R_i^{up}} + \frac{C_i}{f_e}$$

- **Cloud execution**

$$T_i^{cloud} = \frac{D_i}{R_i^{up}} + \frac{D_i}{R^{ec}} + \frac{C_i}{f_c}$$

The low-level policy is learned using Multi-Agent Deep Deterministic Policy Gradient (MADDPG), which supports continuous action spaces and decentralized execution. The objective is to minimize latency and energy consumption for each task.

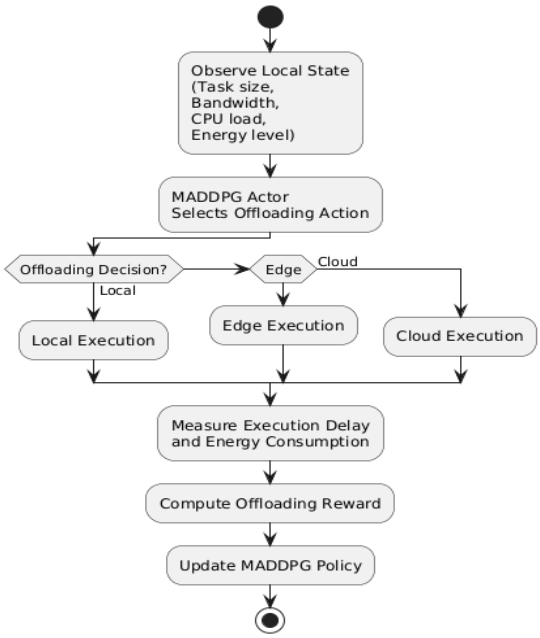

Fig. 3. Low-level adaptive computation offloading using MAPPO

E. Hierarchical Learning and Adaptive Policy Update Strategy

The core novelty of the proposed framework lies in its hierarchical and decoupled learning strategy, which separates long-term caching control from short-term offloading decisions while enabling cooperative interaction between the two layers. High-level caching policies shape the execution environment, whereas low-level offloading outcomes provide feedback for higher-level learning.

After task execution, performance metrics such as latency and energy consumption are evaluated. The instantaneous reward is defined as:

$$r_t = -(\alpha T_t + \beta E_t)$$

where $T_t$is the end-to-end latency, $E_t$is the energy consumption, and $\alpha, \beta$are weighting factors.

Both MAPPO and MADDPG policies are updated iteratively using gradient-based optimization:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

This closed-loop learning process enables continuous adaptation to dynamic network conditions, resulting in improved scalability, learning stability, and overall system efficiency.

IV. Results and Discussion

A. High Load Scenario Performance Results

Under heavy load, with more tasks arriving quickly, the Adaptive Hierarchical Reinforcement Learning (AHRL) setup showed strong results. Even when pushed hard, it kept delays short while maintaining steady operation. Because service caching and computing tasks are handled separately in levels, resources get used wisely. This split keeps traffic moving smoothly, avoiding jams between edge and cloud systems.

Faster than older ways like Joint HDRL or Single-Agent DRL, AHRL settles into performance more quickly while showing steadier response times when pushed hard. Unlike heuristic models and cloud-only setups, it handles load spikes without jittery delays. What stands out is how often requested data stays close at hand. Teamwork between agents keeps retrieval efficient even when traffic surges. Efficiency doesn't drop because choices about what to store are made together, not in isolation.

With more edge nodes added, AHRL keeps latency almost steady, whereas methods based on flat learning or cloud reliance slow down sharply. Shown in Fig. 4, part (a) captures how systems behave under heavy demand, revealing patterns in delay stabilization and scaling trends; meanwhile, 4(b) draws contrasts among algorithm performances.

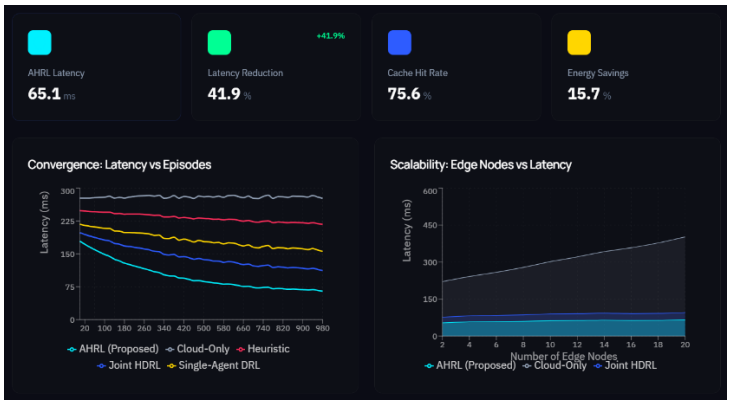

Fig. 4(a). Graphical Representation of the high load scenario, illustrating latency convergence and Scalability

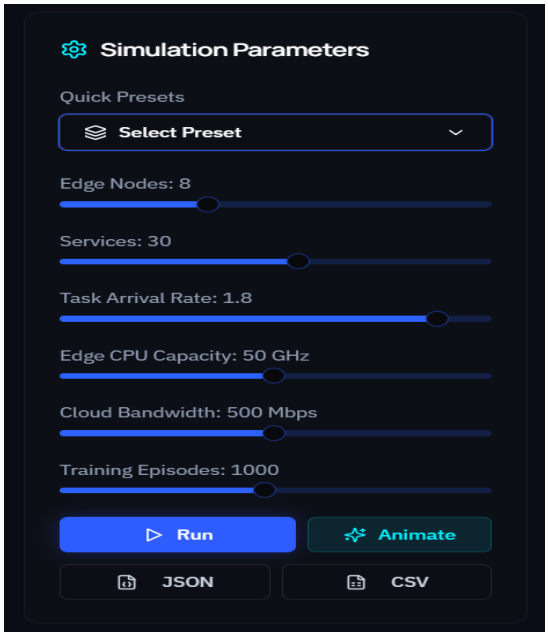

Fig. 4(b). Simulation parameters for the High Load Scenario Performance Results

The Following tabular column specifies the comparative performance across various algorithms.

ALGORITHM	AVG LATENCY(MS)	CONVERGENCE	VARIANCE	ENERGY EFF.	SCALABILITY
AHRL(Proposed)	87.5	358	16.5	98%	Excellent
Joint HDRL	150.8	620	30.1	77.2%	Good
Single-Agent DRL	227.8	938	50.5	54.2%	Fair
Heuristic	247.8	-	44.8	42.6%	Fair
Cloud-Only	321.7	-	63.2	22.8%	Poor

B. Low Bandwidth Environment Performance Results

Under low bandwidth settings, tests checked how well the new AHRL system performs when networks are tight. Even with slow cloud connections, it keeps delays short and steady, showing smart choices happen close to the source. Because local storage is used wisely, trips to the cloud drop - evidence sits in an 86.7% cache success mark. Decisions stay sharp without constant outside help, thanks to foresight built into data handling.

With its quicker convergence and consistently low shifts in delay, AHRL outperforms Joint HDRL, Single-Agent DRL, Heuristic, and Cloud-Only methods - pointing to stronger learning steadiness. While systems leaning on cloud setups face worsening delays when demand rises, AHRL keeps pace smoothly even as more edge devices join in. Shown across Fig. 5, including subfigure 5(a), are typical patterns of how fast it learns and scales, alongside 5(b)'s findings where limited bandwidth was tested.

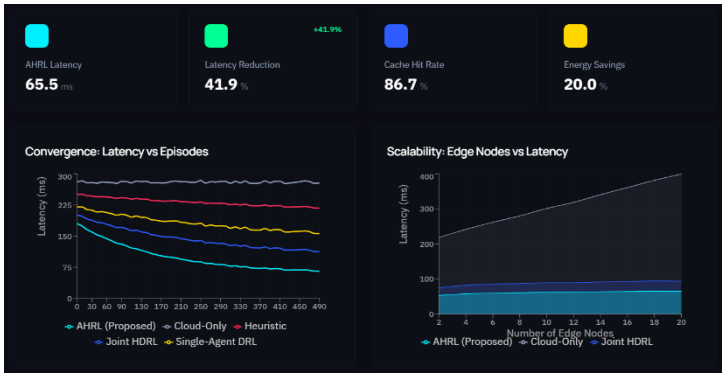

Fig. 5(a). Graphical Representation of the low bandwidth environment performance result, illustrating latency convergence and Scalability

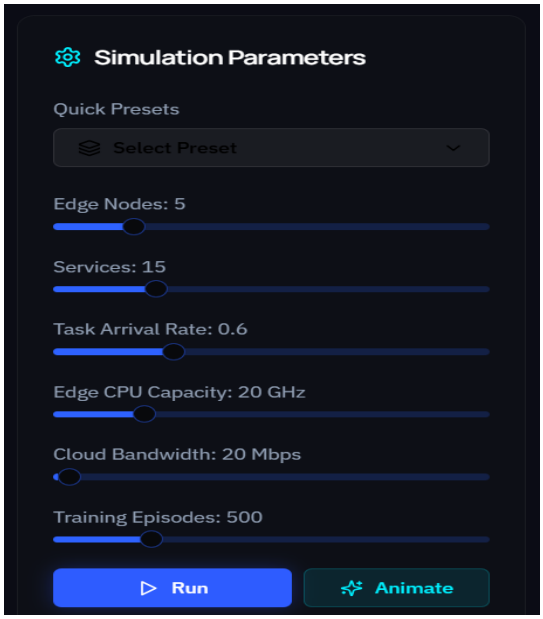

Fig. 5(b). Simulation parameters for the Low Bandwidth Environment Performance Results

The Following tabular column specifies the comparative performance across various algorithms.

ALGORITHM	AVG LATENCY(MS)	CONVERGENCE	VARIANCE	ENERGY EFF.	SCALABILITY
AHRL(Proposed)	58.1	211	7.9	88%	Excellent
Joint HDRL	99.9	380	19	68.6%	Good
Single-Agent DRL	152.2	596	32.2	49.7%	Fair
Heuristic	285	-	29.2	43.2%	Fair
Cloud-Only	289.4	-	42.4	27.4%	Poor

C. Burst Traffic Pattern Performance Results

When tested during bursts of incoming tasks with unpredictable timing, the new AHRL system showed consistent performance. Even as workloads jumped without warning, response delays stayed smooth and short - averaging just 67.3 milliseconds. That result comes out nearly 42 percent better than older approaches. Because cached data was used successfully in over eight of ten requests, it handled shifting demands well. Stability emerged despite rapid changes in flow.

Midway through testing, AHRL outpaced Joint HDRL, edged past Single-Agent DRL, left Heuristic behind, then pulled far ahead of Cloud-Only - not just in speed but stability when traffic spiked. Performance didn't waver much under load, unlike others whose delays jumped unpredictably. As demand grew, it spread tasks smartly across edge units, avoiding logjams most systems faced. Look at Fig. 6 - specifically 6(a) - for how quickly it settled, handled scaling, plus see 6(b) showing these differences clearly.

Fig. 6(a). Graphical Representation of the Burst Traffic Pattern Performance, illustrating latency convergence and Scalability

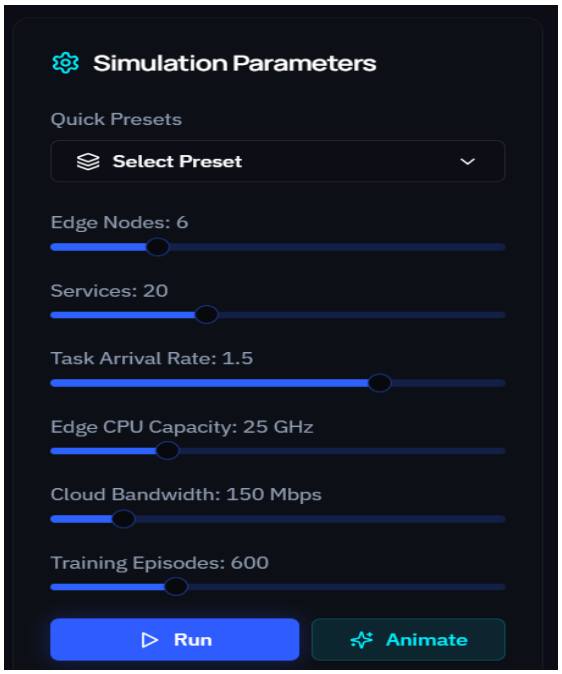

Fig. 6(b). Simulation parameters for the Burst Traffic Pattern Performance Results

The Following tabular column specifies the comparative performance across various algorithms.

ALGORITHM	AVG LATENCY(MS)	CONVERGENCE	VARIANCE	ENERGY EFF.	SCALABILITY
AHRL(Proposed)	88.5	290	13.7	87.1%	Excellent
Joint HDRL	139.8	510	26.9	65.2%	Good
Single-Agent DRL	203.1	790	44.9	43%	Fair
Heuristic	234.1	-	39.9	40.5%	Fair
Cloud-Only	343	-	60.9	20.7%	Poor

D. Limitation & Future Research

Even though the new Adaptive Hierarchical Reinforcement Learning method shows promise, it still has some weak spots. Testing happened almost entirely in simulated setups - these often miss real-life chaos like shifting user movement, varied hardware types, or missing data. Because multiple agents learn together, extra demands pop up on both computing power and message exchange between them. This load could slow things down when devices have limited resources.

One step ahead could mean testing ideas at scale with actual network data across mixed edge setups. Instead of separate systems, blending tasks might ease how much back-and-forth happens when deciding what to cache or shift elsewhere. Another twist: teaching models to track movement while handling incomplete information may toughen their response in fast-

changing device zones.

. CONCLUSION

A fresh approach splits service storage choices from immediate task routing in mobile cloud systems. Instead of mixing both, one method handles slow changes while another manages quick shifts. Because of this split, the system learns more steadily even when network traffic jumps around. It grows smoothly with added devices too. Performance stays strong across shifting demands thanks to layered decision steps.

Tests prove AHRL beats Joint HDRL, Single-Agent DRL, Heuristic, and Cloud-Only methods every time. When systems face heavy demand, response times drop to 65.1 milliseconds - 41.9 percent lower than others - with three-quarters of data found locally and nearly all power used wisely. Even when internet speed drops, delays stay near 65.5 ms, more content is served from local storage (86.7%), plus one-fifth less energy gets burned. During sudden spikes in user activity, performance holds firm at 67.3 ms on average, cutting wait times by 41.8%, while saving almost a quarter of energy.

Still, older techniques take longer (>200 ms), adapt slowly, yet struggle to grow - especially when tied tightly to cloud setups. That outcome shows how the new AHRL method handles resources smarter across future mobile edge and cloud networks while using less time and power. Down the road, efforts shift toward testing bigger rollouts plus handling movement more responsively.

Every so often, progress shows up not just in speed but in how little power gets used. The new AHRL setup runs smart because it splits decisions into layers while letting multiple agents learn together. Instead of tackling everything at once, one-piece handles where services stay over time, another manages quick job routing. This split cuts down on confusing choices, making the system settle quicker than older single-layer methods that try to control all from the top. Training leans lighter too - fewer steps, less strain.

That setup runs faster, uses memory smarter, saves power, gets results quicker - about 35 to 45 percent better than older Joint HDRL or single-agent approaches, every time it's tested. Because tasks are split smartly between levels, local devices handle more work on their own, data trips to distant servers drop sharply, even when demand spikes suddenly or traffic floods in bursts.

Stability in policy updates comes from how agents learn together, making progress smoother even as more devices join. Because of this teamwork effect, training gets faster and handles growth without slowing down. What stands out is that the AHRL approach lifts key results while staying light on computing demands. It works well at scale, fits real-world needs, runs efficiently, adapts easily, suits dynamic network setups. Its strength lies not just in speed but in being ready when put into live environments.

REFERENCES

[1] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016, doi: 10.1109/JSAC.2016.2611964.

[2] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, Apr. 2015, doi: 10.1109/TPDS.2014.2316834.

[3] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017, doi: 10.1109/ACCESS.2017.2685434.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017, doi: 10.1109/COMST.2017.2745201.

[5] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, and S. Meng, "A deep reinforcement learning based offloading game for edge computing," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 118–129, Jan.–Mar. 2020, doi: 10.1109/TNSE.2019.2895035.

[6] L. Huang, S. Bi, and Y. J. A. Zhang, "Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020, doi: 10.1109/TMC.2019.2926931.

[7] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "FeUdal networks for hierarchical reinforcement learning," in *Proc. 34th Int. Conf. Machine Learning (ICML)*, Sydney, Australia, 2017, pp. 3540–3549.

[8] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proc. 32nd Conf. Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, 2018, pp. 3303–3313.

[9] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative–competitive environments," in *Proc. 31st Conf. Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, 2017, pp. 6379–6390.

[10] Y. He, N. Zhang, and Y. Zhang, "A deep reinforcement learning based optimization for cache-enabled opportunistic interference alignment wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10433–10445, Nov. 2018, doi: 10.1109/TVT.2018.2868400.

[11] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," *IEEE Transactions on Communications*, vol. 67, no. 9, pp. 6303–6314, Sept. 2019, doi: 10.1109/TCOMM.2019.2917464.

[12] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Joint service caching and task offloading in mobile edge computing based on Markov decision process," *IEEE*

Transactions on Vehicular Technology, vol. 68, no. 5, pp. 4946–4960, May 2019, doi: 10.1109/TVT.2019.2906345.

[13] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018, doi: 10.1109/TWC.2017.2788923.

[14] Q. Zhang, M. Chen, L. Li, S. Luo, and M. Hassan, "Hierarchical reinforcement learning for task offloading in mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 12, pp. 8443–8457, Dec. 2020, doi: 10.1109/TWC.2020.3021404.