ATLAS: Adaptive Task Offloading System in Multi-UAV-assisted Multi-Access Edge Computing

Taewon Song and Taeyoon Kim

***Abstract*—Multi-UAV task offloading in edge computing environments requires coordinated decision-making under dynamic network conditions and limited computational resources. This paper presents an A3C-based multi-UAV task offloading system that leverages parameter sharing among distributed workers to achieve superior generalization performance across diverse operational conditions. We formulate the problem as a partially observable Markov decision process where UAV agents select optimal offloading actions, specifically, local processing, MEC offloading, or task discard, based on computational state, queue status, and network conditions. The proposed system employs a recurrent actor-critic architecture with layer normalization to handle sequential dependencies and stabilize asynchronous training across multiple workers. Experimental evaluation demonstrates that the A3C-based approach with parameter sharing significantly outperforms independent learning strategies, achieving enhanced mean performance, improved stability, and superior worst-case robustness. Through systematic ablation studies, we validate our architectural design choices and identify critical system parameters that ensure A3C's advantages. The results provide practical guidelines for deploying distributed reinforcement learning in multi-UAV edge computing systems, demonstrating when coordinated learning approaches provide genuine benefits over independent policies.**

***Index Terms*—UAV, Task Offloading, A3C, Deep Reinforcement Learning, Multi-Agent Systems, Multi-Access Edge Computing**

I. Introduction

The proliferation of Unmanned Aerial Vehicles (UAVs) in applications ranging from surveillance to emergency response has created unprecedented opportunities for distributed edge computing. Multi-UAV systems must make real-time decisions about computational task offloading while operating under stringent energy constraints, dynamic network conditions, and limited onboard resources. As UAV deployments scale to swarms of tens or hundreds of agents, the challenge of coordinating task distribution across heterogeneous edge computing infrastructure becomes increasingly critical for system performance and mission success.

While existing approaches have explored various task offloading strategies [1]–[5], a fundamental gap remains in understanding how to effectively coordinate learning among distributed UAV agents. Traditional centralized optimization methods struggle with the dynamic, partially observable nature of multi-UAV environments, while independent learning approaches fail to leverage collective experience across the swarm. The key challenge lies in designing a distributed reinforcement learning system that can share knowledge effectively among heterogeneous workers operating in diverse conditions, maintain stable performance across varying network topologies and resource availability, and generalize to previously unseen operational scenarios without requiring complete retraining.

Consider a disaster response scenario where multiple UAVs must coordinate to process sensor data, execute path planning, and transmit critical information to ground stations. Each UAV is equipped with a local mobile edge computing (MEC) server and faces a continuous stream of computational tasks. For each incoming task, the UAV must decide whether to (1) process it locally using its onboard computing resources (consuming battery energy but avoiding communication delays), (2) offload it to cloud servers on the core network through wireless backhaul links (leveraging superior computational capacity but incurring transmission costs, network latency, and potential communication failures), or (3) discard the task when neither local nor cloud processing is feasible (sacrificing task completion to conserve critical resources). The optimal decision depends on the current system state observed by each UAV, which includes local computational state, queue state, cloud server state, local computation capacity, and the wireless channel conditions. Moreover, environmental heterogeneity means that UAVs operating in different regions encounter vastly different network conditions and resource availability, making coordinated learning through parameter sharing essential for achieving robust system-wide performance.

In this paper, we present an A3C-based multi-UAV task offloading system that leverages parameter sharing among distributed workers to achieve superior generalization performance across diverse operational conditions. Our system employs a recurrent actor-critic architecture with layer normalization to handle sequential decision dependencies and stabilize asynchronous training across multiple heterogeneous workers.

We formulate the multi-UAV task offloading problem as a partially observable Markov decision process (POMDP) where each UAV agent selects actions based on the observations. The proposed architecture integrates recurrent neural networks to capture temporal dependencies in wireless channel conditions and resource availability on the cloud size, which might be hidden on the UAV side, while layer normalization ensures stable gradient updates across asynchronously training workers. Through systematic experimentation across varying resource constraints, network velocities, and exploration parameters,

T. Song is with the Department of Internet of Things, College of SW Convergence, Soonchunhyang University, 22 Soonchunhyang-ro, Shinchang-myeon, Asan-si, Chungcheongnam-do, 31538, Korea (e-mail: twsong@sch.ac.kr) and T. Kim is with ... (e-mail: 2000kty@dankook.ac.kr), Corresponding author: T. Kim.

This work was supported in part by ..., and in part by ..., and in part by ...

we validate our design choices and identify the operational regimes where distributed learning provides genuine advantages over independent policies.

The main contributions of this paper are as follows:

- We design and implement an A3C-based multi-UAV task offloading system with a carefully architected recurrent actor-critic network that achieves superior generalization performance through parameter sharing among distributed workers.
- We conduct comprehensive ablation studies across architectural components, hyperparameters, and environmental factors to validate our design choices and establish practical deployment guidelines.
- We identify critical operational regimes—including resource availability, exploration requirements, and environmental dynamics—that determine whether A3C provides genuine benefits, offering practitioners clear criteria for algorithm selection in multi-UAV deployments.

The remainder of this paper is organized as follows. Section II reviews related work in UAV task offloading and multi-agent reinforcement learning. Section III formulates the problem as a POMDP and describes our system architecture. Section IV details the experimental setup and evaluation methodology. Section V presents performance evaluation results and ablation study findings. Section VI discusses implications and limitations, and Section VII concludes the paper.

II. Related Works

The intersection of UAV-assisted mobile edge computing and deep reinforcement learning has emerged as a critical research area, with various approaches addressing the challenges of distributed task offloading optimization. This section reviews existing work across three key themes: reinforcement learning methods for task offloading, multi-agent coordination and parameter sharing strategies, and architectural components for handling partial observability in sequential decision-making systems.

A. Reinforcement Learning for UAV Task Offloading

Deep reinforcement learning has been extensively applied to UAV task offloading problems due to its ability to handle complex, dynamic environments without requiring explicit models of system dynamics.

[1], [6], [7], [4], [5], [8],

While these approaches demonstrate the effectiveness of deep RL for task offloading, they primarily focus on single-agent scenarios or do not systematically compare centralized parameter sharing versus independent learning strategies. Moreover, limited attention has been given to A3C algorithms specifically designed for multi-UAV coordination with parameter sharing. **Gap 1:** Existing work does not systematically compare A3C's global parameter sharing against independent learning strategies, nor does it identify the operational regimes where distributed learning provides genuine advantages.

B. Distributed Learning and Parameter Sharing

Multi-agent reinforcement learning for edge computing systems requires careful consideration of how knowledge is shared among distributed agents.

[9], [10], [11], [12], [13], [14],

While various distributed learning paradigms have been explored, existing work lacks systematic comparison between A3C with global parameter sharing and fully independent worker training in the context of UAV task offloading. **Gap 2:** Current research provides limited guidance on when distributed parameter sharing provides genuine advantages over independent learning. The operational regimes—including resource availability, exploration requirements, and environmental dynamics—that determine algorithm effectiveness remain unclear, particularly across heterogeneous operational environments.

C. Architecture Components in Deep Reinforcement Learning

The design of neural network architectures significantly impacts the performance and stability of deep RL systems, particularly in partially observable environments.

While individual architectural components have been studied in isolation, comprehensive ablation studies that systematically validate design choices across multiple dimensions remain limited. **Gap 3:** Existing studies lack comprehensive ablation analysis that validates architectural design choices (RNN, layer normalization) in the context of multi-agent asynchronous training across architectural components, hyperparameters, and environmental factors. The interaction effects between recurrent components and normalization techniques in multi-agent asynchronous training have not been thoroughly investigated, particularly regarding their impact on training stability and generalization performance.

Addressing these three research gaps, this paper presents a carefully designed A3C-based multi-UAV task offloading system with comprehensive ablation studies that provide both system design contributions and fundamental insights into when and why distributed learning outperforms independent approaches.

III. POMDP Formulation

A. Problem Formulation

We formulate the multi-UAV task offloading optimization problem as a Partially Observable Markov Decision Process (POMDP), where each UAV agent operates as an intelligent decision maker in a dynamic edge computing environment. The POMDP framework enables UAV agents to make sequential decisions for incoming computational tasks while adapting to changing environmental conditions and resource availability. Furthermore, the POMDP formulation captures the inherent uncertainty in UAV operations, such as fluctuating network conditions and the computation capability of the cloud servers for task offloading.

Formally, the ATLAS framework models the task offloading problem as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the state transition function, $\mathcal{R}$ is

the reward function, and $\gamma \in [0,1]$ is the discount factor. At time t, each UAV agent can be in a state $s_t \in \mathcal{S}$, then selects an action $a_t \in \mathcal{A}$ according to its policy $\pi(a_t|s_t)$, and receives a reward $r_t = \mathcal{R}(s_t, a_t)$ while transitioning to the next state $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

The primary objective is to learn an optimal policy π^* that maximizes the expected cumulative discounted reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi\right]. \tag{1}$$

This formulation enables UAV agents to balance multiple competing objectives including energy efficiency, processing latency minimization, and task completion success rates in dynamic edge computing scenarios.

B. State Space

In the POMDP formulation, the state space $\mathcal{S}$ consists of both observable and hidden components, reflecting the partial observability inherent in multi-UAV edge computing environments. UAV agents can directly observe their local states, environmental context, and nearby MEC server status through edge feedback, while remote cloud infrastructure and wireless channel conditions remain hidden due to limited sensing capabilities and communication constraints.

We formally decompose the state space into observable and hidden components:

$$\mathcal{S} = \mathcal{S}_{obs} \times \mathcal{S}_{hidden} \tag{2}$$

where $\mathcal{S}_{obs} = \mathbf{S_{local}} \times \mathbf{S_{context}} \times \mathbf{S_{mec}}$ represents **observable states** directly measurable by the UAV agent, and $\mathcal{S}_{hidden} = \mathbf{S_{cloud}} \times \mathbf{S_{channel}}$ represents **hidden states** that are not directly observable due to the asynchronous nature of cloud information ($\mathbf{S_{cloud}}$) or limitations in wireless channel sensing ($\mathbf{S_{channel}}$). The recurrent neural network architecture enables the agent to maintain internal memory and infer hidden state components through temporal patterns in observable measurements.

1) Observable State Components: The observable state components provide direct measurements available to each UAV agent:

Local Device State ($\mathbf{S_{local}}$): This component characterizes the UAV's local computational resources, task queue, and processing feedback:

$$\mathbf{S_{local}} = \mathbf{C_l} \times \mathbf{E_r} \times \mathbf{C_q} \times \mathbf{T_q} \times \mathbf{I_l} \times \mathbf{I_o} \tag{3}$$

where $\mathbf{C_l} = [0,1]$ represents a set of available computation units, $\mathbf{E_r} = [0,1]$ denotes a set of remaining epochs during one episode, $\mathbf{C_q} = [0,1]$ and $\mathbf{T_q} = [0,1]$ capture sets of queue computation units and processing times, and $\mathbf{I_l}, \mathbf{I_o} = \{0,1\}$ provide binary success indicators for local and offload operations.

Environmental Context ($\mathbf{S_{context}}$): This encompasses dynamic environmental conditions directly measurable by the UAV:

$$\mathbf{S_{context}} = \mathbf{V} \times \mathbf{C} \tag{4}$$

where $\mathbf{V} = [0,1]$ represents normalized velocity context, and $\mathbf{C} = [0,1]$ indicates normalized local computation capacity context. We set the minimum and the maximum velocities to $30\,\mathrm{km/h}$ and $100\,\mathrm{km/h}$ [15], respectively, and $\mathbf{V}$ is normalized according to these marginal velocity values.

MEC Server State ($\mathbf{S_{mec}}$): This component captures the observable state of nearby, or physically attached mobile edge computing servers through feedback mechanisms:

$$\mathbf{S_{mec}} = \mathbf{C_m} \times \mathbf{T_m} \tag{5}$$

where $\mathbf{C_m} = \prod_{k \in K} \mathbf{C_{m,k}}$ represents normalized computation units, with $\mathbf{C_{m,k}} = [0,1]$ for each task $k \in K$, and $\mathbf{T_m} = \prod_{k \in K} \mathbf{T_{m,k}}$ denotes processing times, with $\mathbf{T_{m,k}} = [0,1]$ for each task $k \in K$. UAV agents can observe MEC state through acknowledgement messages and status updates from edge infrastructure.

2) Hidden State Components: The hidden state components represent system information not directly observable by UAV agents, requiring inference through recurrent processing:

Cloud Server State ($\mathbf{S_{cloud}}$): This component models the remote cloud infrastructure availability in the core network, which is hidden from direct UAV observation because of the aggregated nature of cloud resources serving numerous MEC servers simultaneously:

$$\mathbf{S_{cloud}} = \mathbf{C_c} \times \mathbf{T_c} \tag{6}$$

where $\mathbf{C_c} = \prod_{k \in K} \mathbf{C_{c,k}}$ represents normalized computation units, with $\mathbf{C_{c,k}} = [0,1]$ for each task $k \in K$, and $\mathbf{T_c} = \prod_{k \in K} \mathbf{T_{c,k}}$ denotes processing times, with $\mathbf{T_{c,k}} = [0,1]$ for each task $k \in K$. The shared and distributed architecture of cloud infrastructure, coupled with the large number of attached MEC nodes, makes direct state monitoring infeasible from individual UAV perspectives.

Channel State ($\mathbf{S_{channel}}$): This captures the wireless communication quality between UAV and infrastructure as

$$\mathbf{S_{channel}} = \mathbf{Q} \tag{7}$$

where $\mathbf{Q} \in \{0,1\}$ denotes binary channel quality indicators, with 0 and 1 representing bad and good channel states, respectively. The channel state follows a finite-state Markov channel (FSMC) formulation, adopting a two-state representation derived from the Gilbert–Elliott model [16], [17] for Rayleigh fading channels [18].

C. Action Space

The action space $\mathcal{A}$ in the ATLAS framework defines the discrete set of decisions available to each UAV agent when processing incoming computational tasks. We formulate the action space as a finite discrete set:

$$\mathcal{A} = \{0, 1, 2\} \tag{8}$$

where where 0, 1, and 2 stand for each defined action, respectively. A specific procedure will be stated as follows.

- **Local Processing ($a = 0$):** When this action is selected, the UAV processes the computational task using its associating MEC server. The local processing action triggers

immediate computation on the UAV, consuming local computational resources but avoiding transmission delays and potential network failures.

- **Offloading** ($a = 1$)**:** This action involves transmitting the computational task to available cloud servers for remote processing. The offloading action incurs transmission energy costs and may introduces network-dependent failures, but potentially reduces local computational load and enables processing of resource-intensive tasks.
- **Task Discard** ($a = 2$)**:** When system conditions are unfavorable for both local processing and offloading, because of insufficient local resources, saturated server queues, or poor channel conditions that prevent successful task completion, the agent may choose to discard the task. Task discard minimizes immediate resource consumption but results in task failure penalties in the reward function.

D. State Transition Function

The state transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines the probability distribution over next states given the current state and action. We denote the current state s and next state s' as:

$$s = \{c_l, e_r, c_q, t_q, i_l, i_o, v, c, \mathbf{c}_m, \mathbf{t}_m, \mathbf{c}_c, \mathbf{t}_c, q\} \tag{9}$$

and

$$s' = \{c'_l, e'_r, c'_q, t'_q, i'_l, i'_o, v', c', \mathbf{c}'_m, \mathbf{t}'_m, \mathbf{c}'_c, \mathbf{t}'_c, q'\} \tag{10}$$

where $\mathbf{c}_m, \mathbf{t}_m, \in \mathbb{R}^K$ and $\mathbf{c}_c, \mathbf{t}_c \in \mathbb{R}^K$ represent local-side MEC and cloud server-side queue state vectors, respectively.

The complete transition function can be decomposed into **endogenous** (agent-internal) and **exogenous** (agent-external or environment-driven) components, as mentioned in (2):

$$\Pr[s'|s, a] = \Pr[s'_{endo}|s, a] \times \Pr[s'_{exo}|s], \tag{11}$$

where $s'_{endo} \in \mathcal{S}_{endo}$ and $s'_{exo} \in \mathcal{S}_{exo}$, respectively, making sure that the dynamics of exogenous states are independent from the agent's action. The endogenous components evolve according to controllable system dynamics influenced by the agent's actions, while the exogenous components evolve independently, modeling external task arrivals and environmental conditions. This decomposition explicitly separates controllable resource allocation from uncontrollable workload variations, enabling the agent to learn robust policies under stochastic task arrivals.

1) Endogenous State Transitions: The endogenous components are directly influenced by the agent's actions and evolve according to system dynamics:

$$\begin{aligned}
\Pr[s'_{\text{endo}}|s, a] = &\Pr[c'_l|c_l, c_q, a] \\
&\times \Pr[e'_r|e_r] \\
&\times \Pr[i'_l|c_l, c_q, \mathbf{c}_m, a] \\
&\times \Pr[i'_o|c_q, a] \\
&\times \prod_{k=1}^{K} \Pr[c'_{m,k}|c_{m,k}, t_{m,k}, a] \\
&\times \prod_{k=1}^{K} \Pr[t'_{m,k}|t_{m,k}]
\end{aligned} \tag{12}$$

2) Exogenous State Transitions: The exogenous components evolve independently of the agent's actions and model external task arrivals and environmental variations:

$$\Pr[s'_{\text{exo}}|s] = \Pr[c'_q] \times \Pr[t'_q] \times \Pr[v'] \times \Pr[c'] \tag{13}$$

Queue Task Arrivals (c_q, t_q)**:** New tasks arrive at each time step with computational requirements and processing times sampled from uniform distributions:

$$c'_q \sim \text{Uniform}(1, 200), \quad t'_q \sim \text{Uniform}(1, 50) \tag{14}$$

This stochastic arrival process models the unpredictable workload in real-world UAV applications.

Environmental Context (v, c)**:** Velocity and computation capacity contexts remain constant within an episode, representing the environmental configuration:

$$v' = v, \quad c' = c \tag{15}$$

The complete state transition function combines both components:

$$\Pr[s'|s, a] = \Pr[s'_{\text{endo}}|s, a] \times \Pr[s'_{\text{exo}}|s] \tag{16}$$

The state transition dynamics are influenced by several key factors:

Computational Resource Evolution: The computational state transitions depend on task processing outcomes and resource consumption:

$$c'_{avail} = \begin{cases} \max(0, c_{avail} - \Delta c_{local}) & \text{if } a_t = a_{LOC} \\ c_{avail} & \text{if } a_t = a_{OFF} \\ c_{avail} & \text{if } a_t = a_{DIS} \end{cases} \tag{17}$$

where Δc_{local} represents the normalized computational resources consumed for local processing.

Queue State Dynamics: The queue state evolves based on task arrivals, processing actions, and completion events:

$$q'_{units} = f_{queue}(q_{units}, a_t, \text{new\_tasks}, \text{completed\_tasks}) \tag{18}$$

Queue updates incorporate the stochastic nature of task arrivals and the deterministic effects of processing actions.

Environment State Changes: MEC server states evolve based on system load and external factors:

$$e'_{mec\_units} \sim P_{MEC}(e_{mec\_units}|e_{mec\_units}, \text{system\_load}) \tag{19}$$

The MEC state transitions follow a semi-Markov process reflecting the dynamic nature of edge computing infrastructure.

Vehicle Mobility: UAV contextual states evolve according to mobility patterns and mission requirements:

$$v'_{velocity} = v_{velocity} + \epsilon_{velocity} \tag{20}$$

where $\epsilon_{velocity}$ represents bounded random variations in UAV velocity based on environmental conditions and flight dynamics.

The complete transition function captures the stochastic nature of the UAV environment while maintaining computational tractability for reinforcement learning optimization.

E. Reward Function

The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ provides immediate feedback to guide the UAV agent's learning process by quantifying the desirability of taking action a_t in state s_t. The ATLAS framework designs a multi-objective reward function that balances energy efficiency, processing latency, and task completion success.

The reward function is formulated as a weighted combination of multiple cost and benefit components:

$$\mathcal{R}(s_t, a_t) = -\alpha \cdot C_{energy}(s_t, a_t) - \beta \cdot C_{latency}(s_t, a_t) - \gamma \cdot C_{failure}(s_t, a_t) + \delta \cdot B_{completion}(s_t, a_t) \tag{21}$$

where α, β, γ, and δ are positive weighting coefficients that control the relative importance of each objective component.

Energy Cost (C_{energy}): This component penalizes energy consumption for different actions:

$$C_{energy}(s_t, a_t) = \begin{cases} E_{local} \cdot q_{units} & \text{if } a_t = a_{LOC} \\ E_{transmission} \cdot q_{units} & \text{if } a_t = a_{OFF} \\ 0 & \text{if } a_t = a_{DIS} \end{cases} \tag{22}$$

where E_{local} and $E_{transmission}$ represent normalized energy costs for local processing and task transmission, respectively.

Latency Cost ($C_{latency}$): This component accounts for processing and communication delays:

$$C_{latency}(s_t, a_t) = \begin{cases} T_{local}(q_{units}, c_{avail}) & \text{if } a_t = a_{LOC} \\ T_{offload}(q_{units}, e_{mec\_times}) + T_{transmission} & \text{if } a_t = a_{OFF} \\ 0 & \text{if } a_t = a_{DIS} \end{cases} \tag{23}$$

where T_{local} and $T_{offload}$ are functions computing processing times based on task requirements and available resources.

Failure Penalty ($C_{failure}$): This component penalizes unsuccessful task processing:

$$C_{failure}(s_t, a_t) = \begin{cases} P_{fail\_local} \cdot \rho_{failure} & \text{if } a_t = a_{LOC} \\ P_{fail\_offload} \cdot \rho_{failure} & \text{if } a_t = a_{OFF} \\ \rho_{discard} & \text{if } a_t = a_{DIS} \end{cases} \tag{24}$$

where P_{fail_local} and $P_{fail\_offload}$ are failure probabilities, and $\rho_{failure}$ and $\rho_{discard}$ are penalty magnitudes.

Completion Benefit ($B_{completion}$): This component provides positive rewards for successful task completion:

$$B_{completion}(s_t, a_t) = \begin{cases} (1 - P_{fail\_local}) \cdot \eta_{success} & \text{if } a_t = a_{LOC} \\ (1 - P_{fail\_offload}) \cdot \eta_{success} & \text{if } a_t = a_{OFF} \\ 0 & \text{if } a_t = a_{DIS} \end{cases} \tag{25}$$

where $\eta_{success}$ represents the positive reward magnitude for successful task completion.

This multi-objective reward design enables the ATLAS system to learn policies that optimize the trade-off between energy consumption, processing latency, and task success rates across diverse operational conditions.

F. A3C Architecture

The A3C implementation employs both feedforward and recurrent neural network configurations to handle the sequential nature of the task offloading decisions.

1) Network Architecture: The actor-critic architecture consists of:

- **Actor Network**: Outputs action probabilities using a softmax policy
- **Critic Network**: Estimates state values for advantage calculation
- **Hidden Layers**: 128 neurons with ReLU activation functions
- **Recurrent Component**: GRU layers for sequence modeling (when enabled)

For the recurrent variant (RecurrentActorCritic), the network processes sequences of observations through GRU layers before the final actor and critic heads, enabling the model to maintain memory of past decisions and environmental states.

2) Training Strategies: We evaluate two distinct training paradigms:

A3C Global Training: In this centralized approach, all workers share updates to a single global model. Each worker interacts with its assigned environment and computes gradients based on the advantage actor-critic loss:

$$L = L_{policy} + \beta \cdot L_{value} - \alpha \cdot H(\pi) \tag{26}$$

where L_{policy} is the policy loss, L_{value} is the value function loss, and $H(\pi)$ is the policy entropy term for exploration.

The global model aggregates updates from all workers asynchronously, promoting knowledge sharing across different environmental conditions and worker experiences.

Individual Worker Training: In this decentralized approach, each worker develops its own specialized policy independently. Workers train separate neural networks without sharing parameters, allowing for environment-specific adaptation and specialized behavior development.

G. Environment Configuration

The experimental framework utilizes a custom UAV environment with the following key parameters:

- Maximum computation units: 200
- Maximum computation units for cloud: 1000
- Maximum epoch size: 100
- Maximum queue size: 20
- Agent velocities: 50 units

Five distinct environmental configurations are evaluated, each representing different operational scenarios with varying computational demands, network conditions, and resource availability patterns.

H. Performance Metrics

Our evaluation framework includes the following key metrics:

- Episode-level total rewards
- Convergence characteristics across training episodes
- Statistical significance of performance differences
- Cross-environment generalization performance

Algorithm 1 Training phase of ATLAS using A3C with RNN

Initialize global actor-critic network θ with recurrent architecture
Initialize shared optimizer and cloud resource state $\mathcal{R}_{\text{cloud}}$
for each worker $w \in \{1, \ldots, W\}$ **in parallel do**
  $\theta_w \leftarrow \theta$ ▷ Copy global parameters
  for episode $e = 1$ to $E_{\max}$ **do**
    Initialize state s_0 and RNN hidden state $h_0 \leftarrow \mathbf{0}$
    while episode not terminated **do**
      // Collect T-step rollout
      for $t = 0$ to $T - 1$ **do**
        Compute $\pi(\cdot|s_t, h_t; \theta_w)$ and $V(s_t, h_t; \theta_w)$
        Sample $a_t \sim \pi(\cdot|s_t, h_t; \theta_w)$
        Execute a_t, observe r_t and s_{t+1}
        Update h_{t+1} via GRU; if done, $h_{t+1} \leftarrow \mathbf{0}$
      end for
      // Compute returns and advantages
      Bootstrap $R \leftarrow V(s_T, h_T; \theta_w)$ if not done, else $R \leftarrow 0$
      for $t = T - 1$ down to 0 **do**
        $R \leftarrow r_t + \gamma R$; $A_t \leftarrow R - V(s_t, h_t; \theta_w)$
      end for
      // Compute loss
      $\mathcal{L} \leftarrow -\sum_t \log \pi(a_t|s_t, h_t; \theta_w) A_t + \beta_v \sum_t (R_t - V_t)^2 - \beta_h \mathcal{H}(\pi)$
      // Asynchronous update
      Compute gradients $\nabla_{\theta_w} \mathcal{L}$ and clip
      Copy gradients to global: $\nabla_\theta \leftarrow \nabla_{\theta_w}$
      Update global: $\theta \leftarrow \theta - \alpha \nabla_\theta$ (SharedAdam)
      Synchronize local: $\theta_w \leftarrow \theta$
    end while
  end for
end for
return Trained global network θ

IV. Experimental Setup

A. Implementation Details

B. Environment Configurations

Five distinct environmental configurations were evaluated, each representing different operational scenarios with varying computational demands and network conditions.

C. Training Parameters

V. Results and Analysis

A. Performance Comparison

B. Episode-level Analysis

C. Distribution Analysis

D. Cross-Environment Performance

E. Statistical Significance

VI. Discussion

VII. Conclusion

References

[1] A. M. Seid, G. O. Boateng, B. Mareri, G. Sun, and W. Jiang, "Multi-agent drl for task offloading and resource allocation in multi-uav enabled iot edge network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4531–4547, 2021.

[2] M. Wang, L. Zhang, P. Gao, X. Yang, K. Wang, and K. Yang, "Stackelberg-game-based intelligent offloading incentive mechanism for a multi-uav-assisted mobile-edge computing system," *IEEE Internet of Things Journal*, vol. 10, no. 17, pp. 15 679–15 689, 2023.

[3] H. Guo, Y. Wang, J. Liu, and C. Liu, "Multi-uav cooperative task offloading and resource allocation in 5g advanced and beyond," *IEEE Transactions on Wireless Communications*, vol. 23, no. 1, pp. 347–359, 2024.

[4] H. Hao, C. Xu, W. Zhang, S. Yang, and G.-M. Muntean, "Joint task offloading, resource allocation, and trajectory design for multi-uav cooperative edge computing with task priority," *IEEE Transactions on Mobile Computing*, vol. 23, no. 9, pp. 8649–8663, 2024.

[5] B. Li, R. Yang, L. Liu, J. Wang, N. Zhang, and M. Dong, "Robust computation offloading and trajectory optimization for multi-uav-assisted mec: A multiagent drl approach," *IEEE Internet of Things Journal*, vol. 11, no. 3, pp. 4775–4786, 2024.

[6] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in uav-assisted mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6949–6962, 2022.

[7] T. Song, "Drqn-based task offloading in uav-assisted mobile edge computing environments with hidden channel conditions," in *2024 15th International Conference on Information and Communication Technology Convergence (ICTC)*, 2024, pp. 2153–2154.

[8] S. Guo, Z. Li, J. Yan, and X. Zhao, "Cognitive uav-assisted offloading for mobile edge computing based on multi-agent deep reinforcement learning," in *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, 2025, pp. 1–6.

[9] H. H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, "Federated reinforcement learning," *CoRR*, vol. abs/1901.08277, 2019. [Online]. Available: http://arxiv.org/abs/1901.08277

[10] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated

deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9441–9455, 2020.

[11] P. Tehrani, F. Restuccia, and M. Levorato, "Federated deep reinforcement learning for the distributed control of nextg wireless networks," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2021, pp. 248–253.

[12] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5g ultradense network," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2238–2251, 2021.

[13] S. Moon and Y. Lim, "Federated deep reinforcement learning based task offloading with power control in vehicular edge computing," *Sensors*, vol. 22, no. 24, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/24/9595

[14] X. Zhao, Y. Wu, T. Zhao, F. Wang, and M. Li, "Federated deep reinforcement learning for task offloading and resource allocation in mobile edge computing-assisted vehicular networks," *Journal of Network and Computer Applications*, vol. 229, p. 103941, 2024.

[15] 3GPP, "TS 22.125: Unmanned Aerial System (UAS) Support in 3GPP," ETSI, Sophia Antipolis, France, Tech. Rep., 2022.

[16] E. N. Gilbert, "Capacity of a Burst-noise Channel," *Bell system technical journal*, vol. 39, no. 5, pp. 1253–1265, 1960.

[17] E. O. Elliott, "Estimates of Error Rates for Codes on Burst-noise Channels," *The Bell System Technical Journal*, vol. 42, no. 5, pp. 1977–1997, 1963.

[18] Q. Zhang and S. A. Kassam, "Finite-state Markov Model for Rayleigh Fading Channels," *IEEE Transactions on communications*, vol. 47, no. 11, pp. 1688–1692, 1999.