ATLAS: Adaptive Task Learning Allocation System for UAV Task Offloading in Edge Computing

Taewon Song and Taeyoon Kim

***Abstract*—Multi-UAV task offloading optimization in edge computing environments represents a critical challenge for enabling efficient distributed computation, as UAV swarms must make coordinated decisions under dynamic network conditions, limited computational resources, and stringent latency constraints. This paper presents ATLAS (Adaptive Task Learning Allocation System), a universal framework that employs shared global model training through Asynchronous Advantage Actor-Critic algorithms to optimize task offloading decisions across distributed UAV networks. The methodology formulates the problem as a multi-agent Markov Decision Process where several UAV workers operating in heterogeneous environments contribute learning experiences to a centralized global model, enabling coordinated decision-making for local processing, MEC server offloading, or task discard actions. Experimental evaluation demonstrates that the shared global model achieves superior reward distribution and convergence characteristics compared to individual worker training, with statistical analysis revealing significant performance improvements across diverse operational scenarios. Cross-environment generalization experiments show enhanced knowledge transfer capabilities, where the global model maintains robust performance when deployed in previously unseen environmental configurations. These findings provide essential insights for designing scalable multi-UAV systems and demonstrate the effectiveness of centralized learning approaches for distributed edge computing applications.**

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

I. Introduction

The proliferation of Unmanned Aerial Vehicles (UAVs) in various applications has created new opportunities for distributed computing and task offloading in edge computing environments. As UAV swarms become increasingly sophisticated, the challenge of optimizing computational task distribution among multiple agents becomes critical for system performance and energy efficiency.

Modern UAV systems are required to process computationally intensive tasks such as real-time image processing, path planning, and sensor data analysis while operating under strict energy and latency constraints. The limited computational resources onboard individual UAVs necessitate intelligent task offloading strategies that can dynamically distribute workloads across the network, including mobile edge computing (MEC) servers and cloud infrastructure.

Reinforcement Learning (RL), particularly deep reinforcement learning approaches, has emerged as a promising solution for addressing the complex decision-making challenges in multi-UAV systems. Among various RL algorithms, Asynchronous Advantage Actor-Critic (A3C) has shown significant potential due to its ability to handle continuous action spaces and its inherent parallelization capabilities. The asynchronous nature of A3C allows multiple agents to learn simultaneously, making it particularly suitable for distributed UAV scenarios where coordination and communication constraints exist.

However, a fundamental question remains regarding the optimal training strategy for A3C in multi-UAV environments: should the system employ a centralized approach with a shared global model, or would decentralized training with individual worker specialization yield superior performance? This question becomes even more critical when considering the heterogeneous nature of operational environments that UAV systems encounter.

This paper investigates two fundamental training strategies for A3C-based multi-UAV task offloading: (1) centralized training using a global model shared across all workers, and (2) decentralized training where individual workers develop specialized policies. Through comprehensive experimental evaluation across multiple environmental configurations, we analyze the performance characteristics, convergence properties, and generalization capabilities of each approach.

To sum up, the contributions of this paper are fourfold:

- We present a comprehensive performance comparison between A3C global and individual training strategies for multi-UAV task offloading optimization, providing the first systematic evaluation of these training paradigms in UAV-assisted edge computing environments.
- We conduct rigorous statistical analysis of reward distributions and convergence characteristics across multiple heterogeneous environments, demonstrating significant performance differences between training approaches.
- We develop an evaluation framework for cross-environment generalization in multi-UAV systems, enabling assessment of training strategy effectiveness across diverse operational scenarios.
- We provide practical insights into optimal training paradigms for different operational scenarios, offering guidelines for UAV system deployment in varying environmental conditions.

The remainder of this paper is organized as follows. Section II offers a comprehensive review of relevant studies in UAV task offloading and multi-agent reinforcement learning. We describe the system model and problem formulation in

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 ...

Section III. The experimental setup and implementation details are presented in Section IV. Performance evaluation results are given in Section V, followed by discussion in Section VI and conclusion in Section VII.

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: deep reinforcement learning approaches for UAV task offloading, multi-agent coordination strategies, and federated learning frameworks.

A. Deep Reinforcement Learning for UAV Task Offloading

Recent advances in deep reinforcement learning have shown significant promise for addressing the complex decision-making challenges in UAV-assisted mobile edge computing environments. Traditional optimization approaches often struggle with the dynamic and uncertain nature of wireless channels, computational demands, and energy constraints.

Song [1] addresses the challenge of hidden channel conditions in UAV-assisted MEC systems by proposing PORTO-MEC, a Deep Recurrent Q-Network (DRQN) based algorithm. The work formulates the problem as a Partially Observable Markov Decision Process (POMDP), where UAVs must make task offloading decisions without complete information about channel states between UAVs and cloud servers. The DRQN approach leverages temporal dependencies in the environment to make informed decisions under uncertainty, achieving 58.82% higher rewards compared to traditional DQN approaches, 92.39% improvement over local-only processing, and 213.51% improvement over offload-only strategies. However, this work focuses on single-agent scenarios and does not address multi-UAV coordination challenges.

Wang et al. [2] and Liu et al. [3] have explored various deep reinforcement learning approaches for UAV-enabled wireless communications and collaborative task offloading. These works demonstrate the effectiveness of DRL methods in handling dynamic environments but primarily focus on individual UAV optimization or simplified multi-UAV scenarios without systematic comparison of training strategies.

B. Multi-Agent Deep Reinforcement Learning Approaches

The complexity of multi-UAV systems necessitates sophisticated coordination mechanisms that can handle the distributed nature of the problem while ensuring efficient resource utilization and performance optimization.

Zhao et al. [4] propose a comprehensive multi-agent deep reinforcement learning framework for task offloading in UAV-assisted mobile edge computing systems. Their approach, named MATD3 (Multi-Agent Twin Delayed Deep Deterministic Policy Gradient), addresses the joint optimization of UAV trajectories, computation task allocation, and communication resource management. The framework formulates the problem as a multi-agent Markov Decision Process and employs a cooperative MADRL approach to handle high-dimensional continuous action spaces. The system demonstrates superior performance compared to traditional optimization approaches, particularly in terms of adaptability to UEs' mobility, robustness to resource changes, and flexibility to dynamic computation demands.

The MATD3 framework adopts a centralized training with decentralized execution strategy, where evaluation critic networks obtain global views during training while individual UAVs execute policies based on local observations during deployment. This approach achieves significant improvements in total system cost reduction while maintaining scalability across different numbers of UAVs and user equipment. However, the work does not systematically compare different training paradigms or investigate the trade-offs between centralized and fully decentralized learning approaches.

Foerster et al. [5] and Gupta et al. [6] have established foundational work in cooperative multi-agent reinforcement learning, providing theoretical frameworks that have influenced subsequent UAV coordination research. However, these approaches have not been specifically evaluated for UAV task offloading scenarios or compared across different training strategies.

C. Federated Learning and Decentralized Training

The challenge of data privacy, communication overhead, and training efficiency in distributed systems has led to the exploration of federated learning approaches in edge computing environments.

Wang et al. [7] introduce FADE (Federated Deep reinforcement learning-based cooperative edge caching), a framework that enables base stations to cooperatively learn shared predictive models while keeping training data localized on individual IoT devices. The approach uses Double Deep Q-Network (DDQN) as the underlying reinforcement learning algorithm and demonstrates 92% loss reduction in the first 100 training steps compared to centralized approaches, along with 60% improvement in system payment efficiency. The framework addresses privacy concerns and communication overhead by sharing only model parameters rather than raw data.

While FADE focuses on edge caching rather than computation task offloading, it provides valuable insights into the benefits and challenges of federated learning in distributed edge computing scenarios. The work demonstrates that decentralized training can achieve comparable or superior performance to centralized approaches while addressing practical deployment constraints.

D. Research Gaps and Motivation

Despite the significant progress in UAV-assisted mobile edge computing and deep reinforcement learning, several critical research gaps remain:

Training Strategy Comparison: Existing work has not systematically compared global versus individual training strategies for A3C in multi-UAV environments. While Zhao et al. [4] demonstrate the effectiveness of centralized training with decentralized execution, and Wang et al. [7] show benefits of federated learning for edge caching, there lacks a comprehensive analysis of pure centralized versus fully decentralized training paradigms specifically for task offloading optimization.

A3C Algorithm Exploration: Limited research has explored the potential of Asynchronous Advantage Actor-Critic (A3C) algorithms for multi-UAV task offloading, despite A3C's inherent advantages in parallel learning and policy gradient methods. Most existing work focuses on DQN variants [1] or actor-critic methods like TD3 [4], leaving A3C's performance characteristics unexplored in this domain.

Cross-Environment Generalization: Current approaches primarily evaluate performance within specific environmental configurations without systematic analysis of generalization capabilities across diverse operational scenarios. This limits the practical applicability of trained models in real-world deployments where UAVs encounter varying environmental conditions.

Statistical Rigor: Many existing studies lack comprehensive statistical analysis of performance differences between training approaches, making it difficult to draw definitive conclusions about optimal strategies for different scenarios.

This work addresses these gaps by providing a systematic comparison of A3C global versus individual training strategies, comprehensive statistical analysis across multiple environmental configurations, and evaluation of cross-environment generalization capabilities in multi-UAV task offloading scenarios.

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. Each UAV agent observes the current system state $s_t \in \mathcal{S}$, 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

The state space $\mathcal{S}$ in the ATLAS framework captures comprehensive information about the UAV's computational environment, task queue status, and contextual factors that influence offloading decisions. We define the state space as a Cartesian product of four fundamental components:

$$\mathcal{S} = \mathcal{C} \times \mathcal{Q} \times \mathcal{E} \times \mathcal{V} \tag{2}$$

where each component represents distinct aspects of the system state:

Computational State ($\mathcal{C}$): This component characterizes the UAV's local computational resources and task processing status:

$$\mathcal{C} = \{c_{avail}, c_{epochs}\} \tag{3}$$

where $c_{avail} \in [0,1]$ represents the normalized available computation units on the UAV, and $c_{epochs} \in [0,1]$ denotes the remaining normalized epochs for task completion.

Queue State ($\mathcal{Q}$): This captures the current status of the UAV's task buffer and processing queue:

$$\mathcal{Q} = \{q_{units}, q_{times}, q_{success}\} \tag{4}$$

where $q_{units} \in [0,1]$ represents the normalized queue computation requirements, $q_{times} \in [0,1]$ indicates normalized processing times, and $q_{success} \in \{0,1\}^2$ provides binary success indicators for local and offload operations.

Environment State ($\mathcal{E}$): This component models the external MEC infrastructure and network conditions:

$$\mathcal{E} = \{e_{mec\_units}, e_{mec\_times}\} \tag{5}$$

where $e_{mec\_units} \in [0,1]$ represents normalized available MEC server computation units, and $e_{mec\_times} \in [0,1]$ denotes normalized MEC processing times.

Vehicle Context ($\mathcal{V}$): This encompasses UAV-specific contextual information:

$$\mathcal{V} = \{v_{velocity}, v_{capacity}\} \tag{6}$$

where $v_{velocity} \in [0,1]$ represents normalized agent velocity, and $v_{capacity} \in [0,1]$ indicates normalized computation capacity.

The complete state vector at time t is represented as:

$$s_t = (c^{(t)}_{avail}, c^{(t)}_{epochs}, q^{(t)}_{units}, q^{(t)}_{times}, q^{(t)}_{success}, e^{(t)}_{mec\_units}, e^{(t)}_{mec\_times}, v^{(t)}_{veloci} \tag{7}$$

All state components are normalized to $[0,1]$ to ensure stable neural network training and enable effective knowledge transfer across different environmental configurations.

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:

$$\mathcal{A} = \{a_{LOCAL}, a_{OFFLOAD}, a_{DISCARD}\} = \{0, 1, 2\} \tag{8}$$

Each action corresponds to a distinct task processing strategy with specific operational characteristics:

Local Processing ($a_{LOCAL} = 0$)**:** When this action is selected, the UAV processes the computational task using its onboard computing resources. This action is optimal when:

- Local computational capacity is sufficient for the task requirements
- Network connectivity to MEC servers is poor or unavailable
- Energy consumption for local processing is acceptable
- Processing latency requirements can be met locally

The local processing action triggers immediate computation on the UAV, consuming local energy resources but avoiding transmission delays and potential network failures.

Offloading ($a_{OFFLOAD} = 1$)**:** This action involves transmitting the computational task to available MEC servers for remote processing. Offloading is advantageous when:

- Local computational resources are insufficient or overloaded
- MEC servers have available capacity and favorable processing times
- Network conditions support reliable task transmission
- Total offloading latency (transmission + processing + result retrieval) is acceptable

The offloading action incurs transmission energy costs and introduces network-dependent delays, but potentially reduces local computational load and enables processing of resource-intensive tasks.

Task Discard ($a_{DISCARD} = 2$)**:** When system conditions are unfavorable for both local processing and offloading, the agent may choose to discard the task. This action is selected when:

- Local resources are critically low
- MEC servers are overloaded or unreachable
- Task deadline cannot be met through any processing option
- Energy conservation is prioritized over task completion

Task discard minimizes immediate resource consumption but results in task failure penalties in the reward function.

The discrete action formulation enables straightforward policy optimization through categorical probability distributions, allowing the neural network to output action probabilities $\pi(a_t|s_t)$ for effective exploration and exploitation during training.

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. In the ATLAS framework, state transitions are governed by the dynamics of the UAV environment, task processing outcomes, and system resource evolution.

The transition probability can be expressed as:

$$\mathcal{P}(s_{t+1}|s_t, a_t) = P(s_{t+1} = (c', q', e', v')|s_t = (c, q, e, v), a_t) \tag{9}$$

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:

$$c'_{avail} = \begin{cases} \max(0, c_{avail} - \Delta c_{local}) & \text{if } a_t = a_{LOCAL} \\ c_{avail} & \text{if } a_t = a_{OFFLOAD} \\ c_{avail} & \text{if } a_t = a_{DISCARD} \end{cases} \tag{10}$$

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:

$$q'_{units} = f_{queue}(q_{units}, a_t, \text{new\_tasks}, \text{completed\_tasks}) \tag{11}$$

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:

$$e'_{mec\_units} \sim P_{MEC}(e_{mec\_units}|e_{mec\_units}, \text{system\_load}) \tag{12}$$

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:

$$v'_{velocity} = v_{velocity} + \epsilon_{velocity} \tag{13}$$

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:

$$\mathcal{R}(s_t, a_t) = -\alpha \cdot C_{energy}(s_t, a_t) - \beta \cdot C_{latency}(s_t, a_t) - \gamma \cdot C_{failure}(s_t, a_t) + \delta \tag{14}$$

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:

$$C_{energy}(s_t, a_t) = \begin{cases} E_{local} \cdot q_{units} & \text{if } a_t = a_{LOCAL} \\ E_{transmission} \cdot q_{units} & \text{if } a_t = a_{OFFLOAD} \\ 0 & \text{if } a_t = a_{DISCARD} \end{cases} \tag{15}$$

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:

$$C_{latency}(s_t, a_t) = \begin{cases} T_{local}(q_{units}, c_{avail}) & \text{if } a_t = a_{LOCAL} \\ T_{offload}(q_{units}, e_{mec\_times}) + T_{transmission} & \text{if } a_t = a_{OFFLOAD} \\ 0 & \text{if } a_t = a_{DISCARD} \end{cases} \tag{16}$$

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:

$$C_{failure}(s_t, a_t) = \begin{cases} P_{fail\_local} \cdot \rho_{failure} & \text{if } a_t = a_{LOCAL} \\ P_{fail\_offload} \cdot \rho_{failure} & \text{if } a_t = a_{OFFLOAD} \\ \rho_{discard} & \text{if } a_t = a_{DISCARD} \end{cases} \tag{17}$$

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:

$$B_{completion}(s_t, a_t) = \begin{cases} (1 - P_{fail\_local}) \cdot \eta_{success} & \text{if } a_t = a_{LOCAL} \\ (1 - P_{fail\_offload}) \cdot \eta_{success} & \text{if } a_t = a_{OFFLOAD} \\ 0 & \text{if } a_t = a_{DISCARD} \end{cases} \tag{18}$$

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:

$$L = L_{policy} + \beta \cdot L_{value} - \alpha \cdot H(\pi) \tag{19}$$

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

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] T. Song, "Drqn-based task offloading in uav-assisted mobile edge computing environments with hidden channel conditions," in *2024 International*

Conference on Information and Communication Technology Convergence (ICTC). IEEE, 2024, pp. 881–884.

[2] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and A. Nallanathan, “Deep reinforcement learning for uav-enabled wireless communications with edge computing,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 1, pp. 131–141, 2020.

[3] J. Liu, Q. Zhang, Z. Chen, and Q. Ni, “Multi-uav collaborative task offloading and resource allocation in edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12 394–12 405, 2020.

[4] 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.

[5] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[6] J. K. Gupta, M. Egorov, and M. Kochenderfer, “Cooperative multi-agent control using deep reinforcement learning,” *International conference on autonomous agents and multiagent systems*, pp. 66–83, 2017.

[7] 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.