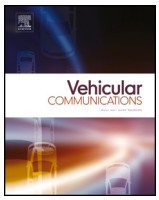
Vehicular Communications 50 (2024) 100858

Contents lists available at ScienceDirect

Vehicular Communications

journal homepage: www.elsevier.com/locate/vehcom



Decentralized multi-hop data processing in UAV networks using MARL

Indu Chandran *, Kizheppatt Vipin

Department of Electrical and Electronics engineering, BITS Pilani, KK Birla Goa Campus, Zuari Nagar, Goa, India

A B S T R A C T

Unmanned Aerial Vehicles (UAVs) have become integral to numerous applications, prompting research towards enhancing their capabilities. For time-critical missions, minimizing latency is crucial; however, current studies often rely on sending data to ground station or cloud for processing due to their limited onboard capacities. To leverage the networking capabilities of UAVs, recent research focuses on enabling data processing and offloading within the UAV network for coordinated decision-making. This paper explores a multi-hop data offloading scheme designed to optimize the task processing and resource management of UAVs. The proposed distributed strategy uses multi-agent reinforcement learning, where UAVs, each with varying computational capacities and energy levels, process and offload tasks while managing energy consumption and latency. The agents, represented as actor-critic models, learn and adapt their actions based on current state and environment feedback. The study considers a consensus-based method to update learning weights, promoting cooperative behavior among the agents with minimum interaction. Through multiple training episodes, the agents improve their performance, with the overall system achieving faster convergence with high rewards, demonstrating the viability of decentralized data processing and offloading in UAV networks.

1. Introduction

Unmanned aerial vehicles have been explored across different industrial sectors including agriculture, disaster relief missions, product delivery and many more [1]. This widespread popularity is mainly attributed to its ease of flight, quick formations, coordinated mission execution, and low cost of deployment [2]. Recent advancements in UAV design have enabled these vehicles to carry higher payloads, including cameras and advanced sensors, facilitating faster and more efficient data collection. Additionally, developments in embedded technology have equipped UAVs with enhanced processing and computational capabilities, making them particularly valuable for time-critical missions such as disaster monitoring and surveillance, where reducing latency in relief efforts is crucial. In such scenarios, UAV nodes that frequently detect and process events may deplete their energy more quickly than their peers. Implementing data offloading schemes in these cases can significantly enhance overall network performance. This approach is particularly relevant in disaster scenarios such as wildfire monitoring, earthquake search and rescue, or flood damage assessment, where communication with the ground station may be unstable. For instance, during an earthquake, the ground dynamics can severely disrupt communication, making on-board processing and offloading a more reliable option to reduce operational latency.

Data offloading decisions are generally modeled as mixed integer problems or as Markov approximations, however, these methods rely on heuristics, and so require considerably higher number of iterations to reach the optimal value [3–5]. Thus, the above methods may not be suitable for real-time decision making where there are frequent changes involved in the working environment. In order to make learning-based decisions, Reinforcement Learning (RL) has evolved as a potential solution overcoming the drawbacks of the conventional heuristic schemes. Although considerable amount of research has been stated in literature, the RL based schemes have been explored widely in meeting the growing computational demands of ground users [14,16]. The studies that focus on offloading within the network during a surveillance have been under explored, while those few have considered agent decisions as independent entities, where the decision of one agent does not rely on the decisions taken by another agent. These studies thus do not address the possibility of collaborative learning and decision making within a UAV network.

In this paper, we propose the use of multi-hop data offloading using Multi-Agent Reinforcement Learning (MARL) to decide the fraction of data that an agent will process locally, and the remaining fraction that is offloaded, while minimizing latency and task completion time. The UAVs are treated as agents and are assumed to fly in formation to accomplish a surveillance mission. The study considers a decentralized actor-critic (AC) framework, and incorporates a consensus update mechanism, significantly enhancing co-operative behavior among UAV agents. Each agent updates its policy and value function by incorporating feedback from other agents, thereby aligning their learning objec-

* Corresponding author.
E-mail addresses: p20200055@goa.bits-pilani.ac.in (I. Chandran), kizheppattv@goa.bits-pilani.ac.in (K. Vipin).

https://doi.org/10.1016/j.vehcom.2024.100858
Received 27 June 2024; Received in revised form 23 October 2024; Accepted 11 November 2024
Available online 15 November 2024

Table 1
A comparison of state-of-the art approaches.

Reference	Architecture	Algorithm	Multi-agent	Consensus-based	Multi-hop	Application	Optimization parameter
[6]	Distributed	Q-Learning	Yes	No	No	IIoT	Computation cost, energy, latency
[7]	Centralized	DNN	No	No	No	MEC	Computation rate, energy
[8]	Decentralized	Q-Learning	No	No	No	IoT	Computation cost, energy, latency
[9]	Decentralized	DNN	Yes	No	No	MEC	Energy, latency
[10]	Distributed	DNN	Yes	Yes	No	MEC	Energy, latency
[11]	Decentralized	Policy gradient	Yes	No	No	MIMO	Energy, delay
[12]	Distributed	Policy gradient	Yes	No	No	MEC	Time, load balancing
[13]	Centralized	DNN	No	No	No	MEC	Computation cost, energy, latency
[14]	Decentralized	AC	Yes	Yes	No	UAV-MEC	Computation cost, energy, latency
[15]	Distributed	DQN	No	No	Yes	UAV-MEC	Computation cost, energy, latency
Proposed	Decentralized	AC	Yes	Yes	Yes	UAV-UAV	Computation cost, energy, latency

tives. The actor update is performed by each agent independently, while the critic update follows a consensus-based learning where the consensus parameters are updated based on the learning progress of each agent. The feedback from the critic is then used in the subsequent step for the actor. Thus, by incorporating information from other agents, each agent can make more informed updates to its policy and value functions. During training, agents decide actions based on current states, perform these actions in the environment, and update their states while dynamically adjusting the consensus weights. This iterative process fosters a cooperative learning strategy, crucial for optimizing overall energy consumption and latency in multi-UAV systems.

2. Background

Several reinforcement learning approaches have been proposed to optimize computational task offloading in different environments. A reinforcement learning approach is proposed in [6] to optimize the offloading of computational tasks in Industrial Internet of Things environments. The authors addressed the challenges of resource allocation and task offloading by formulating these problems as a sum cost delay framework and using Q-learning to derive optimal offloading decisions. The authors of [17] addressed the challenge of communication efficiency among agents that collaborate to learn policy values based on private local rewards and joint state-action observations. In [18], off-policy reinforcement learning is extended to multi-agent settings, introducing a multi-agent version of emphatic temporal difference for policy evaluation. Despite proving convergence under linear function approximation, the adaptability of this method in non-linear scenarios remains untested. A comprehensive review of theories and algorithms in multi-agent reinforcement learning is provided in [19], with a focus on the theoretical foundations of MARL, reviewing algorithms within the frameworks of Markov/stochastic games and extensive-form games. Reference [20] discussed a binary offloading policy for mobile edge computing (MEC) servers, presenting an efficient order-preserving policy generation method. However, it does not address the potential latency issues in high-density user scenarios.

In [21], a self-learning strategy for task offloading in UAV networks is proposed, aiming to optimize edge computing resources. Although it reduces task completion time and energy consumption, the reliance on dynamic predictor selection can introduce computational overhead. The authors of [5] proposed a solution to low computing capability in low-power wireless networks using radio frequency-based wireless power transfer and edge computing technologies. The approach, focusing on binary computation offloading policy, may face challenges in energy harvesting efficiency. A deep reinforcement learning-based online offloading framework is proposed in [7] to optimize task offloading and resource allocation in wireless powered mobile-edge computing networks. The framework uses a deep neural network to learn offloading decisions from past experiences, significantly reducing computational complexity. Recent advancements in decentralized MARL schemes are reviewed in [8], covering various algorithmic frameworks. In [22], an RL-based offloading scheme using energy harvesting is discussed, allowing IoT devices to optimize offloading decisions without requiring MEC model knowledge. A fully decentralized actor-critic algorithm relying on neighbor-to-neighbor communication is analyzed in [23], aiming to maximize the network-wide averaged long-term return. This approach, though innovative, might suffer from communication delays in dense networks. Reference [24] proposed a value propagation method combining softmax temporal consistency and primal-dual decentralized optimization. This approach enables agents to learn collaboratively by exchanging information only with their neighbors, preserving privacy and security. An extension of actor-critic methods is discussed in [25], that incorporates centralized training with decentralized execution, allowing the critic to access additional information about other agent policies while the actor operates with local information. This approach facilitates learning policies that require complex coordination among agents, however, the dependency on centralized training data can be a limitation.

Reference [9] explored a deep reinforcement learning framework for optimizing offloading decisions and resource allocation in mobile-edge computing systems by mapping the input states, such as wireless channel gains and edge CPU frequency, to binary offloading decisions. A distributed reinforcement learning in multi-agent systems is proposed in [26], emphasizing the importance of collaborative learning and decentralized decision-making. The approach combines consensus dynamics with local innovations, enabling agents to update their value functions based on both local observations and information received from neighboring agents. Reference [10] addressed joint offloading decisions and bandwidth allocation in MEC networks, proposing a distributed deep learning-based offloading algorithm. While generating efficient offloading decisions, it may require significant computational resources. In [11], a decentralized dynamic computation offloading strategy for multi-user MEC systems is proposed, aiming to minimize long-term average computation costs. The study considered deep deterministic policy gradient (DDPG) to enable each user to independently learn efficient offloading policies from local observations. A multi-agent load balancing distribution algorithm based on deep reinforcement learning is introduced in [12] to address task offloading and resource allocation challenges in edge computing. The algorithm, while enhancing system robustness and scalability, may face issues with real-time adaptability.

A fully decentralized MARL framework is discussed in [27], proposing two decentralized actor-critic algorithms that use local information for the actor step and achieve consensus on value function estimates in the critic step. A distributed architecture leveraging MARL is proposed in [14] to dynamically offload tasks from UAVs to the edge cloud. The approach is based on actor-critic framework where each agent performs the actor step individually while sharing value function estimates with neighbors during the critic step, aiming to minimize overall latency and energy usage.

Table 1 outlines the relevant studies on data offloading schemes in UAVs. Notably, no studies closely relate to offloading data to neighbors in a multi-hop fashion, and they often treat tasks as independent entities, simplifying the analysis in coordinated environments. Although a

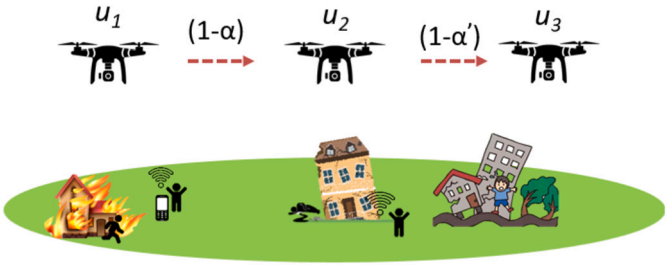


Fig. 1. Multi-hop data offloading scenario in a disaster environment, where α denotes the fraction of data processed locally, and $(1-\alpha)$ is the fraction that is offloaded.

multi-hop offloading scenario is discussed in [15], it considers a centralized learning approach. The lack of focus on decentralized, multi-hop offloading strategies highlights a critical gap in the literature, particularly for applications in disaster response, where real-time coordination and resilience against network disruptions are paramount.

3. System model

In our study, the agents are the UAVs collaboratively operating in a disaster environment. The UAV network consisting of N UAVs distributed in an area, tasked with performing a coverage mission to identify trivial events such as wild fires or landslides. During disaster missions, a lot of data is generated in the form of images and videos. Sending it to the ground control station for processing in a harsh and dynamic environment may not be feasible as the network may experience frequent link disconnections. In such scenarios, data offloading can be viable solution, where each UAV can process a portion of the data and offload the remaining in a multi-hop fashion to its neighboring UAVs. Fig. 1 depicts an example environment with α, α' representing the fraction of data locally processed by UAVs u_1 and u_2 respectively. An important question to be answered here is on deciding the fraction of data that a UAV can process locally based on its available energy and computational resources, and the remaining fraction that it offloads to its peers without burdening the network.

A task generated by a UAV can be defined in terms of its size and the number of clock cycles to process 1 bit of information, and is represented in the form of a tuple (s, c) where s denotes the task size and c denotes the clock cycles. The duration for which a UAV processes the fraction of data is considered as a virtual time slot. Based on the amount of data to be processed and the capabilities, the duration of the time slots may vary. The study aims at computing the best processing and offloading decision that minimizes the overall latency in processing a task and the energy consumption of the nodes involved in data offloading.

Considering a disaster mission, the number of UAVs may also vary over time due to failures, energy exhaust or loss of communication. However, for simplicity and to avoid ambiguity, the number of UAVs is always represented as N. Moreover, UAVs are assumed to fly in formation for the ease of control in such critical missions. Formations such as linear or V-shape are generally preferred [28]. In these formations, the neighbor UAVs are more or less fixed as the nodes share data on their positions to remain in formation. Without losing generality, the route for task offloading is taken as $u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow \cdots \rightarrow u_n$, where n is the last UAV in the formation. A UAV u_i processes a fraction α of the data that it receives from UAV u_{i-1}, and offloads $(1-\alpha)$ of the data to the next UAV. Therefore the sub-tasks are dependent. Further, it is assumed that the total task size generated by any UAV does not exceed the overall processing capacity of the network. After the task is processed, each UAV sends the information back to the UAV that generated the task along the same route. Therefore, time taken to send back the information also adds to overall latency. This often depends on the time taken by the last UAV to send back the information as the previous UAVs start processing the tasks at an early stage. However, since it is a multi-hop processing, the amount of data that the last UAV processes will be comparatively smaller considering that only the remaining fraction is forwarded. Thus, the latency to send back the information can be ignored as is commonly assumed in similar studies [14,15].

3.1. Mathematical formulation of UAV network dynamics

Considering the dynamic and harsh operating environment for the UAVs, variations in the channel parameters can affect the offloading choice of each UAV.

3.1.1. Channel model

Since UAVs are operating in free-space at a considerable altitude from the ground level, it can be assumed that Line-of-Sight (LoS) path exists between UAVs. The channel gain $h_g(i,j)$ between two UAVs u_i and u_j can be represented as given in (1) and is influenced by various factors such as distance, path loss and fading.

$$h_{i,j}(t) = \left(\frac{4\pi f}{c}\right)^{-2} \mu_{\text{LoS}} d_{i,j}^{-\beta_{\text{LoS}}(t)} |h_{i,j}^{\text{Rice}}(t)|^2 \tag{1}$$

Here, $h_{i,j}^{\text{Rice}}(t) \sim \text{Rice}(v, \delta)$; $\text{Rice}(v, \delta)$ refers to the Ricean distribution with parameters v and δ; $d_{i,j}(t)$ is the Euclidean distance between UAVs u_i and u_j and is calculated as, $d_{i,j}(t) = \|\mathbf{r}_i(t) - \mathbf{r}_j(t)\|$, where $\mathbf{r}_i(t) = (x_i(t), y_i(t), z_i(t))$ and $\mathbf{r}_j(t) = (x_j(t), y_j(t), z_j(t))$ are the position coordinates of UAV i and j respectively. The term $(\frac{4\pi f}{c})^{-2}$ represents the scaling factor for signal attenuation due to path loss in free-space propagation, where f is the frequency of the transmitted signal and c is the speed of light. μ_{LoS} is the attenuation factor, and $\beta_{\text{LoS}}(t)$ is the path loss exponent for LoS that varies with distance d.

3.1.2. Transmission rate

The transmission rate of a UAV u_i to offload data to its neighbor u_{i+1} during a virtual time slot t is represented as (2), and is a factor of transmission power and bandwidth.

$$T^r_{u_i,u_{i+1}} = B \log_2 \left(1 + \frac{P_{u_i}(t)}{P_{\text{n}}} h_{u_i,u_{i+1}}\right) \tag{2}$$

Here, B represents the bandwidth allocated to each node, while P_n denotes the noise power. The term $h_{u_i,u_{i+1}}$ refers to the channel gain between node u_i and node u_{i+1}; $P_{u_i}(t)$ is the transmission power of u_i, and $P^{max}_{u_i}$ is its maximum value.

3.1.3. Energy consumption

Each UAV has its own energy and computational capability. A UAV decides to process α fraction of the task it receives and offloads $(1-\alpha)$. Therefore, the total energy consumption is the sum of the energy consumed to process the data fraction and the energy consumed to offload the remaining data to its neighboring UAV.

The energy consumed by UAV u_i to process the data locally at time slot t can be formulated as (3).

$$E^i_{comp}(t) = sc\alpha_t \prod_{t'=0}^{t-1} (1-\alpha_{t'}) E_{u_i} \tag{3}$$

where E_{u_i} represents the CPU energy consumption needed to execute one cycle at node u_i, and $E_{u_i^{max}}$ is its maximum value; α is the fraction of task processed locally, and $(1-\alpha')$ refers to the fractions of the task that were offloaded in previous time slots up to $t-1$; s and c are the task size and the clock cycles required to process 1 bit, respectively. The product term aggregates the energy consumed across all previous time slots $t' = 0$ to $t' = t-1$ since the remaining task size to be processed locally at each time slot depends on what was offloaded in previous slots.

The energy required to offload the data to the next UAV at time slot t is computed as (4).

$$E^i_{off}(t) = s(1-\alpha_t)\prod_{t'=0}^{t-1}(1-\alpha_{t'})T^r_{u_i,u_{i+1}}P_{u_i}(t) \tag{4}$$

where the transmit power $P_{u_i}(t)$ of node u_i remains below its maximum allowed value $P^{max}_{u_i}$; $(1-\alpha)$ is the fraction of task offloaded in time slot t, and $(1-\alpha')$ refers to the fractions of the task that were offloaded in previous time slots up to $t-1$; $T^r{}_{u_i,u_{i+1}}$ is the transmission rate.

3.1.4. Latency

The duration D of a time slot t depends on the time taken by a UAV to process the data locally and the time taken for offloading. The processing time can be represented as (5).

$$T^i_{comp}(t) = \frac{sc\alpha_t\prod_{t'=0}^{t-1}(1-\alpha_{t'})}{R_{u_i}(t)} \tag{5}$$

where $R_{u_i}(t)$ represents the available computation resource for node u_i at t, measured in cycles per second, and is less than the computation capacity limit $R^{max}_{u_i}$ of the UAV.

The offloading time can be represented as (6), where the numerator represents the total amount of data to be transmitted, and the denominator is the transmission rate as defined in (2).

$$T^i_{off}(t) = \frac{s(1-\alpha_t)\prod_{t'=0}^{t-1}(1-\alpha_{t'})}{T^r_{u_i,u_{i+1}}} \tag{6}$$

Since processing and offloading happen in parallel, the slot duration is the maximum of both the values as denoted in (7).

$$D^i_t = \max(T^i_{comp}(t), T^i_{off}(t)) \tag{7}$$

3.2. Proposed MARL algorithm formulation

In an MARL environment, each agent independently makes decisions based on its local observations and the joint observations gathered through interactions with other agents in the environment. To achieve this, all agents agree upon a global policy that aligns with their collective goals while simultaneously adhering to their individual local policies. The decision of each agent is influenced by a reward value that indicates the effectiveness of their actions, and operates with the goal of maximizing these rewards. In this study, we apply MARL framework using the Actor-Critic (AC) method to find the most efficient offloading strategy for each agent in the network. The actor represents the policy $\pi(a|s)$, which selects actions based on the current state s and the critic evaluates the actions taken by the actor by estimating value functions. It provides feedback by assessing how well the actions are, and guides the policy towards optimal actions. The decision-making process is influenced by the current state of the UAV, which includes the remaining task size, energy levels, and computational capacity.

The optimization problem of task offloading is defined as $< S, A, P, R >$, where S, A, P, R are the elements of RL and is referred to as state space, action space, transition probability, and reward function respectively. The state space S at time slot t defines the state of the environment, and in the study, it consists of the following elements:

- *Remaining Task Size (s)*: The amount of data (in bits) the UAV still needs to process or offload.
- *Energy Level (E)*: The remaining energy of the UAV at time t, measured in joules or percentage.
- *Computation Capacity (C)*: The number of CPU cycles per second available for task processing.

The action space A consists of discrete set of actions that the UAVs take at slot t and can be represented as (8) where a value 1 indicates that the data received is completely offloaded to the next UAV, and 0 if the UAV can process the entire data and no fraction is offloaded. While the state set S and action set A are shared among the UAVs, the actions a_u taken by each UAV, and reward r_u it receives remains local to the UAV, thus, making the system decentralized in nature.

$$A = \left\{\alpha_i \in (0,1) \cup \left\{0, \frac{1}{k}, \frac{1}{2k}, \ldots, \frac{k-1}{k}, 1\right\}\right\}^n_{i=1} \tag{8}$$

The utility value (W_i) that represents how good it is to remain in a state for a given action, is represented as (9), where $E^i_{comp}(t)$ and $E^i_{off}(t)$ defined in (3) and (4), represents the energy costs, D^i_t denotes the latency cost, and $p_1, p_2 \in [0,1]$ are the associated weights.

All the UAVs collaborate to find the optimal global policy π_{θ_g} that is to maximize the average long-term return across all UAVs, using the locally available information. In the study, rewards are assigned according to how well the offloading strategy of each UAV contributes to minimizing the overall latency of task completion and the energy consumption as given in (10). This way, we aim to guide the UAVs towards policies that effectively balance task processing and offloading to achieve the best overall performance in terms of both latency and energy efficiency. The reward function is therefore calculated as (9) and the policy is denoted as (10),

$$W^i_t = p_1(E^i_{comp}(t) + E^i_{off}(t)) + p_2 D^i_t \tag{9}$$

$$f^{\pi}_{min} = \sum_{t=0}^{\infty}\gamma_t W^i_t(s_t, a_t)\Bigg| \sum(T^i_{comp} + T^i_{off}) \le T_{max} \tag{10}$$

where $\gamma_t \in [0,1]$ is the discount factor, a_t is the action taken by the policy at state s_t at time t and T_{max} is the maximum permitted task completion time that depends on the application. The overall time to process a task must not exceed T_{max}.

The gradient of long-term average $J(\theta)$, action-value Q^{θ} and advantage function $A^{\pi\theta}$ in the MARL environment can be formulated as,

$$\nabla_{\theta_u} J(\theta_g) = \mathbb{E}_{\pi_g}\left[\nabla_{\theta_u}\log\pi_{\theta_u}(a_u \mid s)A^{\pi_g}(s,a)\right] \tag{11}$$

where, $\nabla_{\theta_u} J(\theta_g)$ represents the gradient of the objective function $J(\theta_g)$ with respect to the parameters of UAV u; $A^{\pi_g}(s,a) = Q^{\pi_g}(s,a) - \hat{V}^{\pi_{\theta_u}}(s, a^{-u})$; $\hat{V}^{\pi_{\theta_u}}(s, a^{-u})$ is the baseline value and gives the average expected reward for UAV u considering the actions of all other UAVs except u. This baseline is introduced to reduce variance in the gradient estimates.

While directly knowing a^{-u} would provide the most accurate gradients, it is not always possible in practical cases. However, with global estimates of action-value and advantage functions, the necessary information can be approximated to compute gradients using the score function. But, local information alone is insufficient to calculate the global estimates, as it relies on the local rewards averaged across all the UAVs. Therefore, a learning based consensus method is proposed to allow UAVs to update their policies based on the information shared. Each UAV calculates its learning progress, which is the difference between its current and previous average return estimates (μ). This progress indicates how much the UAV has learned recently. For each UAV u, the learning progress is calculated as,

$$l_u = \mu^t_u - \mu^{t-1}_u, \quad \forall u \in U \tag{12}$$

Using this learning progress, the UAVs calculate consensus weights, which determine how much influence other UAVs have on its actions. The weight C_w between two UAVs i and $j : i, j \in U$ and is computed as,

$$C_{w_{ij}} = r_i \cdot l_i + r_j \cdot l_j, \quad \forall i, j \in U \tag{13}$$

where r_i and r_j are the learning rates for UAVs, l_i and l_j are the learning progress for UAVs i and j respectively. These weights are updated using a smoothing factor ϕ as shown in (14), where C_t is the consensus matrix representing the relationship between UAV u and all other UAVs in the environment.

$$C_t[i,j] = \phi \cdot C_{t-1}[i,j] + (1-\phi)\cdot C_{w_{ij}} \tag{14}$$

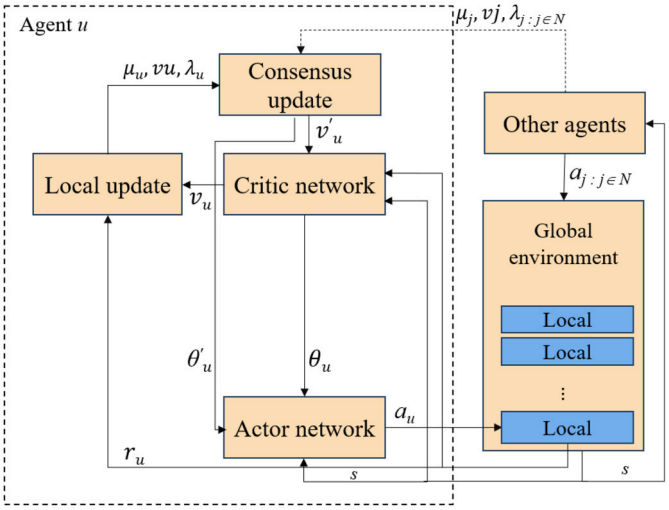


Fig. 2. MARL framework with consensus update.

The smoothing factor helps in gradually adjusting the weights rather than making abrupt updates. Finally, the consensus matrix C_t is normalized so that the weights for each UAV sum up to one, maintaining a valid probability distribution, as denoted in (15),

$$C_t[i,:] = \frac{C_t[i,:]}{\sum_j C_t[i,j]} \tag{15}$$

This normalization ensures that the influence of all UAVs in decision-making is balanced.

Once the consensus weights are calculated, each UAV updates its average return estimate μ. Instead of relying solely on its own rewards, an UAV now considers a weighted sum of rewards from all UAVs, where the weights are derived from the consensus matrix. This allows the UAV to benefit from the experiences of other UAVs, leading to a more comprehensive understanding of the environment. The updated average return estimate for each UAV is calculated as,

$$\mu^i_{t+1} = (1-\varepsilon_{v,t})\cdot\hat{\mu}^i_t + \varepsilon_{v,t}\cdot\sum_j C_t[i,j]\cdot k^j_{t+1} \tag{16}$$

where $\hat{\mu}^i_t = (1-\varepsilon_{v,t})\cdot\mu^i_t + \varepsilon_{v,t}\cdot k^i_{t+1}$. The term $(1-\varepsilon_{v,t})\cdot\mu^i_t$ ensures that the new estimate is influenced by the past estimate, providing stability and smoothing the updates over time. The first term of (16) carries forward the preliminary updated average return estimate $\hat{\mu}^i_t$ scaled by $(1-\varepsilon_{v,t})$. The second term of (16) aggregates the rewards received by all UAVs, weighted by the consensus matrix $C_t[i,j]$ and scaled by the step size $\varepsilon_{v,t}$.

Each UAV then updates its state-value function, which estimates how good a particular state is. The value function gradient update is performed as follows,

$$\hat{v}^i_t = v_t + \varepsilon_{v,t}\cdot\delta^i_t\cdot\nabla_v V_t(v^i_t) \tag{17}$$

where, $\delta^i_t = k^i_{t+1} - \mu^i_t + V_{t+1}(v^i_t)$ is the Temporal Difference (TD) error, k_{t+1} is the immediate reward, μ^i_t is the average return estimate, and v_t, $\hat{v}_t$ are the value estimates before and after the action. The value function is then updated based on the consensus weights as,

$$v^i_{t+1} = \sum_{j\in N} C_t[i,j]\cdot\hat{v}^j_t \tag{18}$$

The UAV also updates its reward function $R(s,a)$ parameterized by λ, based on its local observations. The parameter λ is adjusted iteratively to minimize the difference between these estimated rewards and the actual observed reward values. This is expressed as,

$$\hat{\lambda}^i_t = \lambda^i_t + \varepsilon_{v,t}\cdot(k^i_{t+1} - \bar{R}_t(\lambda^i_t))\cdot\nabla_\lambda\bar{R}_t(\lambda^i_t) \tag{19}$$

Algorithm 1 Decentralized MARL with Consensus Update.

Initialize parameters:
  θ_g: global policy parameter
  ω: action-value function parameter
  v: state-value function parameter
  λ: reward function parameter
  μ: average return estimate
  C_w: consensus weight
  γ: discount factor
  $\varepsilon_{v,t}, \varepsilon_{\omega,t}, \varepsilon_{\theta,t}$: step sizes
for each UAV u **do**
  Initialize local estimates $\theta_u, \omega_u, v_u, \lambda_u, \mu_u$
  Initialize consensus weights C_{w_u}
  Initialize local policy π_u
end for
for each episode **do**
  for each time slot t **do**
    for each UAV u **do**
      Observe current global state s
      Select action a_u based on local estimate of global policy $\pi_u(s;\theta_u)$
      Execute action a_u and observe reward r_u and next global state s'
      Compute TD error: $\delta_t = k_{t+1} - \mu_t + Q(s_{t+1}, a_{t+1})$
      Update local value function parameters $v_{t+1} = v_t + \varepsilon_{v,t}\cdot\delta_t\cdot\nabla_v V(s_t; v_t)$
      Compute advantage function: $A_u = Q_u(s, a_u; \omega_u) - V_u(s; v_u)$
      Update local policy parameters: v_u, ω_u, θ_u
      Compute learning progress l_u
      Compute consensus weights C_{w_u} based on learning progress
      Update local average return estimate μ_u based on C_{w_u}
      Normalize consensus matrix C_t
      Update local estimates of global parameters θ_u, v_u, λ_u based on consensus weights and information from neighboring UAVs
    end for
    Average all local estimates of global parameters to update the global parameters θ_g, v, μ
  end for
end for
return optimized global policy π_g for all UAVs

Here, $\bar{R}_t$ represents the average estimate of the reward function across all the UAVs. A consensus step ensures alignment of λ across UAVs, and is obtained as,

$$\lambda^i_{t+1} = \sum_{j\in N} C_t[i,j]\cdot\lambda^j_t \tag{20}$$

It is to be noted that these updates are performed by the critic of the UAVs. On the other hand, the actor tries to improve the policy based on the evaluations provided by the critic. This involves updating the policy parameters θ_i to improve the selection of actions. The actor uses the estimate $\bar{R}_t(\lambda^i)$ to evaluate the globally averaged TD error,

$$\hat{\delta}^i_t = \bar{R}_t(\lambda^i_t) - \mu^i_t + V_{t+1}(v^i_t) - V_t(v^i_t) \tag{21}$$

It further updates θ_i as,

$$\theta^i_{t+1} = \theta^i_t + \varepsilon_{\theta,t}\cdot\hat{\delta}^i_t\cdot\varphi^i_t \tag{22}$$

where $\varphi^i_t = \nabla_{\theta_i}\log\pi^i_{\theta_i}(s_t, a^i_t)$. The parameters μ, v, and λ are shared across the UAVs. By sharing and aligning these parameters, the UAVs can leverage the collective knowledge and experiences to make more effective and coordinated learning decisions. Fig. 2 visually represents the MARL framework with consensus updates. The detailed procedure of the proposed method is outlined in Algorithm 1.

4. Performance evaluation

In this section, we evaluate the performance of the proposed MARL solution. The evaluations are performed on Intel(R) Core(TM) i5-

Table 2
Parameter values.

Parameter	Value
Number of UAVs	5-200
Computation capacity	500-2500
Initial energy	5350 mAh
Initial task size	1000 bits
Transmission power	0.1 W
Energy consumption per CPU cycle	1×10^{-9} J/cycle
Number of CPU cycles per bit	1000 cycles per bit
Carrier frequency	2.4 GHz
Speed of light	3×10^{8} m/s
Path loss exponent	2
Power of noise	10^{-13} W
Bandwidth	10^{6} Hz
Distance between UAVs	100 meters
T_{max}	300 milliseconds
Rice factor	0.5 + 0.5j
Discount factor	0.995
Dimension of the state space	4
Number of discrete actions	11
epsilon	0.1
ϕ	0.5
p_1, p_2	0.5
Number of episodes	2000

10300H processor operating at 2.50 GHz using python *TensorFlow* libraries.

4.1. Network setup

In the study, we initialize a set of environmental parameters critical for evaluating UAV communication and computation offloading. The transmission power for each UAV is set at 0.1 W, ensuring energy-efficient operation while maintaining connectivity. The energy consumption per computation cycle is defined as 1×10^{-9} J, which aligns with modern low-power processing units. Each bit of data requires 1000 CPU cycles to process, reflecting the complexity of computational tasks in real-time scenarios. The carrier frequency is set at 2.4 GHz, a common frequency used in UAV communication systems due to its balance between range and bandwidth. The speed of light, 3×10^{8} m/s, is considered for signal propagation calculations. The path loss exponent for line-of-sight (LoS) communication is set to 2, reflecting typical free-space conditions. The noise power is set at 10^{-13} W, indicative of low-noise environments. The communication bandwidth is set to 10^{6} Hz, suitable for high-speed data transmission. An additional path loss factor of 1.5 is included, accounting for additional losses in realistic scenarios. Table 2 lists the network parameter values used in the study.

The environment is instantiated with specific parameters such as state dimension, number of agents, agent ID, hidden units in the neural network, and learning rate. Each agent in the AC model involves two neural networks: the actor network, which determines the actions, and the critic network, which evaluates these actions by estimating the value functions. Both the networks are constructed using $TensorFlow$ API.

The actor network consists of two layers: a hidden layer with 64 units, that uses the $ReLU$ activation function for introducing non-linearity, and an output layer with units equal to the number of possible actions, which in our case is 11 discrete values between 0 and 1. It uses the $softmax$ activation function to produce a probability distribution over the actions, ensuring that the sum of probabilities is 1. Similar to the actor network, the critic network has a hidden layer with the same number of units and $ReLU$ activation function. This layer processes the input state to extract relevant features for value estimation. The output layer of the critic network has a single unit, providing a scalar value that represents the estimated value of the current state. The training of both networks involves gradient descent using *TensorFlow GradientTape*. To prevent the gradients from becoming too large and destabilizing the training process, gradient clipping is applied, which restricts the gradients within a specified range. The consensus weights are initialized to a uniform distribution, ensuring that initially, each agent gives equal importance to every other agent in the system, and the step sizes $\epsilon_{\theta,t}$ and $\epsilon_{v,t}$ are set to 0.001 and 0.01 respectively to balance stability and convergence speed as considered in [14].

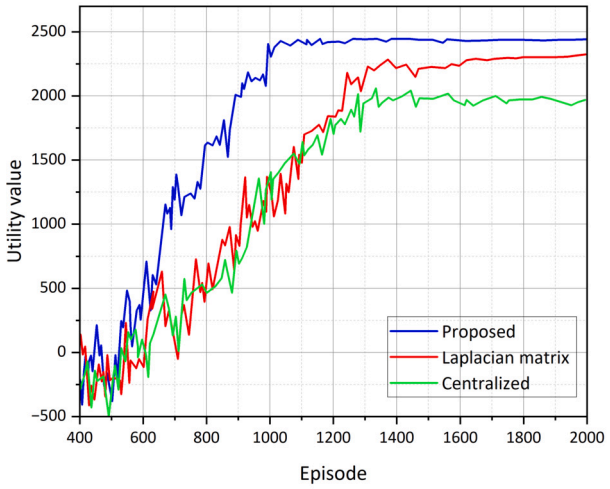


Fig. 3. Utility plot of different multi-agent algorithms for varying episodes.

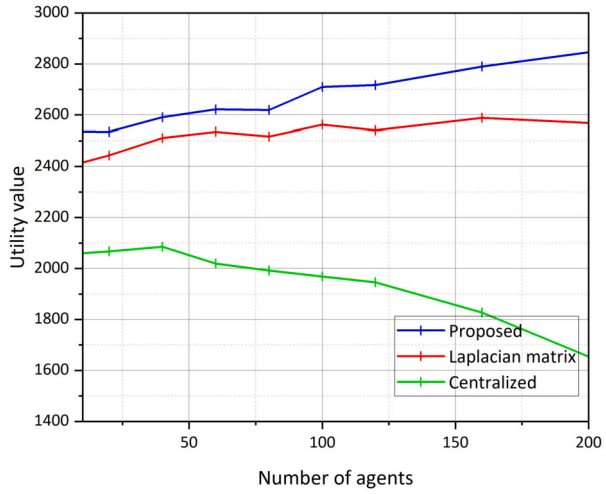


Fig. 4. Utility plot for varying number of agents.

4.2. Result analysis

In this section, we present the analytical results and compare the proposed solution with similar studies. While no directly comparable work was found in the literature on multi-hop data offloading, we have benchmarked our analysis against studies [15] and [14], which are recent and relevant to offload decision-making in UAV networks. As discussed in Section 2, [14] examines a decentralized actor-critic MARL environment, using a Laplacian matrix update for consensus weights. The weights are predetermined and do not adapt to changing conditions or the performance of individual agents. The authors employed softmax and ReLU activation functions in the neural network with 64 hidden units, similar to our study. However, they assume tasks to be independent and the data is not processed in a multi-hop manner, simplifying the formulation. In contrast, [15] investigates a centralized control actor-critic RL environment, exploring the multi-hop offloading with a detailed analysis of the concepts.

Fig. 3 illustrates the performance comparison of the proposed learning consensus-based update MARL with the state-of-the-art schemes. The plot clearly justifies the advantage of the proposed method over

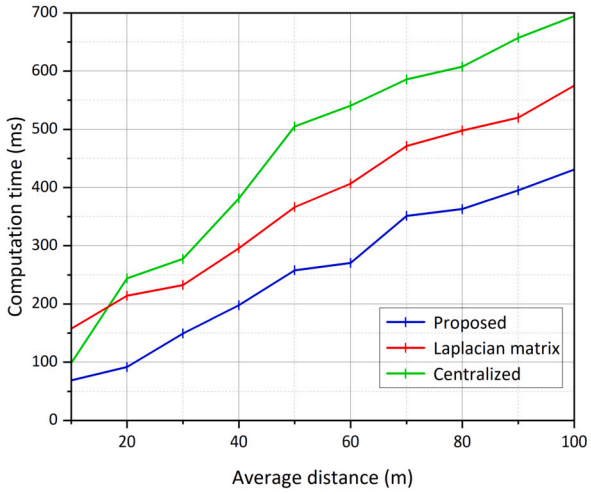


Fig. 5. Computation time for varying distance between agents.

the static laplacian matrix and centralized MARL approaches in optimizing task processing and resource management in UAV networks. The proposed learning based method achieves higher utility values and converges faster than the other methods demonstrating its effectiveness in quickly learning and adapting optimal policies. In contrast, the plot for the laplacian matrix method discussed in [14] shows a slower increase in utility value and converges at a lower level. This indicates that while the static consensus weights derived from the laplacian matrix ensure some degree of co-operation among UAVs, it lacks the adaptability required to respond quickly to dynamic changes in the environment or agent states. Faster convergence is particularly important in time-sensitive applications where rapid decision-making can significantly impact the success of the mission. On the other hand, the plot for centralized MARL approach discussed in [15] shows a much slower convergence rate, and this could be due to its dependence on the central controller to gather information and make decisions for the entire network.

A superior scalability of the proposed learning based method is evident in Fig. 4 which shows a plot of utility value with varying number of agents in the network. As the number of agents increases, the proposed method shows a steady rise in utility because each UAV can autonomously decide how to manage its energy and computational resources, thereby optimizing task completion times and reducing latency. This adaptive decision-making ensures that as more UAVs are introduced, tasks are distributed more efficiently across the network, leading to improved overall system performance. In contrast, the Laplacian matrix method shows a more gradual improvement, as it relies on a static consensus approach that lacks the real-time adaptability seen in the proposed method. Meanwhile, the centralized method suffers from scalability issues, as the increasing number of UAVs leads to communication bottlenecks and inefficient task management, which results in declining utility as the network size grows. Therefore, the analysis affirms that the proposed method is well-suited for large-scale deployments where a high number of UAVs are involved. On the other hand, while the laplacian matrix method may be suitable for moderate-scale deployments, the centralized approach is less ideal for large-scale UAV networks due to its poor scalability and lower utility values, making it less efficient for applications requiring extensive coordination among agents.

Fig. 5 compares total computation time as a function of average distance between agents. As distance increases, the total computation time rises for all methods. As observed in the plot, the laplacian matrix method is more efficient in terms of computation time, making it suitable for scenarios where minimizing computation time is critical, and the environment is less dynamic. The adaptive consensus mechanism in the proposed method likely introduces additional computational overhead due to dynamic adjustments, which contributes to higher computation times compared to the static laplacian method. However, its

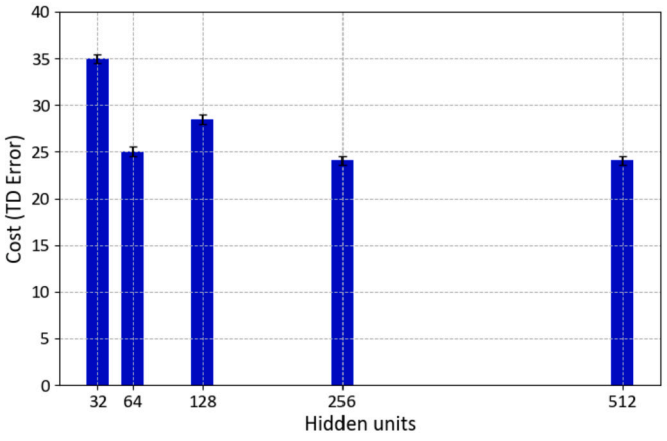


Fig. 6. Cost analysis with varying number of hidden units in the actor-critic network.

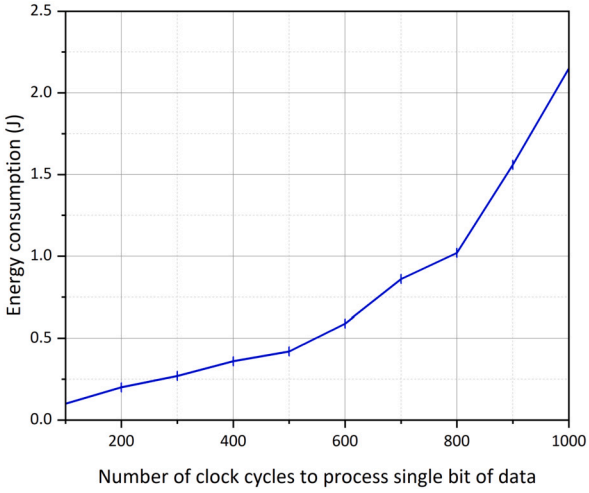


Fig. 7. Energy consumption for varying clock cycles per bit processing.

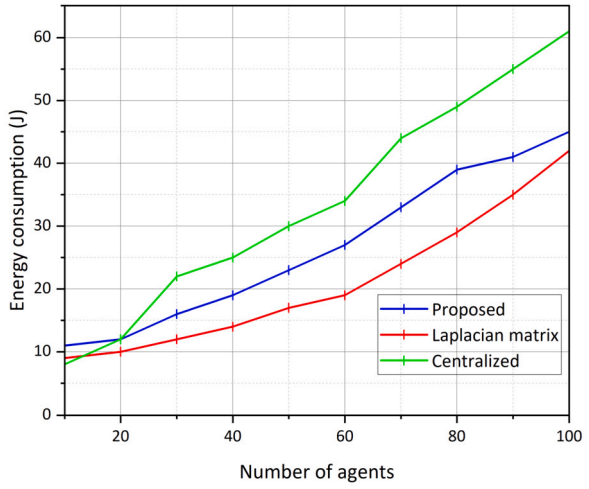


Fig. 8. Energy consumption with varying number of agents.

ability to adapt and handle dynamic environments better justifies its use in complex and large-scale UAV networks where such flexibility and robustness are critical, despite their slightly higher computation time.

To achieve optimal performance, it is important to select an appropriate number of hidden units in the neural network for the actor and the critic. The results plotted in Fig. 6 indicate that a model with 64 hidden

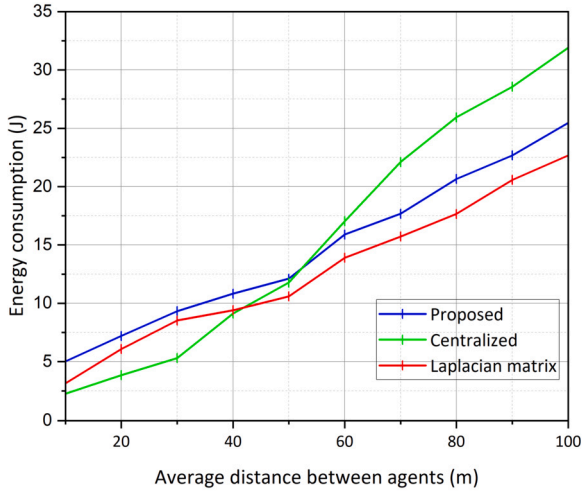


Fig. 9. Energy consumption for varying distance between agents.

units achieves optimal performance, balancing model complexity and prediction accuracy. With 32 units, the TD error is the highest, around 40. This suggests that the model is under-fitting, making it less suitable for capturing the complexity of the environment and in making accurate predictions. With 64 units, the model shows reduced TD error value, suggesting that 64 hidden units provide a better representation of the environment. The error remains relatively low, slightly above 20, suggesting that an increase from 64 to 128 does not significantly improve the model performance. With 256 hidden units, a marginal increase in error is observed, possibly due to over-fitting where the model becomes too complex and captures noise in the training data. Therefore, we have set the number of hidden units to 64 in the study.

For battery-operated UAV networks, maintaining energy efficiency is crucial. To better evaluate and understand the performance of the network in terms of energy, Fig. 7 plots the correlation between the number of clock cycles required to process a single bit of data and the energy consumed. At lower clock cycles, the energy consumption is minimal, around 0.1 to 0.2 joules. As the number of clock cycles increases to around 400-600, the energy consumption rises steadily to about 0.5 joules. A more significant increase is observed as the number of clock cycles approaches 1000, reaching up to approximately 2.0 joules. The significant rise in energy consumption beyond 600 clock cycles implies that it is critical to optimize computational tasks to stay within a lower clock cycle range. Therefore, maintaining an optimal range of clock cycles can significantly enhance the energy efficiency of the network, allowing UAVs to operate longer and perform more tasks.

As the number of agents increases, the energy consumption of the agents also rises, as illustrated in Fig. 8. This increase in energy consumption indicates a corresponding increase in communication overhead with the growing number of agents. It is clear that as the number of agents increases, the energy consumed by the agent that acts as the central controller in the centralized scheme depletes its energy at a faster rate compared to the other two approaches. Although the proposed method primarily involves sharing only three key parameters among agents to achieve consensus, these parameters must be disseminated across the entire network to ensure effective coordination. The need for frequent updates of these parameters—due to the learning involved—results in higher energy consumption compared to the static Laplacian matrix method. However, the increase in overhead and communication costs represents a trade-off for the substantial benefits offered by decentralized approaches, including reduced computation time, better scalability, adaptability, and reduced dependency on a central controller.

Fig. 9 shows the relationship between energy consumption and the average distance between agents for all three methods. The laplacian matrix approach has the lowest increase, suggesting better energy efficiency in managing the increased distances between agents, while the centralized method exhibits the highest energy consumption across all distances, indicating inefficiency in handling longer distances between agents. The proposed method shows an increase in energy consumption with distance but remains lower than [15] and only slightly higher than [14]. This increase in energy consumption compared to static methods can be due to the computations for policy updates, value function estimations, and adaptation based on feedback from the environment. Therefore, this is a reasonable trade-off for the significant advantages it provides in terms of adaptability, scalability, and improved decision-making.

5. Conclusion and future work

The work proposed a multi-agent reinforcement learning approach for multi-hop data offloading in UAV networks. By dynamically adjusting the consensus weights based on the learning progress of each agent, the system ensures that all agents benefit from the collective learning experience. This adaptability is crucial in environments where conditions can change rapidly, such as in disaster response missions. The laplacian matrix method discussed in literature, while better than the centralized approach, falls short in adapting to larger networks due to its static consensus weights. The centralized approach, with its inherent communication and control limitations, shows the poorest performance, highlighting the inefficiency of centralized decision-making in large and dynamic environments. While the proposed method shows a moderate increase in computation time and energy consumption, this is a trade-off achieved for the adaptability, improved decision-making capabilities and overall efficiency achieved by the network.

Future study will focus on evaluating agent performance when handling multiple tasks in parallel, which may introduce increased computational complexity but offer a more realistic assessment. Additionally, investigating the fault tolerance of these networks during data offloading and assessing the impact of environmental factors such as wind and obstacles through real-world experiments would be valuable. Addressing security challenges by developing secure communication protocols and intrusion detection mechanisms also represents a critical area for future work.

CRediT authorship contribution statement

Indu Chandran: Writing – review & editing, Writing – original draft, Software, Methodology, Formal analysis, Conceptualization. **Kizheppatt Vipin:** Writing – review & editing, Visualization, Validation, Supervision, Conceptualization.

Funding

This manuscript received no funding.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] S.A.H. Mohsan, M.A. Khan, F. Noor, I. Ullah, M.H. Alsharif, Towards the unmanned aerial vehicles (UAVs): a comprehensive review, Drones 6 (6) (2022) 147.
[2] I. Chandran, K. Vipin, Multi-uav networks for disaster monitoring: challenges and opportunities from a network perspective, Drone Syst. Appl. 12 (2024) 1–28.
[3] B. Liu, W. Zhang, W. Chen, H. Huang, S. Guo, Online computation offloading and traffic routing for uav swarms in edge-cloud computing, IEEE Trans. Veh. Technol. 69 (8) (2020) 8777–8791.

[4] T.X. Tran, D. Pompili, Joint task offloading and resource allocation for multi-server mobile-edge computing networks, IEEE Trans. Veh. Technol. 68 (1) (2018) 856–868.
[5] S. Bi, Y.J. Zhang, Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading, IEEE Trans. Wirel. Commun. 17 (6) (2018) 4177–4190.
[6] M.S. Hossain, C.I. Nwakanma, J.M. Lee, D.-S. Kim, Edge computational task offloading scheme using reinforcement learning for iiot scenario, ICT Express 6 (4) (2020) 291–299.
[7] L. Huang, S. Bi, Y.-J.A. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, IEEE Trans. Mob. Comput. 19 (11) (2019) 2581–2593.
[8] K. Zhang, Z. Yang, T. Başar, Decentralized multi-agent reinforcement learning with networked agents: recent advances, Front. Inf. Technol. Electron. Eng. 22 (6) (2021) 802–814.
[9] J. Yan, S. Bi, Y.J.A. Zhang, Offloading and resource allocation with general task graph in mobile edge computing: a deep reinforcement learning approach, IEEE Trans. Wirel. Commun. 19 (8) (2020) 5404–5419.
[10] L. Huang, X. Feng, A. Feng, Y. Huang, L.P. Qian, Distributed deep learning-based offloading for mobile edge computing networks, Mob. Netw. Appl. 27 (3) (2022) 1123–1130.
[11] Z. Chen, X. Wang, Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach, EURASIP J. Wirel. Commun. Netw. 2020 (1) (2020) 188.
[12] Z. Zhang, C. Li, S. Peng, X. Pei, A new task offloading algorithm in edge computing, EURASIP J. Wirel. Commun. Netw. 2021 (1) (2021) 17.
[13] X. Li, Y. Qin, H. Zhou, Y. Cheng, Z. Zhang, Z. Ai, Intelligent rapid adaptive offloading algorithm for computational services in dynamic Internet of things system, Sensors 19 (15) (2019) 3423.
[14] A. Sacco, F. Esposito, G. Marchetto, P. Montuschi, Sustainable task offloading in uav networks via multi-agent reinforcement learning, IEEE Trans. Veh. Technol. 70 (5) (2021) 5003–5015.
[15] N.T. Hoa, N.C. Luong, D. Le Van, D. Niyato, et al., Deep reinforcement learning for multi-hop offloading in uav-assisted edge computing, IEEE Trans. Veh. Technol. (2023).
[16] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2015) 2795–2808.
[17] J. Ren, J. Haupt, A communication efficient hierarchical distributed optimization algorithm for multi-agent reinforcement learning, in: Real-World Sequential Decision Making Workshop at International Conference on Machine Learning, 2019.
[18] W. Suttle, Z. Yang, K. Zhang, Z. Wang, T. Başar, J. Liu, A multi-agent off-policy actor-critic algorithm for distributed reinforcement learning, IFAC-PapersOnLine 53 (2) (2020) 1549–1554.
[19] K. Zhang, Z. Yang, T. Başar, Multi-agent reinforcement learning: a selective overview of theories and algorithms, in: Handbook of Reinforcement Learning and Control, 2021, pp. 321–384.
[20] Y. Xu, Y. Wang, J. Yang, Meta-heuristic search based model for task offloading and time allocation in mobile edge computing, in: Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence, 2020, pp. 117–121.
[21] A. Sacco, F. Esposito, G. Marchetto, P. Montuschi, A self-learning strategy for task offloading in uav networks, IEEE Trans. Veh. Technol. 71 (4) (2022) 4301–4311.
[22] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, W. Zhuang, Learning-based computation offloading for iot devices with energy harvesting, IEEE Trans. Veh. Technol. 68 (2) (2019) 1930–1941.
[23] K. Zhang, Z. Yang, T. Basar, Networked multi-agent reinforcement learning in continuous spaces, in: 2018 IEEE Conference on Decision and Control (CDC), IEEE, 2018, pp. 2771–2776.
[24] C. Qu, S. Mannor, H. Xu, Y. Qi, L. Song, J. Xiong, Value propagation for decentralized networked deep multi-agent reinforcement learning, Adv. Neural Inf. Process. Syst. 32 (2019).
[25] R. Lowe, Y.I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, Adv. Neural Inf. Process. Syst. 30 (2017).
[26] S. Kar, J. Moura, H.V. Poor, Qd-learning: a collaborative distributed strategy for multi-agent reinforcement learning through consensus, arXiv preprint, arXiv:1205.0047, 2012.
[27] K. Zhang, Z. Yang, H. Liu, T. Zhang, T. Basar, Fully decentralized multi-agent reinforcement learning with networked agents, in: International Conference on Machine Learning, PMLR, 2018, pp. 5872–5881.
[28] M. De Benedetti, F. D'Urso, G. Fortino, F. Messina, G. Pappalardo, C. Santoro, A fault-tolerant self-organizing flocking approach for uav aerial survey, J. Netw. Comput. Appl. 96 (2017) 14–30.