IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 April 2024, accepted 11 May 2024, date of publication 15 May 2024, date of current version 22 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3401115

RESEARCH ARTICLE

Task Offloading and Resource Allocation in UAV-Aided Emergency Response Operations via Soft Actor Critic

SHATHEE AKTER[1], **DAT VAN ANH DUONG**[1], **DAE-YOUNG KIM**[2], **(Member, IEEE), AND SEOKHOON YOON**[1], **(Member, IEEE)**
[1]Department of Electrical, Electronic and Computer Engineering, University of Ulsan, Ulsan 44610, South Korea
[2]Department of Computer Software Engineering, Soonchunhyang University, Asan 31538, South Korea

Corresponding author: Seokhoon Yoon (seokhoonyoon@ulsan.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2021R1I1A3051364.

ABSTRACT In modern emergency responses, unmanned aerial vehicles (UAVs) play a crucial role in redefining disaster management through diverse task execution (e.g., object detection). However, UAVs are usually resource-constrained. To address this issue, UAV mother-ships with edge servers (UMECs) can be placed near UAV scouts for remote task processing. UMECs can have a larger scale of energy consumption and battery capacity than UAV scouts. Therefore, minimizing the total or maximum energy consumption may result in ignoring the energy consumption in UAV scouts and focusing on UMECs, consequently reducing the network's lifetime. Furthermore, UMECs might lack the capacity to process all types of tasks owing to memory or software restrictions, and faster task execution often necessitates both central processing unit (CPU) and graphics processing unit (GPU), which are rarely considered by existing works. Therefore, this paper studies a task offloading and resource (computation capacity and power) allocation (TORA) problem with the goal of minimizing the maximum energy consumption ratio among UMECs and UAV scouts alongside task execution latency ratio and total energy consumption, where tasks are executed using both CPU and GPU, each UMEC can execute a specific set of tasks, and devices have limited resources. We then formulate the problem as a non-convex mixed-integer nonlinear programming problem and decompose it into multiple sub-problems. Finally, a soft-actor-critic (SAC) based TORA algorithm (SATORA) is proposed to address the problem, which can adapt to the time-varying environment scenario. Numerical simulation results show that SATORA outperforms other baseline algorithms.

INDEX TERMS Deep reinforcement learning, resource allocation, task offloading, multi-access edge computing.

I. INTRODUCTION

Every year, natural and man-made disasters such as wildfires, landslides, floods, terrorism, and industrial accidents cause significant damage to lives, property, the environment, and the economy. Emergency response operations after a disaster, including search and rescue, evacuations, emergency shelters, and damage assessment, play a crucial role in saving lives, providing aid, and facilitating the recovery process [1], [2], [3]. To improve the speed and efficiency of operations, advanced technologies such as unmanned aerial vehicles (UAVs) are increasingly integrated into disaster response strategies in recent years because of their flexibility, cost-effectiveness, and easy deployment in hard-to-reach and hazardous areas [4], [5], [6]. Despite their advantages, the tasks performed by UAVs (e.g., video streaming, object detection, real-time mapping, and tracking) are computationally demanding and highly time-sensitive, whereas the computational and energy resources of UAVs are limited. To meet the requirements of these

The associate editor coordinating the review of this manuscript and approving it for publication was Hongwei Du.

resource-demanding applications, a promising solution called mobile edge computing (MEC) has emerged, where tasks can be offloaded to edge servers with a more powerful computational capacity deployed at base stations (BSs) or in UAVs (working as aerial base stations) or both [7], [8], [9]. The success of an MEC system is highly dependent on the task offloading (TO) decision (whether a task should be offloaded or not, and where to offload it) and resource allocation (RA) since different edge servers and amounts of allocated resources (such as power and computational capacity) will lead to different energy consumption and execution times.

Many researchers, therefore, have proposed strategies for jointly optimizing TO and RA over the past years in their respective problem scenarios [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]. Existing studies mostly aim to minimize total or maximum energy consumption in the system. However, the energy capacity of each device in a MEC system can be different, as well as the scale of energy consumption.

For example, assume a UAV network aided by a MEC system in a search and rescue operation with the mission objective of detecting humans in a disaster area. The network consists of resource-constrained UAV scouts that generate the task (i.e., human detection using deep learning technique), edge servers equipped with UAV mother-ships (UMECs), and a base station equipped with cloud servers (BSC), where UMECs and BSC provide computing services. The task can be processed locally in the UAV scout or offloaded either directly to a UMEC or to the BSC through an associated UMEC. Furthermore, the total operation time is divided into time slots, and the mobility of the UAV scouts and UMECs follows the grid pattern (i.e., the area is divided into grids, and the UAVs move from one grid to another after staying a given amount of time at each grid). At each time slot, a task is generated by each UAV scout, and the available resources (i.e., energy and computational capacity) of the UAV scouts and UMECs are limited. We need to find a task offloading, transmission power, and resource allocation solution such that the total resources consumed do not exceed the budget of the devices (at each timeslot). In this scenario, the energy consumption scale and the energy capacity of the UMECs can be much higher than the UAV scouts. Thus, minimizing total or maximum energy consumption will mostly focus on minimizing the energy consumption of UMECs due to their larger scale of energy consumption. This can lead to a higher percentage of energy consumption in the UAV scouts, compared to their capacity, due to lack of optimization, and as a consequence, the lifetime of the UAV network may be reduced.

Furthermore, existing works consider that all types of task can be executed in a MEC server. However, different types of tasks may have different requirements (e.g., memory, computation capacity, database, and service-software) for execution, and in many cases, it is impossible to prepare all services demanded when MEC servers are UAVs. Moreover, most existing studies assumed that only the CPU executes the tasks, whereas faster performance can be achieved by harnessing the combined processing power of a graphics processing unit (GPU) and a CPU in the case of most deep learning techniques.

Therefore, in this paper, we consider a task offloading and resource (i.e., transmission power, CPU cycles, and GPU FLOPS) allocation (TORA) problem with the aim of minimizing the maximum energy consumption ratio among UAV scouts and UMECs, the maximum task execution latency ratio, and the total energy of the system while taking into account the task execution limitations of UMECs and the resource budgets. The maximum energy consumption ratio of a processing device (UAV scout or UMEC) is the ratio of actual maximum energy consumption and approximated probable maximum energy consumption with respect to the device. The task execution latency ratio is the total execution latency over the deadline of the task. The considered problem contains an integer and a non-convex problem, which makes it very complex and hard to solve.

In our previous paper [23], although we have considered a similar problem using a meta-heuristic approach based on a messy genetic algorithm (mGA), it has the following limitations. First, it considers a different objective, i.e., minimize the maximum energy consumption ratio and the total task execution latency ratio. Second, a meta-heuristic strategy based on the memetic genetic algorithm (mGA) is used, which lacks adaptability to variations in parameters in the problem scenario or environment (such as data sizes of tasks that need to be offloaded and the required CPU cycles and GPU FLOPS for task processing). Consequently, it necessitates re-execution each time a parameter changes, introducing a potentially longer decision-making time. Third, the mGA-based strategy is more prone to falling into local optima due to losing genetic diversity.

Therefore, in this paper, a deep reinforcement learning (DRL) based TORA approach, which is called soft actor-critic (SAC) based TORA (SATORA) algorithm, is proposed that can adapt to the dynamic environment, i.e., can find a solution within a very short time in an unseen environment by learning general patterns and representations of the environment. Furthermore, DRL continuously explores different actions and their outcomes in the environment (states and rewards) by refining its strategies based on past experiences, which reduces the likelihood of getting stuck into local optima, making it a more appropriate choice for our dynamic problem scenario. The main contributions of this paper are summarized as follows:

- A TORA problem in a MEC-enabled hierarchical UAV network is studied in this paper with the objective of minimizing the maximum energy consumption ratio among UAV scouts and UMECs, maximum task execution latency ratio, and total energy consumption in the system. The considered problem is first formulated as a mixed-integer non-linear programming (MINLP)

problem while taking into consideration the task performing capabilities of UMECs, both GPU and CPU processing of tasks, the computational resources, the transmission power budget, and the energy consumption constraints of UMECs and UAV scouts. The formulated MINLP problem consists of integer and continuous variables, which is highly challenging to solve. Thus, we decompose the problem into two problems (a TO problem and an RA problem), which enables the use of multiple solution-searching strategies. TO is an integer programming problem and is NP-hard, whereas RA is a continuous problem.
- To solve the TO and RA problems, we propose a SAC-based joint TORA (SATORA) strategy, which can cope with the dynamic environment. In SATORA, the integer programming problem is solved by using a SAC-based TO algorithm, whereas a Karush–Kuhn–Tucker (KKT)-based simple power and computational resource allocation strategy [23] is employed to obtain the RA solution.
- The proposed algorithm is evaluated by conducting simulations under various problem scenarios, employing multiple performance metrics. The results verify that SATORA can outperform other approaches by attaining a longer network lifetime, lower task execution latency, and lower total energy consumption in the system.

The remainder of this paper is organized as follows. In Section II, related works are discussed. The system model, problem formulation, and decomposition are described in Section III. The proposed algorithm is presented in Section IV. In Sections V and VI, respectively, the result analysis of the proposed method and the conclusion is presented.

II. RELATED WORKS

A. TASK OFFLOADING AND RESOURCE ALLOCATION

Over the past few years, there has been significant research conducted on the simultaneous optimization of TO and RA in multi-access edge computing [9], [10], [11], [12], [13], [14], [15]. However, these studies made the assumption that MEC servers are deployed within base stations (BSs), which can introduce significant delays for users located far from the BSs, lack flexibility, and restrict the availability of MEC services in areas without infrastructure or outside the coverage range of the BSs.

Therefore, recently, the concept of UAV-aided MEC networks has gained attention, where UAVs consist of edge servers and operate as base stations (BSs) [16], [17], [18], [19], [20], [21], [22]. For example, in [16], the authors considered a fog node carried by a UAV that provides radio and computation services to mobile users. They optimized the TO, computation frequency, and transmission power to minimize energy consumption and delay. In [17], a mixed edge-cloud computing framework with space-aerial assistance was introduced, where a joint optimization of UAV computational resources, computation task allocation, transmission power, association control, deployment position, and bandwidth allocation was performed, aiming to minimize the maximum computation delay among IoT devices. In [18], a two-layer optimization strategy was presented, focusing on UAV trajectory optimization, bit allocation, and task scheduling to minimize energy consumption for ground users. The upper layer employed a dynamic programming-based method for task scheduling, whereas the lower layer addressed the UAV trajectory and bit allocation problems by using an alternating direction method of multipliers. Messous et al. [20] proposed a TO method that finds a balance between energy consumption and time delay in a UAV network comprising end-users (UAVs), base stations, and edge stations (ESs) where end-users can offload tasks to the BS or to ESs. Chen et al. [21] minimized the average service latency of tasks by introducing an intelligent TO algorithm (iTOA) that utilizes the deep Monte Carlo Tree Search.

While the aforementioned studies explored various tasks and applications in their MEC systems, they assumed that edge servers equipped UAVs are capable of performing any type of task, whereas in reality, this assumption is impractical due to the limited resources of UAVs. Additionally, none of these studies took into account the scenario where tasks are computed using both CPU and GPU, which makes execution faster than when processed only by the CPU. Moreover, current works only minimize the UAVs' total or maximum energy usage, overlooking varying levels of energy consumption across UAV types. This oversight can lead to unfair energy consumption among different UAV categories, such as UAV scouts and UMECs, relative to their energy capacities.

In contrast, our study considers that MEC servers (i.e., UMECs) can only process limited types of tasks, and the tasks are computed on the CPU as well as a GPU. Moreover, we consider a multi-user and multi-server MEC system within a hierarchical UAV network, and propose a comprehensive DRL-based framework that minimizes the maximum energy consumption ratio among UAV scouts and UMECs with respect to their energy capacity, task execution latency ratio, and total energy consumption in the system.

B. DRL-BASED TASK OFFLOADING AND RESOURCE ALLOCATION

In recent years, several studies on TO have started focusing on DRL-based solutions because of their generalization capability and lower time complexity in high-dimensional and complex problem scenarios. For example, Seid et al. [24] proposed a model-free DRL-based collaborative computation offloading and resource allocation scheme to minimize task execution delay and energy consumption in emergency situations. In particular, they proposed a deep deterministic policy gradient (DDPG)-based method for multi-UAV-assisted IoT networks while taking into consideration time-varying computational node capacity, channel quality, and tasks. The authors in [25] studied a network lifetime maximization problem in an IoT network consisting of multiple users and a MEC server. They proposed a SAC-based DRL framework called DeepLM to jointly optimize the local CPU-cycle frequency, task splitting ratio, and bandwidth allocation. In [26], the authors combined two DRL algorithms (multi-agent DDPG and SAC) to find the target server for TO and the amount of data offloaded in a wireless power transfer-enabled scenario. Zhang et al. [27] aimed to jointly optimize the cooperative partial TO and computational RA for industrial Internet of Things (IIoT) devices. They developed an improved SAC-based framework to find the TO and computational RA decisions for each IIoT device. The study in [28] focused on cooperative computation offloading to address the limitations of single-edge servers and nonuniform task distribution in edge servers. They aimed to offload tasks from an edge server with a heavy load to other neighbor ESs and cloud servers, and proposed two algorithms based on SAC to find the solution. The authors in [29] introduced a DRL-based framework to minimize computation overhead (execution delay and energy consumption) in a non-orthogonal multiple access (NOMA)-MEC system. Their proposed model takes both continuous and discrete action space and time-varying fading channels into account.

However, the considered problem scenarios or system models or objectives in the above studies are very different from our problem scenario, objective, and optimization problems. Therefore, these methods are not directly applicable to our problem.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this paper, an MEC-enabled UAV network that performs emergency response operations is considered, as shown in Fig. 1. The network consists of a base station equipped with edge servers (BSC) g, a set of UAV mother-ships with edge servers (UMECs) $\mathbb{M}$ ($\mathbb{M} = \{1, \ldots, |\mathbb{M}|\}$), and a set of UAV scouts $\mathbb{S}$ ($\mathbb{S} = \{1, .., |\mathbb{S}|\}$). The operation time is assumed to be divided into time slots with a constant duration. At the beginning of each time slot, at most, one task can be generated by each UAV scout. The task can be performed locally or offloaded to a UMEC or to the BSC through the associated UMEC (which has the communication link with both the UAV scout and the BSC), according to the decision given by the lead UMEC after executing the decision-making algorithm.

UAV scout i has the following specifications: task size in bits, s_i, required CPU cycles, c_i and GPU FLOPS, g_i, for performing the task, desired task execution time or deadline before which the task is expected to be processed, d_i, type of the task, V_i ($V_i \in \mathbb{Z}^+$) (represents different tasks such as pose recognition or object detection), CPU capacity, f_i (cycle/second), GPU capacity, u_i (in FLOPS), maximum transmission power, b_i, and battery capacity for data transmission and reception, e_i. $\mathbb{Z}^+$ is the set of positive integer.

UMEC m is associated with the following: a set of specific tasks that it has the capability to perform, $\mathbb{T}_m$ ($\mathbb{T}_m \in \mathbb{Z}^+$), transmission power budget, F_m, computation resources (i.e., CPU in cycles/second, w_m and GPU in FLOPS, h_m), and battery capacity a_m for transmitting and receiving data.

A. COMMUNICATION MODEL

In a UAV network, line-of-sight channels exhibit more dominance since both UMECs and UAV scouts fly at high altitudes. Therefore, we assume that the channels between UAV scouts and UMECs follow the free-space path loss model, and the channel access scheme is considered to be the orthogonal frequency division multiple access (OFDMA) scheme. Then, the attainable bit rates (uplink) r_{im} between UAV scout i and UMEC m when offloaded to a UMEC or r_{mg} when transmitting from UMEC m to BSC g are calculated as follows:

$$r_{im} = W \log_2(1 + \frac{w_{im} p_i}{\sigma_{im}}) \tag{1}$$

$$r_{mg} = W \log_2(1 + \frac{w_{mg} o_{img}}{\sigma_{mg}}), \tag{2}$$

where W denotes the bandwidth. w_{im} and w_{mg} represent the channel gains from UAV scout i to UMEC m and UMEC m to BSC g, respectively. p_i represents the transmission power assigned by the UAV scout for task offloading, and o_{img} is the transmission power assigned by UMEC m to the task of UAV scout i when transmitting the task to the BSC. σ_{im} and σ_{mg} are the noise powers.

B. TASK EXECUTION LATENCY MODEL

The task execution delay, when processed locally, comprises control signal transmission and reception time for task requests and decisions. However, the time needed for the control signal and for decision-making is ignored in this work since control signals use the dedicated channel, and the decision-making algorithm takes a negligible amount of time to run (the proposed DRL-based method is trained before deployment and runs in a UMEC). Therefore, the task execution latency for local processing is given by,

$$I_i = \frac{c_i}{f_i} + \frac{g_i}{u_i}. \tag{3}$$

However, if UAV scout i decides to offload the task to a UMEC or the BSC, then the task transmission time, propagation delay, and post-processing output retrieval time also need to be taken into account, whereas the output data transmission time is ignored here because of the negligible data size and higher downlink data rate.

Furthermore, MEC servers have limited computation capacities; thus, the computation resources assigned to each task for execution need to be taken into consideration. Let b_{im} and h_{im}, respectively, denote the CPU and GPU resources allocated to UAV scout i by UMEC m. The CPU cycles and GPU FLOPS allocated to each UAV scout i by BSC g are denoted by z_{ig} and u_{ig}, respectively.

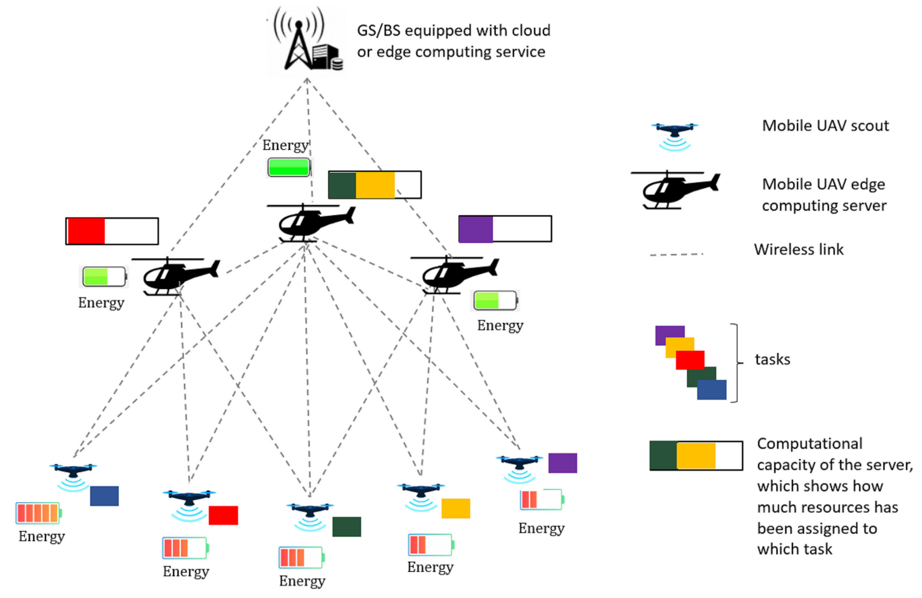

GS/BS equipped with cloud or edge computing service
Energy
Energy
Energy
Energy
Energy
Energy
Energy
Energy
Mobile UAV scout
Mobile UAV edge computing server
Wireless link
tasks
Computational capacity of the server, which shows how much resources has been assigned to which task

FIGURE 1. An example of a MEC-enabled UAV network.

Then, the task execution delay for offloading to a UMEC m, I_{im}, or to BSC g, I_{img}, can be calculated as follows:

$$I_{im} = \frac{s_i}{r_{im}} + \frac{D_{im}}{C} + \frac{c_i}{b_{im}} + \frac{g_i}{h_{im}} \tag{4}$$

$$I_{img} = \frac{s_i}{r_{im}} + \frac{s_i}{r_{mg}} + \frac{D_{im}+D_{mg}}{C} + \frac{c_i}{z_{ig}} + \frac{g_i}{u_{ig}} \tag{5}$$

where C is the light speed. In eq. (4), the first term is task transmission time, the second term is propagation delay, the third part represents task execution time in the CPU, and the fourth term, task processing time in the GPU. In eq. (5), the first two terms represent the task transmission times from the UAV scout to the associated UMEC (which works as the relay between the BSC and the UAV scout), and from the UMEC to the BS, respectively. The third term represents the propagation delay from the UAV scout to the UMEC and from the UMEC to the BSC, whereas the fourth and fifth terms represent the task execution times in the CPU and GPU, respectively.

Note that in this paper, we assume that the GPU mostly processes the computationally intensive functions of the tasks; therefore, we only focus on the GPU resource allocation, whereas CPU resources are assumed to be divided equally among all assigned tasks in the processing device.

C. ENERGY CONSUMPTION MODEL

When a task is offloaded to a UMEC, the energy consumption of the task includes the energy needed for transmission and execution (in the CPU and GPU), whereas for local processing, it only includes the energy needed for execution in the CPU and GPU. Therefore, the energy usage for task execution locally, E_i, for transmitting the task to a UMEC, E_{im}, and for processing the task in a UMEC, R_{im}, are calculated in eqs. (6)-(8), respectively:

$$E_i = q_i \times c_i + v_i \times g_i \tag{6}$$

$$E_{im} = p_i \times \frac{s_i}{r_{im}} \tag{7}$$

$$R_{im} = q'_m \times c_i + v'_m \times g_i, \tag{8}$$

where q_i signifies the energy used per CPU cycle, v_i denotes the energy consumed per GPU FLOP in UAV scout i. q'_m stands for energy consumption per CPU cycle, and v'_m indicates energy expenditure per GPU FLOP in UMEC m.

The BSC is considered to have a continuous power supply; therefore, the energy consumption for task execution in the BSC only includes transmission energy consumption from UAV scout i to UMEC m to BSC g. Then, the energy consumption for transmission from UMEC m to BSC g is

$$E_{img} = o_{img} \times \frac{s_i}{r_{mg}}. \tag{9}$$

D. PROBLEM FORMULATION

In this subsection, the considered TORA problem is formulated as a mixed integer non-linear programming (MINLP) problem. Optimization variables used in the problem formulation are defined as follows:

$$x_{im} = \begin{cases} 1, & \text{if UAV scout } i \text{ offloads its task to the UMEC } m \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$$x_{img} = \begin{cases} 1, & \text{if UAV scout } i \text{ offloads its task to the BSC } g \\ & \text{through associated UMEC } m \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

TABLE 1. Model parameters.

UAV scout	
$\mathbb{S}$	set of UAV scouts.
e_i	energy capacity of UAV scout i
u_i	GPU capacity of UAV scout i
p_i	power assigned for task transmission by UAV scout i
b_i	transmission power budget of UAV scout i
f_i	CPU capacity of UAV scout i
c_i	number of CPU cycles required for task execution
u_i	GPU capacity of UAV scout i
d_i	desired execution latency before which the task of UAV scout i is expected to be completed
s_i	task size(in bits)
g_i	number of GPU FLOPS required to process the task of UAV scout i
V_i	task type of UAV scout i and $V_i^k \in \mathbb{Z}^+$
q_i	energy consumption (per CPU cycle) of UAV scout i .
v_i	per GPU FLOP energy consumption of UAV scout i .
UMEC	
$\mathbb{M}$	set of UMECs.
$\mathbb{T}_m$	tasks set that UMEC m can execute.
a_m	energy capacity of UMEC m
w_m	CPU capacity of UMEC m
h_m	GPU capacity of UMEC m
o_{img}	power assigned to the task by UMEC m to transmit to the BSC g
F_m	transmission power budget of UMEC m
h_{im}	the number of GPU FLOPS assigned to the task of UAV scout i by UMEC m
b_{im}	the number of CPU cycles assigned to the task of UAV scout i by UMEC m
v'_m	energy consumption of UMEC m per GPU FLOP.
q'_m	per CPU cycle energy consumption of UMEC m.
τ_m	energy consumed by UMEC m for receiving each bit of task.
Others	
W	bandwidth
g	base station
D_{mg}	distance between UMEC m and BSC g
D_{im}	distance between UAV scout i and UMEC m
w_{mg}	uplink channel gain between UMEC m and BSC g
w_{im}	channel gain (on uplink) from UAV scout i to UMEC m
r_{mg}	bit rate from UMEC m and BSC g
r_{im}	bit rate from UAV scout i and UMEC m
u_{ig}	number of GPU FLOPS assigned by BSC g to the task of UAV scout i
z_{ig}	number of CPU cycles assigned by BSC g to the task of UAV scout i
z_g	local GPU capacity of BSC g
C	light speed.
$\mathbb{Z}^+$	set of positive integer

where both x_{img} and x_{im} equaling 0 represents the local task processing.

The total energy usage of UAV scout i and UMEC m therefore can be written as:

$$O_i = \sum_{m \in \mathbb{M}} E_{im}\big(x_{im} + x_{img}\big) + \Big(1 - \big(x_{im} + x_{img}\big)\Big)E_i \quad (12)$$

and

$$O_m = \sum_{i \in \mathbb{S}} \Big(\Big(\tau_m \times s_i + R_{im}\Big)x_{im} + \big(\tau_m \times s_i + E_{img}\big)x_{img}\Big), \quad (13)$$

where τ_m is the reception energy consumption of UMEC m per bit. Then, the energy consumption ratios of UAV scout i and UMEC m, respectively, are calculated as $\frac{O_i}{e_i^{max}}$ and $\frac{O_m}{a_m^{max}}$, where $e_i^{max} = \min\{e_i, O_i^{max}\}$ and $a_m^{max} = \min\{a_m, O_m^{max}\}$. O_i^{max} and O_m^{max} are the approximated probable maximum energy consumption for an UAV scout and a UMEC, respectively.

The task execution delay of UAV scout i can be rewritten as follows:

$$T_i = \big(I_{im}x_{im} + I_{img}x_{img}\big) + \big(1 - (x_{im} + x_{img})\big)I_i \quad (14)$$

To take into account the associations between devices, we define two matrixes, $Q \in \mathbb{B}^{|\mathbb{M}| \times |\mathbb{G}|}$ and $Z \in \mathbb{B}^{|\mathbb{S}| \times |\mathbb{M}|}$, to represent UMEC-BSC and UAV scout-UMEC associations, respectively. Each element $Q_{(m,g)}$ and $Z_{(i,m)}$ in matrixes Q and Z, respectively, can have either value "1" or "0". The value "1" represents the association between devices, whereas "0" signifies there is no communication link. For example, in matrix z, $Z_{(i,m)} = 1$ signifies that UAV scout i is associated with UMEC m; otherwise,

$Z_{(i,m)} = 0$. The remaining notations (that are used in the problem formulation) definitions are presented in Table 1.

The main problem can then be formulated as follows:

$$\min w_1 \max_{i \in \mathbb{S}, m \in \mathbb{M}} \left\{ \frac{O_i}{e_i^{max}}, \frac{O_m}{a_m^{max}} \right\} + w_2 \max_{i \in \mathbb{S}} \left\{ \frac{T_i}{d_i} \right\} + w_3 \left(\sum_{i \in \mathbb{S}} O_i + \sum_{m \in \mathbb{M}} O_m \right) \quad (15)$$

subject to:

$$\sum_{m \in \mathbb{M}} x_{im} + x_{img} \leq 1, \qquad \forall i \in \mathbb{S} \quad (16)$$

$$x_{im} = Z_{(i,m)} x_{im}, \qquad \forall m \in \mathbb{M}, \forall i \in \mathbb{S} \quad (17)$$

$$2x_{img} = (Z_{(i,m)} + Q_{(m,g)}) x_{img}, \qquad \forall m \in \mathbb{M}, \forall i \in \mathbb{S} \quad (18)$$

$$|\{V_i\} \cap \mathbb{T}_m| + (1 - x_{im}) \geq 1, \quad \forall i \in \mathbb{S}, \forall m \in \mathbb{M} \quad (19)$$

$$\sum_{i \in \mathbb{S}} h_{im} x_{im} \leq h_m, \qquad \forall m \in \mathbb{M} \quad (20)$$

$$\sum_{i \in \mathbb{S}} \sum_{m \in \mathbb{M}} u_{ig} x_{img} \leq z_g \quad (21)$$

$$p_i (x_{im} + x_{img}) \leq b_i, \qquad \forall i \in \mathbb{S} \quad (22)$$

$$\sum_{i \in \mathbb{S}} o_{img} x_{img} \leq F_m, \qquad \forall m \in \mathbb{M} \quad (23)$$

$$O_i \leq e_i, \qquad \forall i \in \mathbb{S} \quad (24)$$

$$O_m \leq a_m, \qquad \forall m \in \mathbb{M} \quad (25)$$

where w_1, w_2, and w_3 are weights for scaling.

The objective of the TORA problem is presented in equation (15), which is minimizing the weighted sum of the maximum energy consumption ratio among UAV scouts and UMECs, maximum task execution latency ratio, and total energy consumption in the network. Inequality (16) ensures that a UAV scout can either offload its task to only one UMEC or to the BSC, or execute it locally. In particular, $\sum_{m \in \mathbb{M}} x_{im} + x_{img} = 0$ means the task of UAV scout i is not offloaded to any of the UMEC or the BSC and is processed locally. If $x_{im} = 1$, then UAV scout i should not offload its task to the BSC or to any other UMECs except UMEC m. Similarly, the task should not be offloaded to the BSC using several relay UMECs or offloaded to the UMECs if $x_{img} = 1$. Equation (17) states that a communication link is necessary to offload the task from a UAV scout to a UMEC. Constraint (18) mandates the association between the BSC and its associated UMEC and between the UAV scout and the UMEC when the task is offloaded to the BSC. Constraint (19) guarantees task assignment to a UMEC that has the capability (in terms of types of tasks). Inequalities (20) and (21) prevent exceeding the GPU computing resource budgets of UMECs and the BSC, respectively. The maximum transmission power constraints of UAV scouts and UMECs, respectively, are expressed in inequalities (22) and (23). The constraints on the energy capacities of UAV scouts and UMECs, respectively, are expressed in eqs. (24) and (25), ensuring total energy consumption remains within capacity.

The above-formulated problem (15) consists of both binary and continuous variables and is non-convex. Therefore, following a similar strategy in [9] and [23], we decompose the problem into an integer problem (TO) and a non-convex continuous problem (RA–power and computational resource) and convert the non-convex continuous problem into a convex problem. The integer problem, i.e., TO, has a very large solution space. Assume that there are M UMECs, K UAV scouts with exactly one task, and G BSCs. A UMEC can process tasks of all K UAV scouts; thus, there are 2^K ways to select a UMEC. For M UMECs, there is 2^{MK} assignment or offloading schemas. Moreover, each BSC can also process tasks of all K UAV scouts, and each UAV scout can offload the task to a BSC through M different UMECs. Therefore, the solution space will contain a total of $2^{MK} + 2^{KMG}$ assignment schemas.

In addition, a simplified model of TO can be mapped to an instance of the generalized assignment problem (GAP), which is a well-known NP-hard problem [30]. In particular, assume an instance of GAP with L tasks, P agents, a cost c_{lp} associated with assigning task l ($l = 1, \ldots, L$) to agent p ($p = 1, .., P$) and a capacity budget for each agent. Each task can be assigned to exactly one agent, and the objective is to minimize the total cost of the assignment subjected to capacity constraints on agents. Then, we simplify the TO problem to only UMECs and UAV scouts (no base station), where UAV scouts can process their tasks locally or offload to the UMECs. Each UAV scout can offload to exactly one UMEC. This simplified form of our problem closely mirrors GAP, where UAV scouts can be mapped to the tasks and UMECs to the agents, except for the explicit capacity constraints on UMECs. However, introducing a pseudo-capacity constraint (e.g., a maximum number of tasks that a UMEC can handle, even if it's a large number) transforms it into a form of GAP. Since GAP is NP-Hard, and our simplified problem can be viewed as a special case or variation of GAP, it implies that TO is at least as hard as GAP.

IV. THE DRL-BASED TORA FRAMEWORK

We assume that the task size, required GPU FLOPS, task types, deadline, and the number of tasks generated in the environment vary in each time slot. Therefore, if the considered problem is solved by a heuristic algorithm, such as a genetic algorithm (GA), it necessitates repeated executions whenever there is a variation in any parameter value within the environment that may need a relatively longer decision-making time, whereas a greedy algorithm usually gives a sub-optimal solution. Contrarily, deep reinforcement learning uses deep learning models to learn the complex patterns and representations of the environment, which enables it to generalize to the unseen environment and obtain solutions within a very short time. Furthermore, in DRL, the agent explores the environment by taking actions based on a policy that maximizes the total reward obtained. The reward guides the agent towards a better solution at each step, which

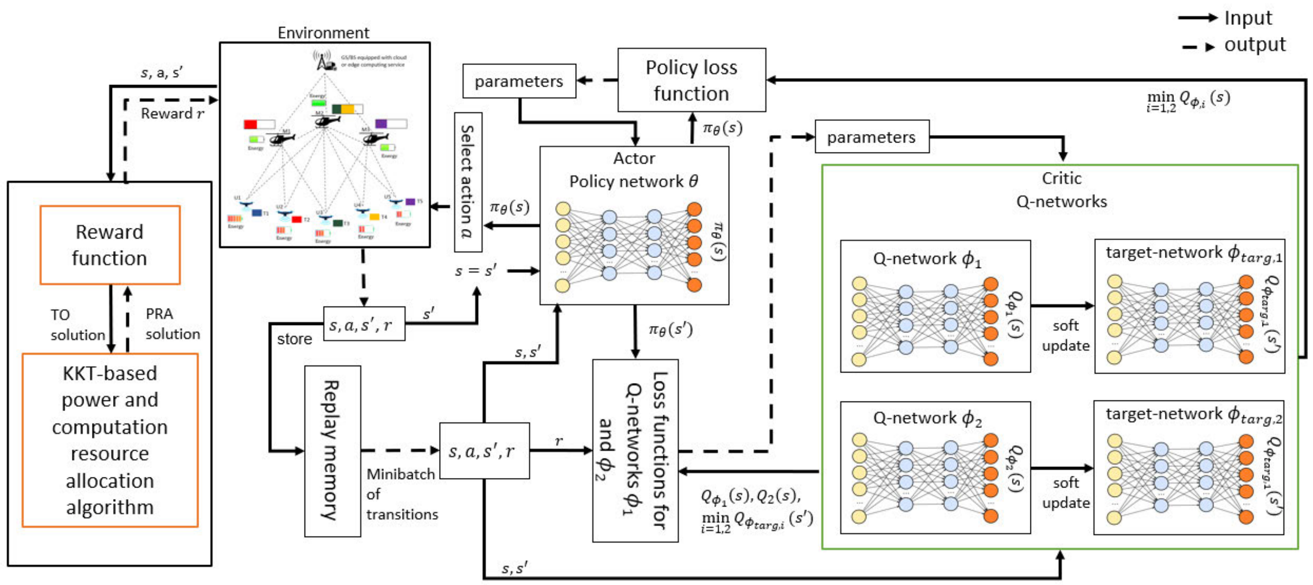


FIGURE 2. SAC-based TORA framework.

encourages the agent to avoid local optima. This ability of the DRL methods to learn and adapt in real-time, combined with its near-optimal or optimal solution searching strategy, makes it highly suitable for our considered problem and we believe that it can outperform traditional methods by efficiently balancing decision-making speed with solution quality. Note that GA, greedy algorithm, and DRL are all widely-used techniques for solving the complex optimization problems. Therefore, in this section, a DRL framework for TO and RA is proposed based on the soft-actor-critic (SAC) [31] algorithm, which can make faster decisions by generalizing when environmental parameters change.

Given the sets of UAV scouts and UMECs, we first model our considered problem as a Markov decision process (MDP), and then, the SAC-based TORA algorithm is presented in detail.

A. MDP MODEL

1) STATE AND ACTION

In this work, the state includes the task specifications (task size, required GPU FLOPS, CPU cycles, and deadlines) of UAV scouts, offloading decision, and distances between processing devices (UMECs, BSC, and UAV scout itself) and UAV scouts. Before defining the state, let us denote each processing device as p and $p \in \mathbb{P}$ (the set of all processing devices in the system). Then, the state s can be expressed as $s = \{d_{ip}, y_{ip}, s_i, g_i, c_i, d_i\}, \forall i \in \mathbb{S}, \forall p \in \mathbb{P}$, where d_{ip} represents the distance and $y_{ip} \in \{0, 1\}$ denotes the offloading decision between UAV scout i and processing device p. If UAV scout i offloads to processing device p, then $y_{ip} = 1$; otherwise, $y_{ip} = 0$. Then, the size of the state space for a given environment is $2^{|\mathbb{S}|\times|\mathbb{P}|}$.

The action in this MDP is choosing a processing device-UAV scout pair, i.e., $a = \{i, p\}$. Action a means the task of UAV scout i will be processed in device p. If p is selected for processing, then the rest of the devices can not be chosen for the task of UAV scout i. Thus, the size of the action space is $|\mathbb{S}| \times |\mathbb{P}|$.

2) REWARD

In DRL algorithms, the reward function assigns a numerical value to each state-action pair, indicating the immediate benefit or cost associated with the agent's decision. The goal is to design a reward function that reflects the objective and the constraints of the TORA problem, i.e., that encourages the agent toward the optimal value of the objective by reinforcing actions leading to better solutions and discouraging those resulting in constraint violations or sub-optimal solutions. Therefore, we first design a utility function as follows:

$$U = -\Bigg(w_1 \max_{i\in\mathbb{S}, m\in\mathbb{M}}\left\{\frac{O_i}{e_i^{max}}, \frac{O_m}{a_m^{max}}\right\} + w_2 \max_{i\in\mathbb{S}}\left\{\frac{T_i}{d_i}\right\} + w_3\left(\sum_{i\in\mathbb{S}} O_i + \sum_{m\in\mathbb{M}} O_m\right) + \beta\Bigg) \tag{26}$$

where β is the penalty for violating the constraints. Then, the reward function can be expressed as follows:

$$r(s, a, s') = \begin{cases} mU - \epsilon, & \text{if } U \text{ is higher in } s' \text{ than } s \\ mU, & \text{otherwise} \end{cases} \tag{27}$$

where $m(m \geq 1)$ is called a magnifying constant in this paper that adds flexibility to the scale of the utility value. The magnifying constant is added in order to avoid situations where the reward differences among states becomes too small to learn. $\epsilon(\epsilon \in [0, 1])$ is a penalty given to prevent the agent from going to relatively worse states than the current one. Moreover, to avoid varied scales of rewards and to allow more stable training, the reward, r, is clipped to within the range $(-n, 0)$. n is an integer and $n > 0$.

B. THE SAC-BASED TORA ALGORITHM

Deep reinforcement learning algorithms leverage the power of deep learning techniques to approximate either the policy function, which maps states to actions, or the value function, which estimates the expected cumulative reward, or sometimes both, in order to learn a policy that maximizes the overall expected reward. DRL algorithms that are frequently employed for solving the optimization problems are deep Q-networks (DQN) [32], advantage actor-critic (A2C) [33], and soft actor-critic [31]. DQN is one of the pioneering and dedicated algorithms designed for the discrete action space, whereas A2C can be adapted to both discrete and continuous action spaces. However, DQN needs a large number of samples to learn a good policy and sometimes struggles with exploration in high-dimensional action spaces and with high complexity, whereas A2C often converges to a sub-optimal policy because of the lack of exploration (only uses an actor network to explore).

Different from other DRL algorithms, SAC uses an entropy-regularized reward function and a nondeterministic policy that encourages exploration and helps prevent premature convergence. Specifically, the goal of SAC is to find a policy that maximizes the cumulative reward and policy entropy, which is expressed as follows:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{T} \mathbb{E}\big[\gamma^t r(s_t, a_t, s_{t+1}) + \alpha H(\pi(.|s_t))\big] \quad (28)$$

where π and π^*, respectively, are the policy and the optimal policy, t is the time step, T is the total number of timesteps in an episode, and γ is the discount factor and $\gamma \in [0, 1]$. α is called the temperature parameter that determines the importance between the reward and the policy entropy $H(\pi(.|s_t))$ at state s, where $H(\pi(.|s_t)) = \mathbb{E}[-log\pi(.|s_t)]$. This entropy-regularized reward prevents assigning an excessive probability mass to a single action, and encourages more exploration in the environment, which helps the agent avoid local optima.

However, SAC is primarily designed to solve problems with a continuous space, whereas many environments may contain a discrete action space. Therefore, an extension of SAC applicable for the discrete action space was derived by [34], called SAC-Discrete. Note that SAC-Discrete holds the same objective as the original SAC except that $\pi(.|s)$ is a probability mass function instead of a density function as in the original SAC. Therefore, this paper proposes a SAC-Discrete-based task offloading and resource allocation (SATORA) algorithm, where SAC makes the TO decision and a KKT-based algorithm is employed to find the power and computation resource allocation solution. Note that from now on, we will use SAC to represent the SAC-Discrete for simplicity throughout the rest of the paper. The proposed algorithm is shown in Fig. 2, which consists of a KKT-based power and computational resource allocation algorithm (used for reward calculation), replay memory (store agent's experience), and five different neural networks (a policy network, two Q-networks, and two target Q-networks) that are used for action selection and evaluation. The actor or policy network approximates the policy, whereas critic networks (two Q-networks and target Q-networks) evaluate the policy by learning the Q-value function. The details of SATORA in Fig. 2 are given below.

At each time step, SAC takes an action a in state s, switches to the next state s', and obtains reward r. The action is chosen according to the probabilities given for each action by the actor. The actor or policy network uses a deep neural network (DNN) parameterized by θ that takes the state as input and outputs the probabilities for each possible action in a state. The agent's experience, i.e., s, a, s', r, is then stored in replay memory or buffer D, which acts as a dataset to train the DNNs in the SAC algorithm.

Next, a batch of transitions, B, is sampled from the replay memory to update the actor and critic networks. The actor-network is updated by minimizing the following loss function:

$$J_\pi(\theta) = \frac{1}{|B|} \sum_{l \in B} \big(\pi_\theta(s)^T [\alpha log\pi_\theta(s) - Q_\phi(s)]\big), \quad (29)$$

where $\pi_\theta(s)$ and $Q_\phi(s)$ are vectors containing probabilities and Q-values, respectively, for all possible actions. Q-values, $Q_\phi(s)$, are approximated by a parameterized DNN, called Q-network or critic, where ϕ denotes the DNN parameter. However, it is possible that the approximated Q-values are higher than the true Q-values in one or many states, which is likely to lead to a overestimation of future expected rewards, resulting in poor policy performance and propagation of estimation errors. Therefore, we use the clipped double-Q trick [35] to avoid the Q-value overestimation problem. Specifically, two different Q-networks, parameterized by ϕ_1 and ϕ_2, are trained to approximate the Q-values, and the minimum Q-values among them are used to update the actor network, which is as follows:

$$J_\pi(\theta) = \frac{1}{|B|} \sum_{l \in B} \big(\pi_\theta(s)^T [\alpha log\pi_\theta(s) - \min_{j=1,2} Q_{\phi_j}(s)]\big) \quad (30)$$

Each Q-network is updated by using the loss function given below:

$$J(\phi_j) = \frac{1}{|B|} \sum_{l \in B} \big(Q_{\phi_j}(s) - y(r, s', d)\big)^2, \forall j \in \{1, 2\}, \quad (31)$$

where $y(r, s', d)$ is the target Q-value, calculates as follows:

$$y(r, s', d) = r + \gamma(1 - d)\big(\pi_\theta(s)^T [\min_{j=1,2} Q_{\phi_{targ,j}}(s') - \alpha log\pi_\theta(s')]\big). \quad (32)$$

$Q_{\phi_{targ,j}}(s')$ represents the vector containing Q-values approximated using a target Q-network ($\phi_{targ,j}$ denotes the DNN parameter of the target Q-network). Similar to the Q-networks, two target Q-networks are used for target value approximations, $Q_{\phi_{targ,1}}(s')$ and $Q_{\phi_{targ,2}}(s')$, and the minimum between the two target Q-approximators is used to compute the target-value.

Algorithm 1 The SAC Based TORA Algorithm (SATORA)

```
1:  Training Phase:
2:  initialize policy/actor network $\theta$, and two Q-networks/critic networks $\phi_1$ and $\phi_2$
3:  initialize target Q-networks $\phi_{targ,1} \leftarrow \phi_1$ and $\phi_{targ,2} \leftarrow \phi_2$
4:  for each episode $e$ in $E$ do        ▷ $E$: Number of episodes
5:      initialize the environment setup
6:      initialize current state $s$
7:      for each step $t$ in $T$ do        ▷ $T$: one episode length
8:          select action $a \sim \pi_\theta(.|s)$
9:          execute $a$ in the environment
10:         observe next state $s'$, reward $r$, and done signal $d$ (indicate episode termination)
11:         store $(s, a, r, s', d)$ in replay buffer $D$
12:         set current state to $s'$
13:         sample a batch of transitions $B$ from replay buffer $D$
14:         compute targets using eq. (32)
15:         update Q-networks using loss function in eq. (31)
16:         update policy using loss function in eq. (30)
17:         update target networks by using eq.(33)
18:         update temperature/coefficient $\alpha$ by
                $J(\alpha) = \pi_\theta(s)^T[-\alpha log\pi_\theta(s) + \bar{H}]$
19:     end for
20: end for
21: Test Phase:
22: load the trained model
23: initialize the environment setup
24: repeat the operations in lines 8, 9, 10, and 12 for a given number of steps
25: apply simulated annealing and find the final TORA solution
```

Lastly, the target networks are updated at each step by using a soft updating method, which is given by

$$\phi_{targ,j} \leftarrow \rho\phi_{targ,j} + (1-\rho)\phi_j, \forall j \in \{1, 2\}, \tag{33}$$

where ρ is the soft update coefficient and $\rho \in [0, 1]$.

The pseudo-code of the SATORA is given in Algorithm 1. In the beginning, the parameters of the policy network, θ, two Q-networks, ϕ_1 and ϕ_2, and two target Q-networks, $\phi_{targ,1}$ and $\phi_{targ,2}$, are initialized in lines 2-3. Note that the policy network, target Q-networks, and Q-networks have the same DNN architecture with two fully connected hidden layers, one input layer, and one output layer, where both hidden layers consist of 200 units. Then, the agents start to play in the environment to learn the optimal policy for a given number of episodes. During each episode (Line 4), the simulation environment setup is initialized, where the location of UAVs, task sizes, task types, required GPU FLOPS, and deadlines are chosen randomly within a set range (Line 5).

Next, in line 8, for each step t, an action a is chosen randomly based on the given action probabilities obtained by feeding the current state into the policy network. Then, the agent executes the action in the environment, obtains reward r and next state s', and stores the observation in the replay memory (lines 9-11). After that, in line 13, a mini-batch of experiences is sampled from the replay memory. The mini-batch of transition taken from the replay memory is usually chosen randomly. However, all stored samples are not equally important for training. Furthermore, bad samples (contain states far from near-optimal or with deadline constraint violations) might be selected more frequently, which may lead to poor performance because of DNN learning the bad experiences. Therefore, in this paper, we put the weight of each sample in the memory according to the reward value and convert it to a probability (dividing each weight by the sum of all weights). If the reward is higher, the weight is higher as well as the probability, and then the samples are chosen randomly according to the probability. The actor network and Q-networks are then updated using the batch in lines 14-16. Lastly, the temperature parameter is updated in Line 18 by minimizing the loss function $J(\alpha)$. This procedure is repeated for a given number of steps in an episode.

Immediate reward $r(s, a, s')$ is obtained by solving the continuous problem, i.e., the RA problem. First, the RA problem is solved by using a KKT-based simple power and computational resource allocation method, called K-SPRA, from [23]. Specifically, in K-SPRA, at first, the closed-form solution of computation resource allocation when offloaded to the UMECs and BSCs is obtained by applying Karush-Kuhn-Tucker conditions, which are given in eqs. (34) and (35):

$$h_{im} = \frac{\sqrt{w_2 g^k}}{\sqrt{d_i}\frac{1}{h_m}\left(\sum_{i\in\mathbb{S}}\frac{\sqrt{w_2 g_i}}{\sqrt{d_i}}\right)} \tag{34}$$

$$u_{ig} = \frac{\sqrt{w_2 g_i}}{\sqrt{d_i}\frac{1}{z_g}\left(\sum_{i\in\mathbb{S}}\sum_{m\in\mathbb{M}}\frac{\sqrt{w_2 g_i}}{\sqrt{d_i}}\right)}. \tag{35}$$

Then, the transmission power allocation solution is derived by exploiting the correlation between the desired task processing latency and the actual task execution latency. Transmission power is assigned to the task of UAV scout i when transmitted to a UMEC (p_i) or to the BSC (o_{img}) according to the following equations:

$$p_i = \frac{\sigma}{w_{im}} \times \left(exp\left(\frac{s_i}{B\left(d_i - \frac{c_i}{b_{im}} - \frac{g_i}{h_{im}}\right)}\right) - 1\right) \tag{36}$$

$$o_{img} = \frac{\sigma_{mg}}{w_{mg}} \times \left(exp\left(\frac{s_i}{B\left(d_i - \frac{c_i}{z_{ig}} - \frac{g_i}{u_{ig}} - \frac{s_i}{r_{im}}\right)}\right) - 1\right), \tag{37}$$

Next, the energy consumption of each UAV scout and UMEC, the task execution latency of each task, and utility value U from equation (26) are obtained by using the RA solution, and the reward value is computed according to the eq. (27).

The SATORA algorithm has two phases: training and testing. In the training phase, the SATORA model is trained through trial and error in different environments. In the testing phase, the trained model is used to predict the solution in a new environment not seen before. In particular, the agent loads the trained model in Line 22, initializes the new environment in Line 23, then repeats lines 8, 9, 10, and 12 for a given number of steps, and obtains the final state. A local optimizer, i.e., simulated annealing (SA), is applied in Line 25 to find the final TORA solution, which takes the final state (i.e., the TO solution) as input. The solution representation in SA is the offloading decision matrix, and the fitness of the solution is calculated using the same equation as the reward, i.e., eq. (27).

V. PERFORMANCE ANALYSIS

In this section, the simulation setup is presented, and SATORA's performance is evaluated by comparing it with other methods under various simulation scenarios.

A. SIMULATION SETUP

The considered simulation scenario consists of a UAV network (MEC-enabled) comprising multiple UMECs, UAV scouts, and a BSC. Each UMEC has communication radius of 300 meters, whereas the coverage area of the BSC is 10 kilometers. The positions of UMECs and UAV scouts are selected randomly within a given range. Specifically, the x and y-coordinates are selected within a 500 × 500 m^2 area, and the position of the base station is set to 3 km away from this area. The height of the UMECs and UAV scouts are selected within the range of 80 to 100 m and 50 to 80 m, respectively. The channel gain between devices (UMEC and UAV scout, and UMEC and BSC) at the reference distance (one meter) is assumed to be equal (for simplicity) with values of $\sigma_{im} = \sigma_{mg} = -100$ dBm, and the noise variance is set to $\mu = -50$ dBm. The processing capacities (CPU and GPU) of the BSC are set to 30 GHz [22] and 34 TFLOPS (Nvidia GeForce RTX 3080 Ti), respectively, whereas for UMECs, it is 24 GHz (Intel Core i5-12400) and 20 TFLOPS (Nvidia GeForce RTX 3070), respectively, and for UAV scouts, it is 5 GHz (Quad-core ARM A57) and 472 GFLOPS (NVIDIA Maxwell), respectively. Energy consumption per CPU cycle is set to 2.708e-09 W and 1e-9 W for UMECs and UAV scouts, respectively, whereas per GPU FLOPS, it is set to 1.1e-11 W (UMECs) and 2.11e-11 W (UAV scouts), respectively. These values are derived from each processor's thermal design power (TDP). Note that kf^2 is the widely recognized metric for calculating per cycle CPU energy consumption, where f denotes the frequency of the CPU, and k is the energy coefficient based on the chip architecture. In most new GPUs and CPUs, the specific value of k is often unavailable. To address this issue, following a similar approach in [23] and [37], we opted to use TDP to estimate the energy consumption in both processing units. Although TDP might not measure the actual GPU and CPU energy usage, it provides a reasonable approximation for our purposes. Other environmental simulation parameters can be found in Table 2, with bold font denoting default values, whereas the hyperparameters of SATORA are presented in Table 3.

The simulations are conducted on a system consisting of an Intel Xeon W-2245 CPU @ 3.90GHz, coupled with an NVIDIA GeForce RTX 3080 Ti graphics card and 128 GB of RAM.

In order to validate the effectiveness of the SATORA, we compare the numerical results obtained from the simulations against the following three baselines:

- mGA-TPR [23]: This study employs a messy genetic algorithm (mGA) for TO determination and two distinct KKT-based methods for RA. In particular, first, a modified mGA is used to obtain the TO decision, incorporating a simple KKT-based RA algorithm to compute each solution's fitness value. Finally, given the TO decision by the mGA, a KKT and gradient

TABLE 2. Experiment settings.

Parameter	Value
Area (in meters)	500 × 500
Number of UAV scouts	{ 4, 6, **8**, 10}
Number of UMECs	4
Tasks desired execution latencies (in seconds)	[0.5, 1]
Maximum power (transmission) of UAV scout and UMEC (in dBm)	80 and 90
Data size of tasks (in KB)	[200, 300], [300, 400], **[400, 500]**, [500, 600], [600, 800]
Required GPU FLOPs (in GFLOPS)	[200, 300], [300, 400], [400, 500] **[500, 800]**, [800, 1000]
Total CPU cycles needed to complete the task (cycles per bit)	400
Bandwidth (in MHz)i	25
Energy capacity of UAV scouts and UMECs (in MJ)	2 and 150

TABLE 3. Hyperparameters of SATORA.

Parameter	Value
Total episodes	3000
Number of steps in an episode	$\lvert\mathbb{S}\rvert \times \lvert\mathbb{P}\rvert$
Mini-batch size	1500
Memory capacity	$1000(\lvert\mathbb{S}\rvert \times \lvert\mathbb{P}\rvert)$
Discount rate γ	0.99
Learning rate of critic and actor network	1e-4 and 1e-5
Optimizer	Adam

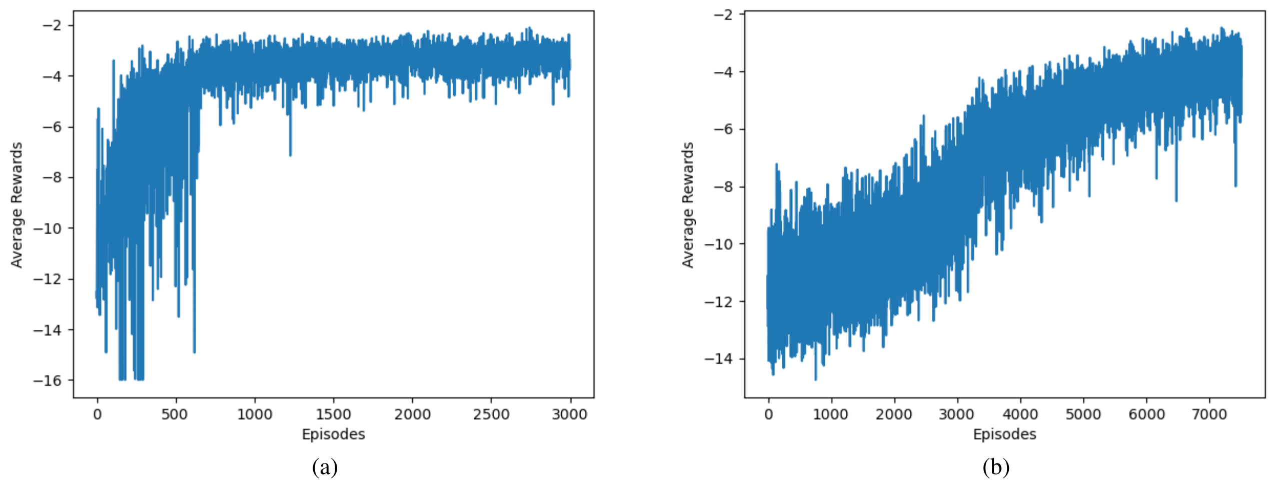


FIGURE 3. **Reward learning curves of SATORA and DQN-TORA along with the increases in the number of episodes: (a) SATORA and (b) DQN-TORA.**

descent-based RA strategy is applied, and the final RA solution is obtained.

- Greedy: In this approach, for each task, the processing device with the lowest utility is selected. The utility is the weighted sum of the total energy consumption in the network, the total task execution latency ratio, and the ratio of tasks that miss the deadline. The RA solution needed to compute the energy consumption and task execution latency is obtained using the same RA strategy used in SATORA.
- DQN-TORA: This approach is similar to SATORA. In DQN-TORA, the deep Q-network is employed to obtain the TO solution instead of SAC, whereas the reward of each state is calculated using the KKT-based power and resource allocation method called K-SPRA.

The results of the proposed and baseline algorithms are compared based on performance metrics such as network lifetime, maximum task execution latency ratio, total energy consumption in the network, and deadline hit ratio. The network lifetime ends when the first device (UAV scouts or UMECs) uses up its energy resource, and the description of the network lifetime is provided in more detail later in the result analysis subsection. The maximum task execution latency ratio is given by $\max_{i\in\mathbb{S}}\left\{\frac{T_i}{d_i}\right\}$, whereas the total energy consumption in the network is calculated as follows: $\sum_{i\in\mathbb{S}} O_i + \sum_{m\in\mathbb{M}} O_m$. Furthermore, we have also shown the total energy consumption of UAV scouts and UMECs separately by different optimization methods. The percentages of tasks processed by each algorithm and the time complexity are also discussed to compare the performance of the baselines and SATORA.

B. RESULT ANALYSIS

In this subsection, we discuss the numerical results of the simulations conducted with various parameters, including different numbers of UAV scouts, task sizes, and required GPU FLOPS. Note that we have trained four models for four different numbers of UAV scouts (4, 6, 8, and 10) by using 50 training environment scenarios. Then, the trained model for the default number of UAV scouts (i.e., 8) is used to collect results for other parameters, such as required GPU FLOPS and task sizes. Each result is collected after averaging over 20 test environment scenarios.

1) CONVERGENCE OF DIFFERENT DRL FRAMEWORK BASED TO STRATEGIES

Figure 3 depicts the learning curves for TO strategies utilizing DQN and SAC, illustrating their progression over increasing episodes of training. Figures 3 (a) and (b) display the learning curves for SATORA and DQN-TORA, respectively. In Figure 3 (a), the learning curves for SATORA exhibit higher fluctuations (rewards ranging between -16 and -2) that reduces after around 600 episodes, stabilizing with moderate improvement. Conversely, the peak-to-trough fluctuations for DQN-TORA start within a narrower range compared to SATORA, subsequently reducing with increasing episode count. This reduction in fluctuation signifies the algorithms' better action selection competence with increasing exposure to the environment.

However, even though DQN-TORA is trained for more than twice the episodes of SATORA, it does not show any sign of convergence. This is attributed to DQN's limited generalization capability to the new states when environments varies in each episode due to changes in UAV locations, task specifications, GPU FLOPS, and deadlines. Note that we trained SATORA for 3000 episodes to avoid overfitting, and chose 7500 episodes for DQN-TORA based on the performance after trying different numbers of episodes, whereas the number of steps in each episode is the same for both algorithms.

2) EFFECT OF DIFFERENT NUMBERS OF UAV SCOUTS

Figure 4 illustrates the impact of different numbers of UAV scouts on baselines (Greedy, mGA-TPR, and DQN-TORA) and SATORA.

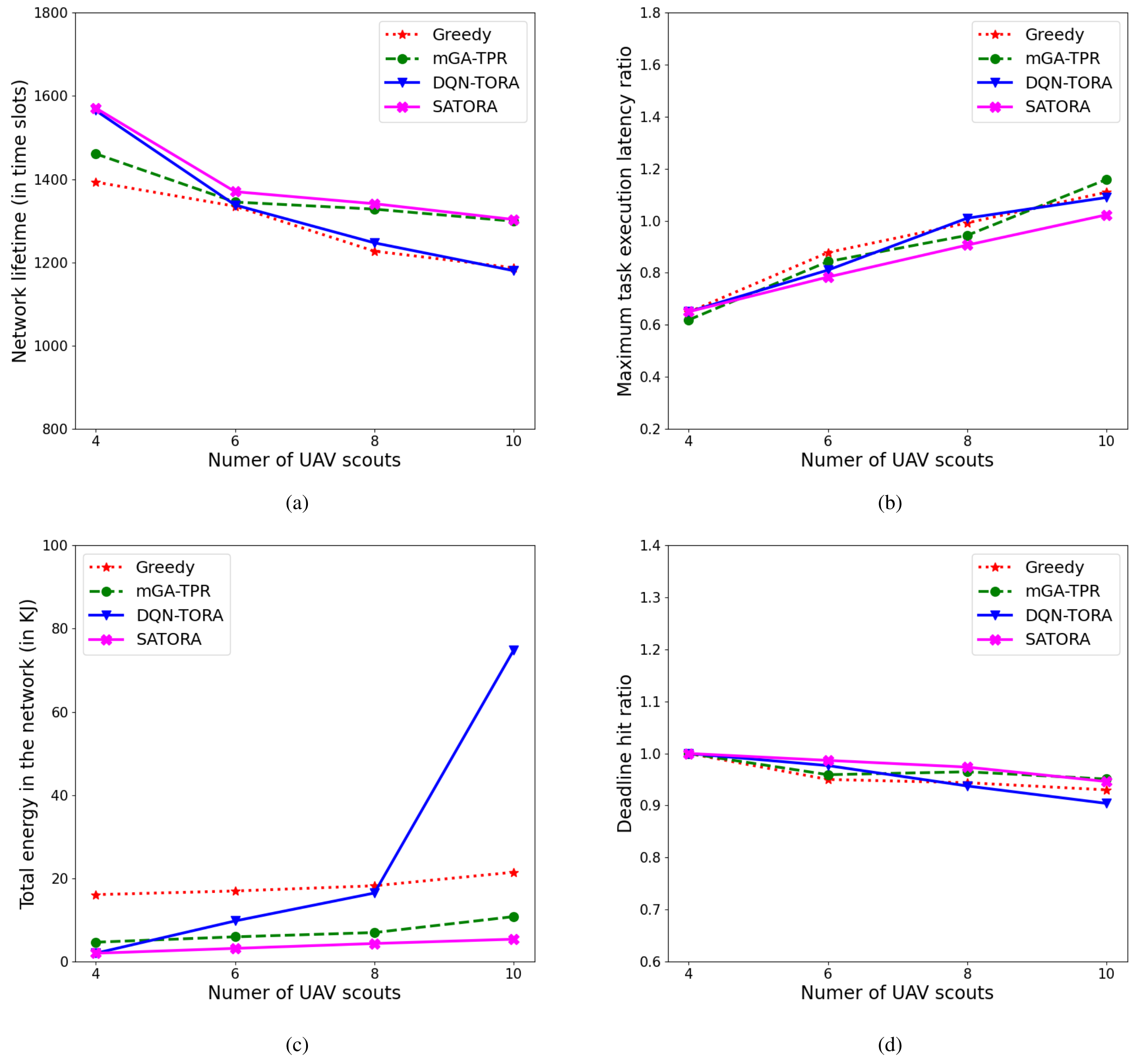


FIGURE 4. Effect of the different number of UAV scouts.

In fig. 4 (a), the network lifetimes (measured in time slots) obtained for varying numbers of UAV scouts under the four methods are depicted. The total operation time is assumed to be divided into time slots, and each time slot duration is 5 seconds. The simulation is run until the first UAV depletes its energy, and the total number of time slots it can survive is counted as the network lifetime. Furthermore, since this paper only focuses on task offloading energy consumption, the movement energy consumption of UAVs is calculated using specifications of some existing real-life UAVs. For each UAV scout, the movement energy consumption is assumed to be 92.77 J/s (calculated using battery specifications and the flight time of the DJI Mavic3). The energy consumption for hovering of each UMEC is assumed to be 18,125 J/s (calculated from the amount of fuel needed for the Sentinel Long-reach 70 UAV to hover for one hour and from the heat of combustion per kilogram of fuel [36]). We can see from the figure that network lifetimes reduce with a higher number of UAV scouts across all algorithms. This is because the number of tasks that need to be processed in the system increases with increments in the number of UAV scouts; thereby, maximum energy consumption increases and the network lifetime reduces.

Figure 4 (b) reveals the effect of UAV scouts on the maximum task execution latency ratio among all tasks. The higher the number of UAV scouts, the higher the number of tasks in the network, and the lower the computation resources assigned to each task. Therefore, the task execution latency increases.

The effect on the total energy consumption in the network is shown in Fig. 4 (c), where all four algorithms (SATORA, DQN-TORA, mGA-TPR, and Greedy) show an upward trend. This is due to the fact that more tasks in the system lead to higher transmission and execution energy consumption, thus the increase in the total energy consumption. Moreover, it can be seen that there is a sudden sharp increase in energy consumption obtained by the DQN-TORA when the number of UAV scouts is 10. This is because, with 10 UAV scouts, the environment becomes more constrained, which contains more infeasible solutions or solutions with lower rewards than the feasible or high-reward solutions. When the complexity of the environment increases, DQN-TORA fails to learn the new dynamics or complex patterns accurately, resulting in poor-quality generalization. However, SATORA can still generalize well and can obtain the lowest energy consumption without a show drastic change in performance similar to the DQN-TORA.

The deadline hit ratio, i.e., the ratio of tasks that are executed within the desired execution latency for each method under different numbers of UAV scouts, is presented in Fig 4 (d). When the number of tasks increases, the opportunity to select the processing devices that can compute the tasks within the desired execution latencies decreases. Thus, it is expected that the deadline-hit ratio will decrease along with the higher number of tasks.

Furthermore, it can be seen from the figures (fig. 4 (a), (b), (c), and (d)) that the proposed SATORA outperforms the other three approaches, whereas DQN-TORA, mGA-TPR, and Greedy outperforms each other in different scenarios. For instance, with eight UAV scouts, the total energy consumed by SATORA is 4.39 KJ. In contrast, DQN-TORA, mGA-TPR, and Greedy consume 34.96 KJ, 7.12 KJ, and 18.24 KJ respectively, indicating 696.3%, 62.1%, and 315.4% higher consumption than SATORA. This is because SATORA employs the SAC algorithm, incorporating distinct DNNs for policy and value approximation, double-Q trick to avoid overestimation, prioritized experience replay to train the model, and follows stochastic policies coupled with entropy regularization that encourages more effective and adaptive exploration, which enables it to discover better task offloading decision. On the other hand, DQN uses epsilon-greedy policies and suffers from overestimation bias, leading to sub-optimal performance. The mGA-TPR is an evolutionary algorithm that partially depends on random evolution and suffers from a loss of genetic diversity, resulting in early convergence. Greedy's step-wise decision-making yields suboptimal solutions as it prioritizes immediate gains without considering broader optimization.

3) EFFECT OF DIFFERENT TASK SIZES

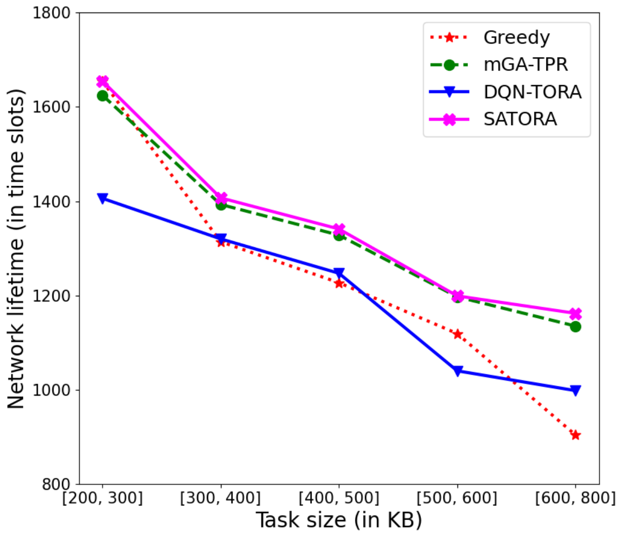
The impacts of varied task size ranges on network lifetime, maximum task execution latency ratio, total energy consumption, and deadline hit ratio is illustrated in Fig. 5. With an increase in the task size, each task needs more transmission power when offloaded, and needs more computation resources for execution, which reduces the resources allocated to tasks. Therefore, along with the larger task sizes, network lifetime decreases, the maximum task execution latency ratio increases, total energy consumption increases, and the deadline hit ratio decreases, which are shown in Figs. 5 (a)-(d), respectively.

It is observable from the figures that in most cases, SATORA and mGA-TPR have better performance than DQN-TORA and Greedy, whereas among them (SATORA and mGA-TPR), SATORA achieves the best performances in terms of network lifetime, maximum task execution latency ratio, total energy consumption, and deadline hit ratio. For instance, when the task size is between 600-800 KB, the worst network lifetime is achieved by the Greedy algorithm (904 time slot), which is 10.3%, 25.5%, and 28.5% higher than DQN-TORA, mGA-TPR, and SATORA, respectively. Furthermore, when the task range is 600-800 KB, Greedy consumed the most total energy among all four algorithms, whereas maximum task execution latency is obtained by mGA-TPR (possibly because of falling into local optima), and DQN-TORA obtains the lowest deadline hit ratio.

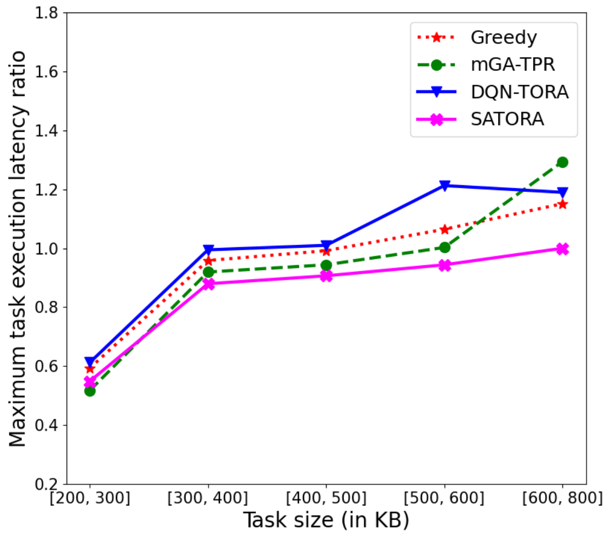
In addition, along with task size increments, network lifetime sharply decreases in all four algorithms, whereas the maximum task execution latency ratio sharply increases initially before a comparatively moderate increment due to the different levels of constrained (loosely or highly constrained) environment. However, although total energy consumption by DQN-TORA and Greedy sharply increases with task sizes, SATORA and mGA-TPR show very moderate increments, which shows their effective task offloading decisions. Moreover, as the task size increases, the performance gaps of DQN-TORA and Greedy with SATORA and mGA-TPR also increases in all four performance metrics.

4) EFFECT OF DIFFERENT RANGES OF REQUIRED GPU FLOPS

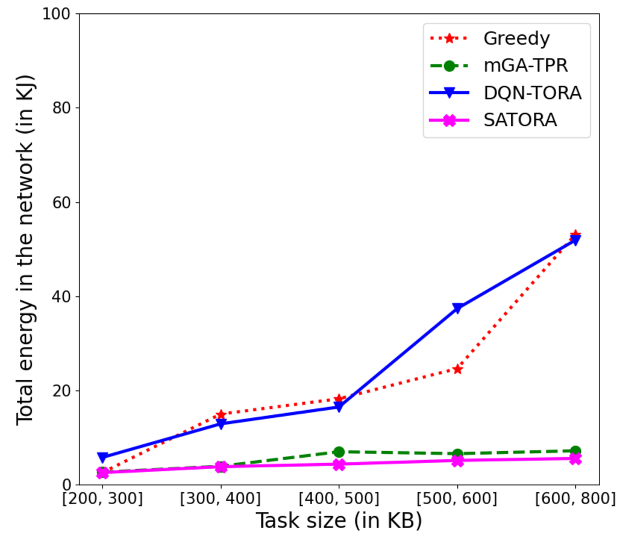
The impacts of different required GPU FLOPS on SATORA, DQN-TORA, mGA-TPR, and Greedy in terms of network lifetime, task execution latency ratio, and deadline hit ratio respectively, are shown in Figs. 6(a)-(c). When tasks require a higher amount of GPU FLOPS, additional computational resources are necessary for each one. This leads to a reduction in the GPU resources allocated per task, and limits the opportunity to select better processing devices. As a consequence, both computation time and energy consumption rise. It is observable from the Figs. 6(b) and (c) that along with the higher number of required GPU FLOPS, the task execution latency ratio of all algorithms increase, whereas the deadline hit ratio decreases. However, in Fig.6 (a), the network lifetime obtained by SATORA and mGA-TORA shows an up-and-down trend, whereas although the DQN-TORA and Greedy show a decreasing trend, it is not smooth. Specifically, when the required GPU FLOPS changes from 200 to 500 GPU FLOPS, all four algorithms show an up-and-down trend, possibly due to falling into local optima. However, when the required GPU FLOPS changes from 800 to 1000 GLOPS, although the network lifetime achieved by DQN-TORA, mGA-TPR, and Greedy decreases, it increases in SATORA. This is due to the following reasons. When the required GPU



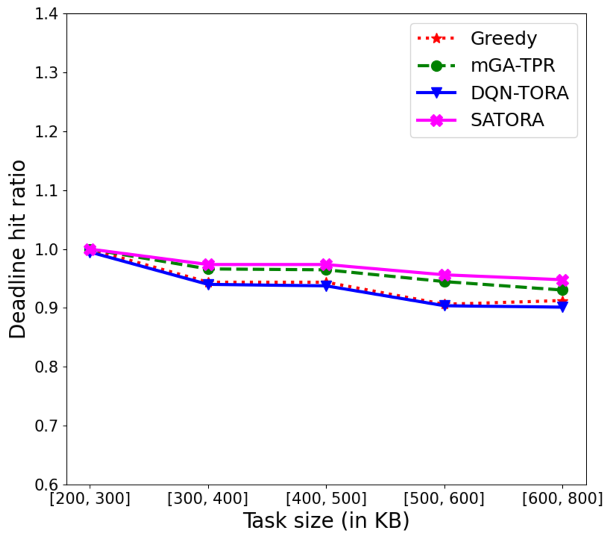
(a)



(b)



(c)



(d)

FIGURE 5. Effect of the different task sizes.

TABLE 4. Percentages of tasks processed by each type of device.

Required GPU FLOPS	Algorithm											
	SATORA			DQN-TORA			mGA-TPR			Greedy		
	Device type			Device type			Device type			Device type		
	US	UMEC	BSC	US	UMEC	BSC	US	UMEC	BSC	US	UMEC	BSC
[200-300]	1.25	98.75	0	2.125	97.625	0.25	0.463	99.512	0.025	0	100	0
[300-400]	0.125	99.875	0	0.75	98.875	0.375	0.0375	99.9375	0.025	0	100	0
[400-500]	0	100	0	0.375	99.125	0.5	0.0625	99.9	0.0375	0	99.375	0.625
[500-800]	0	100	0	0.375	99.125	0.5	0.125	99.75	0.125	0	99.375	0.625
[800-1000]	1.25	98.75	0	1.375	97.25	1.5	1.2875	98.45	0.2625	0	98.125	1.875

FLOPS is high, the environment becomes very constrained, and it is hard to find processing devices for tasks that satisfy the deadlines and have lower task execution latency, even at the cost of higher energy consumption. Therefore, SATORA finds a task offloading decision that has lower maximum energy consumption because of the trade-off between energy consumption, task execution time, and deadline missing ratio.

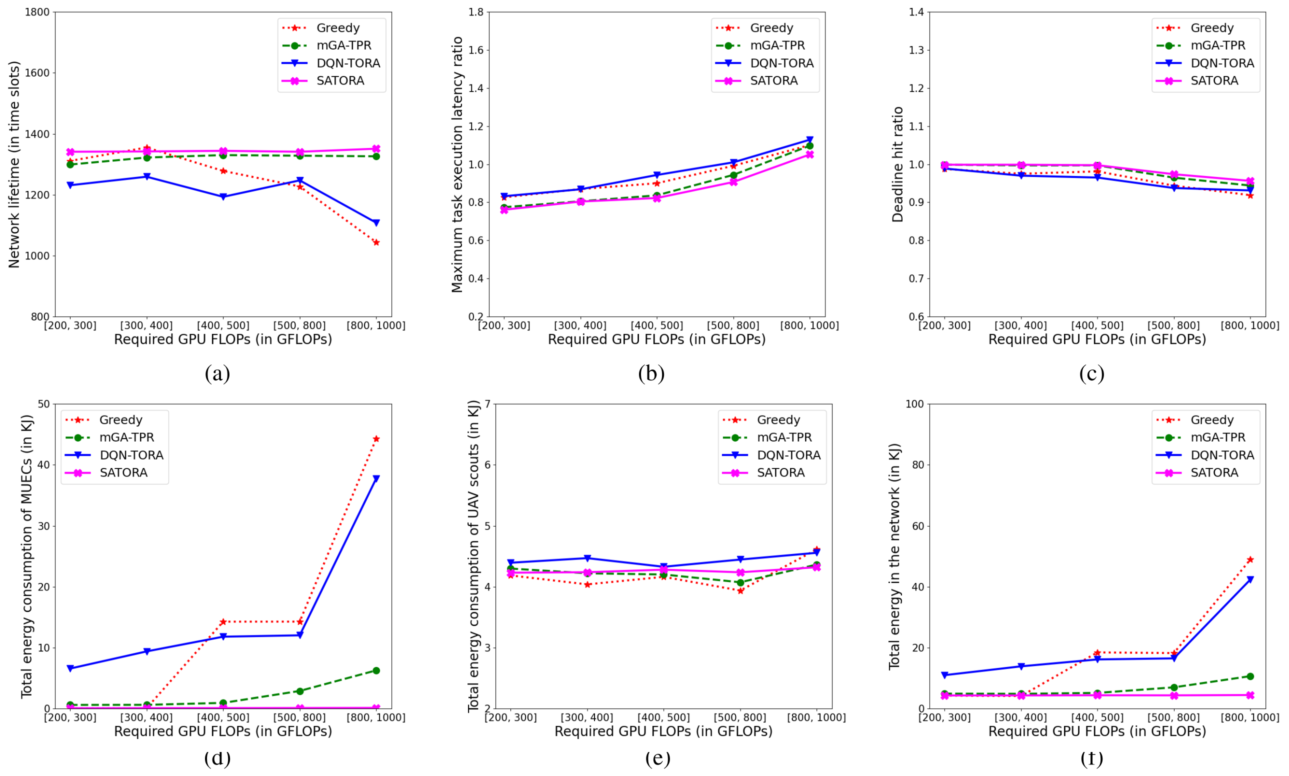


FIGURE 6. Effect of the different required GPU FLOPS.

TABLE 5. Time complexity (in seconds).

No. of UAV scouts	Algorithm			
	Greedy	SATORA	DQN-TORA	mGA-TPR
4	0.026	0.048	0.079	0.049
6	0.049	0.059	0.092	0.062
8	0.0637	0.071	0.125	0.129
10	0.0912	0.082	0.144	0.162

Total energy consumption of UMECs, UAV scouts, and the network is shown in Figs. 6(d)-(f), respectively. From Fig. 6(d), it can be seen that for UMECs, SATORA consumes the least energy, whereas the highest energy is consumed by DQN-TORA (when the required GPU FLOPs changes from 200 to 400 GFLOPS) and the Greedy (when the required GPU FLOPS ranges from 400 to 1000 GFLOPS). In Fig. 6(e), all algorithms outperform each other in different situations, i.e., for UAV scouts, the highest and lowest energy consumption is achieved by the different algorithms in different situations. However, in the case of the overall energy consumption of the network, illustrated in Fig. 6(f), SATORA outperforms all other algorithms, whereas Greedy shows the worst performance when the required GPU FLOPs increase. This is because SATORA mainly chooses to process the tasks in the UMECs (more than 98% of thetasks) and locally (less than 2% of the tasks), as presented in Table 4. Table 4 shows the percentage of tasks performed in the UAV scouts (US), UMECs, and the BSC by each algorithm. On the contrary, it can be seen from the Table 4 that Greedy offloads all its tasks to the UMECs and the BSC, whereas DQN-TORA and mGA-TPR employ all three types of processing device (i.e., local, UMEC, and BSC). Although a very small percentage of tasks is processed in the BSC by the algorithms, processing in the BSC consumes much higher energy (i.e., transmission energy) than in the UMECs and locally because of the distance between the UAV scouts and the BSC. Therefore, the energy consumption in SATORA is the lowest among all the methods.

Moreover, our proposed method, SATORA achieves the best performance across a majority of performance metrics, whereas DQN-TORA exhibits the least favorable performance in most instances. For example, when GPU FLOPS requirements range between 400 and 500, SATORA achieves network lifetime, maximum task execution latency ratio, total energy consumption in the network, and deadline hit ratio of 1344 time slots, 0.823, 4.29 KJ, and 0.9975, respectively. In comparison, mGA-TPR, Greedy, and DQN-TORA attain network lifetimes of 1330 time slots, 1278 time slots, and 1193 time slots, respectively; obtain task execution latency ratio of 0.836, 0.90, and 0.943, respectively; consume total network energy of 5.12 KJ, 18.4 KJ, and 17.1 KJ, respectively; and achieve deadline hit ratio of 0.997, 0.98, and 0.965, respectively.

5) TIME COMPLEXITY

Each algorithm's time complexity is illustrated in table 5 for different numbers of UAV scouts. It is observable that with an increment in the number of UAV scouts, the computational time also increases as a result of the exponential expansion of the problem size and solution space. However, in comparison to other algorithms (except Greedy), SATORA takes the shortest execution time when the number of UAV scouts rises to 10 from 4. In contrast, DQN-TORA needs the longest time for finding the solution in case of lower number of UAV scouts, whereas mGA-TPR needs the most time among all four algorithms when the number of UAV scouts increases.

VI. CONCLUSION

In this paper, a TORA problem consisting of different types of UAVs for emergency operations has been studied. In particular, we considered a UAV network (which is MEC-assisted and hierarchical), where UAV scouts generate tasks, and MEC servers equipped UAVs and a BSC provide computing services to UAV scouts. Each task in the network can have heterogeneous specifications, each UAV has a power budget and energy capacity limit, each UMEC has a specific number of tasks it is capable of performing, and each device (UAVs and BSC) has a computational resource budget. Each UAV scout can perform its task locally or offload it to either a UMEC or the BSC. Moreover, the considered problem was formulated as a MINLP problem with the objective of minimizing the weighted sum of the maximum task execution latency ratio, the maximum energy consumption ratio, and total energy consumption. Next, a decomposition technique was employed to decompose the problem into a TO and an RA problem that enables solving the considered problem using multiple strategies. Furthermore, it is assumed that the UAV locations and task specifications are time-varying. Therefore, in this paper, a SAC-based approach called SATORA was proposed to find the TORA solution in a dynamic environment. Simulation results showed that SATORA can achieve a higher network lifetime, a lower task execution latency ratio and total energy, and a higher deadline hit ratio with a shorter run time than other comparing methods.

REFERENCES

[1] ReliefWeb. (2017). *Natural Disasters 2017*. [Online]. Available: https://reliefweb.int/report/world/natural-disasters-2017

[2] Y.-J. Zheng, Q.-Z. Chen, H.-F. Ling, and J.-Y. Xue, "Rescue wings: Mobile computing and active services support for disaster rescue," *IEEE Trans. Services Comput.*, vol. 9, no. 4, pp. 594–607, Jul. 2016, doi: 10.1109/TSC.2015.2401598.

[3] A. Masood, D. Scazzoli, N. Sharma, Y. L. Moullec, R. Ahmad, L. Reggiani, M. Magarini, and M. M. Alam, "Surveying pervasive public safety communication technologies in the context of terrorist attacks," *Phys. Commun.*, vol. 41, Aug. 2020, Art. no. 101109.

[4] Y. Wang, Z. Su, N. Zhang, and D. Fang, "Disaster relief wireless networks: Challenges and solutions," *IEEE Wireless Commun.*, vol. 28, no. 5, pp. 148–155, Oct. 2021, doi: 10.1109/MWC.101.2000518.

[5] M. Aljehani and M. Inoue, "Performance evaluation of multi-UAV system in post-disaster application: Validated by HITL simulator," *IEEE Access*, vol. 7, pp. 64386–64400, 2019.

[6] S. Yin, Y. Zhao, and L. Li, "UAV-assisted cooperative communications with time-sharing SWIPT," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[7] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[8] M. T. Beck, M. Werner, S. Feld, and T. Schimper, "Mobile edge computing: A taxonomy," in *Proc. 6th Int. Conf. Adv. Future Internet*, 2014, pp. 48–54.

[9] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[10] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3435–3447, Apr. 2017.

[11] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018, doi: 10.1109/JSAC.2018.2815360.

[12] Z. Song, Y. Liu, and X. Sun, "Joint task offloading and resource allocation for NOMA-enabled multi-access mobile edge computing," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1548–1564, Mar. 2021, doi: 10.1109/TCOMM.2020.3044085.

[13] L. Tan, Z. Kuang, L. Zhao, and A. Liu, "Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1960–1972, Mar. 2022, doi: 10.1109/TWC.2021.3108641.

[14] W. Chu, P. Yu, Z. Yu, J. C. S. Lui, and Y. Lin, "Online optimal service selection, resource allocation and task offloading for multi-access edge computing: A utility-based approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4150–4167, Jul. 2022.

[15] H. Li, H. Xu, C. Zhou, X. Lü, and Z. Han, "Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 10214–10226, Sep. 2020, doi: 10.1109/TVT.2020.3003898.

[16] X. Wei, C. Tang, J. Fan, and S. Subramaniam, "Joint optimization of energy consumption and delay in cloud-to-thing continuum," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2325–2337, Apr. 2019, doi: 10.1109/JIOT.2019.2906287.

[17] S. Mao, S. He, and J. Wu, "Joint UAV position optimization and resource scheduling in space-air-ground integrated networks with mixed cloud-edge computing," *IEEE Syst. J.*, vol. 15, no. 3, pp. 3992–4002, Sep. 2021, doi: 10.1109/JSYST.2020.3041706.

[18] Y. Luo, W. Ding, and B. Zhang, "Optimization of task scheduling and dynamic service strategy for multi-UAV-enabled mobile-edge computing system," *IEEE Trans. Cognit. Commun. Netw.*, vol. 7, no. 3, pp. 970–984, Sep. 2021, doi: 10.1109/TCCN.2021.3051947.

[19] C. Zhan, H. Hu, X. Sui, Z. Liu, and D. Niyato, "Completion time and energy optimization in the UAV-enabled mobile-edge computing system," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7808–7822, Aug. 2020.

[20] M.-A. Messous, S.-M. Senouci, H. Sedjelmaci, and S. Cherkaoui, "A game theory based efficient computation offloading in an UAV network," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4964–4974, May 2019, doi: 10.1109/TVT.2019.2902318.

[21] J. Chen, S. Chen, S. Luo, Q. Wang, B. Cao, and X. Li, "An intelligent task offloading algorithm (ITOA) for UAV edge computing network," *Digit. Commun. Netw.*, vol. 6, no. 4, pp. 433–443, Nov. 2020.

[22] A. A. Ashraf Ateya, A. Muthanna, R. Kirichek, M. Hammoudeh, and A. Koucheryavy, "Energy- and latency-aware hybrid offloading algorithm for UAVs," *IEEE Access*, vol. 7, pp. 37587–37600, 2019, doi: 10.1109/ACCESS.2019.2905249.

[23] S. Akter, D.-Y. Kim, and S. Yoon, "Task offloading in multi-access edge computing enabled UAV-aided emergency response operations," *IEEE Access*, vol. 11, pp. 23167–23188, 2023, doi: 10.1109/ACCESS.2023.3252575.

[24] A. M. Seid, G. O. Boateng, S. Anokye, T. Kwantwi, G. Sun, and G. Liu, "Collaborative computation offloading and resource allocation in multi-UAV-assisted IoT networks: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12203–12218, Aug. 2021, doi: 10.1109/JIOT.2021.3063188.

[25] A. R. Heidarpour, M. R. Heidarpour, M. Ardakani, C. Tellambura, and M. Uysal, "Soft actor–critic-based computation offloading in multiuser MEC-enabled IoT—A lifetime maximization perspective," *IEEE Internet Things J.*, vol. 10, no. 20, pp. 17571–17584, Oct. 2023, doi: 10.1109/JIOT.2023.3277753.

[26] H. Lu, C. Gu, F. Luo, W. Ding, S. Zheng, and Y. Shen, "Optimization of task offloading strategy for mobile edge computing based on multi-agent deep reinforcement learning," *IEEE Access*, vol. 8, pp. 202573–202584, 2020, doi: 10.1109/ACCESS.2020.3036416.

[27] F. Zhang, G. Han, L. Liu, M. Martinez-Garcia, and Y. Peng, "Deep reinforcement learning based cooperative partial task offloading and resource allocation for IIoT applications," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 5, pp. 2991–3006, Aug. 2022.

[28] C. Sun, X. Wu, X. Li, Q. Fan, J. Wen, and V. C. M. Leung, "Cooperative computation offloading for multi-access edge computing in 6G mobile networks via soft actor critic," *IEEE Trans. Netw. Sci. Eng.*, early access, p. 1, Apr. 2021, doi: 10.1109/TNSE.2021.3076795.

[29] C. Shang, Y. Sun, H. Luo, and M. Guizani, "Computation offloading and resource allocation in NOMA-MEC: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 10, no. 17, pp. 15464–15476, Sep. 2023, doi: 10.1109/JIOT.2023.3264206.

[30] L. Özbakir, A. Baykasoglu, and P. Tapkan, "Bees algorithm for generalized assignment problem," *Appl. Math. Comput.*, vol. 215, no. 11, pp. 3782–3795, Feb. 2010.

[31] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1856–1865.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[33] M. Lapan, *Deep Reinforcement Learning Hands-on*. Birmingham, U.K.: Packt publishing, 2020.

[34] P. Christodoulou, "Soft actor-critic for discrete action settings," 2019, *arXiv:1910.07207*.

[35] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, *arXiv:1802.09477*.

[36] T. Zhang, G. Liu, H. Zhang, W. Kang, G. K. Karagiannidis, and A. Nallanathan, "Energy-efficient resource allocation and trajectory design for UAV relaying systems," *IEEE Trans. Commun.*, vol. 68, no. 10, pp. 6483–6498, Oct. 2020, doi: 10.1109/TCOMM.2020.3009153.

[37] T. Do, S. Rawshdeh, and W. Shi, "PTop: A process-level power profiling tool," in *Proc. 2nd Workshop Power Aware Comput. Syst.*, 2009, pp. 1–5.

SHATHEE AKTER received the M.Sc. degree in computer engineering from the University of Ulsan, South Korea, in 2020, where she is currently pursuing the Ph.D. degree. Her current research interests include mobile crowd-sensing, optimization algorithms, multi-access edge computing, the Internet of Things, and deep reinforcement learning.

DAT VAN ANH DUONG received the B.S. degree in electronics and telecommunications engineering from Hanoi University of Science and Technology, Vietnam, in 2015, and the Ph.D. degree in computer engineering from the University of Ulsan, South Korea, in 2022. After the Ph.D. studies, he was a Postdoctoral Researcher with the University of Ulsan, where he has been a Research Professor, since 2024. His research interests include human mobility, opportunistic networks, deep learning-based stock market strategy, and UAV-based networks.

DAE-YOUNG KIM (Member, IEEE) received the B.E. degree in electronics engineering and the M.S. and Ph.D. degrees in computer engineering from Kyung Hee University, South Korea, in 2004, 2006, and 2010, respectively. From 2010 to 2013, he was a Research Staff with the Communication Research and Development Laboratory, LIG Nex1 Company Ltd., South Korea, and a Research Staff with AirPlug Inc., South Korea, from 2013 to 2015. Since 2015, he has been an Assistant Professor with the Department of Software Engineering, Changshin University, South Korea. Since 2017, he has also been an Assistant Professor with the School of Computer Software, Daegu Catholic University, South Korea. He is currently an Assistant Professor with the Department of Computer Software Engineering, Soonchunhyang University, South Korea. His research interests include mobile networking and computing, intelligent systems, the IoT services, and machine learning for network systems.

SEOKHOON YOON (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science and engineering from The State University of New York at Buffalo (SUNY Buffalo), in 2005 and 2009, respectively. After the Ph.D. studies, he was a Senior Research Engineer with the defense industry, where he designed several tactical wireless network solutions. He is currently a Professor with the University of Ulsan, South Korea, where he leads the Advanced Mobile Networks and Intelligent Systems Laboratory. His research interests include opportunistic networking, human mobility, intelligence-defined networking, and machine learning-based IoT services.