



AM-DQN: Action-Masked Deep Q-Network for Task Offloading and Resource Allocation in Mobile Edge Computing

Le Van Hoa ^a, Pham Trong Nguyen ^{b,c}, Vo Viet Minh Nhat ^d,*

^a School of Hospitality and Tourism – Hue University, Viet Nam

^b University of Sciences, Hue University, Viet Nam

^c Saigon University, Viet Nam

^d Department of Academic and Students' Affairs – Hue University, Viet Nam

ARTICLE INFO

Keywords:

Mobile Edge Computing
Task offloading
Resource allocation
Deep reinforcement learning
Action masking

ABSTRACT

Mobile Edge Computing (MEC) is envisioned to help reduce latency and energy overhead incurred by resource-limited IoT devices. The joint Task Offloading and Resource Allocation (TORA) problem has been shown to be challenging to solve due to time-varying constraints, such as computational capacity, remaining energy, maximum latency, connectivity, etc. Traditional Deep Reinforcement Learning (DRL) methods select actions during training through blind exploration, which can lead to feasibility violations. This paper presents a feasibility-aware DRL framework, called Action-Masked Deep Q-Network (AM-DQN), for tackling the TORA problem. A dynamic action-masking mechanism is introduced to the DQN model, which masks the infeasible actions during the agent's decision-making process. A multi-objective reward function is also designed to achieve a trade-off between latency minimization and energy saving. Simulation results show that the AM-DQN model outperforms the traditional DQN and heuristic baselines, achieving reductions of 8.81% in energy consumption, 6.98% in latency, and 5.31% in task failure rate.

1. Introduction

The exponential proliferation of mobile applications requiring computation-intensive and low-latency operations — such as Augmented Reality/Virtual Reality (AR/VR), real-time video streaming, and autonomous Internet of Things (IoT) — has posed significant challenges due to the limited computational capacity and energy of user devices [1]. Traditional cloud computing, characterized by its centralized processing architecture, often fails to meet real-time requirements due to transmission latency and backhaul congestion. To mitigate these issues, edge computing, notably Mobile Edge Computing (MEC), has been proposed as a practical solution to bring computation and storage resources closer to edge user devices. Edge computing enables user devices to offload tasks to edge servers for execution, rather than to remote cloud servers, thereby significantly improving the Quality of Experience (QoE) [2].

The typical architecture of an MEC system comprises two main layers: the user device layer and the edge layer, in which edge servers are collocated with base stations, as illustrated in Fig. 1. Data exchange occurs via wireless connections between user devices and base stations. Within the scope of this study, to focus on the optimization of computational resource allocation, we consider a quasi-static communication environment where small-scale signal impairments are managed by physical-layer protocols.

* Corresponding author.

E-mail addresses: levanhhoa@hueuni.edu.vn (L. Van Hoa), ptnguyen.dhkh25@hueuni.edu.vn (P.T. Nguyen), vvmnhat@hueuni.edu.vn (V.V.M. Nhat).

<https://doi.org/10.1016/j.pmcj.2026.102273>

Received 3 May 2026; Received in revised form 30 June 2026; Accepted 26 July 2026

Available online 31 July 2026

1574-1192/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

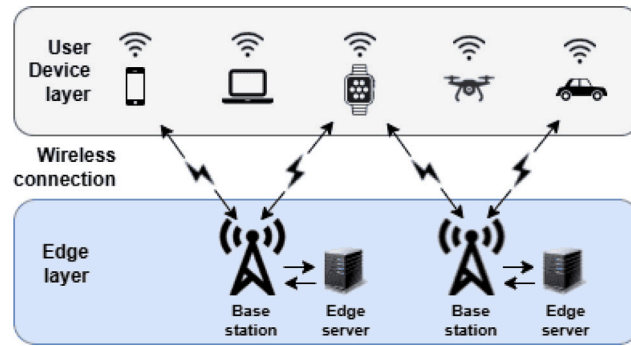


Fig. 1. The typical architecture of an MEC system.

In an MEC-enabled IoT system, tasks originating from user devices are highly diverse, including camera image processing, environmental data collection, interactions in AR/VR, etc. These tasks can be processed locally at the user device or offloaded to an edge server for execution, particularly when the user device has limited computational capabilities. The problem of optimizing decision-making (local processing vs. offloading) within an MEC system, subject to various constraints such as connectivity, latency, energy, and server capacity, is known as the Joint Task Offloading and Resource Allocation (TORA). Given that TORA is inherently NP-hard, learning-based and metaheuristic approaches are often preferred to find efficient solutions, serving as alternatives to traditional greedy or brute-force approaches [3].

Recently, Deep Reinforcement Learning (DRL), particularly Deep Q-Networks (DQN), has emerged as a promising approach for the TORA problem, owing to its ability to adapt flexibly to dynamic and uncertain network environments [4,5]. However, a significant limitation of most existing DQN studies is the blind exploration of the solution space and the insufficient consideration of multiple constraints in real-world MEC systems. By neglecting these constraints during the exploration phase, the DRL agent sometimes selects invalid or infeasible actions, thereby degrading the system's reliability and operational efficiency during actual deployment.

This study proposes an Action-Masked Deep Q-Network (AM-DQN) model, an improvement over the traditional DQN that incorporates an Action Masking (AM) mechanism. Through the masking operation, invalid or infeasible actions are dynamically pruned from the action space. Furthermore, a multi-objective reward function is designed to optimize the trade-off between processing latency and energy consumption. These improvements not only ensure the model adheres to the practical constraints of MEC systems but also significantly enhance learning stability and convergence speed compared to the traditional DQN.

In summary, the contributions of this research are as follows:

- The TORA problem is formulated as a constrained Markov Decision Process (MDP) that considers user device, task, and server heterogeneity, where state transitions occur due to variations in task profile, connection matrix, remaining energy, and queue load.
- An Action Masking (AM) mechanism is proposed for integration with the traditional DQN (resulting in AM-DQN) to proactively mask invalid and infeasible actions arising from system resource fluctuations. The masking is performed before an action is selected, thereby eliminating invalid or infeasible actions during training. As a result, the AM-DQN model avoids blind exploration and achieves faster convergence than the traditional DQN.
- A reward function is designed to balance latency minimization and energy saving. The reward function penalizes invalid or infeasible actions to ensure that the learned policy remains applicable to practical IoT systems.
- Simulation results show that the AM-DQN model significantly outperforms the traditional DQN, reducing energy consumption by 8.81%, latency by 6.98%, and task failure rate by 5.31%.
- Finally, the proposed action masking mechanism demonstrated significant improvement in quality of service (QoS) with minimal runtime and is thus suitable for deployment at the edge-tier.

The remainder of this paper is organized as follows. Section 2 reviews the related works. Section 3 describes the TORA problem formulation. Section 4 details the proposed AM-DQN model with action masking. Section 5 provides simulation results and discussion. Finally, Section 6 concludes this paper and presents future work.

2. Related work

Mobile Edge Computing (MEC) is a promising computing architecture that deploys computation and storage capabilities in proximity to the network edge. It aims to reduce the transmission latency and workload from the remote cloud. One representative problem studied in this context is Task Offloading and Resource Allocation (TORA), which is often formulated as a multi-objective optimization problem for joint latency-energy minimization [1,2].

Existing works on the TORA problem often adopted conventional optimization methods such as convex optimization models [3], multi-user computation offloading models [6], or joint radio-computational resource allocation models [7]. While yielding analytical solutions and showing optimality in small-scale static systems, such traditional approaches are often invalid in the MEC context due to uncertainties introduced by wireless channels, time-varying edge server queue loads, and dynamics of task arrivals [1].

To this end, recent works utilize DRL methods to train agents. By building upon the DQN framework [8], agents are able to obtain near-optimal policies by interacting with the environment over time. In addition, since model-free RL does not require an explicit characterization of the underlying state transition probability model, this kind of algorithm can be well-suited for high-dimensional dynamic environments [5,9,10].

Literature has also made efforts to scale these DRL models to more practical heterogeneous settings. Examples include vehicular MEC networks [11,12] which deal with high-mobility scenarios and fast-varying wireless channels, energy harvesting IoT devices [10] which introduce hard constraints on the availability of power, cooperative models with multiple users and cloud-edge collaboration [13,14] which significantly scale up the modeling complexity. However, most existing works make independence assumptions on the offloading decision for computational resources and offloaded data, thereby reducing offloading into a pure action-selection problem. These works often ignore or relax constraints that enforce strict feasibility of actions under operational settings.

Namely, traditional DQN models would encourage exploration by allowing invalid or infeasible actions, such as offloading task to an overloaded server or one without any established link. Such actions slow down training procedures and cause unstable policies. This problem falls into a category of constrained Markov decision processes [15]. Although constrained RL works have often approached the problem with Lagrangian multipliers/policies [16], tuning these hyper-parameters can be tedious and on-policy constraint violations.

However, current DRL-based TORA algorithms suffer from some limitations when implemented in practical MEC environments. Firstly, most prior research focuses on minimizing latency and energy consumption while neglecting or relaxing crucial system constraints. Secondly, blind exploration over the action space incurs additional computation cost, slow down convergence speed and may select those actions invalid or infeasible under constraints such as computing power, remaining energy, queue capacity and completion deadline.

Action masking shows the potential for avoiding the invalid or infeasible actions and expediting the training process in RL [17,18]. Nonetheless, its applicability to TORA problems in MEC has been rarely addressed in the literature. Therefore, this paper developed an Action Masking Deep Q-Network (AM-DQN) framework to make the offloading decisions in dynamic MEC systems. Instead of developing new DRL algorithms, the proposed model augments the original DQN model through adding a feasibility-aware action masking mechanism prior to the action selection. The action masking mechanism filters invalid or infeasible actions during the action selection phase to shorten the period of irrelevant exploration, hasten the learning progress, and ensure that the selected actions satisfy the practical constraints in MEC systems.

3. The TORA problem in a MEC system

3.1. System model

We consider a heterogeneous MEC system comprising a set $\mathcal{N} = \{1, 2, \dots, N\}$ of User Devices (UDs) and a set $\mathcal{M} = \{1, 2, \dots, M\}$ of Edge Servers (ESs), as illustrated in Fig. 1. The UD_{*i*} represent IoT entities, including smartphones, environmental sensors, autonomous vehicles, industrial robots, etc. Each $UD_i \in \mathcal{N}$ is characterized by inherent resource limitations, such as available computing power, remaining energy, and low uplink power. Consequently, these devices primarily function as data sources or perform only lightweight pre-processing.

In contrast, ESs are co-located with base stations or access points, offering higher computational capacity and near-infinite energy. Through wireless links, ESs can concurrently receive and execute computation-intensive tasks offloaded from multiple UD_{*s*}.

3.2. Connectivity model

In a practical MEC system deployment, communication topology between the UD_{*s*} and ES_{*s*} does not always form a full graph. Limited signal coverage, physical barriers or administrative policies can prevent certain UD_{*s*} from communicating with certain ES_{*s*}. Such spatial-temporal connectivity between UD_{*s*} and ES_{*s*} can be abstracted as a binary connection matrix $\mathbf{L} \in \{0, 1\}^{N \times M}$ illustrated in Fig. 2, where $L_{i,j} = 1$ represents UD_i is covered by ES_j , and $L_{i,j} = 0$ otherwise. This connection matrix would be the first input to our proposed action masking scheme below, in which the DQN agent would never attempt actions to offload tasks to unavailable servers.

3.3. Task model

For each task i generated from UD_i , we use $\mathcal{W}_i = \langle d_i, c_i, T_i^{\max} \rangle$ to represent its profile, including task size (d_i in bits), computational workload requirement (c_i in CPU cycles), and maximum latency ($T_i^{\max}$ as deadline). Since modern IoT applications can be highly heterogeneous, we do not make any assumption on how d_i and c_i are related to each other. For example, during advanced image processing or encryption, even a small increase in d_i could lead to exponential growth in c_i .

		Edge Server						
		0	1	2	3	...		M
User Device	1	0	0	1	0	1	0	0
	2	1	0	0	0	1	0	0
	3	0	0	1	0	1	0	0
	...	0	1	0	0	1	0	0
		0	0	0	1	0	1	0
	N	0	0	0	0	0	0	1

Fig. 2. Representation of a binary connection matrix.

3.4. User device model

Each $UD_i \in \mathcal{N}$ is defined by its hardware constraints: $\langle f_i^{loc}, P_i^{tx}, E_i^{res}, \kappa_i \rangle$. Specifically, f_i^{loc} is the local CPU frequency, P_i^{tx} is the uplink power, E_i^{res} is the remaining energy, and κ_i is the effective switched capacitance determined by the chip architecture.

3.5. Edge server model

Each $ES_j \in \mathcal{M}$ has its own queue following the First-In-First-Out (FIFO) rule. An ES_j is denoted by $\langle f_j^{srv}, Q_j^{max} \rangle$, where f_j^{srv} is its service rate (CPU cycles/s) and Q_j^{max} indicates its queue maximum capacity (CPU cycles). In contrast to UD, ESs are assumed to have a continuous power supply, so their energy consumption will not be considered in the reward function. Nevertheless, the service capabilities of each ES_j are subject to its current queue workload Q_j^{curr} , which is measured by the total CPU cycles of all tasks waiting in the queue of ES_j .

3.6. Task computing model

In local processing mode, the latency T_i^{loc} and the energy E_i^{loc} can be represented as below:

$$T_i^{loc} = \frac{c_i}{f_i^{loc}} \quad (1)$$

$$E_i^{loc} = \kappa_i (f_i^{loc})^2 c_i \quad (2)$$

If task i is offloaded from UD_i to ES_j , the overall offloading latency $T_{i,j}^{off}$ can be represented as the summation of uplink latency, queuing latency and execution latency. Similar to previous works [1], we ignore the downlink latency from the ES_j to UD_i since the output data are much smaller than task i 's input data d_i in most cases:

$$T_{i,j}^{off} = \frac{d_i}{R_{i,j}} + \frac{Q_j^{curr} + c_i}{f_j^{srv}}, \quad (3)$$

where the transmission rate $R_{i,j}$ is determined according to the Shannon capacity formula, $R_{i,j} = W \log_2(1 + SNR_{i,j})$, where W denotes the channel bandwidth and $SNR_{i,j}$ represents the signal-to-noise ratio. In addition, Q_j^{curr} denotes the current workload in the queue of ES_j , which consists of all unfinished tasks that arrived before task i and are waiting to be processed.

According to the assumption in Section 3.5, we do not consider the energy consumption at edge servers. Therefore, the energy consumption for task offloading only contains the uplink power of UD_i :

$$E_{i,j}^{off} = P_i^{tx} \cdot \frac{d_i}{R_{i,j}} \quad (4)$$

Each task i is either processed by user device UD_i or offloaded to edge server ES_j , the overall latency T_i and energy consumption E_i can be given by:

$$T_i = \begin{cases} T_i^{loc}, & \text{if processed locally} \\ T_{i,j}^{off}, & \text{if offloaded to } ES_j \end{cases} \quad (5)$$

$$E_i = \begin{cases} E_i^{loc}, & \text{if processed locally} \\ E_{i,j}^{off}, & \text{if offloaded to } ES_j \end{cases} \quad (6)$$

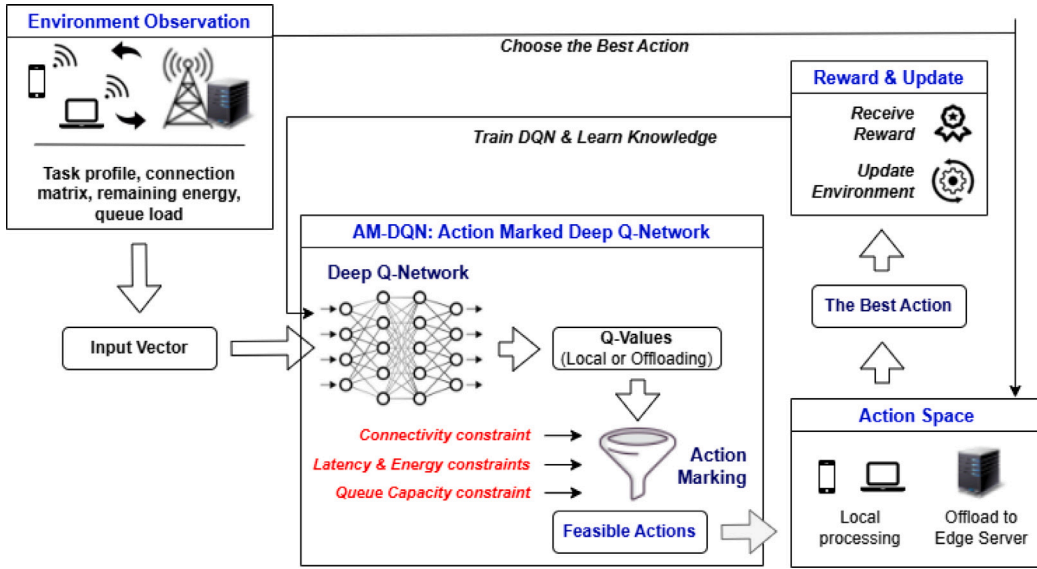


Fig. 3. The local execution and task offloading decision workflow of the AM-DQN model.

4. Action masked deep Q-network

Fig. 3 depicts the workflow of AM-DQN. Specifically, it contains three parts: system initialization, invalid/infeasible-action masking and decision making, and experience-driven learning for policy improvement. An improvement over the traditional DQN, the AM-DQN model takes advantage of the action masking mechanism during decision making, so that the offloading decisions made by agents can always be aware of the physical and operation constraints in MEC systems.

In each time slot t , the agent first observes the state $s(t)$ of the environment (including task profile, available computing power, remaining energy, connectivity, queue load, and deadline) and encodes it into a state vector. Then the DQN agent produces Q-values for estimating the expected actions. Before deciding to choose an action, the action masking mechanism is triggered to eliminate invalid (e.g., unavailable connection or insufficient remaining energy) or infeasible (e.g., overload queue or deadline violation) actions. The agent chooses an action $a(t)$ from the feasible action set $\mathcal{A}^{feasible}$ according to the ϵ -greedy policy. After executing the selected action, the environment moves into a new state $s(t+1)$ and a system reward $R^{sys}(t)$ is returned to update the Q-network.

4.1. TORA formulation as a MDP

We formulate the TORA problem as a Markov Decision Process (MDP) defined by tuple $\langle S, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where:

S : The state space describing the multidimensional joint status of all UD_s and ES_s.

$\mathcal{A}$: The action space capturing the local computing and offloading decisions.

$\mathcal{P}$: The state transition probability $P(s(t+1)|s(t), a(t))$, depicting the changes in the MEC environment.

$\mathcal{R}$: The reward function characterizing the latency-energy joint optimization target.

4.1.1. State space

At decision epoch t , the AM-DQN agent observes global MEC system state $s(t) \in S$ that contains information collected from all user devices and edge servers. This state is given by

$$s(t) = \{o_1(t), o_2(t), \dots, o_N(t), Q_1^{curr}(t), Q_2^{curr}(t), \dots, Q_M^{curr}(t)\}, \quad (7)$$

where $o_i(t) = [W_i(t), L_i(t), E_i^{res}(t)]$ contains the task profile, connection status, and remaining energy of UD_i . Here, $Q_j^{curr}(t)$ represents current computational backlogs at edge server j .

With global system observation, the DQN agent decides how to process the currently arriving task accounting for overall system conditions.

4.1.2. Action space

Assume that decisions of task processing for all N user devices can be made at once. Then the action space can be defined as the Cartesian product of all actions of the user devices. The available actions of each UD_i is defined as $\mathcal{A}_i = \{0, 1, 2, \dots, M\}$, where $a_i(t) = 0$ denotes local execution at UD_i , and $a_i(t) = j \in \{1, \dots, M\}$, indicates that the task i is offloaded to edge server ES_j . Then

there are totally $(M + 1)^N$ joint actions. This size of action space scales exponentially with the number of user devices and edge servers, hence applying the AM-DQN model directly is computationally prohibitive.

Instead of selecting a single action for all N user devices, the decision-making process is broken down into a sequence of sequential decisions within the same time slot. Specifically, the agent sequentially determines the action for each user device based on the current global state of the MEC system as well as decisions already assigned to previously processed devices. Thanks to this approach, the combinatorial action space of size $(M + 1)^N$ is transformed into a series of sub-decision spaces, each containing only $M + 1$ possible actions. This transformation reduces action space explosion and enhances the scalability of the AM-DQN model in large-scale MEC systems.

4.1.3. State transition

After executing the selected action, the MEC system transitions from state $s(t)$ to state $s(t + 1)$.

If the task is executed locally or offloaded to an edge server, the remaining energy of the corresponding user device UD_i is updated as

$$E_i^{res}(t + 1) = \max(0, E_i^{res}(t) - E_i(a(t))), \quad (8)$$

where $E_i(a(t))$ denotes the energy consumed by the selected action $a(t)$.

Meanwhile, if the task is offloaded to edge server ES_j , the server backlog evolves according to

$$Q_j^{curr}(t + 1) = \max(0, Q_j^{curr}(t) + c_i - f_j^{srv}), \quad (9)$$

where c_i denotes the computational workload of the arriving task i and f_j^{srv} represents the service rate of edge server ES_j .

Consequently, the next state reflects the updated energy levels of user devices and the computational loads of edge servers after task execution.

4.1.4. Reward function

The reward function is designed to jointly optimize task execution latency and energy consumption while satisfying MEC system constraints.

For the currently scheduled task, the reward is defined as

$$r_i(t) = \mathbb{I}_i(t) \cdot \left[\psi \left(\frac{T_i^{max} - T_i(t)}{T_i^{max}} \right) + (1 - \psi) \left(\frac{E_i^{res} - E_i(t)}{E_i^{res}} \right) \right] - (1 - \mathbb{I}_i(t)) \cdot \zeta \quad (10)$$

where $\mathbb{I}_i(t) = 1$ if $T_i(t) \leq T_i^{max}$ and $E_i(t) \leq E_i^{res}$, and $\mathbb{I}_i(t) = 0$, otherwise. The indicator $\mathbb{I}_i(t)$ acts as a constraint satisfaction flag. It ensures that positive rewards are only given when both the latency and energy requirements of the task are met. If either constraint is violated, the reward is replaced with a penalty. Therefore, $\mathbb{I}_i(t)$ incentivizes the agent to learn task processing policies that meet QoS constraints while minimizing latency and energy consumption.

Here, $T_i(t)$ and $E_i(t)$ denote the task i 's completion delay and energy consumption associated with the selected action $a(t)$, respectively. The parameter $\psi \in (0, 1)$ controls the trade-off between latency and energy objectives, while ζ is a penalty applied when any system constraint is violated.

The system reward $R^{sys}(t)$ encourages global cooperation:

$$R^{sys}(t) = \sum_{i=1}^N \omega_i \cdot r_i(t) \quad (11)$$

where ω_i denotes the reward weight for UD_i and $\sum_{i=1}^N \omega_i = 1$.

After receiving reward $R^{sys}(t)$, the target value is computed according to

$$y(t) = R^{sys}(t) + \gamma \max_{a'} Q(s(t + 1), a'; \theta^-), \quad (12)$$

where γ is the discount factor and θ^- is the target network parameter.

The loss function is given by

$$Loss = \frac{1}{B} \sum_{k=1}^B (y(t) - Q(s(t), a(t); \theta))^2, \quad (13)$$

where B is the mini-batch size.

The primary network parameters are updated using gradient descent:

$$\theta(t + 1) = \theta(t) - \eta \nabla Loss, \quad (14)$$

where η is the learning rate.

By continuously interacting with the environment, the agent learns an optimal decision policy that maximizes the cumulative expected reward.

4.1.5. Action masking mechanism

Traditional DQNs explore actions blindly, including actions that are invalid or infeasible under current MEC system constraints. To guarantee feasibility, we incorporate an action masking mechanism before action selection. Specifically, for an arriving task, an action ($a \in \mathcal{A}$) is considered feasible if

$$C(a) = \begin{cases} 1, & (L_{i,j}(a) = 1) \wedge (E_i(a) \leq E_i^{res}) \wedge (Q_j^{curr}(a) + c_i \leq Q_j^{max}) \wedge (T_i(a) \leq T_i^{max}) \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

Eq. (15) gives the connectivity, remaining-energy, queue-capacity, and deadline constraints the MEC system must meet when servicing any incoming task. Since the system's MEC resources are not always stable, not all constraints can be evaluated with certainty when the offloading decision is made. Whereas the system's remaining energy levels and the status of the network can be directly determined from the current system, estimates must be used when assessing queue capacity and deadline of any particular task, as those rely on future system behavior and cannot be precisely known. This uncertainty may cause an action to appear valid from the decision making point-of-view, which later leads to a constraint violation, yielding a task failure rate greater than zero in spite of the use of the AM-DQN method (This problem is further discussed in Section 5.6).

Before applying either the ϵ -greedy policy or greedy action selection, the masked Q-value is defined as

$$\hat{Q}(s, a) = \begin{cases} Q(s, a), & C(a) = 1, \\ -\infty, & C(a) = 0. \end{cases} \quad (16)$$

The action selected by the agent is then $\arg \max_a \hat{Q}(s, a)$.

By assigning invalid and infeasible actions a value of $-\infty$, the action masking mechanism guarantees that only feasible task-processing decisions can be selected, while significantly reducing ineffective exploration during training.

4.1.6. AM-DQN algorithm

The 1 algorithm outlines the implementation process of the AM-DQN model to optimize TORA. For ease of presentation, the AM-DQN algorithm is divided into three stages:

Stage 1 (Initialization): The reinforcement learning process start by initializing the parameters of the primary Q-network (θ) and the target Q-network (θ^-). To begin learning, The algorithm also initialize a replay buffer D to save transition experience tuples and initialize the exploration rate ϵ to 1.0. The MEC environment is set and the initial state $s(0)$ is observed.

Stage 2 (Action masking and selection): At each time slot t , the agent determines the feasible decision space $\mathcal{A}^{feasible}$ by evaluating all candidate offloading decisions against MEC system constraints (e.g., connectivity, deadline, remaining energy, queue load). A non-empty set of feasible actions are constructed dynamically.

Stage 3 (Experience-driven learning): The agent then executes the action $a(t)$, receives the reward $R^{sys}(t)$, and transitions to the next state $s(t+1)$. The resulting transition tuple $(s(t), a(t), r(t), s(t+1))$ is stored in the replay buffer D . During training, random mini-batches of transitions are selected from D to update the parameters of the primary network, θ , by minimizing the corresponding loss function through gradient descent. Finally, the parameters of the target network, θ^- , are updated by copying the weights from the primary network, which helps stabilize the learning process and promotes the convergence.

5. Simulation and result analysis

5.1. Parameters and scenarios

We implemented all experiments in Python scripts running in the same simulation environment. The simulation is conducted on a desktop PC with Intel Core i5 @ 3.4 GHz CPU, 16 GB memory, and Windows OS. The MEC system involves $N = 100$ user devices and $M = 5$ edge servers. At each decision epoch, a task arrives to a user device and is processed according to the chosen offloading strategy. The simulator accounts for dynamic connection states, computing resources, and task arrivals to resemble the real-world MEC environment.

The wireless transmission is modeled by the signal-to-noise ratio ($SNR_{i,j}$) and its resultant uplink rate ($R_{i,j}$). The parameter settings and ranges adopted in this study follow those commonly used in MEC settings [1,9,19] and are listed in Table 1.

The parameters for the agent and the physical constants used across all scenarios are summarized in Table 2.

5.1.1. Evaluation metrics

To evaluate the effectiveness of the AM-DQN model, four baseline schemes are compared:

1. Local execution (Local): Execute all tasks locally on user devices;
2. Full offloading (FullOff): Blindly offload all tasks to edge servers ;
3. Random selection (Random): Randomly select an action from the action space;
4. Traditional DQN (DQN): Apply the traditional DQN model without infeasible-action masking.

We then study the following metrics to evaluate the system efficiency from different aspects:

- *Latency reduction*: Measure the average task completion time under different policies to represent task processing time.

Algorithm 1 AM-DQN for TORA

Require: $N, M, \alpha, \gamma, \epsilon$
Ensure: $\mathcal{A}^{feasible}$

```

1: // Phase 1: Initialization
2: Initialize the primary Q-network  $Q(s, a, \theta)$  and the target Q-network  $Q(s, a, \theta^-)$  with  $\theta^- \leftarrow \theta$ 
3: Initialize the replay buffer  $\mathcal{D}$  and the exploration rate  $\epsilon \leftarrow 1.0$ 
4: for each episode do
5:   Reset MEC environment and observe the initial state  $s(0)$ 
6:   for each time slot  $t = 0, \dots, T - 1$  do
7:     // Phase 2: Action masking and selection
8:     Initialize the feasible action set  $\mathcal{A}^{feasible} \leftarrow \emptyset$ 
9:     for each candidate action  $a(t) \in \mathcal{A}$  do
10:      if  $a(t)$  satisfies Eq. (15) then  $\mathcal{A}^{feasible} \leftarrow \mathcal{A}^{feasible} \cup a(t)$ 
11:    end if
12:  end for
13:  if  $\text{rand}() < \epsilon$  then
14:    Select  $a(t)$  randomly from  $\mathcal{A}^{feasible}$ 
15:  else
16:    Compute the Q-value  $\hat{Q}(s(t), a)$  according to Eq.(16)
17:    Select  $a(t) = \arg \max_a \hat{Q}(s(t), a)$ 
18:  end if
19:  // Phase 3: Experience-driven learning
20:  Execute  $a(t)$ , compute  $R^{sys}(t)$  and observe the next state  $s(t+1)$ 
21:  Update  $E_i^{res}(t+1)$  and  $Q_j^{curr}(t+1)$  according to Eq.(8) and Eq.(9)
22:  Store the experience  $(s(t), a(t), r(t), s(t+1))$  in  $\mathcal{D}$ 
23:  Select a mini-batch from  $\mathcal{D}$ 
24:  Compute the target value  $y(t)$  according to Eq.(12)
25:  Compute  $Loss$  according to Eq.(13)
26:  Update the parameter  $\theta$  according to Eq.(14)
27:  Update target network  $\theta^- \leftarrow \theta$ 
28: end for
29: end for

```

Table 1

Dataset parameters and ranges.

Data field	Symbol	Range/Value	Description
User ID	i	[1, 100]	Identifier for UDs.
Local capacity	f_i^{loc}	[0.5, 2.5] GHz	UD's processing power.
Local energy	E_i^{res}	[10, 20] J	UD's remaining energy.
Uplink power	p_i^{tx}	[0.1, 0.5] W	Transmission power.
Task size	d_i	[0.5, 5.0] MB	Input data workload.
Task complexity	c_i	[0.1, 1.0] $\times 10^9$	Total computational workload (CPU cycles).
Deadline	T_i^{max}	[100, 500] ms	Latency constraint.
Server capacity	f_j^{srv}	[5, 25] GHz	ES processing power.
Channel quality	$SNR_{i,j}$	[0, 20] dB	Wireless signal quality.

Table 2

Agent and physical parameters.

Parameter	Symbol	Value
Learning rate	α	0.001
Discount factor	γ	0.95
Trade-off weight	ψ	0.5
Effective capacitance	κ	10^{-28}
Hidden layers	–	$2 \times [128, 64]$
Replay buffer size	$ \mathcal{D} $	20,000

- **Energy efficiency:** Measure the total amount of energy consumed to perform task execution through local computations and data transmission.
- **Reward convergence:** Observe the reward updating process to reveal the agent's tradeoff across objectives.
- **Task success rate:** Evaluate whether the system can handle dynamic workloads while satisfying task requirements.
- **Runtime:** Measure the average execution time of each policy to assess its computational overhead and suitability for real-time MEC environments.

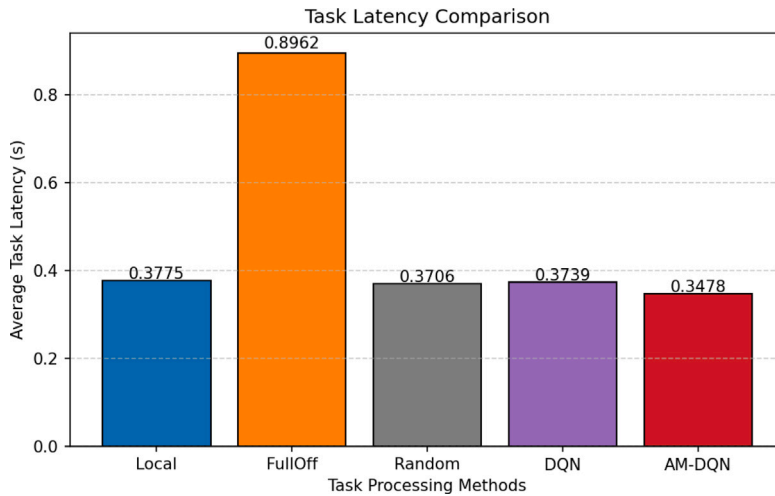


Fig. 4. Comparison of the processing latency across different models.

5.2. Latency reduction

Fig. 4 represents the average processing latency obtained from different models. AM-DQN achieves the lowest average latency of 0.3478 s, which represents a latency improvement of 6.98% with regard to the latency of DQN (0.3739 s) and 6.15% relative to Random (0.3706 s). This observation reveals the efficiency of the action masking mechanism to remove infeasible offloading decisions such as trying to offload a task, while the connection to the target server is unavailable, or the server is overloaded.

It is observed from Fig. 4, that FullOff obtained the largest latency of 0.8962 s, which is almost 2.6 times higher than AM-DQN. This considerable latency increase occurred because the overloaded edge servers receive every possible task without checking if they are able to process the task. The queues at the edge servers are full and every task stays at the queues before being processed by the server, hence they take more time.

the Local model resulted in 0.3775 s. This is the second best in terms of average latency since tasks do not need to be offloaded and are not queued at edge servers. However, it should not be forgotten that all tasks are executed locally on user devices and, for resource-constrained systems, that could run out quickly from computing and battery capacity.

5.3. Energy efficiency

Fig. 5 illustrates the averaged energy consumption of each model. As we can see, AM-DQN consistently outperforms other approaches in terms of energy efficiency. It consumes an average energy of 0.0476 J. Compared with DQN (0.0522 J) and Random (0.0511 J), AM-DQN saves around 8.81% and 6.85% energy, respectively. This validates that AM-DQN not only accelerates task processing performance but also avoids unnecessary energy waste caused by infeasible offloading decisions.

The reason behind the relatively high energy consumption of DQN lies in its ignorance of action feasibility during the decision-making process. Thus the agent tends to offload computational tasks when channel conditions are bad or edge servers are overloaded. Such issue is especially severe during early training stage when $\epsilon = 1$. As actions are ϵ -greedy selected in this phase, it is even possible to offload tasks over unavailable connections. Different from DQN, the action masking module in AM-DQN disallows taking all infeasible actions before conducting decision making process. In such a way, it avoids wasting energy on infeasible actions.

Among all approaches, FullOff achieves the highest energy consumption with the average value of 0.1257 J. It consumes around 2.6 times more energy than AM-DQN does. This again indicates that naively offloading all tasks to edge servers is extremely energy inefficient in practical MEC systems. Large communication overhead and high rates of blind offloading attempts make FullOff redundant in terms of energy consumption, which reinforces our motivation of enforcing feasibility awareness into the offloading decision making process.

5.4. Reward performance

Fig. 6 depicts the total accumulated reward obtained by each model throughout training. Recall that our reward function is defined as a weighted multi-objective combination of latency and energy. Therefore, Fig. 6 can also be used to interpret the trade-off between evaluated methods. AM-DQN achieves the highest total reward of -0.2853 compared to any other approach. This is equivalent to having around a 3.87% improvement on DQN which achieved a reward of -0.2968 .

When unaware of the feasibility of actions, DQN searches the action space and may choose an action that violates system constraints. However, taking an infeasible action will heavily penalize the reward and lead to poorer performance. The action

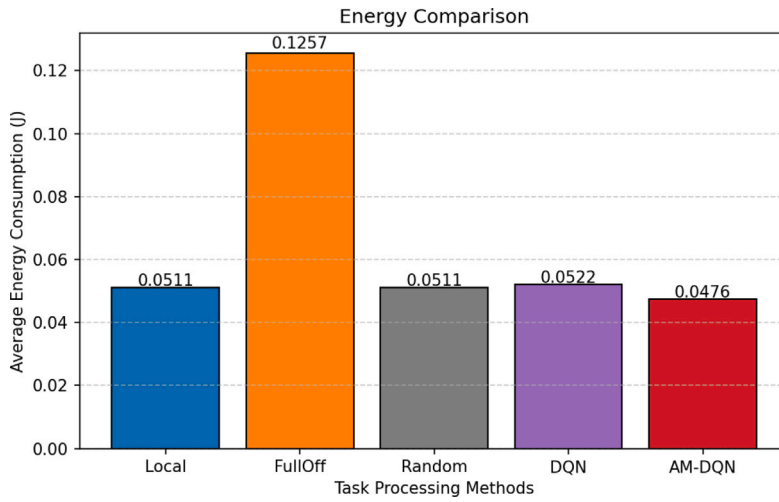


Fig. 5. Comparison of the energy consumption across different models.

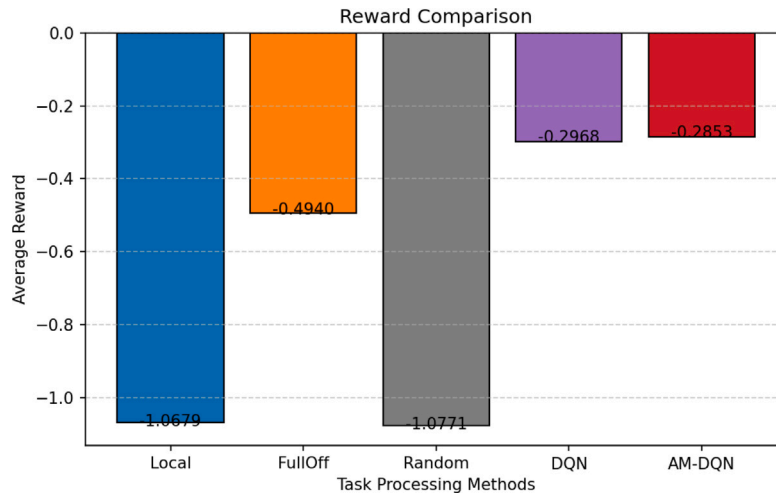


Fig. 6. Comparison of the aggregate reward across different models.

masking mechanism allows AM-DQN to only explore feasible actions and therefore enables the agent to learn efficiently and converge to a higher quality policy.

Random and Local achieve the worst performance with the rewards of -1.0771 and -1.0679 , respectively. These are around 73% worse than AM-DQN. This illustrates the poor decision making strategy of selecting actions without any regard to the state of the environment, such as device compute capacity or queue lengths on each server. FullOff also significantly under-performs the DQN-based methods with a reward of -0.4940 . This is around 40% worse than AM-DQN.

5.5. Task success rate

The task completion (success) rates of DQN-based approaches and baselines are shown in Fig. 7. These results show that AM-DQN outperforms other baseline methods (Random with 26.92% and DQN with 45.44%), achieving the highest success rate of 51.00%, representing a 12.24% improvement over DQN, under simulated heterogeneous task arrivals with different energy levels and connection fluctuations. This promising result verifies the importance of learning feasible strategies in MEC with resource-constrained IoT devices. The relatively modest success rate reflects the stringent deadline and energy constraints imposed under highly dynamic MEC environments.

DQN achieves a lower task success rate than AM-DQN because it frequently explores infeasible actions and violates system constraints, such as task deadline requirements and energy limitations. In contrast, the action masking mechanism in AM-DQN acts as a pre-filter, preventing infeasible actions from being considered and thereby ensuring constraint-aware decision making

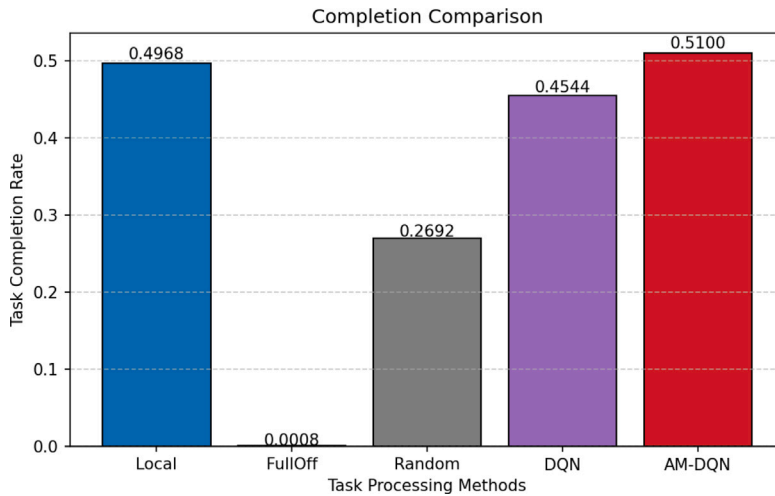


Fig. 7. Comparison of the task success rate across different models.

Table 3

Comparison of the task failure rate among different models.

Model	Task failure rate (%)
Local	49.44
FullOff	99.96
Random	75.04
DQN	50.88
AM-DQN	48.18

during both exploration and exploitation. As a result, AM-DQN is able to maintain a higher proportion of successful task executions. Meanwhile, FullOff achieves a success rate of only 0.08%, primarily due to severe constraint violations caused by excessive queuing delays at edge servers. We also see that blindly offloading tasks to MEC servers is not useful in dense MEC systems where queuing latency is the primary performance bottleneck.

5.6. Task failure rate and runtime

Table 3 reports the average task failure rate over 1000 simulation time slots. We can see that the Local solution suffers from a high task failure rate of 49.44% because the limited computing and energy resources of user devices are often insufficient to meet task deadlines. This problem is even more severe for FullOff due to limited uplink connectivity and server resource constraints. Compared to DQN, AM-DQN achieves a lower task failure rate because the action masking mechanism prevents agents from selecting actions that violate system constraints.

Specifically, across more than 1000 simulated time slots, AM-DQN masked an average of 118 invalid actions and 176 infeasible actions out of 600 actions from 100 devices, reducing the number of actions to be considered by 49%. However, task errors could not be completely eliminated because some violations could not be identified beforehand during the decision-making process. For example, whether an offloaded task would meet its deadline depends on several uncertainties, including transmission latency, queuing latency, and server processing latency. Despite these uncertainties, AM-DQN achieved the lowest task error rate (48.18%) among all evaluated methods, an improvement of 5.31% compared to DQN, due to its ability to eliminate a large proportion of invalid and infeasible actions before action selection.

AM-DQN also reduces the runtime. As shown in Fig. 8, the runtime of AM-DQN is 1.8376 s, 24.31% faster than DQN (2.4279 s). This is achieved because the reduction in feasible actions decreases action-selection overhead, resulting in lower runtime. Though DRL-based models have much higher execution times than heuristic baselines (larger than average, ranging from 9.91 to 18.43 times) due to the inference time of Deep Neural Networks (DNNs), the introduced acceleration from AM demonstrates the improved efficiency.

These results demonstrate that the action masking not only improves decision-making performance but also speeds up runtime, allowing DRL to become feasible for real-time MEC settings.

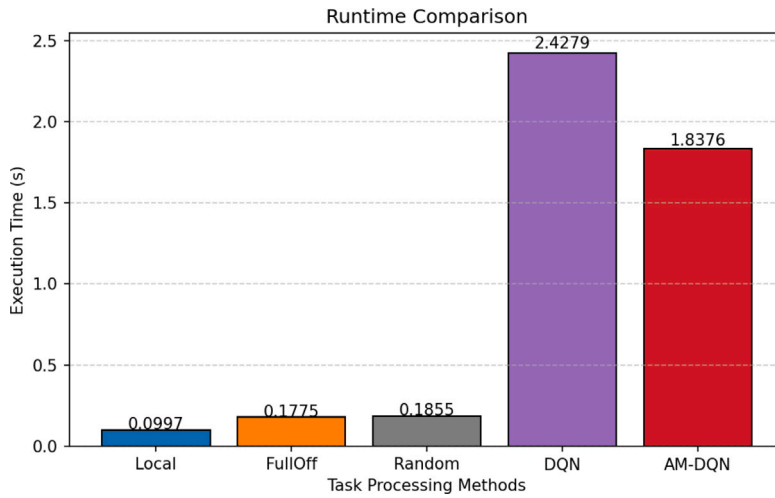


Fig. 8. Comparison of runtime across different models.

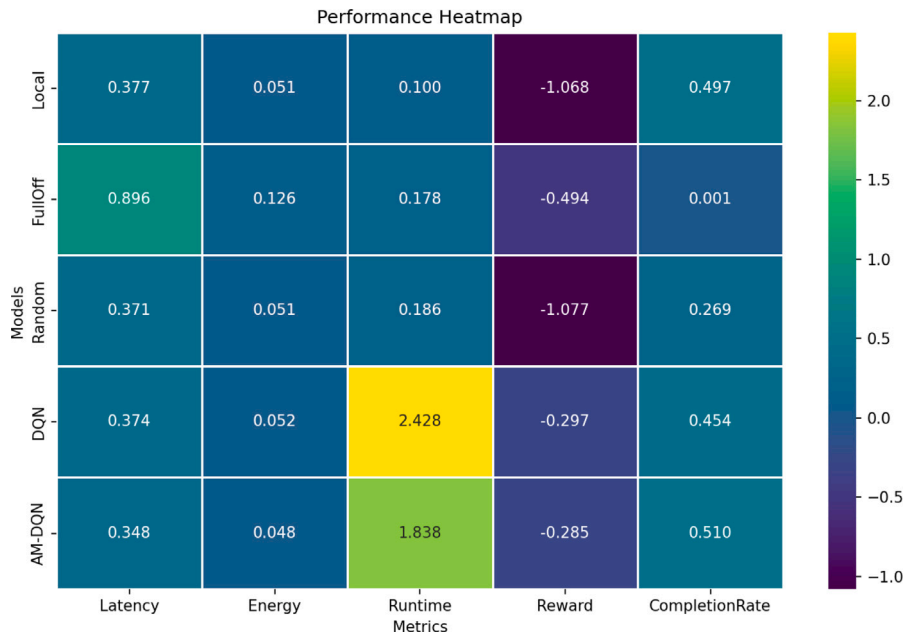


Fig. 9. Correlation heatmap of performance indicators.

5.7. Multi-indicator correlation analysis and trade-offs

Fig. 9 offers another view to discuss the trade-offs between each performance metric via heatmap visualization over all evaluated metrics. When looking at the system-level performance heatmap holistically, we note several observations about the inner relations between performance metrics for MEC systems:

- **Energy-latency trade-offs:** Notice a highly positive correlation between latency and energy consumption in the heatmap. Policies with higher than tolerable latency (e.g., FullOff in 0.8962 s), tend to consume the most energy (0.1257 J). Latency and energy can therefore be seen as two aspects working in tandem rather than representing conflicting objectives in dense IoT settings, where meeting one immediately contributes to meeting the other.
- **Reward's sensitivity to QoS:** Whereas overall reward is negatively correlated with latency and energy consumption (i.e., processing latency, saved energy), success rate clearly benefited the most from reductions in latency. Both AM-DQN's latency (≈ 0.348 s) and energy (≈ 0.048 J) is the lowest among all models. Correspondingly, it also achieves the best reward

accumulation and completion rate. Therefore we can conclude that the reward function successfully directed the agent to learn policies around the Pareto frontier of the decision space.

- **Independence of algorithm runtime:** Last but not least, we see that the running time of the algorithm does not affect the KPIs of a deployed policy (e.g. Latency, Success rate). This can be seen as both DRL-based models incur a higher runtime cost for inference (≈ 2 s vs. ≈ 0.15 s for baselines). Despite the higher numerical value, the inference time represents a fixed cost to pay in order to allow for adaptable policies. This adaptive overhead does not compromise the quality of offloading decisions and leads to an overall better task completion time seen at the system-level.
- **Action masking efficacy:** Finally, we also see that AM-DQN has a lower runtime compared to the traditional DQN (1.8376 s vs. 2.4279 s), which verifies that introducing the action masking mechanism does not harm its inference time. Combined with hardware acceleration on edge devices, we believe AM-DQN is applicable for real-time edge usage.

6. Conclusion

We model the TORA problem in MEC systems as a constrained decision problem, and propose a DRL-based decision framework, namely an Action-Masked Deep Q-Network (AM-DQN), together with a reward mechanism penalizing the sum of processing latency and energy consumption. The AM-DQN is aware of the current environment state consisting of incoming task information, connectivity state, remaining energy level of each device, and queue size of each server. Since system resources vary dynamically, a decided action (either locally processing or offloading) may become infeasible at subsequent decision epochs.

To solve this problem, we apply a feasibility-aware action-masking mechanism which filters out invalid and infeasible decisions both in exploration and exploitation phases and ensures that only currently feasible actions are considered, tackling one inherent obstacle for conventional DRL algorithms in meeting the resource constraint in the highly dynamic MEC environment.

Based on extensive simulation results, simulation results demonstrate that AM-DQN outperforms the benchmarks including local processing, full offloading, random selection, and conventional DQN in terms of reducing the processing delay, saving the energy consumption, and increasing the task success rate by 6.98%, 8.81%, and 5.31%, respectively compared to conventional DQN. In addition, the conversion of a search in the entire decision space into a sequential decision search over the subspace action spaces of each user device leads to the decrease of the computational cost, thus the proposed algorithm exhibits scalability in MEC networks.

Although it is difficult to eliminate the overheads of DRL-based approaches, it is able to learn a reliable decision-making policy in dynamically changing circumstances. In the future, we could extend the AM-DQN framework into a multi-agent DRL architecture to facilitate large-scale MEC with mobility and decentralized control.

CRedit authorship contribution statement

Le Van Hoa: Writing – original draft, Software, Methodology, Investigation, Formal analysis. **Pham Trong Nguyen:** Writing – original draft, Software, Data curation. **Vo Viet Minh Nhat:** Writing – review & editing, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Le Van Hoa, Pham Trong Nguyen, Vo Viet Minh Nhat report financial support was provided by Hue University. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partially supported by Hue University under the Core Research Program, Grant No. NCTB.DHH.2025.09.

Data availability

The data that has been used is confidential.

References

- [1] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, *IEEE Commun. Surv. Tutorials* 19 (4) (2017) 2322–2358.
- [2] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, *IEEE Commun. Surv. Tutorials* 19 (3) (2017) 1628–1656.
- [3] S. Sardellitti, G. Scutari, S. Barbarossa, Joint optimization of radio and computational resources for multicell mobile-edge computing, *IEEE Trans. Signal Inf. Process. over Netw.* 1 (2) (2015) 89–103.
- [4] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, *IEEE/ACM Trans. Netw.* 24 (5) (2016) 2795–2808.
- [5] H. Ye, G.Y. Li, B.H. Juang, Deep reinforcement learning based resource allocation for V2V communications, *IEEE Trans. Veh. Technol.* 68 (4) (2019) 3163–3173.

- [6] C. Wang, C. Liang, F.R. Yu, Q. Chen, L. Tang, Computation offloading and resource allocation in wireless cellular networks with mobile edge computing, *IEEE Trans. Wirel. Commun.* 16 (8) (2017) 4924–4938.
- [7] Y. Mao, J. Zhang, S.H. Song, K.B. Letaief, Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems, *IEEE Trans. Wirel. Commun.* 16 (9) (2017) 5994–6009.
- [8] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, et al., Human-level control through deep reinforcement learning, *Nature* 518 (7540) (2015) 529–533.
- [9] Y. He, J. Ren, G. Yu, Y. Cai, DQN-based computation offloading strategy for mobile edge computing, *IEEE Wirel. Commun. Lett.* 8 (2) (2019) 420–423.
- [10] Y. Wu, K. Zhang, K.B. Letaief, Online computation offloading in mobile edge computing with energy harvesting devices, *IEEE Trans. Wirel. Commun.* 18 (12) (2019) 5960–5975.
- [11] K. Zhang, Y. Mao, S. Leng, Y. He, Y. Zhang, Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading, *IEEE Veh. Technol. Mag.* 12 (2) (2017) 36–44.
- [12] X. Gu, G. Zhang, Y. Cao, Cooperative mobile edge computing–cloud computing in Internet of Vehicles: Architecture and energy-efficient workload allocation, *IEEE Internet Things J.* 7 (7) (2020) 6165–6177.
- [13] X. Xu, Q. Lui, Y. Luo, H. Peng, H. Zhang, S. Zhang, A computation offloading method over big data for IoT-enabled cloud–edge computing, *Future Gener. Comput. Syst.* 95 (2019) (2018) 522–533.
- [14] Y. Dai, D. Xu, S. Maharjan, Y. Zhang, Joint computation offloading and user association in multi-task mobile edge computing, *IEEE Trans. Veh. Technol.* 68 (12) (2019) 12313–12325.
- [15] E. Altman, *Constrained Markov Decision Processes*, CRC Press, 1999.
- [16] C. Tessler, D.J. Mankowitz, S. Mannor, Reward constrained policy optimization, in: *Proceedings of the International Conference on Learning Representations, ICLR*, 2019.
- [17] S. Huang, S. Ontañón, A closer look at invalid action masking in policy gradient algorithms, *Proc. AAAI Conf. Artif. Intell.* 34 (04) (2020) 4386–4393.
- [18] K. Zhang, K.W. Ross, On the benefits of action masking in reinforcement learning, in: *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021.
- [19] J. Zhang, X. Hu, Z. Ning, E.C.H. Ngai, L. Zhou, J. Wei, B. Hu, Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, *IEEE Internet Things J.* 5 (4) (2018) 2633–2645.