Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.078998

ARTICLE

# A Deep Reinforcement Learning-Based Pre-Allocation Mechanism for Efficient Task Offloading in Mobile Edge Computing

**Chaobin Wang**[1,2], **Xianghong Tang**[1,2,*], **Jianguang Lu**[1,2], **Jing Yang**[1,2] **and Panliang Yuan**[1,2]

[1]College of Computer Science and Technology, Guizhou University, Guiyang, China
[2]State Key Laboratory of Public Big Data, Guizhou University, Guiyang, China

*Corresponding Author: Xianghong Tang. Email: xhtang@gzu.edu.cn

Received: 12 January 2026; Accepted: 17 March 2026; Published: 08 May 2026

**ABSTRACT:** Mobile Edge Computing (MEC) facilitates the rapid response and energy-efficient execution of tasks on mobile devices. However, determining whether and where to offload tasks remains a significant challenge due to the constantly changing character of workloads in MEC environments. To address this issue, this paper proposes PreAlloc-A2C—a deep reinforcement learning actor-critic-based framework that calculates allocation scores by leveraging both task features (task size, required completion time, and waiting time) and server features (queue length and historical workload). This design enables fully distributed task offloading decisions without centralized coordination. Additionally, a Long Short-Term Memory (LSTM) network is integrated to forecast impending server loads, thereby supporting adaptive scheduling. A tailored reward function is also designed to jointly optimize three key performance metrics: task delay, device energy consumption, and task drop rate. Extensive experiments are conducted to evaluate PreAlloc-A2C against five baseline algorithms: Particle Swarm Optimization (PSO), Advantage Actor-Critic (A2C), Deep Q-Network (DQN), Double Deep Q-Network (DDQN), and Dueling Deep Q-Network (Dueling DQN). The results show that PreAlloc-A2C outperforms all baselines, achieving lower latency, reduced energy consumption, and a lower task drop rate.

**KEYWORDS:** Mobile edge computing; task offloading; deep reinforcement learning

## 1 Introduction

In recent decades, the explosive growth of end devices (e.g., smartphones, IoT sensors) and emerging applications (e.g., augmented reality [1], real-time video analytics [2]) has posed stringent requirements on task processing latency and device energy efficiency—requirements that cloud-centric architectures struggle to meet due to their remote resource deployment. Mobile edge computing (MEC) [3] has therefore gained increasing attention as a promising paradigm to address this mismatch: by bringing edge resources closer to end users, MEC fundamentally enables enhanced responsiveness and energy efficiency in task handling while significantly reducing task dropping rates, thereby laying a critical foundation for next-generation communication and computing systems.

However, two fundamental questions arise in mobile edge computing: (i) whether to execute tasks locally or offload them to edge servers, and (ii) if so, which edge server should be selected for execution. Consequently, achieving efficient task offloading in MEC remains highly challenging. Early studies primarily relied on classical optimization methods, such as dynamic programming [4], linear relaxation [5], and branch-and-bound techniques [6], to formulate and solve the task offloading problem. These methods

aimed to obtain globally optimal decisions under simplified system models. While such approaches provide theoretical insights and optimality guarantees, they often suffer from high computational complexity and poor scalability in large-scale or dynamic MEC environments, where system states and task arrivals change rapidly. Therefore, classical methods gradually became less practical for large-scale and real-time task offloading scenarios.

To mitigate these drawbacks,, heuristic and metaheuristic approaches were subsequently proposed to achieve near-optimal solutions with lower computational overhead. Li et al. [7] devised an offloading framework (EIPSO) based on an enhanced PSO approach for MEC that minimizes system cost by balancing delay and energy consumption. Singh and Kim [8] presented a genetic algorithm–based heuristic for decoupled task placement and resource scheduling in MEC, enabling operators to optimize energy and latency. Similarly, Chakraborty and Mazumdar [9] proposed a GA-based dynamic edge server selection framework for Sensor Mobile Edge Computing that accounts for inter-task dependencies, achieving reduced energy consumption and delay. Wang et al. [10] introduced a Grey Wolf Optimizer–based metaheuristic for joint task offloading and power assignment in MEC, effectively reducing task completion time under device energy constraints. More recently, Cong et al. [11] developed a BCD-CONGW method combining convex optimization and a Grey Wolf algorithm to solve task offloading and resource allocation in vehicular MEC, achieving simultaneous minimization of service delay and power expenditure.

Although heuristic and metaheuristic strategies have achieved success in achieving lower service delay and improved energy efficiency, they still face inherent limitations. These methods are often scenario-specific and require extensive parameter tuning when system conditions change. Their iterative search processes lead to high computational overhead, causing delays in large-scale MEC systems. Moreover, most heuristics optimize only the current state without considering long-term factors such as future energy consumption or queue stability.

Deep reinforcement learning (DRL) has recently gained traction as a powerful alternative to heuristic approaches. In contrast to heuristic methods, DRL enables adaptive and long-term optimization by continuously learning from time-varying operating contexts, which is highly effective for optimizing offloading decisions in MEC. Consequently, a number of representative works have employed DRL to optimize task placement within the Mobile Edge Computing setting. Shen et al. [12] proposed SAC-PP, a DRL-based offloading strategy for MEC that jointly optimizes privacy protection and computation cost. Subsequently, Li et al. [13] introduced SMDRL, a scheduled multi-agent DRL scheme for MEC that enables efficient task offloading and coordination under resource- and communication-constrained scenarios. Wang et al. [14] introduced a SAC-based deep reinforcement learning framework incorporating a two-phase matching strategy for joint MEC and data collection in multi-AAV systems, effectively reducing latency and increasing collected data volume. Li et al. [15] introduced a multi-agent deep reinforcement learning (MADRL)–based distributed offloading algorithm for MEC, enabling collaborative task offloading and load balancing to reduce task-processing cost and timeout rates under dynamic conditions. Similarly, Rahmati et al. [16] introduced a MADRL-based distributed offloading method for MEC that reduces task-processing cost and balances computational load. More recently, Yang et al. [17] presented a SAC-based DRL approach for rotatable STAR-RIS-assisted MEC, jointly optimizing system configuration and task offloading to significantly reduce energy consumption.

Although existing DRL-based approaches have achieved remarkable progress in MEC task offloading, they still exhibit several inherent limitations. First, some studies employ heuristic or decoupled optimization mechanisms (e.g., matching-based or hierarchical strategies) that simplify the decision process but compromise the end-to-end adaptability and theoretical convergence of the learning framework. Second, some existing multi-agent DRL architectures often rely on intensive inter-agent coordination and frequent

information exchange, resulting in high communication overhead and limited scalability in dynamic network environments. Third, most existing methods focus on optimizing only one optimization objective, e.g., latency, energy consumption, or privacy, neglecting to account for the trade-offs and coupling among multiple objectives in real-world systems. Furthermore, many DRL-based schemes depend heavily on static environmental or system-level parameters while neglecting the intrinsic heterogeneity and feature correlations between tasks and edge servers, which are crucial for achieving adaptive and fine-grained scheduling. Therefore, designing a unified and generalizable framework that can exploit task and server feature representations and jointly optimize multiple performance objectives in real time remains a key open challenge in MEC research.

Building upon these observations, we propose PreAlloc-A2C, a novel actor–critic–based framework that fundamentally differs from existing DRL-based MEC offloading methods in its decision paradigm. Unlike conventional approaches that make per-task offloading decisions based solely on current system states or enhanced representations (e.g., structured states or recurrent models), PreAlloc-A2C introduces a pre-allocation mechanism that jointly incorporates both task features and edge server characteristics into the decision process. Specifically, it computes an allocation score for each task–server pair, enabling distributed and fine-grained decision-making without requiring inter-device communication. This design allows the system to capture the coupling between task requirements and server states, which is not explicitly modeled in existing methods. The salient contributions of our work are outlined below.

- **End-to-End Distributed Offloading Framework:** We propose *PreAlloc-A2C*, an end-to-end actor–critic–based task offloading framework that avoids heuristic or decoupled optimization stages, thereby enabling adaptive decision-making under dynamic MEC environments.
- **Communication-Efficient and Scalable Design:** Unlike existing multi-agent DRL approaches that rely on inter-agent coordination or centralized training signals, *PreAlloc-A2C* enables fully distributed offloading decisions without inter-device communication, significantly reducing communication overhead while maintaining decision quality.
- **Pre-Allocation-Based Decision Mechanism:** We introduce a novel pre-allocation mechanism that transforms the offloading problem from a per-task decision into a task–server matching problem. By computing allocation scores for all task–server pairs, the proposed method captures the coupling between task demands and server states, which is not explicitly addressed in existing DRL-based MEC solutions.
- **Multi-Objective Optimization with Workload Awareness:** A tailored reward function is developed to jointly optimize task delay, device energy consumption, and task drop rate. In addition, an LSTM-based workload prediction module is incorporated to proactively adapt scheduling decisions to time-varying server loads.
- **Extensive Performance Evaluation:** Extensive experiments demonstrate that *PreAlloc-A2C* consistently outperforms baseline methods including A2C, DQN, DDQN, and Dueling DQN in terms of latency, energy consumption, and task drop rate.

The structure of this paper is presented below. Section 2 provides a discussion of related work on MEC task offloading. Section 3 outlines the system model. Section 4 establishes the task offloading problem formulation. Section 5 elaborates on the proposed DRL-based PreAlloc-A2C framework. Section 6 validates the performance of the proposed scheme through extensive simulations and comparisons with baseline methods. Finally, Section 7 draws conclusions and identifies future research directions.

## 2 Related Work

In recent years, deep reinforcement learning (DRL) has emerged as a powerful paradigm for resource allocation and task offloading across edge, cloud, and vehicular computing systems, enabling adaptive decision-making in highly dynamic environments without explicit system modeling.

In stochastic edge–cloud scenarios, A3C- and A2C-based schedulers enhanced with recurrent structures have been proposed to capture temporal workload dynamics and improve energy efficiency, SLA satisfaction, and operational cost [18,19]. Multi-agent DRL frameworks further enable decentralized task offloading with reduced communication overhead, such as cooperative schemes based on variational recurrent neural networks [20]. For cloud–edge collaboration under mobility and task correlation, DRL architectures like DRL-CCMCO have demonstrated improved execution efficiency in industrial networks [21].

To address dynamic edge loads and delay-sensitive applications, several studies integrate DRL with temporal modeling. LSTM-enhanced dueling and double DQN methods effectively reduce task drop ratio and latency [22]. For applications with task dependencies, DAG-based offloading strategies leveraging DQN [23] and sequence-to-sequence learning [24] have been introduced to support parallel and dependency-aware scheduling. In caching-enabled MEC systems, joint optimization of computation and caching resources has been explored using soft actor-critic methods [25], while meta-learning frameworks such as DMRO accelerate adaptation in fast-changing IoT edge–cloud environments [26]. For wireless powered MEC networks, online DRL approaches like DROO eliminate combinatorial optimization complexity and achieve near-optimal performance with low latency [27].

Under strict deadline constraints and partial observability, DDPG-based approaches model task offloading as a POMDP to improve energy efficiency and task completion rates [28]. Multi-objective DRL frameworks based on PPO have been proposed to jointly optimize latency and energy without predefined weights, achieving improved Pareto efficiency [29]. Other studies introduce attention mechanisms and hypernetworks for scalable multi-entity scheduling [30], or bound Q-value estimation to mitigate overestimation in latency-critical scenarios [31].

DRL has also been widely applied in vehicular edge computing, including digital twin-enabled resource allocation and maintenance scheduling [32], fog-based vehicular task offloading under realistic mobility models [33], and game-theoretic DRL methods for energy-efficient scheduling with asynchronous arrivals and time-varying channels [34]. Pre-trained RL models, such as PTMB for satellite scheduling [35], and soft actor-critic-based joint computing, caching, and pushing strategies [36], have demonstrated improved decision efficiency in large-scale and high-dimensional settings.

Overall, existing DRL-based approaches exhibit strong robustness and versatility in dynamic edge environments. However, most methods rely on coarse-grained, system-level state representations, overlooking the intrinsic heterogeneity of tasks and edge servers. This limitation restricts their ability to perform fine-grained task–server matching, motivating the need for structured state modeling that explicitly captures task characteristics and node-specific resource attributes.

## 3 System Model

We consider a mobile edge computing system, depicted in Fig. 1, which consists of $M$ mobile devices and $N$ edge nodes. We define the set of mobile devices as $\mathcal{M} = \{1, 2, \ldots, M\}$ and the set of edge nodes as $\mathcal{N} = \{1, 2, \ldots, N\}$. The system operation is observed over $T$ time slots within each training episode. The duration of each time slot, $t \in \mathcal{T} = \{1, 2, \ldots, T\}$, is fixed at $\Delta$ seconds.

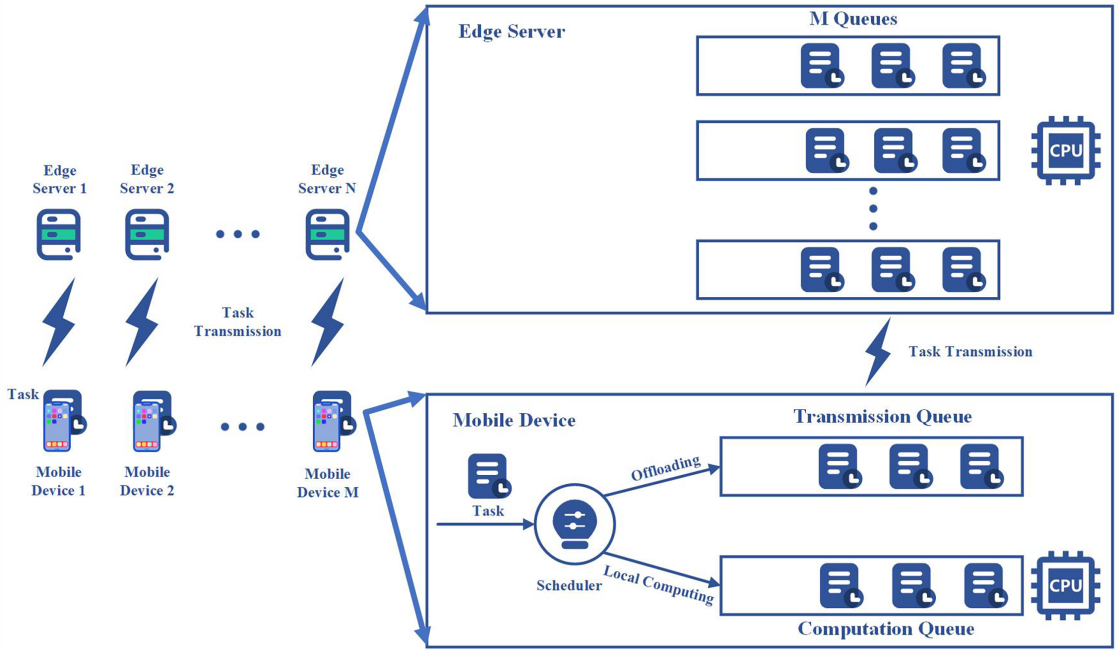



**Figure 1:** MEC system.

### *3.1 Task Model*

During each episode, a new task is generated for mobile device $m \in \mathcal{M}$ at the start of every time slot $t \in \mathcal{T}$ with probability $p_a$. The unique index of the arriving task is denoted by $\text{Task}_m(t) \in \mathbb{Z}^+$; if no task is generated at time $t$, we set $\text{Task}_m(t) = 0$. The task's data size is $L_m(t)$ (in bits), where $L_m(t) = 0$ also signifies no arrival. The computational requirement for processing one bit of data is defined by $c_m$ (in CPU cycles per bit). Each task $\text{Task}_m(t)$ has a deadline, $\tau_m$ (in time slots); if execution is not finalized before this deadline, the task is dropped. Crucially, every task is indivisible and subject to an all-or-nothing allocation constraint: it must be either executed locally or wholly transferred to an edge server.

### *3.2 Mobile Device Model*

In our system, each mobile device $m \in \mathcal{M}$ is modeled as an agent equipped with an internal scheduler that serves as its decision-making brain. Upon the arrival of a new task $\text{Task}_m(t)$, the scheduler first determines whether the task should be executed locally or offloaded to an edge server. We define a binary decision variable $x_m(t) \in \{0,1\}$ to indicate whether the task is executed locally or offloaded to an edge node. Specifically, $x_m(t) = 0$ denotes local execution, while $x_m(t) = 1$ represents task offloading to the edge. If local execution is selected, the task is placed into a local computation queue for processing. Otherwise, if offloading is chosen, the scheduler must further select an appropriate edge server to handle the task. We introduce a binary indicator variable $y_{m,n}(t) \in \{0,1\}$ to specify whether the task of mobile device $m$ at time slot $t$ is offloaded to edge node $n$. Specifically, $y_{m,n}(t) = 0$ indicates that the task is not offloaded to edge node $n$, whereas $y_{m,n}(t) = 1$ indicates that the task is offloaded to node $n$. Each task can be offloaded to at most one edge node, which can be formulated as

$$\sum_{n \in \mathcal{N}} y_{m,n}(t) = \mathbb{1}\big(x_m(t) = 1\big), \quad m \in \mathcal{M},\ t \in \mathcal{T}. \tag{1}$$

The task is then placed into a transmission queue and transmitted to the selected edge server via a wireless link. At the beginning of time slot $t \in \mathcal{T}$, $L_m(t)\big(1 - x_m(t)\big)$ denotes the number of task bits arriving at the computation queue of mobile device $m$, whereas $L_m(t)x_m(t)$ denotes the number of task bits arriving at its transmission queue. Both the computation queue and the transmission queue operate under a first-come, first-served (FCFS) policy. If a task completes its processing (or transmission) within a time slot, the next task in the queue begins execution (or transmission) at the start of the following time slot.

In the following, we provide a detailed description of the computation queue and the transmission queue, respectively.

*3.2.1 Computation Queue*

Every mobile device $m \in \mathcal{M}$ utilizes a dedicated CPU for its local computation queue. This computational capability is constant and defined as $f_m^{\text{device}}$. The time slot marking the completion or drop of a task, $\text{Task}_m(t)$, is denoted by $t_m^{\text{comp}}(t) \in \mathcal{T}$. If $\text{Task}_m(t)$ is not assigned to the local queue (i.e., $\text{Task}_m(t) = 0$), then $t_m^{\text{comp}}(t) = 0$. Tasks entering this queue are not serviced instantly; instead, processing is deferred until all preceding workloads have been fully executed. The resultant waiting duration, $t_m^{\text{wait}}(t)$ (in time slots), is calculated as follows:

$$t_m^{\text{wait}}(t) = \max\left\{\max_{t' \in \{0,1,\ldots,t-1\}} t_m^{\text{comp}}(t') - t + 1, 0\right\}, \tag{2}$$

here, $t \in \mathcal{T}$ denotes the current time slot, and the term $\max_{t' \in \{0,1,\ldots,t-1\}} t_m^{\text{comp}}(t')$ represents the latest time slot prior to $t$ by which each task in the computation queue has either been fully processed or explicitly discarded. The max operator ensures that the waiting time of each task is non-negative. It is worth noting that Eq. (2) does not explicitly include the historical workload term (e.g., $L_m(t')$ for $t' < t$), as commonly used in classical queueing theory. Instead, the proposed model implicitly captures the accumulated workload through the variable $t_m^{\text{comp}}(t')$, which records the completion time of all previously arrived tasks. Specifically, the term $\max_{t' \in \{0,1,\ldots,t-1\}} t_m^{\text{comp}}(t')$ represents the latest time slot at which all prior tasks have been processed or dropped. Therefore, the waiting time of the current task is determined by the residual processing time of all previously scheduled tasks, making Eq. (2) equivalent to a workload-aware queueing formulation in a discrete-time scheduling system. This formulation is consistent with a virtual queue representation, where the queue state is tracked via task completion timelines rather than explicit backlog accumulation. The value of $t_m^{\text{comp}}(t)$ is calculated as follows:

$$t_m^{\text{comp}}(t) = \min\left\{t + t_m^{\text{wait}}(t) + \left\lceil \frac{L_m(t)}{f_m^{\text{device}}\Delta/c_m} \right\rceil - 1, t + \tau_m - 1\right\}, \tag{3}$$

here, $t + t_m^{\text{wait}}(t)$ denotes the time slot at which the task begins processing, and $\left\lceil \frac{L_m(t)}{f_m^{\text{device}}\Delta/c_m} \right\rceil$ represents the number of time slots required for mobile device $m \in \mathcal{M}$ to process the task. The term $t + \tau_m - 1$ indicates the task's deadline. The min operator implies that the task is either fully completed by its deadline or explicitly discarded, in which case $t_m^{\text{comp}}(t) = t + \tau_m - 1$. Notably, $t_m^{\text{comp}}(0) = 0$.

In addition, local execution of a task incurs energy consumption, which can be expressed as

$$E_{\text{loc}_m} = \kappa_{\text{device}} L_m(t)\, c_m \left(f_m^{\text{device}}\right)^2, \tag{4}$$

where $\kappa_{\text{device}}$ denotes the energy consumption coefficient of the mobile device.

*3.2.2 Transmission Queue*

Following selection, the task is immediately scheduled in the transmission buffer for transfer to the chosen edge node. We adopt a wireless network model in which mobile devices conduct data transmission over orthogonal channels. Wireless signal delivery from mobile devices to edge nodes is impaired by both path loss and small-scale fading. Let $t_m^{\text{tran}}(t) \in \mathcal{T}$ signify the specific time slot in which a task $\text{Task}_m(t)$, placed into the transmission queue of mobile device $m \in \mathcal{M}$ at time $t \in \mathcal{T}$, is either successfully forwarded or dropped. If task $\text{Task}_m(t)$ is not designated for transmission (i.e., $\text{Task}_m(t) = 0$), then $t_m^{\text{tran}}(t) = 0$. Notably, tasks placed in this buffer do not commence processing instantly; they must wait until all preceding workloads in the queue are completely sent. The associated waiting duration for task $m$ at time $t$ in the transmission buffer, denoted by $t_m^{\text{wait}}(t)$ (in time slots), is calculated as follows:

$$t_m^{\text{wait}}(t) = \max\left\{\max_{t' \in \{0,1,\ldots,t-1\}} t_m^{\text{tran}}(t') - t + 1, 0\right\}, \tag{5}$$

here, $t \in \mathcal{T}$ denotes the current time slot, and the term $\max_{t' \in \{0,1,\ldots,t-1\}} t_m^{\text{tran}}(t')$ represents the latest time slot, prior to $t$, at which all tasks placed in the transmission queue have either been successfully transmitted or discarded. The max operator guarantees that the waiting time for task transmission is non-negative. The computation of $t_m^{tran}(t)$ is given as follows:

$$t_m^{\text{tran}}(t) = \min\left\{t + t_m^{\text{wait}}(t) + \left\lceil \sum_{n \in \mathcal{N}} \frac{y_{m,n}(t) L_m(t)}{\gamma_{m,n}^{\text{tran}} \Delta} \right\rceil - 1, t + \tau_m - 1\right\}. \tag{6}$$

Specifically, $t + t_m^{\text{wait}}(t)$ denotes the time slot at which the task begins transmission. The term $\left\lceil \sum_{n \in \mathcal{N}} \frac{y_{m,n}(t) L_m(t)}{\gamma_{m,n}^{\text{tran}} \Delta} \right\rceil$ represents the number of time slots required to complete the transmission of the task, where $\gamma_{m,n}^{\text{tran}}$ (in bits per second) denotes the transmission rate from mobile device $m$ to edge node $n$, computed as

$$r_{m,n}^{\text{tran}} = W \log_2\left(1 + \frac{|h_{m,n}|^2 P_m}{\sigma^2}\right), \quad m \in \mathcal{M},\ n \in \mathcal{N}, \tag{7}$$

where $W$ is the channel bandwidth, $|h_{m,n}|^2$ is the channel gain from mobile device $m \in \mathcal{M}$ to edge node $n \in \mathcal{N}$, $P_m$ is the transmission power of the mobile device $m$, and $\sigma^2$ is the noise power at the receiving edge node. For tractability, the transmission rate $r_{m,n}^{\text{tran}}$ is assumed to be constant for each device–server pair during a time slot. Under this assumption, the transmission delay mainly depends on task data size. This abstraction captures the dominant impact of communication latency while avoiding excessive modeling complexity at the MAC and routing layers. The term $t + \tau_m - 1$ specifies the deadline of the task. The min operator ensures that a task is either fully transmitted before its deadline or discarded (i.e., $t_m^{\text{tran}}(t) = t + \tau_m - 1$). In particular, $t_m^{\text{tran}}(0) = 0$.

Task offloading also incurs energy consumption, which can be calculated as

$$E_{\text{off}_m} = P_m\, T_{\text{off}}\, \Delta, \tag{8}$$

where $P_m$ denotes the transmission power of mobile device $m$, and $T_{\text{off}}(t)$ represents the number of time slots required to transmit the task. That is to say, $T_{off} = \left\lceil \sum_{n \in \mathcal{N}} \frac{y_{m,n}(t) L_m(t)}{\gamma_{m,n}^{\text{tran}} \Delta} \right\rceil$.

### 3.3 Edge Node Model

Within the defined system, every edge node $n \in \mathcal{N}$ maintains $M$ distinct queues, each dedicated to a specific mobile device $m \in \mathcal{M}$. When an offloaded task $\text{Task}_m(t)$ is received from device $m$ in time slot $t \in \mathcal{T}$, the node places it into the dedicated queue during the subsequent time slot. We define $\text{Task}_{m,n}^{\text{edge}}(t)$ as the task residing in $n$'s queue for $m$ at time $t$, with $L_{m,n}^{\text{edge}}(t)$ representing its data size. Should device $m$ not offload any task to node $n$ at time $t$, both the task identifier and size are set to zero: $\text{Task}_{m,n}^{\text{edge}}(t) = 0$ and $L_{m,n}^{\text{edge}}(t) = 0$.

We next present a detailed description of the queues at the edge nodes.

#### 3.3.1 Queues at Edge Nodes

Task queues at every edge node adhere to the First-Come, First-Served (FCFS) protocol. Let $q_{m,n}^{\text{edge}}(t)$ (in bits) specify the queue size for mobile device $m \in \mathcal{M}$ awaiting service at edge node $n \in \mathcal{N}$ in time slot $t \in \mathcal{T}$. A queue is deemed active if, at time $t$, it either receives a new task from device $m\left(L_{m,n}^{\text{edge}}(t) > 0\right)$ or was non-empty in the preceding time slot $t-1\left(q_{m,n}^{\text{edge}}(t-1) > 0\right)$. The set of active queues at node $n$ in time $t$ is denoted by $\mathcal{B}_n(t)$, which is formally defined as:

$$\mathcal{B}_n(t) = \left\{ m \in \mathcal{M} \;\middle|\; L_{m,n}^{\text{edge}}(t) > 0 \text{ or } q_{m,n}^{\text{edge}}(t-1) > 0 \right\}. \tag{9}$$

The number of active queues at edge node $n$ is given by the cardinality of the set $\mathcal{B}_n(t)$:

$$B_n(t) = |\mathcal{B}_n(t)|. \tag{10}$$

Every edge node ($n \in \mathcal{N}$) operates with a dedicated, single CPU to execute tasks awaiting in its queues. The computational power of edge node $n$ is defined by $f_n^{\text{edge}}$ (in CPU cycles per second). In any time slot $t \in \mathcal{T}$, the available CPU resources are divided equally among all currently active queues at node $n$. As the set of active queues, $B_n(t)$, changes dynamically and is unknown *a priori*, the computational capability assigned to each queue fluctuates over time. The queue length, $q_{m,n}^{\text{edge}}(t)$, consequently evolves according to:

$$q_{m,n}^{\text{edge}}(t) = \max\left\{ q_{m,n}^{\text{edge}}(t-1) + L_{m,n}^{\text{edge}}(t) - \frac{f_n^{\text{edge}}\Delta}{c_m B_n(t)} \mathbf{1}(m \in \mathcal{B}_n(t)) - e_{m,n}^{\text{edge}}(t),\ 0 \right\}, \tag{11}$$

here, $q_{m,n}^{\text{edge}}(t-1)$ denotes the queue length in the previous time slot, $L_{m,n}^{\text{edge}}(t)$ is the number of bits of tasks arriving at time slot $t$, $\frac{f_n^{\text{edge}}\Delta}{c_m B_n(t)} \mathbf{1}(m \in \mathcal{B}_n(t))$ represents the number of bits processed, $e_{m,n}^{\text{edge}}(t)$ (in bits) denotes the number of discarded bits at the end of time slot $t$, and the max operator ensures that the queue length remains non-negative.

#### 3.3.2 Task Processing or Dropping at Edge Nodes

We denote by $t_{m,n}^{\text{comp}}(t) \in \mathcal{T}$ the time slot at which the task $\text{Task}_{m,n}^{\text{edge}}$ is either completed or dropped at the edge node. If $\text{Task}_{m,n}^{\text{edge}} = 0$, then $t_{m,n}^{\text{comp}}(t) = 0$, and particularly $t_{m,n}^{\text{comp}}(0) = 0$. The completion time $t_{m,n}^{\text{comp}}(t)$ satisfies the following constraints:

$$\sum_{t'=\hat{t}_{m,n}^{\text{edge}}(t)}^{t_{m,n}^{\text{comp}}(t)} \frac{f_n^{\text{edge}}\Delta}{c_m B_n(t')} \mathbf{1}\left(m \in \mathcal{B}_n(t')\right) \geq L_{m,n}^{\text{edge}}(t), \tag{12}$$

$$\sum_{t'=\hat{t}^{\text{edge}}_{m,n}(t)}^{t^{\text{comp}}_{m,n}(t)-1} \frac{f_n^{\text{edge}}\Delta}{c_m B_n(t')}\mathbf{1}\big(m \in \mathcal{B}_n(t')\big) < L^{\text{edge}}_{m,n}(t). \tag{13}$$

That is, the size of task $\text{Task}^{\text{edge}}_{m,n}$ is no greater than the cumulative processing capacity allocated to mobile device $m$ by edge node $n$ over the time slots from $\hat{t}^{\text{edge}}_{m,n}(t)$ to $t^{\text{comp}}_{m,n}(t)$, and strictly greater than the cumulative capacity over the time slots from $\hat{t}^{\text{edge}}_{m,n}(t)$ to $t^{\text{comp}}_{m,n}(t)-1$. Here, $\hat{t}^{\text{edge}}_{m,n}(t)$ denotes the starting time slot when task $\text{Task}^{\text{edge}}_{m,n}$ begins processing, computed as

$$\hat{t}^{\text{edge}}_{m,n}(t) = \max\left\{t, \max_{t'\in\{0,1,\ldots,t-1\}} t^{\text{comp}}_{m,n}(t')+1\right\}. \tag{14}$$

In other words, a task is either processed immediately upon arrival or waits until all previously arrived tasks before time slot $t$ are completed. The max operator ensures that the processing start time of a task is not earlier than its arrival time.

Similarly, task execution at an edge node also incurs energy consumption, which can be expressed as

$$E_{\text{edge}_n} = \kappa_{\text{edge}}\, L^{\text{edge}}_{m,n}(t)\, c_m \left(f_n^{\text{edge}}\right)^2, \tag{15}$$

where $\kappa_{\text{edge}}$ denotes the energy consumption coefficient of the edge node.

## 4 Problem Formulation

Based on the system model in Section 3, our objective is to jointly minimize the task processing delay and energy consumption for all mobile devices in the MEC system.

Let $T_m(t)$ and $E_m(t)$ denote the total delay and total energy consumption of task $\text{Task}_m(t)$, respectively, defined as:

$$T_m(t) = \big(1-x_m(t)\big)\big(t^{\text{comp}}_m(t)-t+1\big) + x_m(t)\big(t^{\text{comp}}_{m,n}(t')-t+1\big), \tag{16}$$

$$E_m(t) = \big(1-x_m(t)\big)E_{\text{loc}_m} + x_m(t)\big(E_{\text{off}_m}+E_{\text{edge}_n}\big), \tag{17}$$

where $x_m(t)\in\{0,1\}$ indicates whether the task is offloaded ($x_m(t)=1$) or executed locally ($x_m(t)=0$), and $y_{m,n}(t)\in\{0,1\}$ specifies which edge node $n$ receives the task if offloaded. The optimization problem can then be formulated as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \sum_{t\in\mathcal{T}}\sum_{m\in\mathcal{M}}\Big[\alpha T_m(t)+\beta E_m(t)\Big] \\ \text{s.t.} \quad & \text{Task offloading constraints in Eqs. (1)–(3), (5)–(7), (11)–(14),} \end{aligned} \tag{18}$$

where $\alpha$ and $\beta$ are weighting coefficients reflecting the relative importance of delay and energy, and $\mathbf{x}=\{x_m(t)\}, \mathbf{y}=\{y_{m,n}(t)\}$ are the decision variables.

The problem is combinatorial and NP-hard due to the binary offloading decisions, indivisible tasks, and dynamic coupling of tasks and edge server resources. The time-varying task arrivals, heterogeneous workloads, and varying edge computing capabilities further increase the complexity. Consequently, exact solutions are computationally intractable for large-scale MEC systems, motivating the use of a reinforcement learning approach that enables mobile devices to learn adaptive task offloading strategies balancing delay, energy, and task drop rates in real time.

## 5 DRL-Based Pre-Allocation Strategy

To solve the optimization challenge as defined in Section 4, we introduce a new A2C scheme, termed *PreAlloc-A2C*. The original optimization objective is reformulated as a reward maximization problem within a deep reinforcement learning framework. The subsequent subsections detail the essential elements of PreAlloc-A2C, which encompass the state representation, action design, reward function, neural network architecture, and policy update procedure.

### *5.1 State*

The state represents the environmental information perceived by a mobile device when making a task offloading decision. In this study, the state of mobile device $m$ at time slot $t$ is defined as

$$\mathbf{s}_m(t) = \left(L_m(t),\ t_m^{\text{comp}}(t),\ t_m^{\text{tran}}(t),\ \mathbf{q}_m^{\text{edge}}(t-1),\ \mathbf{H}(t)\right), \tag{19}$$

where the components are detailed as follows: (i) $L_m(t)$ denotes the size of the task; (ii) $t_m^{\text{comp}}(t)$ represents the waiting time for computation; (iii) $t_m^{\text{tran}}(t)$ denotes the waiting time for transmission; (iv) $\mathbf{q}_m^{\text{edge}}(t-1) = \left(q_{m,n}^{\text{edge}}(t-1),\ n \in \mathcal{N}\right)$ describes the queue lengths corresponding to device $m$ at all edge nodes $n \in \mathcal{N}$; (v) $\mathbf{H}(t)$ represents the historical load levels of edge nodes during the previous $T^{\text{step}}$ time slots. Specifically, $\mathbf{H}(t)$ is a matrix of size $T^{\text{step}} \times N$, where each element $\{H(t)\}_{i,j}$ is defined as

$$\{H(t)\}_{i,j} = B_j(t - T^{\text{step}} + i - 1), \tag{20}$$

indicating the number of active queues at edge node $j$ during time slot $t - T^{\text{step}} + i - 1$. Accordingly, the state space can be expressed as

$$\mathcal{S} = \Lambda \times \{0, 1, \ldots, T\}^2 \times \mathcal{Q}^N \times \{0, 1, \ldots, M\}^{T^{\text{step}} \times N}, \tag{21}$$

where $\Lambda$ denotes the set of possible task sizes over $T$ time slots, and $\mathcal{Q}$ represents the set of feasible queue lengths of edge nodes within $T$ time slots.

In this work, we assume that mobile devices can access certain edge server information, such as queue length and historical workload. In practical MEC systems, such information can be obtained through periodic status broadcasting or lightweight signaling from edge servers via control channels. In addition, many MEC architectures include edge orchestrators or base stations that maintain system-level information and can assist in disseminating server status. Therefore, the adopted assumption is reasonable and consistent with practical MEC system designs.

### *5.2 Action*

In this study, the action corresponds to the task allocation decision of each mobile device. Specifically, at the beginning of time slot $t \in \mathcal{T}$, if a task $\text{Task}_m(t)$ arrives at mobile device $m \in \mathcal{M}$, the device must select between executing the task locally or transferring it to an edge server. If offloading is chosen, the device further selects the target edge server for task assignment. The action of device $m$ at time slot $t$ is defined as

$$a_m(t) = \left(x_m(t), \mathbf{y}_m(t)\right), \tag{22}$$

where $\mathbf{y}_m(t) = \left(y_{m,n}(t),\ n \in \mathcal{N}\right)$. The action space is denoted by $\mathcal{A} = \{0,1\}^{1+N}$, where the first dimension $x_m(t)$ indicates the local/offload decision, and the subsequent $N$ dimensions in $\mathbf{y}_m(t)$ specify the selected edge node.

### 5.3 Reward Function

The devised reward mechanism elucidates the objective in mobile device learning, directly influencing the quality of the policy learned by these devices. Our reward formulation is formulated to simultaneously account for the balance between task latency and power expenditure, while penalizing unfinished tasks. Formally, the reward at time slot $t$ is defined as

$$r(t) = \begin{cases} -2t_{\text{delay}}^{\max}, & \text{if the task is unfinished,} \\ -\left(\alpha T_m + \beta E_m\right), & \text{otherwise,} \end{cases} \tag{23}$$

where $T_m$ denotes the actual task delay, $t_{\text{delay}}^{\max}$ is the maximum allowable delay, $E_m$ is the energy consumption. The coefficients $\alpha$ and $\beta$ are non-negative weights used to establish the relative priority between task latency and power expenditure. For unfinished tasks, a penalty proportional to the maximum delay is imposed to strongly discourage task dropping. Compared to a standard linear penalty, this design amplifies the cost of task failure and encourages the agent to prioritize successful task completion. This design ensures that unfinished tasks incur a severe penalty, whereas completed tasks receive a reward inversely proportional to their normalized delay and energy consumption.

### 5.4 Neural Network Architecture

We employ the *PreAlloc-A2C* network to approximate the mapping from states to actions. The overall architecture for a mobile device $m \in \mathcal{M}$ is illustrated in Fig. 2. Specifically, the state information of the environment is first fed into the input layer. To capture temporal dynamics of edge node utilization, we incorporate an LSTM layer that predicts near-future load levels based on historical records. The state representation is then decomposed into two components:

$$\mathbf{s}_m^{task}(t) = \left(L_m(t),\, t_m^{\text{comp}}(t),\, t_m^{\text{tran}}(t)\right), \tag{24}$$

which denotes task-related features, and

$$\mathbf{s}_m^{node}(t) = \left(\mathbf{q}_m^{\text{edge}}(t-1),\, \mathbf{H}(t)\right), \tag{25}$$

which denotes node-related features. Both task and node features are transformed into a unified embedding space through dedicated embedding networks, ensuring compact representations that preserve essential information. These embeddings serve as inputs to the decision network: the actor branch generates allocation scores for each edge node, which are normalized via a softmax function, resulting in a selection probability across candidate actions. Conversely, the critic branch is responsible for determining the utility of potential actions. Finally, an execution action is then sampled from this distribution, yielding the task allocation decision along with its corresponding value estimate. After the action is executed, the environment returns a reward signal, which is leveraged to jointly optimize the policy and value networks in an iterative manner, thereby enhancing subsequent task allocation decisions. A comprehensive breakdown of the five network components is presented next, including the *LSTM* network, the *task feature embedding* network, the *node feature embedding* network, the *actor* network, and the *critic* network.

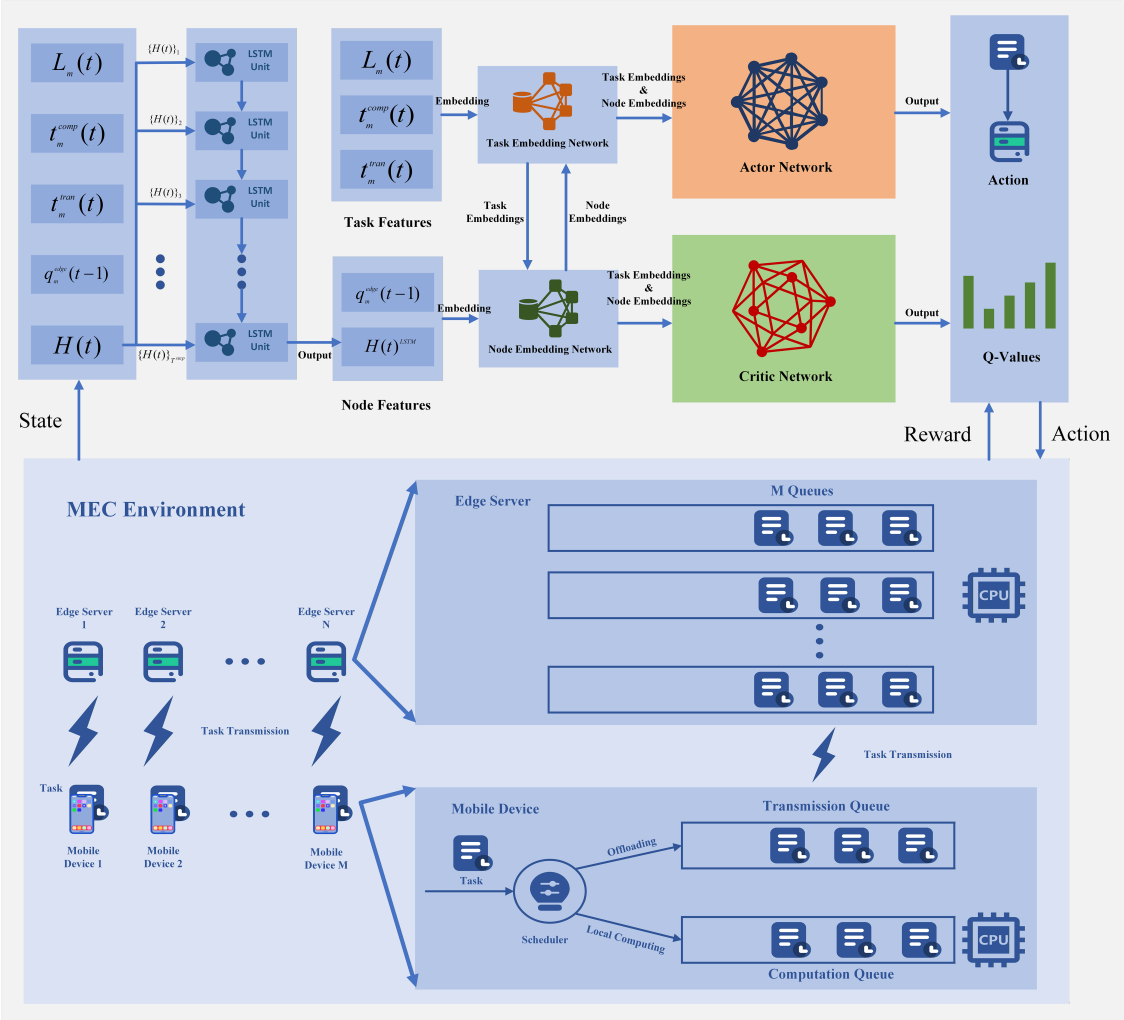



**Figure 2:** Neural network architecture.

### *5.4.1 LSTM Network*

We adopt a *LSTM* network to model the time-dependent evolution of edge nodes' workloads. Specifically, the input to the LSTM is the historical workload matrix $\mathbf{H}(t) \in \mathbb{R}^{T^{\text{step}} \times N}$, where $T^{\text{step}}$ signifies the total past time steps and $N$ indicates the total edge nodes. The LSTM is unrolled over $T^{\text{step}}$ time steps, where each row of $\mathbf{H}(t)$ is sequentially processed by one LSTM cell. This recurrent structure enables the model to track variations in workload levels across time slots and to learn the underlying temporal dependencies. The hidden state of the final LSTM cell represents the aggregated temporal information, which serves as a compact encoding of the workload trajectory and provides predictive cues for the near-future workload of edge nodes.

### *5.4.2 Task and Node Feature Embedding Networks*

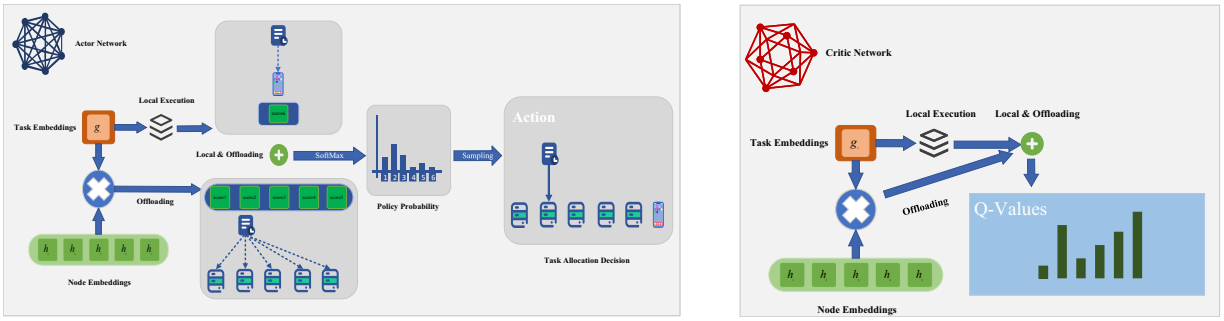
To obtain expressive representations of task characteristics, we employ a feed-forward embedding network. The input to this module is the raw task feature vector $\mathbf{s}_m^{task}(t) = \left(L_m(t),\ t_m^{\text{comp}}(t),\ t_m^{\text{tran}}(t)\right)$, which is first mapped into a hidden space through a linear transformation, followed by a ReLU nonlinearity.

A subsequent fully connected layer projects the intermediate representation into a compact embedding vector. This task embedding effectively encodes the salient attributes of each incoming task, thereby facilitating downstream decision-making.

For edge node representation, we design a dedicated embedding network that integrates both the instantaneous queue lengths $\mathbf{q}_m^{\text{edge}}(t-1)$ and the predicted workload dynamics $\mathbf{H}(t)$ obtained from the LSTM module. Specifically, the queue length of each node is first mapped into a hidden space through a linear layer, while the LSTM output is projected into the same dimensional space to capture temporal load information. The two representations are then concatenated and fused via another linear transformation, yielding the final node embedding. This design ensures that both static and temporal characteristics of each node are jointly captured, providing a comprehensive representation for task offloading decisions.

### *5.4.3 Actor and Critic Networks*

The actor network is responsible for generating a probability distribution over candidate offloading decisions. Its input consists of two components: the embedded task representation and the embedded node representation. The task embedding is obtained via the task feature embedding network, while the node embedding integrates both instantaneous queue lengths and workload dynamics predicted by the LSTM module. The actor network computes the compatibility between task and node embeddings through a bilinear interaction, yielding a score for each node. In addition, a separate score is computed for local execution. The resulting logits are normalized with a softmax function to produce a probability distribution over all possible offloading actions, including local computation. During decision making, the final action is selected through probabilistic sampling from this distribution, as illustrated in Fig. 3a.




**(a)** **(b)**

**Figure 3:** Architectures of the actor and critic networks: (**a**) actor network and (**b**) critic network.

The critic network is designed to estimate the state–action value for each candidate decision. Similar to the actor, it receives both the task embedding and the node embedding as inputs. The critic evaluates the potential utility of offloading a given task to each node by computing the inner product between task and node embeddings. In parallel, a separate linear projection estimates the value of local execution. These estimates are concatenated to form the final vector of Q-values, one for each possible offloading action. By providing value estimates, the critic guides the actor's policy updates through advantage estimation, thereby stabilizing and accelerating the learning process, as shown in Fig. 3b.

### *5.5 Policy Learning*

We adopt the Advantage Actor–Critic (A2C) framework to jointly optimize the actor and critic networks. The critic network $Q_\phi(s,a)$ is trained to approximate the state-action value function, while the actor network $\pi_\theta(a|s)$ is optimized to improve the policy according to the estimated advantage.

Temporal-Difference: Target Given a transition $(s_t, a_t, r_t, s_{t+1})$, the temporal-difference (TD) target is defined as

$$y_t = r_t + \gamma Q_\phi(s_{t+1}, a_{t+1}), \tag{26}$$

where $\gamma \in (0,1)$ is the discount factor and $a_{t+1} \sim \pi_\theta(\cdot|s_{t+1})$ is the next action. The TD error is then computed as

$$\delta_t = y_t - Q_\phi(s_t, a_t). \tag{27}$$

Actor Update: The policy parameters $\theta$ are updated to maximize the expected cumulative reward. Specifically, the actor loss is defined as

$$\mathcal{L}_{\text{actor}}(\theta) = -\mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\left[\log \pi_\theta(a_t|s_t)\,\delta_t\right], \tag{28}$$

which encourages actions leading to positive advantages while discouraging actions with negative advantages.

Critic Update: The critic parameters $\phi$ are optimized by minimizing the mean squared error between the TD target and the estimated $Q$-value:

$$\mathcal{L}_{\text{critic}}(\phi) = \mathbb{E}_{(s_t,a_t)}\left[\left(Q_\phi(s_t,a_t) - y_t\right)^2\right]. \tag{29}$$

Joint Training: During training, gradients of both losses are backpropagated, and the parameters $\theta$ and $\phi$ are updated simultaneously using stochastic gradient descent:

$$\theta \leftarrow \theta - \alpha_\theta \nabla_\theta \mathcal{L}_{\text{actor}}, \quad \phi \leftarrow \phi - \alpha_\phi \nabla_\phi \mathcal{L}_{\text{critic}}, \tag{30}$$

where $\alpha_\theta$ and $\alpha_\phi$ are the learning rates for the actor and critic networks, respectively. To clearly illustrate the training procedure of the proposed PreAssign framework, we summarize the overall workflow in Algorithm 1.

**Algorithm 1:** Training Procedure of the Pre-A2C method

**Input:** Quantity of mobile devices $M$, count of edge nodes $N$, total episodes $E$, maximum delay $D_{\max}$.
**Output:** Trained actor and critic networks for each mobile device.
Initialize the environment $\mathcal{E}$ with $(M, N, D_{\max})$;
**for** *each mobile device* $m \in \mathcal{M}$ **do**
  Initialize actor network and critic network;
**for** *episode* = 1 **to** $E$ **do**
  Generate task arrival matrix $\mathbf{B} \sim \mathcal{U}(b_{\min}, b_{\max})$;
  Reset environment and obtain initial states $\mathbf{s}_m = \left(L_m,\ t_m^{\text{comp}},\ t_m^{\text{tran}},\ \mathbf{q}_m^{\text{edge}},\ \mathbf{H}\right), m \in \mathcal{M}$;
  **while** *not terminated* **do**
    **for** *each mobile device m* **do**
      Extract task embedding: $\mathbf{z}_m^t = f_t(L_m, t_m^{\text{comp}}, t_m^{\text{tran}})$;

(Continued)

**Algorithm 1 (continued)**

Predict server load: $\mathbf{H}' = f_h(\mathbf{H})$;
Extract node embedding: $\mathbf{z}_m^n = f_n(\mathbf{q}_m^{\text{edge}}, \mathbf{H}')$;
Compute assignment scores for all servers using actor network;
Compute Q-values for all possible offloading actions using critic network;
Select action $a_m$ via policy sampling;
Execute actions $a_m$ in $\mathcal{E}$;
Observe next states $\mathbf{s}'_m$, processing delay $d_m$, energy consumption $e_m$, and task drop ratio $dr_m$;
Compute reward:

$$r_m = \begin{cases} -2D_{\max}, & \text{if task unfinished,} \\ -(0.5d_m + 0.5e_m), & \text{otherwise.} \end{cases}$$

Update recurrent state $\mathbf{s}_m \leftarrow \mathbf{s}'_m$;
**for** *each mobile device m* **do**
Update critic network parameters $\phi_m$ and actor network parameters $\theta_m$;
Apply soft update to target networks with coefficient $\tau$;
Record episode metrics: average reward, delay, energy consumption, and drop ratio;
Save trained models and training statistics;

### *5.6 Computational Complexity Analysis*

The computational complexity of the proposed *PreAlloc-A2C* method can be analyzed in both training and online execution phases.

**(1)** **Training Complexity:** During training, the algorithm iterates over $E$ episodes. In each episode, for every time slot, each of the $M$ mobile devices performs task embedding extraction, server load prediction via the LSTM network, node embedding extraction, and actor and critic forward passes to compute assignment scores and Q-values for $N$ edge nodes. Assuming the complexity of each embedding extraction and network forward pass is $O(L_{\text{NN}})$, the complexity per episode is approximately $O\Big(T \cdot M \cdot N \cdot L_{\text{NN}}\Big)$, leading to an overall training complexity of $O\Big(E \cdot T \cdot M \cdot N \cdot L_{\text{NN}}\Big)$.

**(2)** **Online Execution Complexity:** Once the networks are trained, each mobile device independently makes offloading decisions at each time slot. The online decision complexity for all devices is $O(M \cdot N \cdot L_{\text{NN}})$, which is significantly lower than the training complexity and can be executed in real time.

It is worth noting that the proposed framework does not require full model training or heavy computation on mobile devices. The training process is performed offline or at edge servers, while mobile devices only execute lightweight inference during deployment. The LSTM-based workload prediction module can be deployed at edge servers, which typically have sufficient computational resources, and the predicted information can be shared with mobile devices. During online decision-making, the actor network involves only a few forward passes, resulting in low computational overhead that is suitable for real-time execution on resource-constrained devices. Therefore, the proposed method can support real-time task offloading decisions with negligible latency. This design is consistent with practical MEC systems, where complex computation is offloaded to edge servers while mobile devices perform lightweight decision-making.

## 6 Performance Evaluation

This section commences by outlining the experimental configuration and performance indicators, subsequently listing the baseline approaches employed for comparison. We then present the experimental results and provide a detailed performance analysis.

### *6.1 Experimental Setup*

Our setup involves a network featuring 50 mobile clients and 5 edge facilities. Key settings governing the simulation context are itemized in Table 1. The key hyperparameters for all algorithms are configured as follows. We configure the learning rate to 0.001, alongside a fixed discount factor of 0.9. Network parameters are updated using the Adam optimization mechanism. For all DQN-based algorithms, the mini-batch size is set to 16 to enhance sample efficiency and stabilize learning. The probability of random exploration is reduced from 1.0 to 0.01 in a linear fashion during training, facilitating a trade-off between exploration and exploitation. The PSO algorithm employs 30 particles and a maximum of 200 iterations for convergence. The inertia weight is configured as 0.7 to balance global exploration and local exploitation. The cognitive and social learning factors are equally fixed at 1.5, ensuring a symmetric influence between individual and global best solutions. In addition to the general hyperparameter settings, the configurations of each algorithm are carefully designed to ensure fair comparison and reproducibility. For baseline methods, such as DQN, DDQN, and Dueling DQN, we adopt network architectures consistent with their original implementations. The dueling architecture explicitly separates value and advantage streams, while DDQN employs a target network for decoupled action evaluation. All hyperparameters are selected based on a combination of widely adopted settings in prior literature and empirical tuning. Specifically, we perform preliminary experiments to tune key parameters such as learning rate and exploration schedule, and apply the same tuning protocol across all methods to ensure fairness. All reported results are obtained by averaging over five independent runs to mitigate the impact of randomness and ensure statistical reliability.

**Table 1:** Parameter settings.

| Parameter | Value |
|---|---|
| $\Delta$ | 0.1 s |
| $f_m^{\text{device}}, m \in \mathcal{M}$ | 2.5 GHz [37] |
| $f_n^{\text{edge}}, n \in \mathcal{N}$ | 41.8 GHz [37] |
| $r_{m,n}, m \in \mathcal{M}, n \in \mathcal{N}$ | 14 Mbps [38] |
| $\lambda_m(t), m \in \mathcal{M}, t \in \mathcal{T}$ | $\{2.0, 2.1, \ldots, 5.0\}$ Mbits [39] |
| $c_m, m \in \mathcal{M}$ | 0.297 gigacycles per Mbits [39] |
| $\tau_m, m \in \mathcal{M}$ | 10 time slots (i.e., 1 s) |
| $\kappa_{\text{device}}$ | $5 \times 10^{-27}$ [40] |
| $\kappa_{\text{edge}}$ | $5 \times 10^{-31}$ [29] |
| Task arrival probability | 0.3 |

### *6.2 Evaluation Metrics*

The following metrics are considered for assessing the effectiveness of the proposed scheme.

- **Task Delay:** Task delay measures the average time required to complete computation tasks, including both transmission and processing latency. This metric provides a direct indication of the system's agility and is particularly critical for latency-sensitive mobile applications. More specifically, the definition of

task delay depends on the execution mode. If a task is executed locally on the mobile device, the task delay corresponds solely to the local processing latency. In contrast, if the task is offloaded to an edge server, the task delay consists of two components: (i) transmission delay from the mobile device to the edge server, and (ii) processing delay at the edge server. The transmission delay is modeled according to the system formulation in Section 3.2.2.

- **Energy Consumption:** This metric quantifies the total energy consumed during task execution, including transmission energy at mobile devices and computation energy at edge servers. Minimizing energy consumption is essential to prolong device battery lifetime and enhance system sustainability. Specifically, the energy consumption depends on the execution mode of each task. If a task is executed locally on the mobile device, the energy consumption corresponds to the local computation energy incurred by the device. In contrast, if the task is offloaded to an edge server, the energy consumption includes two components: (i) the transmission energy consumed by the mobile device during data offloading, and (ii) the computation energy consumed by the edge server for task processing.
- **Task Drop Rate:** Task drop rate represents the proportion of tasks that fail to be completed before their deadlines. This measure indicates the system's capacity to handle resource constraints and ensures quality-of-service (QoS) guarantees for end users.

Together, these measures collectively offer a comprehensive perspective on the system's performance concerning efficiency, sustainability, and reliability.

### *6.3 Baseline Algorithms*

To evaluate our method, we compare it with the following baselines:

- **PSO:** A swarm-intelligence-based stochastic optimizer that updates particle positions and velocities according to individual and global bests, achieving fast convergence to near-optimal solutions.
- **DQN:** Combines Q-learning with deep networks to approximate the state-action value function, employing experience replay and a target network for stable learning over large state spaces.
- **DDQN:** Addresses DQN's overestimation by decoupling action selection (online network) from value evaluation (target network), improving prediction accuracy and stability.
- **Dueling DQN:** Splits the Q-function into state-value and advantage networks, enabling more robust value estimation, particularly when multiple actions yield similar outcomes.
- **A2C:** A synchronous actor-critic method using the advantage function to reduce gradient variance, jointly training actor and critic networks for stable policy learning in discrete or continuous action spaces.

It is worth noting that these baseline methods are implemented and adapted to our problem setting for a fair comparison.

### *6.4 Experimental Results and Analysis*

#### *6.4.1 Convergence Analysis of the Proposed Algorithm*

For the assessment of the convergence properties of the proposed algorithm, we first examine its performance under various learning rate settings, as depicted in Fig. 4a. The results demonstrate that, despite minor fluctuations in the early stages, all learning rates eventually converge to a stable reward region. Notably, learning rates of 0.001 and 0.0001 exhibit smoother convergence, whereas larger values (0.1 and 0.01) exhibit marginally more fluctuations during the intermediate episodes. These observations indicate that the algorithm is robust to a wide range of learning rates, with smaller rates providing more stable convergence behavior.

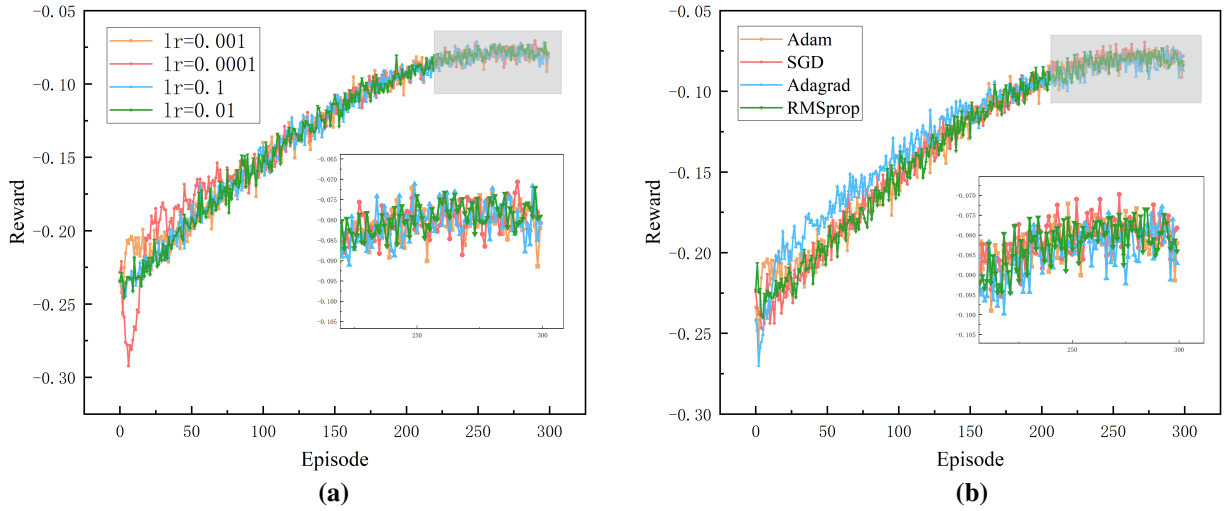



**Figure 4:** Evaluating the convergence behavior under diverse experimental setups: (**a**) different learning rates and (**b**) different optimizers.

Furthermore, Fig. 4b compares the performance of different optimization strategies, including Adam, SGD, Adagrad, and RMSprop. It is observed that all optimizers are capable of driving the algorithm toward convergence, but Adam and RMSprop achieve slightly faster and more stable improvements in reward, while Adagrad demonstrates competitive performance with relatively higher fluctuations. These findings highlight that the proposed method maintains stable convergence regardless of the underlying optimizer, which confirms its robustness and adaptability across different training settings.

### *6.4.2 Comparative Analysis of Performance Metrics*

Fig. 5 compares the performance of Pre-A2C with baseline methods (DQN, DDQN, Dueling DQN, A2C, and PSO) under varying numbers of mobile devices, evaluating task drop ratio, latency, and energy consumption. As device count increases, computational and transmission loads grow, highlighting algorithm scalability and robustness. Pre-A2C consistently outperforms all DRL baselines, achieving the lowest task drop ratio, shortest delay, and minimal energy consumption across scales. DQN-based methods show similar trends but degrade under higher device density, while conventional A2C performs slightly worse. PSO exhibits significantly poorer performance, with task drop ratio, delay, and energy consumption rising sharply as device count exceeds 80, reflecting its inability to adapt to dynamic workloads. The superior performance of Pre-A2C derives from its hierarchical state representation. By separating task-specific and node-specific features, dedicated embedding networks capture task urgency and computational characteristics, as well as server resource states. These embeddings are jointly processed in the A2C framework to generate assignment scores, enabling effective task–server matching. Conventional DRL methods, relying on flat state vectors, fail to model these structured interactions, yielding less accurate value estimates and suboptimal scheduling. Heuristic PSO, lacking learning capabilities, cannot adapt to environmental dynamics, explaining its poor performance across all metrics.

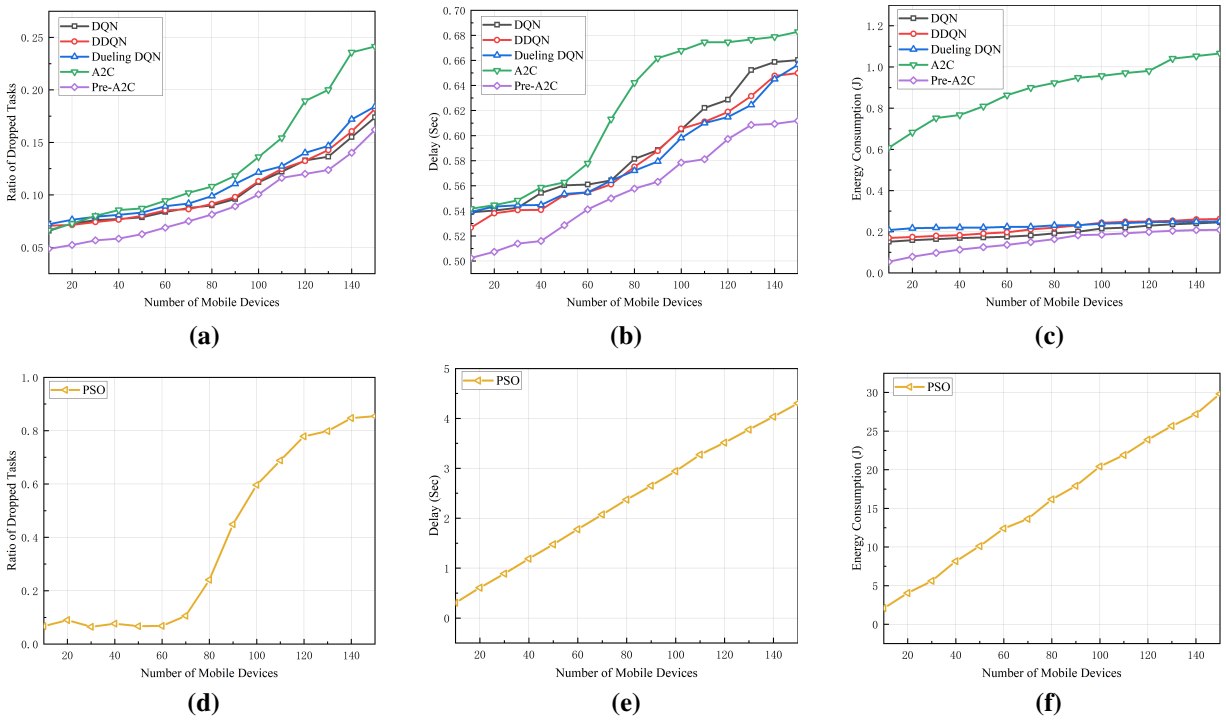



**Figure 5:** Evaluation across a varying number of mobile devices: (**a**) DRL ratio of dropped tasks; (**b**) DRL delay; (**c**) DRL energy consumption; (**d**) PSO ratio of dropped tasks; (**e**) PSO delay; (**f**) PSO energy consumption.

Fig. 6 illustrates algorithm performance under varying numbers of edge nodes. As edge capacity increases, all DRL-based methods benefit from additional resources, reducing task drop ratio, latency, and energy consumption. Pre-A2C consistently achieves the lowest drop ratio, shortest delay, and minimal energy usage across all configurations. DQN and its variants show similar but slightly inferior performance, while conventional A2C exhibits higher latency and energy consumption. PSO demonstrates poor adaptability: task drop ratio remains high under low edge capacity and energy consumption is consistently higher, reflecting its static heuristic nature. Pre-A2C's superior performance stems from its structured state representation and hierarchical feature extraction. By decomposing the global environment into task- and node-specific features, two embedding networks encode dynamic task characteristics and heterogeneous server capacities, enabling precise policy learning within the A2C framework. This design allows the agent to fully exploit additional resources as the edge network scales. In contrast, conventional DRL methods rely on flat state vectors, limiting task–node relationship modeling, while PSO lacks learning capability, yielding suboptimal scheduling under dynamic conditions.

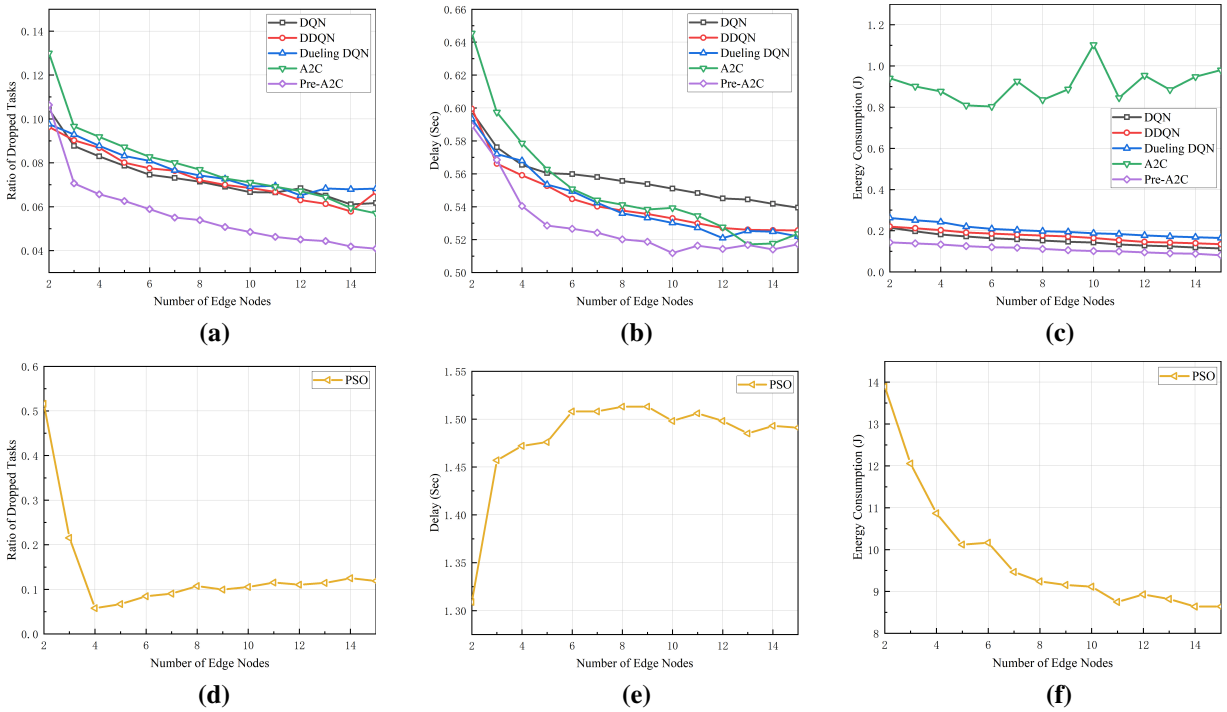



**Figure 6:** Evaluation across varying edge node quantity: (**a**) DRL ratio of dropped tasks; (**b**) DRL delay; (**c**) DRL energy consumption; (**d**) PSO ratio of dropped tasks; (**e**) PSO delay; (**f**) PSO energy consumption.

Fig. 7 compares algorithm performance under varying task arrival probabilities, evaluating task drop ratio, latency, and energy consumption. Pre-A2C consistently outperforms all baselines, maintaining the lowest drop ratio, minimal delay, and lowest energy usage across all load levels. DQN, DDQN, and Dueling DQN exhibit moderate scalability but higher drop ratio, latency, and energy consumption. Standard A2C performs worse, struggling to maintain stable scheduling under high arrival rates. PSO shows the weakest performance, with high drop ratios, energy consumption, and delays. The superior performance of Pre-A2C arises from its structured policy learning, which incorporates task–server matching information within the actor–critic framework. This enables effective modeling of complex task–resource relationships, yielding efficient scheduling and resource utilization. In contrast, DQN-based methods rely on discrete value estimation and lack smooth policy updates, while A2C without structural guidance cannot fully capture task–server interactions, leading to suboptimal energy allocation and increased latency. PSO, as a heuristic, cannot adapt to stochastic arrivals, resulting in frequent task rejections and excessive energy usage. In summary, these results demonstrate that integrating prior knowledge and adaptive learning mechanisms, as in the proposed Pre-A2C, is crucial for maintaining low latency, minimizing task loss, and achieving energy-efficient resource scheduling in dynamic edge computing environments.

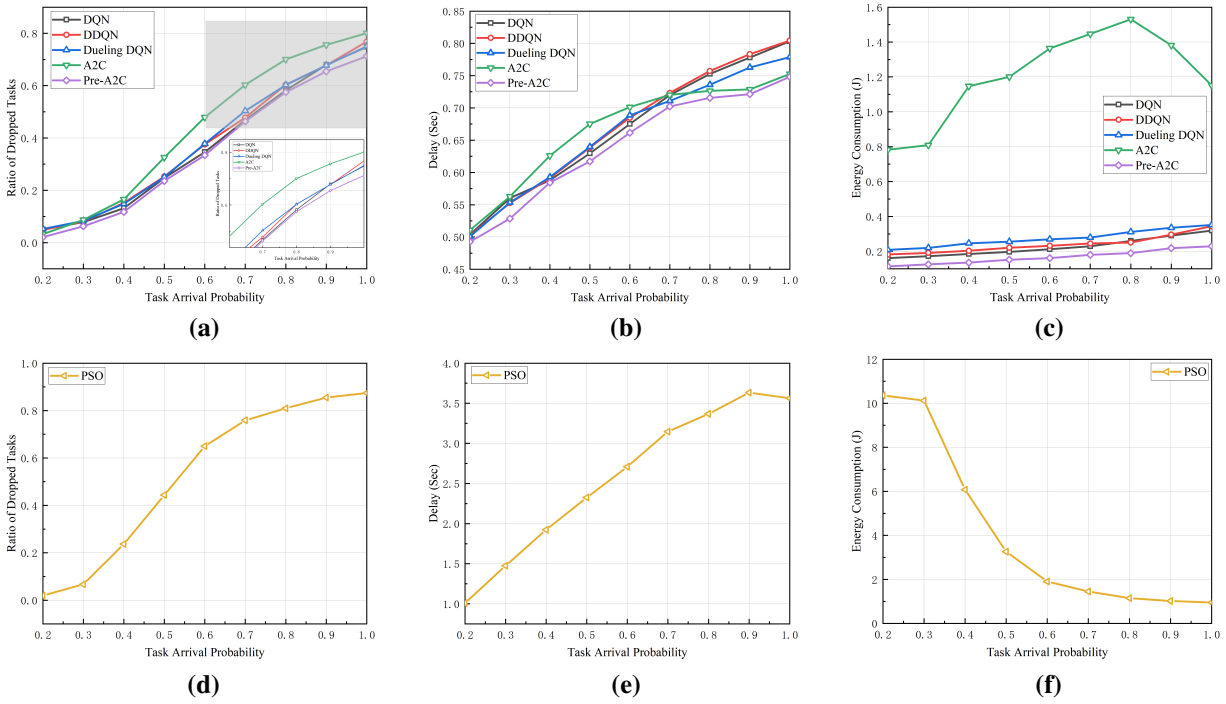



**Figure 7:** Assessment of performance across varying task arrival probabilities: (**a**) DRL ratio of dropped tasks; (**b**) DRL delay; (**c**) DRL energy consumption; (**d**) PSO ratio of dropped tasks; (**e**) PSO delay; (**f**) PSO energy consumption.

Fig. 8 compares algorithm performance under varying task deadlines, evaluating task drop ratio, latency, and energy consumption. Pre-A2C consistently achieves the lowest drop ratio, minimal delay, and energy usage across all deadlines. DQN, DDQN, and Dueling DQN exhibit similar performance, while vanilla A2C performs worse. PSO consistently lags behind, with the highest delay and energy consumption, highlighting its limited adaptability to dynamic deadlines. Pre-A2C's advantage stems from its two-stream embedding architecture, which separates task and node features and encodes them via dedicated embedding networks. This captures fine-grained task–server interactions, enabling context-aware offloading decisions that reduce task drops, latency, and energy consumption. In contrast, value-based DQN variants cannot fully exploit structural relationships, A2C lacks specialized embeddings for effective state representation, and PSO relies on static heuristics, resulting in suboptimal scheduling under dynamic conditions.

As shown in Fig. 9, with increasing mobile device processing capacity, all reinforcement learning–based methods achieve clear improvements in task drop rate and delay. Specifically, both DRL-based methods and PSO exhibit a rapid decline in task drop rate that gradually stabilizes at higher capacities, indicating that enhanced local resources effectively alleviate system load and reduce task rejections. In terms of delay, DRL-based methods consistently outperform PSO, with overall delay decreasing as processing capacity increases. Among them, Pre-A2C achieves the best performance, demonstrating superior adaptability in resource scheduling and task allocation. In contrast, PSO shows a slight increase in delay, likely due to admitting and processing more tasks as the drop rate decreases. Regarding energy consumption, DRL-based methods show an increasing trend with higher processing capacity due to the higher energy cost of increased operating frequencies. Nevertheless, Pre-A2C maintains relatively lower energy consumption among DRL methods, highlighting its energy efficiency. In contrast, PSO's energy consumption remains nearly constant, suggesting a preference for offloading tasks to edge servers. Overall, the proposed Pre-A2C method consistently achieves

superior comprehensive performance across different device capability settings, effectively reducing task drop rate and delay while maintaining energy consumption at a controlled level.

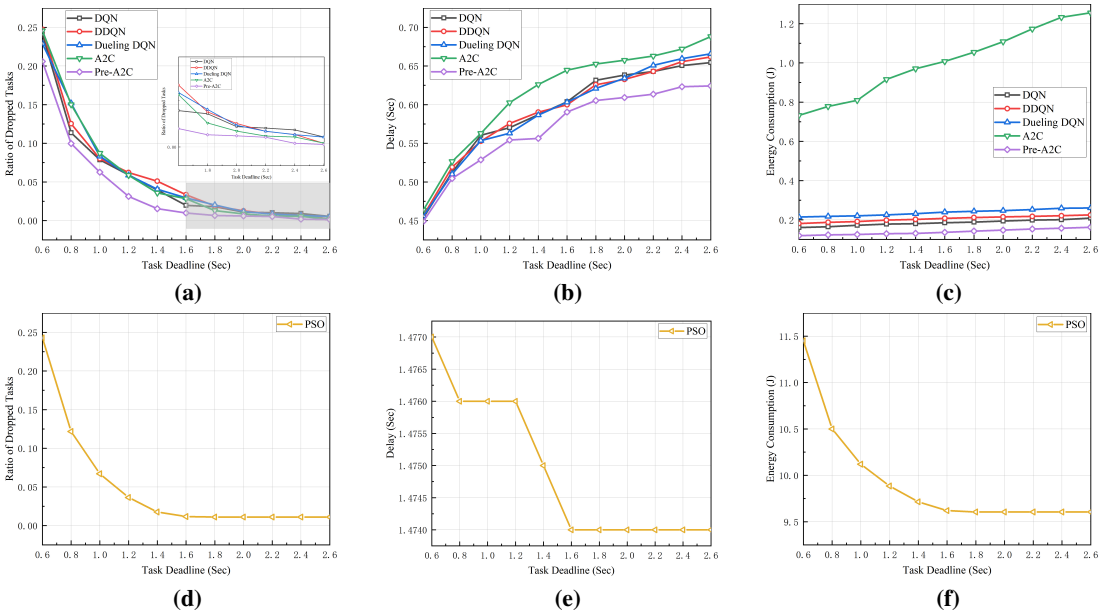



**Figure 8:** Evaluation across varying deadline constraints: (**a**) DRL ratio of dropped tasks; (**b**) DRL delay; (**c**) DRL energy consumption; (**d**) PSO ratio of dropped tasks; (**e**) PSO delay; (**f**) PSO energy consumption.

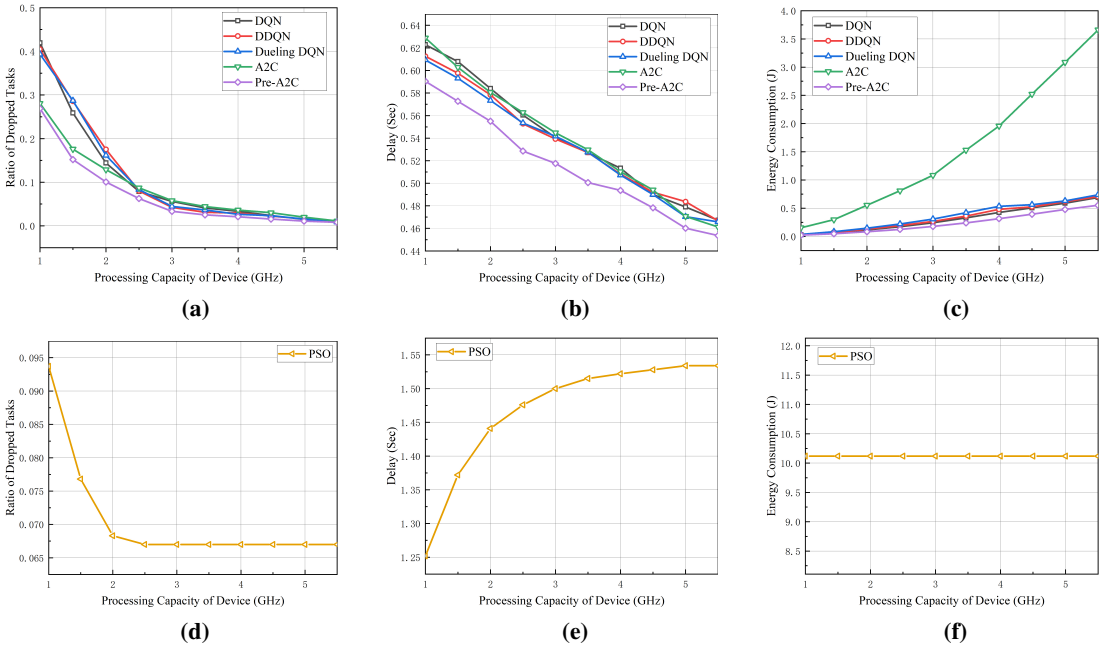



**Figure 9:** Performance under different processing capacities of each mobile device: (**a**) DRL ratio of dropped tasks; (**b**) DRL delay; (**c**) DRL energy consumption; (**d**) PSO ratio of dropped tasks; (**e**) PSO delay; (**f**) PSO energy consumption.

Collectively, the above experiments evaluate the performance of different algorithms under diverse system configurations, including varying numbers of mobile devices, edge nodes, task arrival probabilities, task deadlines, and processing capacities of mobile devices. These settings naturally introduce heterogeneity and varying workload conditions into the MEC environment. From a robustness perspective, the results in Figs. 5–9 demonstrate that the proposed PreAlloc-A2C consistently outperforms baseline algorithms across a wide range of system parameters. In particular, as the task arrival rate increases, the system becomes more challenging due to higher workload and resource contention. Nevertheless, PreAlloc-A2C maintains lower task drop rates and competitive delay and energy performance, indicating strong adaptability to varying system conditions. Furthermore, although each experiment is conducted under a fixed parameter setting, the evaluation across multiple configurations with varying workload intensity and resource conditions provides strong evidence of the model's robustness and generalization capability. These observations confirm that the proposed method is robust to variations in system parameters.

To further evaluate the scalability of the proposed PreAlloc-A2C framework, experiments are conducted under progressively larger MEC scenarios by jointly increasing the number of mobile devices and edge servers, i.e., (50 devices, 5 servers), (100 devices, 10 servers), and (150 devices, 15 servers). As illustrated in Fig. 10, as the system scale grows, all methods exhibit certain degrees of performance degradation due to increased workload and intensified resource contention. However, PreAlloc-A2C shows significantly smaller performance degradation compared with baseline methods. In particular, both the task drop rate and delay increase only marginally, indicating strong robustness under heavy-load conditions. Moreover, across all three scales, PreAlloc-A2C consistently achieves the lowest task drop rate, lowest energy consumption, and competitive delay performance, demonstrating its effectiveness in coordinating task offloading decisions even as the system becomes more complex. With the increase in both devices and servers, the decision space expands rapidly, yet the proposed method is still able to maintain efficient scheduling, suggesting that the learned policy generalizes well to larger environments and successfully captures underlying resource allocation patterns. Overall, these results indicate that the proposed framework scales effectively with both system size and workload, maintaining stable energy efficiency and system performance, and thus is well-suited for large-scale MEC networks.

To justify the design of the reward function, a sensitivity analysis is conducted with respect to both the weighting factors $(\alpha, \beta)$ and the penalty coefficient for unfinished tasks. As shown in Fig. 11, varying the relative importance between delay and energy consumption (i.e., $\alpha = 0.2, 0.5, 0.8$ with corresponding $\beta = 0.8, 0.5, 0.2$) results in only marginal differences in task drop rate, average delay, and energy consumption. Similarly, adjusting the penalty coefficient from 1× to 3× has a very limited impact on overall system performance. These observations indicate that the proposed PreAlloc-A2C framework is not overly sensitive to specific reward parameter settings. This is because the structured state representation and policy learning mechanism enable the agent to capture the intrinsic relationship between task characteristics and resource availability, rather than relying heavily on finely tuned reward weights. Therefore, the selected parameter configuration provides a reasonable balance between delay and energy objectives while ensuring stable and robust performance across different settings.

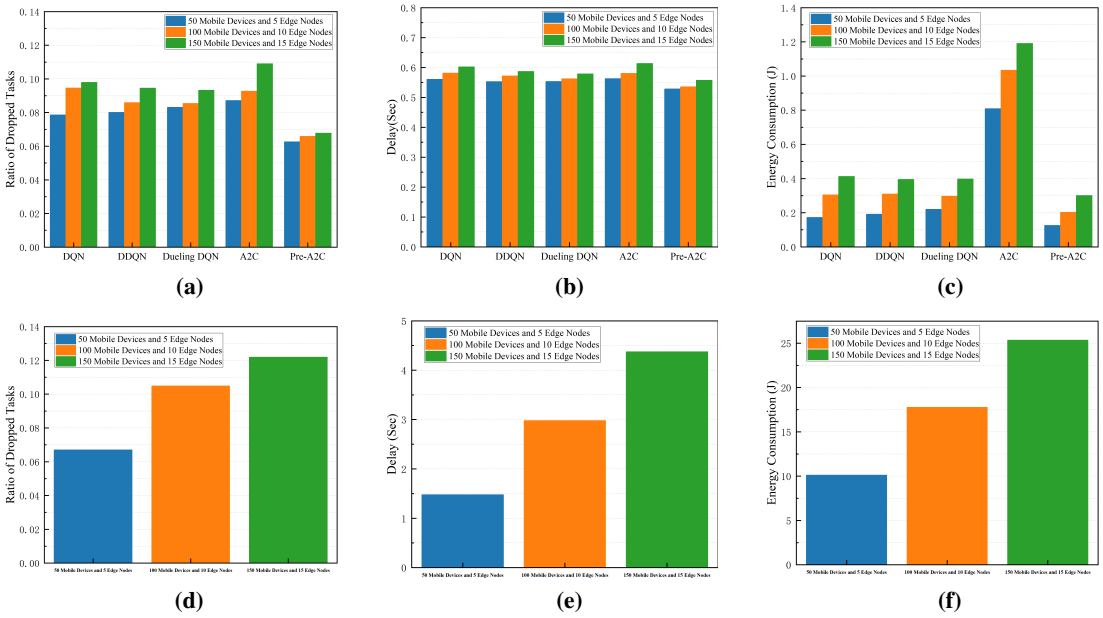



**Figure 10:** Scalability evaluation under different MEC system scales: (**a**) DRL ratio of dropped tasks; (**b**) DRL delay; (**c**) DRL energy consumption; (**d**) PSO ratio of dropped tasks; (**e**) PSO delay; (**f**) PSO energy consumption.

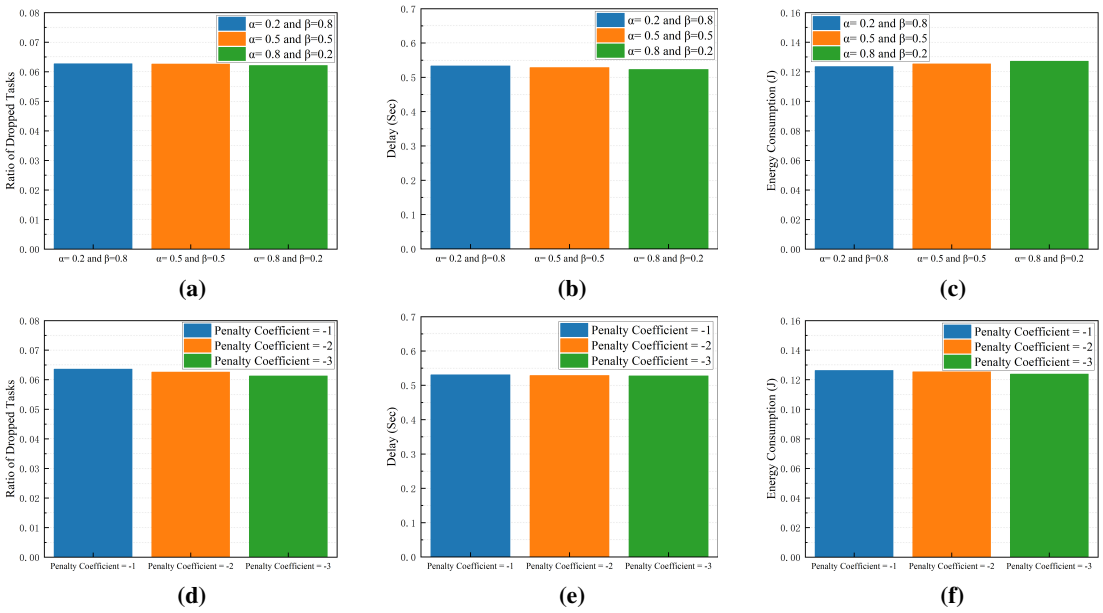



**Figure 11:** Sensitivity analysis of the reward function under different parameter settings: (**a**) ratio of dropped tasks with varying $\alpha$ and $\beta$; (**b**) delay with varying $\alpha$ and $\beta$; (**c**) energy consumption with varying $\alpha$ and $\beta$; (**d**) ratio of dropped tasks with varying penalty coefficients; (**e**) delay with varying penalty coefficients; (**f**) energy consumption with varying penalty coefficients.

### *6.4.3 Variants of Pre-A2C*

To examine the impact of different offloading structures, we propose two Pre-A2C variants. The first, **Pre-A2C (Virtual Node)**, treats the local device as an additional edge node, integrating its features into

the unified scoring framework. The second, **Pre-A2C (Two-Stage)**, adopts a sequential decision process: it first decides whether to offload, then selects the target edge node if offloading is chosen. Fig. 12a shows that both variants exhibit similar reward trends, with initial fluctuations during exploration (episodes 0–50) and convergence to nearly identical reward levels by episode 300. The similar performance of Pre-A2C (Virtual Node) and Pre-A2C (Two-Stage) stems from the fact that both variants effectively capture the essential trade-offs in task offloading decisions. Pre-A2C (Virtual Node) benefits from a unified representation where the local device is treated as just another node, allowing the model to learn the relative merits of local vs. edge offloading within a single scoring framework. Pre-A2C (Two-Stage), on the other hand, decomposes the decision into two logical steps but still relies on the same underlying feature representations and scoring mechanisms for edge node selection as in the virtual node approach. Moreover, the core factors influencing task offloading performance, such as task characteristics and edge server states, are equally well-captured by both structures. Thus, despite the difference in the decision-making flow, both variants can make informed offloading decisions, leading to the observed similar reward performances.

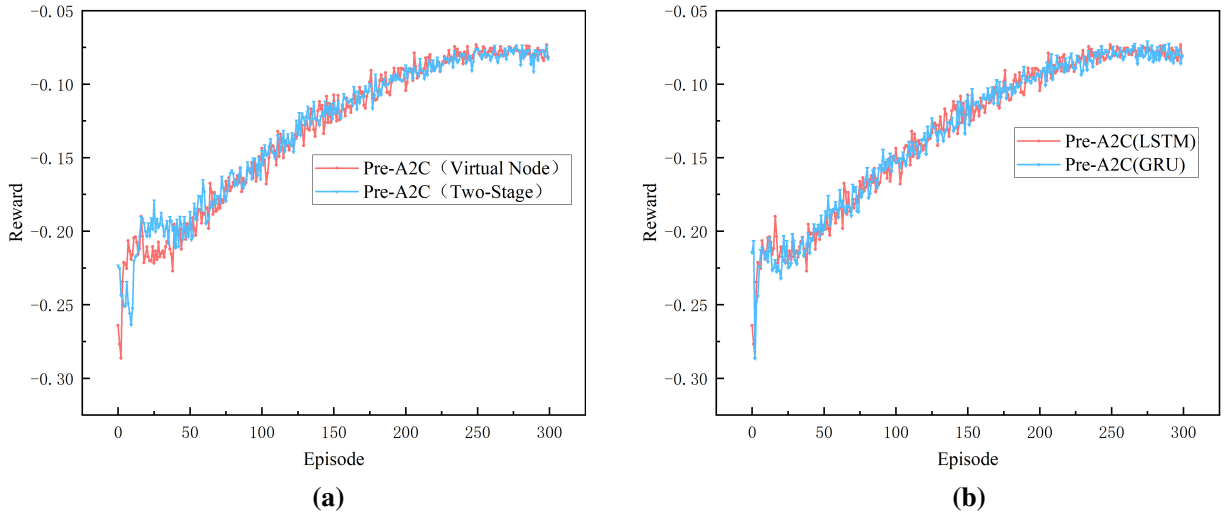



**Figure 12:** Comparison of reward values among different method variants: (**a**) different offloading approaches and (**b**) different recurrent neural networks.

We further investigate sequence modeling by replacing the LSTM network used for predicting server loads with a GRU. Both LSTM and GRU can capture long-term dependencies in sequential data; LSTM uses input, forget, and output gates, while GRU simplifies gating with an update and reset gate, reducing computational overhead. As shown in Fig. 12b, Pre-A2C (LSTM) and Pre-A2C (GRU) achieve nearly identical reward curves. The similarity in performance between Pre-A2C (LSTM) and Pre-A2C (GRU) can be attributed to the fundamental capability of both LSTM and GRU to effectively capture temporal dependencies in server load sequences. Although LSTM offers a more fine-grained control over information flow via its multiple gates, GRU's simplified gating mechanism still suffices to model the essential temporal patterns in server load data for the task offloading problem. Thus, the core functionality of predicting server loads, which is crucial for making informed task offloading decisions, is adequately fulfilled by both architectures, leading to the observed similar reward performances.

## 7 Conclusion and Future Work

This paper addresses task offloading in mobile edge computing by proposing a distributed algorithm that leverages both task-specific and node-specific features. The approach enables autonomous task placement by mobile devices while ensuring allocation to suitable servers. Extensive experiments under varying learning rates, optimization algorithms, numbers of devices and servers, task arrival rates, and deadlines demonstrate consistent convergence, stable learning, and superior performance in latency, energy consumption, and task drop ratio compared with baseline methods. Variants incorporating different recurrent networks (LSTM, GRU) and alternative offloading strategies (virtual nodes, two-stage decision) yielded comparable results, highlighting the framework's flexibility and robustness.

Future research may explore more advanced time-series prediction models, such as temporal convolutional networks or Transformers, to enhance adaptability. Multi-objective optimization could benefit from Pareto-based or evolutionary approaches for improved trade-offs among latency and energy. Adopting multi-agent reinforcement learning methods (e.g., MADDPG, QMIX, mean-field MARL) can facilitate coordinated, scalable policy learning. Finally, extending the framework to complex scenarios—vehicular networks, 6G infrastructures, or smart cities—will further validate its practical applicability and resilience.

**Acknowledgement:** Not applicable.

**Funding Statement:** This work was supported by the Guizhou Provincial Key Technology R&D Program under Grant (QKHZC(2022)YB074) and Guizhou University Science and Technology Group [2024]07.

**Author Contributions:** The authors confirm contribution to the paper as follows: study conception and design: Chaobin Wang and Xianghong Tang; algorithm design and implementation: Chaobin Wang; simulation experiments and data analysis: Chaobin Wang; manuscript preparation: Chaobin Wang; supervision and manuscript revision: Xianghong Tang; manuscript review and editing: Jianguang Lu, Jing Yang, and Panliang Yuan. All authors reviewed and approved the final version of the manuscript.

**Availability of Data and Materials:** The authors confirm that the data supporting the findings of this study are available within the article.

**Ethics Approval:** Not applicable.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Dargan S, Bansal S, Kumar M, Mittal A, Kumar K. Augmented reality: a comprehensive review. Arch Computat Meth Eng. 2023;30(2):1057–80. doi:10.1007/s11831-022-09831-7.
2. Zhang Q, Sun H, Wu X, Zhong H. Edge video analytics for public safety: a review. Proc IEEE. 2019;107(8):1675–96. doi:10.1109/jproc.2019.2925910.
3. Mao Y, You C, Zhang J, Huang K, Letaief KB. A survey on mobile edge computing: the communication perspective. IEEE Commun Surv Tutor. 2017;19(4):2322–58. doi:10.1109/comst.2017.2745201.
4. Zhao P, Tian H, Fan S, Paulraj A. Information prediction and dynamic programming-based RAN slicing for mobile edge computing. IEEE Wire Commun Lett. 2018;7(4):614–7. doi:10.1109/lwc.2018.2802522.
5. Wang Y, Sheng M, Wang X, Wang L, Li J. Mobile-edge computing: partial computation offloading using dynamic voltage scaling. IEEE Trans Commun. 2016;64(10):4268–82. doi:10.1109/TCOMM.2016.2599530.
6. Nguyen PX, Tran DH, Onireti O, Tin PT, Nguyen SQ, Chatzinotas S, et al. Backscatter-assisted data offloading in OFDMA-based wireless-powered mobile edge computing for IoT networks. IEEE Internet Things J. 2021;8(11):9233–43. doi:10.1109/jiot.2021.3057360.

7. Li S, Ge H, Chen X, Liu L, Gong H, Tang R. Computation offloading strategy for improved particle swarm optimization in mobile edge computing. In: Proceedings of the 2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA); 2021 Apr 24–26; Chengdu, China. p. 375–81.
8. Singh S, Kim DH. Profit optimization for mobile edge computing using genetic algorithm. In: Proceedings of the 2021 IEEE Region 10 Symposium (TENSYMP); 2021 Aug 23–25; Jeju, Republic of Korea. p. 1–6.
9. Chakraborty S, Mazumdar K. Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing. J King Saud Univ-Comput Inform Sci. 2022;34(4):1552–68. doi:10.1016/j.jksuci.2022.02.014.
10. Wang X, Zhou Z, Chen H, Zhang Y. Task offloading and power assignment optimization for energy-constrained mobile edge computing. In: Proceedings of the 2021 Ninth International Conference on Advanced Cloud and Big Data (CBD); 2022 Mar 26–27; Xi'an, China. p. 302–7.
11. Cong Y, Xue K, Wang C, Sun W, Sun S, Hu F. Latency-energy joint optimization for task offloading and resource allocation in MEC-assisted vehicular networks. IEEE Trans Vehic Technol. 2023;72(12):16369–81. doi:10.1109/tvt.2023.3289236.
12. Shen S, Hao X, Gao Z, Wu G, Shen Y, Zhang H, et al. SAC-PP: jointly optimizing privacy protection and computation offloading for mobile edge computing. IEEE Trans Netw Serv Manag. 2024;21(6):6190–203. doi:10.1109/TNSM.2024.3447753.
13. Li K, Wang X, He Q, Wang J, Li J, Zhan S, et al. Computation offloading in resource-constrained multi-access edge computing. IEEE Trans Mobile Comput. 2024;23(11):10665–77. doi:10.1109/tmc.2024.3383041.
14. Wang B, Kang H, Li J, Sun G, Sun Z, Wang J, et al. AAV-assisted joint mobile edge computing and data collection via matching-enabled deep reinforcement learning. IEEE Internet Things J. 2025;12(12):19782–800.
15. Li H, Meng S, Sun J, Cai Z, Li Q, Zhang X. Multi-agent deep reinforcement learning based multi-task partial computation offloading in mobile edge computing. Future Gener Comput Syst. 2025;172:107861. doi:10.2139/ssrn.4930170.
16. Rahmati I, Shah-Mansouri H, Movaghar A. QECO: a QoE-oriented computation offloading algorithm based on deep reinforcement learning for mobile edge computing. IEEE Trans Netw Sci Eng. 2025;12(4):3118–30. doi:10.1109/TNSE.2025.3556809.
17. Yang D, Li B, Niyato D. Energy-aware task offloading for rotatable STAR-RIS-enhanced mobile edge computing systems. IEEE Internet Things J. 2025;12(12):20239–50. doi:10.1109/JIOT.2025.3542463.
18. Tuli S, Ilager S, Ramamohanarao K, Buyya R. Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks. IEEE Trans Mobile Comput. 2020;21(3):940–54. doi:10.1109/tmc.2020.3017079.
19. Lu J, Yang J, Li S, Li Y, Jiang W, Dai J, et al. A2C-DRL: dynamic scheduling for stochastic edge-cloud environments using A2C and deep reinforcement learning. IEEE Internet Things J. 2024;11(9):16915–27. doi:10.1109/JIOT.2024.3366252.
20. Yang J, Yuan Q, Chen S, He H, Jiang X, Tan X. Cooperative task offloading for mobile edge computing based on multi-agent deep reinforcement learning. IEEE Trans Netw Service Manag. 2023;20(3):3205–19. doi:10.1109/tnsm.2023.3240415.
21. Chen S, Chen J, Miao Y, Wang Q, Zhao C. Deep reinforcement learning-based cloud-edge collaborative mobile computation offloading in industrial networks. IEEE Trans Signal Inform Process Netw. 2022;8(1):364–75. doi:10.1109/tsipn.2022.3171336.
22. Tang M, Wong VW. Deep reinforcement learning for task offloading in mobile edge computing systems. IEEE Trans Mobile Comput. 2020;21(6):1985–97. doi:10.1109/tmc.2020.3036871.
23. Chen X, Hu S, Yu C, Chen Z, Min G. Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning. IEEE Trans Parallel Distrib Syst. 2024;35(3):391–404. doi:10.1109/tpds.2023.3349177.
24. Wang J, Hu J, Min G, Zhan W, Zomaya AY, Georgalas N. Dependent task offloading for edge computing based on deep reinforcement learning. IEEE Trans Comput. 2021;71(10):2449–61. doi:10.1109/tc.2021.3131040.

25. Dong C, Li W, Zhou Z, Chen X, Tian Z, Wen W. Delay-sensitive task offloading with edge caching through martingale-based deep reinforcement learning. IEEE Trans Mob Comput. 2025;24(7):6225–42. doi:10.1109/tmc.2025.3540413.
26. Qu G, Wu H, Li R, Jiao P. DMRO: a deep meta reinforcement learning-based task offloading framework for edge-cloud computing. IEEE Trans Netw Service Manag. 2021;18(3):3448–59. doi:10.1109/TNSM.2021.3087258.
27. Huang L, Bi S, Zhang YJA. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans Mobile Comput. 2019;19(11):2581–93. doi:10.1109/tmc.2019.2928811.
28. Huang H, Ye Q, Zhou Y. Deadline-aware task offloading with partially-observable deep reinforcement learning for multi-access edge computing. IEEE Trans Netw Sci Eng. 2021;9(6):3870–85. doi:10.1109/tnse.2021.3115054.
29. Yang N, Wen J, Zhang M, Tang M. Multi-objective deep reinforcement learning for mobile edge computing. In: Proceedings of the 2023 21st International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt); 2023 Aug 24–27; Singapore. p. 1–8.
30. Gong A, Yang K, Lyu J, Li X. A two-stage reinforcement learning-based approach for multi-entity task allocation. Eng Appl Artif Intell. 2024;136:108906. doi:10.1016/j.engappai.2024.108906.
31. Yang W, Liu Z, Liu X, Ma Y. Deep reinforcement learning-based low-latency task offloading for mobile-edge computing networks. Appl Soft Comput. 2024;166(1):112164. doi:10.1016/j.asoc.2024.112164.
32. Xie Y, Wu Q, Fan P. Digital twin vehicular edge computing network: task offloading and resource allocation. In: Proceedings of the 2024 7th International Conference on Information Communication and Signal Processing (ICICSP); 2024 Sep 21–23; Zhoushan, China. p. 1137–41.
33. Toopchinezhad MP, Ahmadi M. Deep reinforcement learning for delay-optimized task offloading in vehicular fog computing. In: Proceedings of the 2025 29th International Computer Conference, Computer Society of Iran (CSICC); 2025 Feb 5–6; Iran, Islamic Republic. p. 1–6.
34. Chen M, Liu W, Wang T, Zhang S, Liu A. A game-based deep reinforcement learning approach for energy-efficient computation in MEC systems. Knowl Based Syst. 2022;235(6):107660. doi:10.1016/j.knosys.2021.107660.
35. Li G, Li X, Li J, Chen J, Shen X. PTMB: an online satellite task scheduling framework based on pre-trained Markov decision process for multi-task scenario. Knowl Based Syst. 2024;284:111339. doi:10.1016/j.knosys.2023.111339.
36. Gao X, Sun Y, Chen H, Xu X, Cui S. Joint computing, pushing, and caching optimization for mobile-edge computing networks via soft actor–critic learning. IEEE Internet Things J. 2023;11(6):9269–81. doi:10.1109/globecom54140.2023.10437459.
37. Neto JLD, Yu SY, Macedo DF, Nogueira JMS, Langar R, Secci S. ULOOF: a user level online offloading framework for mobile edge computing. IEEE Trans Mobile Comput. 2018;17(11):2660–74. doi:10.1109/TMC.2018.2815015.
38. Intelligence S. Speedtest market reports: Canada average mobile upload speed based on Q2–Q3 2019 data. 2020 [cited 2020 Aug 5]. Available from: https://www.speedtest.net/reports/canada/.
39. Wang C, Liang C, Yu FR, Chen Q, Tang L. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. IEEE Trans Wireless Commun. 2017;16(8):4924–38. doi:10.1109/twc.2017.2703901.
40. Zhang J, Du J, Shen Y, Wang J. Dynamic computation offloading with energy harvesting devices: a hybrid-decision-based deep reinforcement learning approach. IEEE Internet Things J. 2020;7(10):9303–17.