Computer Science Review 53 (2024) 100656

Contents lists available at ScienceDirect

Computer Science Review

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

Review article

A survey on computation offloading in edge systems: From the perspective of deep reinforcement learning approaches

Peng Peng [a,b], Weiwei Lin [c,b], Wentai Wu [d,*], Haotong Zhang [e], Shaoliang Peng [f], Qingbo Wu [b], Keqin Li [g]

[a] School of Future Technology, South China University of Technology, Guangzhou, 510641, China
[b] Peng Cheng Laboratory, Shenzhen, 518000, China
[c] School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China
[d] College of Information Science and Technology, Jinan University, Guangzhou, 510632, China
[e] School of Software Engineering, South China University of Technology, Guangzhou, 510006, China
[f] College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, China
[g] Department of Computer Science, State University of New York, NY, 12561, USA

ARTICLE INFO

Keywords:
Edge computing
Deep reinforcement learning
Computation offloading

ABSTRACT

Driven by the demand of time-sensitive and data-intensive applications, edge computing has attracted wide attention as one of the cornerstones of modern service architectures. An edge-based system can facilitate a flexible processing of tasks over heterogeneous resources. Hence, computation offloading is the key technique for systematic service improvement. However, with the proliferation of devices, traditional approaches have clear limits in handling dynamic and heterogeneous systems at scale. Deep Reinforcement Learning (DRL), as a promising alternative, has shown great potential with powerful high-dimensional perception and decision-making capability to enable intelligent offloading, but the great complexity in DRL-based algorithm design turns out to be an obstacle. In light of this, this survey provides a comprehensive view of DRL-based approaches to computation offloading in edge computing systems. We cover state-of-the-art advances by delving into the fundamental elements of DRL algorithm design with focuses on the target environmental factors, Markov Decision Process (MDP) model construction, and refined learning strategies. Based on our investigation, several open challenges are further highlighted from both the perspective of algorithm design and realistic requirements that deserve more attention in future research.

Contents

* Corresponding author.
E-mail addresses: pengp@pcl.ac.cn (P. Peng), linww@scut.edu.cn (W. Lin), wentaiwu@jnu.edu.cn (W. Wu), sezhanght@mail.scut.edu.cn (H. Zhang), slpeng@hnu.edu.cn (S. Peng), wuqb@pcl.ac.cn (Q. Wu), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.cosrev.2024.100656
Received 20 March 2024; Received in revised form 6 June 2024; Accepted 16 June 2024
Available online 29 June 2024

1. Introduction

Over the past decade, the paradigm of service delivery has been reshaped and is still evolving. The development of edge computing stems from the popularity of cloud computing, where resources are centralized in data centers to fulfill the demands of end users. However, the limitation of cloud-centric service architecture stands out when it comes to service delivery at the network edge, such as Internet of Things (IoT) scenarios where a myriad of devices with latency-sensitive applications are involved. According to Statista, the number of edge-enabled IoT devices is forecast to reach 6.5 billion by 2030 [1], which further increases data transmission and task processing requirements, leading to network congestion and high response latency.

Driven by this trend, edge computing has emerged and developed into a widely-used service paradigm. The overall architecture of an edge computing system, depicted in Fig. 1, comprises a multi-layered distributed framework, where smaller entities like routers and antenna towers function as edge nodes (EN) to either independently or collaboratively provide services. Thus, computational tasks, which are not feasible to be processed efficiently and promptly by end devices, could be offloaded to ENs in the vicinity for fast responses [2]. This offers great advantages in transmission delays caused by physical distance and relieves network congestion, typically resulting in a stronger guarantee in service latency. Because of these features, edge computing has been widely adopted in the industry especially when enabled by the next-generation communication technology in new scenarios such as high-density, large-scale IoT networks [3].

However, the dynamic, distributed, and inherently complex nature of edge computing systems poses significant challenges in terms of how to better utilize the computing power at the edge, where the strategy of task offloading is one of the most critical aspects for enhancing the efficiency and throughput of edge systems. While traditional optimization methods have demonstrated the ability to achieve adequate performance, they often struggle when facing the current large-scale and highly dynamic systems. To this end, recent research explores the potential of DRL to tackle the intricacies of edge computing systems. Supported by the powerful feature-extracting capabilities of deep neural networks (DNNs), DRL-based approaches have shown remarkable proficiency within complex and uncertain environments. Nonetheless, the heterogeneous nature of realistic edge environments inevitably results in the diversity of system models and the corresponding DRL algorithm design.

Therefore, this survey endeavors to provide an in-depth exposition of algorithm design principles as well as the recent advances in DRL-based computation offloading strategies within the context of edge computing.

1.1. Computation offloading in edge computing

In the realm of edge computing, computation offloading delineates the migration of computation tasks from local devices to edge servers. It allows compute-intensive applications to leverage the better processing capabilities of edge servers, thus augmenting the system's ability to deliver sophisticated services. Distinct from the conventional cloud computing paradigm, offloading tasks to proximal edge servers circumvents long data transmission delays, which is critical for latency-sensitive applications.

A computation offloading strategy decides whether tasks should be executed locally or offloaded to a selected EN. Offloading can improve computing latency but introduces additional transmission delay and energy costs. Meanwhile, communication and computation resources, such as bandwidth, transmit power and CPU frequency, should be adjusted simultaneously to ensure service quality and system efficiency. These considerations combined make the design of offloading strategies challenging and thus attracted broad interest in recent years [4].

The inherent diversity in the specifications of ENs and local devices and the varying requirements of services introduce complexity to the decision-making problem. Firstly, the geographical distribution and the strong heterogeneity of ENs and devices can lead to large differences in benefit between decisions. Secondly, the prevalent usage of wireless communication could be adversely affected by numerous factors such as unstable connection and channel interference. Thirdly, an edge computing system is highly scalable and thus requires adaptive management and coordination. In addition, specific considerations, such

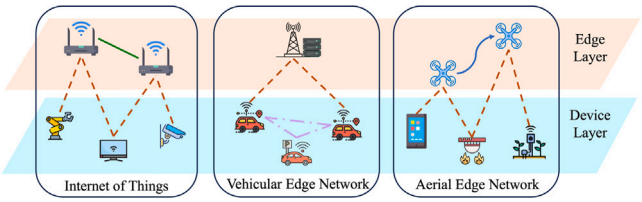

Fig. 1. Edge computing network.

as the mobility of devices, the dependence between subtasks and the limited energy budgets, must be meticulously accounted for in realistic scenarios. These factors collectively underscore the intricate nature of computation offloading, highlighting the need for advanced, adaptable solutions.

Much effort has been invested to address the computation offloading optimization challenges. Approaches such as convex optimization, heuristic algorithms, and approximation algorithms have been extensively proposed and adopted, demonstrating efficacy in achieving adequate performance [5]. Despite their widespread application, these traditional optimization techniques often falter when confronted with the distributed, dynamic, and scalable nature of edge computing systems. Consequently, conventional strategies often experience performance degradation or long decision-making delays, which underscores the importance of finding stronger approaches to the problem of computation offloading.

1.2. DRL-based offloading strategies

Reinforcement Learning (RL) is an experience-driven machine learning paradigm that excels in complex dynamic decision-making scenarios. The core of RL involves the observation and utilization of the system state, interaction with the environment, and the incentivization of reward feedback. By adapting to the environment based on interaction experiences, RL agents are capable of continually learning to improve policies in response to the evolving environment and making decisions that maximize long-term returns. Recently, the integration of DNN with RL, known as DRL, has attracted widespread attention.

The fast development of DRL, coupled with the attributes of edge systems, has given rise to DRL-based computation offloading methods. It not only extends the applicability of RL to a wider range of challenging scenarios, but also improves the precision, adaptability, and quality of the policy. Its benefits can be attributed to the following points. Firstly, DRL inherits the continual learning capability of RL to adapt to the changing dynamics of edge computing systems. Secondly, the integration of DNN enables DRL to more accurately perceive and interpret the complex state of large-scale edge systems. Thirdly, unlike traditional methods that prioritize immediate rewards, DRL focuses on maximizing long-term returns thereby preventing the pitfalls of short-term benefits. In addition, the advent of multi-agent DRL introduces decentralized decision-making. It reduces the need for extensive communication of states and provides benefits for autonomous edge system participants.

Recent studies have highlighted the effectiveness of DRL-based approaches in managing computation offloading to achieve notable performance [6]. Despite their diversity, the effectiveness of DRL-based approaches largely depends on three aspects of DRL, namely, the target environment, the MDP model construction, and the learning strategy of agents. As shown in Fig. 2, the target environment refers to the realistic system the agent interacts with. Its characteristics are critical for MDP model construction, agent design, and policy training. The MDP model outlines the states observed, the actions to be taken, and the reward function that drives learning. The learning strategy of agents is the core of DRL algorithms and is related to how agents utilize the acquired information for decision making and learning from experience. Better learning strategies would result in faster convergence and better performance.

Therefore, this review summarizes recent works from the perspective of these key elements of DRL. In other words, for task offloading in edge systems, we aim to provide three DRL-specific angles for the readers to understand how different solutions work and how they differ from each other. More specifically, instead of discussing real-world application scenarios, we first outline the environment factors that need to be considered in research. Then, we describe the MDP model construction in related work so that the readers can better understand how the agents make offloading decisions. Finally, from the perspective of the learning strategy, we summarize novel techniques for learning strategy improvement that concern how to "help" a DRL agent learn better.

1.3. Related surveys

In recent years, computation offloading management has garnered significant interest and led to the publication of numerous comprehensive surveys. We analyzed relevant reviews published since 2021 and recognized the need for an updated perspective that reflects the fundamentals and challenges of applying DRL as the recipe. Table 1 summarizes and compares existing surveys in this domain 1.

Due to the diverse application domains of edge computing such as the Internet of Vehicles (IoV) and Aerial Mobile Edge Computing, some surveys provide specialized analysis to address domain-specific challenges and developments. For instance, Liu et al. [7] focus on vehicular edge systems. It is structured by the diverse service providers and discusses the usage of RL/DRL algorithms within this area. Hamdi et al. [8] outline some requirements that should be considered in computation offloading for vehicular edge computing systems. Song et al. [9] address the management of UAV-assisted edge systems. Instead of specializing in offloading, they include control plane aspects such as UAV deployment and trajectory optimization. While these reviews offer valuable insights into specific application areas, it is hard for the readers to understand why these DRL algorithms work and how to make them work in other scenarios.

DRL can also be utilized for techniques beyond task offloading. For example, Chen et al. [10] discuss the applications of DRL in communication, computation, caching, and control within five key IoT scenarios including smart grids, intelligent transportation systems, and industrial IoT environments. Frikha et al. [11] explore the application of RL/DRL in communication and networking, including routing, spectrum access, mobility management, and caching strategies. Li et al. [12] specialize in the usage of multi-agent RL for optimization in IoT, addressing areas such as transmission scheduling, computation offloading, trajectory planning, and security measures. Although these studies provide insight into the use of RL/DRL in edge systems, there is still a gap in the literature where a more focused and detailed examination of recent developments in computation offloading from the perspective of RL/DRL.

Under the topic of computation offloading, some existing works offer in-depth analyses and categorizations of the diverse strategies employed within edge systems. For example, Feng et al. [13] and Kumari et al. [14] investigate recent research efforts based on objectives. It is also very common to group related works into algorithmic categories such as traditional optimizations, heuristic methods, and machine learning-based approaches [15–17]. Kar et al. [18] categorizes these works based on network architecture and compares traditional with DRL-based strategies. Akhlaqi and Mohd Hanapi [19] discuss related works in terms of problem definition, algorithm comparison, and performance evaluation. However, we believe these efforts are insufficient to expose the subtleties and challenges of DRL-based offloading strategies.

Our scope partially overlaps with [20] which summarizes DRL-based offloading solutions from 2020 to 2021 and groups them by

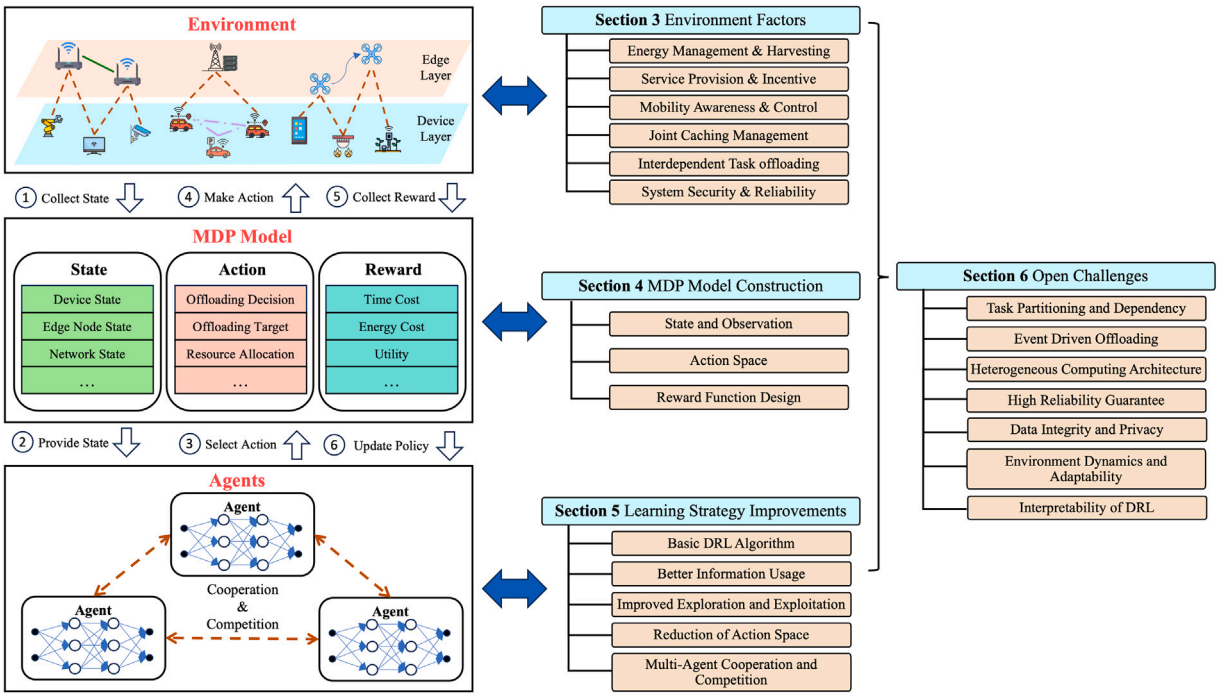

Fig. 2. The left side shows the core elements of DRL, including the environment, the MDP model design, and the agents' learning strategy. The right side shows the structure of this survey, with the main sections matching the core elements of DRL.

Table 1
Comparison of related surveys.

	Coverage (up-to)	Covered scenarios			Focus on offloading	Focus on RL/DRL	Main perspective
		Internet of Things	Vehicular edge network	Aerial Edge Network			
[7]	2021		✓		✓	✓	Network architecture
[8]	2021		✓		✓		Scenario-specific problems
[9]	2021			✓			Scenario-specific problems
[10]	2020	✓	✓				Applications
[11]	2020	✓	✓			✓	Scenario-specific problems
[12]	2021	✓	✓	✓		✓	Scenario-specific problems
[13]	2021	✓	✓		✓		Optimization objectives
[14]	2021	✓	✓		✓		Optimization objectives
[15]	2021	✓	✓	✓	✓		Algorithmic categorization
[16]	2022	✓	✓	✓	✓		Algorithmic categorization
[17]	2021	✓	✓		✓		Algorithmic categorization
[18]	2022	✓	✓		✓		Network architecture
[22]	2022	✓	✓	✓	✓		Network architecture
[19]	2022	✓	✓	✓	✓		Algorithmic categorization
[20]	2021	✓	✓	✓	✓	✓	applications
[6]	2021	✓	✓		✓	✓	Scenario-specific problems
[21]	2022	✓	✓	✓	✓	✓	Algorithmic categorization
[4]	2021	✓	✓	✓	✓	✓	Algorithmic categorization
Ours	2023	✓	✓	✓	✓	✓	Principles of algorithm design

real-world scenarios. While this survey is comprehensive in its exploration of the target environment, it falls short in adequately discussing the detailed agent design. A similar shortcoming is noted in [6], which also misses in-depth details about the algorithms. Abdulazeez and Askar [21] and Zabihi et al. [4] provide helpful references for understanding the application of various RL/DRL algorithms in common scenarios including IoT, vehicular edge networks and aerial edge networks. However, they do not cover the latest advances since late 2022 and fail to provide valuable future directions. Hence, a more up-to-date review that provides both fundamentals and insights into this active field is still desired.

1.4. Contribution and organization

In contrast to the existing literature, our survey casts light on DRL-based approaches with the primary focus on the core elements of DRL, namely, the target environment factors, the MDP model construction, and the learning strategies. We cover related studies from 2020 to 2023 in which period DRL became increasingly popular in the domain. We summarize and discuss several well-established and popular target environment factors in real-world systems and related research to highlight the issues and challenges involved. We then introduce the details of MDP model construction, discussing the state space, the action space,

and the reward design as the fundamental role of applying DRL. An in-depth review and synthesis of recent studies is provided, categorized by angle of improvements in learning strategies. We conclude our review by outlining future research directions that deserve exploration. The contributions of this review are as follows.

1. Our survey revisits state-of-the-art DRL-based methods for offloading computation in edge computing from the perspective of algorithm design principles. We aim to offer a technical exposition and guide the future application and enhancement of DRL in this field.
2. We organize the review based on three key DRL elements, namely, the target environment factors, the MDP model construction, and the learning strategy improvements. Unlike existing surveys, this structure can offer a unique angle from which one can better understand the rationale behind algorithm design by looking into the technical differences between DRL-based offloading strategies.
3. Based on our review of the current literature, we identify and discuss several open challenges and suggest future research directions under this topic. Despite existing efforts in modeling edge environments and improving DRL performance, the dynamic and heterogeneous nature of edge systems with security and reliability requirements still deserves future exploration.

The remainder of this survey is organized as follows. Section 2 introduces the basic formulation of task offloading and the background of DRL. Section 3 summarizes several common environment factors that are common in realistic scenarios and have been widely analyzed. Section 4 presents a statistical analysis of the elements frequently used in MDP model construction. Section 5 provides an in-depth review of learning strategy improvements in recent research. Section 6 discusses open challenges for further research. The paper concludes in Section 7. An overview of the scope and content of the survey is presented in Fig. 2.

2. Background

2.1. Computation offloading in edge computing

In general, as shown in Fig. 3, tasks are generated by end devices. The task offloading procedure includes three steps. First, task data should be uploaded to the selected edge server. It is then processed by edge servers, and then results should be transmitted back to the end devices. The optimization objective of task offloading often relates to the transmission cost and computing latency, both of which depend on environment factors as well as the decisions. In this regard, principled models are commonly used in the literature.

The transmission model reveals the relationship between network resources, such as transmit power and bandwidth, and several performance measures such as transmission rate, latency, and energy costs. Despite the wide range of transmission schemes available in modern communication networks, the Shannon Theorem provides the fundamental equation for estimating transmission rates, expressed as $\upsilon = B\log_2(1+\mathrm{SINR})$, where B signifies the allocated bandwidth, and SINR is the signal-to-interference-plus-noise ratio. Based on the transmission rate, given the task data size, the associated latency and energy costs could be estimated intuitively.

The model of communication has a fundamental influence on the effectiveness of computation offloading. Predominantly, Time Division Multiple Access (TDMA) and Orthogonal Frequency Division Multiple Access (OFDMA) are adopted in the majority of the related studies [23,24]. These techniques allocate exclusive time slots or frequency bands to different transmission schemes. Besides. Non-orthogonal Frequency Division Multiple Access (NOMA) has gained increasing adoption as a key enabling technique for 5G/6G featuring high bandwidth

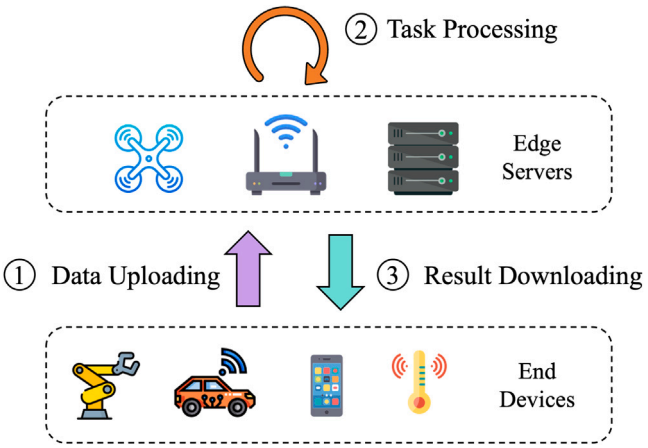

Fig. 3. Task offloading procedures. The task data is uploaded to the selected edge server, processed there, and the result is downloaded back to the end device.

efficiency [25]. NOMA facilitates concurrent data transmission over a shared channel and employs Successive Interference Cancellation (SIC) to sequentially decode the overlaid signals. In this model, signals designated for subsequent decoding are considered as co-channel interference.

Furthermore, additional factors in wireless transmission, such as line-of-sight (LoS) and non-line-of-sight (nLoS) transmissions, air-ground communication, and Intelligent Reflecting Surface (IRS) technology, need to be taken into account [26–28]. These factors increase the complexity of theoretical communication models and should be analyzed in the context of real-world environments.

The computation model evaluates the latency and energy costs associated with task processing. Typically, new tasks are queued at the edge servers or end devices upon their arrival and must wait while previous tasks are still being processed. Only after the completion of preceding tasks can the computation phase for the new tasks begin [18,29]. In the majority of studies, the focus is primarily on CPU-intensive tasks, with the CPU frequency serving as the metric for assessing the processing rate. By given the number of CPU cycles required to complete a task, it is possible to accurately calculate the corresponding latency and energy consumption associated with its processing. A few works explored offloading over heterogeneous computing architectures such as Field Programmable Gate Arrays (FPGA) or Graph Processing Units (GPU) [30,31], but they simply use a predefined suitability coefficient to represent the performance difference among the CPU processors.

The decision variables in computation offloading are mainly about whether to offload, where to offload, and the percentage of computation to offload [32]. These decisions are pivotal in optimizing the performance and efficiency of distributed edge systems. Whether to offload represents a binary choice in computation offloading. Choosing to perform local processing avoids data transmission, which reduces transmission energy consumption but often results in higher latency due to the limited computing resources. Conversely, offloading tasks to the edge can significantly reduce latency but introduce transmission overheads. In scenarios involving a large number of end devices, it is essential to strategically select the offload target to facilitate efficient load balancing among ENs, thereby optimizing network efficiency and maintaining service quality. Several studies follow partial task offloading, where tasks can be fractionally divided. They focus on how many ratios of tasks are selected to offload, which transforms the binary decision into a continuous space decision problem.

In addition, computation offloading is usually coupled with the allocation of resources. For example, setting a lower transmit power can decrease the energy cost per unit time, but may lead to longer transmission times; increasing the CPU frequency would reduce the

computing latency but at the expense of a higher energy cost. Besides, some specific scenarios require other decision variables such as caching location and EN movement control [33,34].

The primary objective of computation offloading management is to enhance system performance which is typically measured by latency and energy cost. Latency impacts Quality of Service (QoS) while energy cost reflects the running costs and efficiency of systems. To find an optimal balance, numerous studies suggest optimizing a weighted sum of latency and energy costs, which creates a trade-off between minimizing latency and reducing energy usage. Some research incorporates additional factors such as task completion gain, task priority, energy pricing, and service provisioning costs. These contribute to composite metrics such as system utility, energy efficiency, and service profit. Yet, the basic ones, e.g., processing latency, energy consumption and system performance, still draw the most attention [13,14].

2.2. Deep reinforcement learning fundamentals

2.2.1. Markov decision process

In general, the very foundation of DRL is to model the system as a Markov Decision Process (MDP). For a given environment or optimization problem, the first step for DRL-based methods is to establish an MDP model that describes the input states, the action space, and the reward. It also defines the transition between states and the cumulative discount factor for future rewards. It is commonly represented by a tuple (S, A, p, r, γ), where

- S represents the set of states that describes the environment. The Markov property constrains that the next state s_{t+1} only depends on the current state s_t and independent to the past states $\{s_0, s_1, \dots, s_{t-1}\}$.
- A is the set of actions. In classic settings, it is assumed that A is a finite, discrete space.
- p is the transition probability function. It maps a state–action pair to the probability distribution of the next state.
- r is the immediate reward obtained based on the action taken.
- γ is the discount factor applied to the reward from future actions. When the MDP is infinitive, a proper γ is required to ensure the cumulative reward (i.e., expected return) converges.

For the state s, if taking action a, an immediate reward $r(s, a)$ is obtained, and the state transition to s' is triggered depending on the transition probability $p(s'|s, a)$. After interacting t times with the environment, the agent has in memory the following trajectory of MDP: $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_t, a_t, r_t)$. The discounted cumulative reward starting from t is represented as $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$.

2.2.2. Value-based DRL

The state–action value function Q is the key element in value-based DRL methods. It represents the expected cumulative reward after taking action a at state s, defined as $Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$. Value-based DRL is required to estimate the Q value for each state–action pair, and the optimal policy is to select the action with the maximum Q value for each state. By minimizing the temporal difference (TD) error $L_{TD} = r_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a) - Q(s_t, a_t)$, the Q value can be iteratively estimated.

A straightforward value-based algorithm is Deep Q-Learning [35]. DQN utilizes deep learning models to approximate the value function of states and actions. For better convergence and stability, the target network is introduced as a smooth copy of the online network with a lower update frequency. In addition, to encourage exploration and avoid local optima, DQN utilizes the ϵ-greedy method, which adds randomness to the selection of actions. It also maintains a storage buffer called experience replay to store interacting transitions. At each learning step, a batch of records is sampled to eliminate the correlation of data from the same trajectory.

To facilitate a smoother learning process, Double Deep Q-Learning [36] is proposed where action selection is based on the online network and the Q value is estimated by the target network. Both networks similarly learn to minimize the TD error and update the parameters as DQN. In addition, from the perspective of the architecture, Dueling DQN [37] is proposed based on the logic that sometimes actions have less impact on the environment but that state is valuable. In this algorithm, after extracting features from the state, two streams are designed to estimate state-value V and advantage-value A separately.

2.2.3. Policy-based DRL

Different from value-based approaches that estimate Q values, policy-based DRL approaches directly optimize the policy network, which generates the probability distribution of actions. Following the goal of RL to maximize the expectation of discount cumulative reward, the policy-based target is $\arg\max_\theta \sum_\tau p_\theta(\tau) G(\tau)$, where $G(\tau)$ is the discounted cumulative reward, and $p_\theta(\tau)$ is the probability of generating a trajectory τ under policy π_θ.

Parameter update in policy-based DRL aims at maximizing the discounted cumulative reward. Using policy gradient, the general update rule can be formulated as $\theta \leftarrow \theta + \eta \mathbb{E}_\tau[G(\tau) \nabla \log p_\theta(\tau)]$, where η is the learning rate. The method avoids calculating and updating the Q value but directly optimizes the policy. Proximal Policy Optimization (PPO) is the most famous policy-based DRL, in which the importance sampling is introduced to leverage previous data to update current policy.

2.2.4. Actor-critic framework

Value-based approaches are simple and interpretable, but cannot properly handle continuous or high-dimensional action space. Policy-based approaches can generate a stochastic distribution of actions and learn the policy but often struggle to converge efficiently. Therefore, a straightforward idea is to design hybrid methods that integrate both types of approaches. The actor-critic framework is proposed with this rationale and has developed into several variants. At the core of the framework is an actor that follows the policy gradient to generate the action and a critic that estimates the V or Q value with value-based approaches to judge the action taken.

Following the gradient ascend method, the actor is firstly optimized based on the predicted Q value provided by the critic. The critic is then trained by minimizing the TD error. To reduce variance and increase stability, a refined framework called Advantage Actor-Critic (A2C) uses the advantage value to calibrate the benefit of actions in a given state. Further, since interacting with the environment to get sufficient data for DRL training is time-consuming, the Asynchronous Advantage Actor Critic (A3C) [38] is proposed, which creates multiple threads for parallel interaction. For each period, the threads are synchronized for network parameters, and trajectories are asynchronously generated by the threads to compute the gradient. After the threads finish interacting and computing the gradient, the parameter server collects the gradients to update the network parameters.

The Actor-Critic framework is also adopted to enhance the training of DQNs. Deep Deterministic Policy Gradient (DDPG) [39] combines the DQN with the Deterministic Policy Gradient (DPG). Unlike stochastic policy networks that generate a distribution of actions for a given state, a deterministic policy outputs a deterministic action, thus the gradient of cumulative discounted reward could be expressed as $\nabla_\theta R_\theta = \mathbb{E}\left[\nabla_\theta p_\theta(s) \nabla_a Q(s, a)\Big|_{a=\pi_\theta(s)}\right]$. DDPG is designed with the actor-critic architecture, where the actor deterministically outputs an action and the critic can estimate the Q value with minimum TD error. Further, based on the DDPG, TD3 [40] is proposed for continuous action space, which alleviates overestimation and improves convergence.

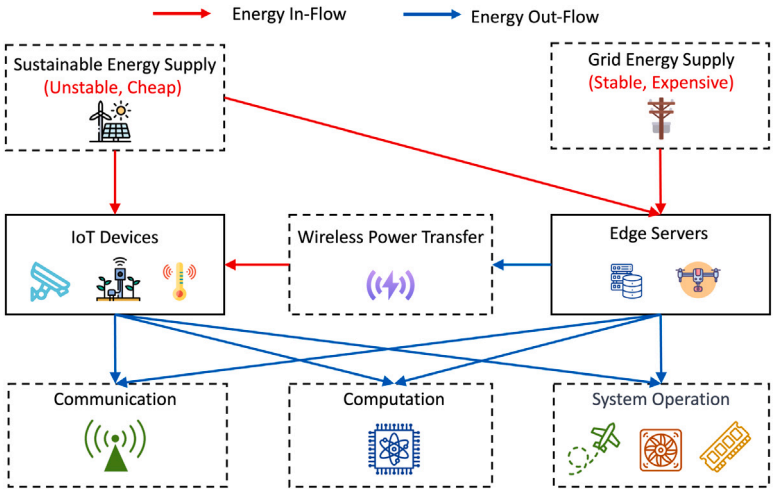

Fig. 4. Common energy in-flows and out-flows in edge computing systems.

2.2.5. Multi-agent DRL

Scenarios with the demand for multiplayer competition and cooperation have led to the development of Multi-Agent RL (MARL). This is also the case for edge computing where each device or EN can act as an agent to make decisions that optimize network performance.

Various multi-agent RL algorithms have been proposed. As an intuitive approach, Independent Q-Learning (IQL) assumes that other agents are part of the environment and maintains a DQN for each agent independently. However, they ignore the influence of actions from other actions, which makes the environment unstable for learning a stable policy. Some approaches decompose the global Q value to local Q value, such as VDN and Q-Mix, to evaluate the contribution of each agent and thus avoid lazy agents caused by independent training. The most widely used paradigm is the actor-critic structure with centralized training and decentralized execution (CTDE), in which the critic network is designed to train agents for cooperatively maximizing long-term system rewards globally. Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [41] is the most widely used algorithm following the CTDE paradigm, where the critic network can fully observe the global state and the joint actions to estimate action values while the actor only obtains limited information to generate actions.

3. Environment factors

The target environment characterizes the scenario of interest and is crucial for the design of DRL-based algorithms. Considering the great diversity in application scenarios such as industrial IoT, IoV, UAV-assisted edge systems, smart grids, and smart homes, it is very necessary to abstract and generalize the specific factors involved. Hence, based on our analysis, we summarize important environment factors that can have major impacts on algorithm design and are frequently discussed in related works.

3.1. Energy management & harvesting

In edge computing systems, end devices and edge services are typically energy-sensitive. In previous works, it is commonly assumed that they can access a stable energy supply and thus energy conservation directly means cost reduction [42,43]. However, in reality, many IoT devices and small ENs are battery-powered via energy harvesting from renewable power sources or wireless power transmission (WPT) (Fig. 4). In this case, energy management and harvesting become a key consideration in the computation offloading scheduling process to ensure the balance of energy income and expense.

In many situations, IoT devices can be fully or partially powered by wind and solar power. Since the renewable energy supply is unstable, a general approach is to treat energy arrivals as a random process. In this context, considering the maximum capacity of the battery b^{max} and the energy costs of the current timeslot E^{cost}, the remaining power at the end of the current timeslot can be expressed as $b^{t+1} = \min\{\max\{b^t - E^{cost} + e^t, 0\}, b^{max}\}$, where e^t is the harvested energy of the current timeslot. For example, Wang and Zhang [44] consider that the energy harvested evolves according to the i.i.d. random process with a maximum value. They set battery power constraints and energy consumption constraints to ensure the actions meet the energy management requirements. Shen et al. [45] discretize the energy into multiple energy packets. The number of energy packets obtained is modeled as a Poisson distribution. A certain amount of packets are selected for processing. Wei et al. [46] allows grid energy to cover energy deficits. The goal is to reduce the fraction of grid energy consumption thus minimizing carbon emissions.

However, it is difficult to ensure the availability of the system by relying solely on an unstable renewable energy supply. In response, some works propose to transmit energy from ENs to IoT devices via WPT as energy sources. Niu et al. [47] assume the WPT and data transmission are mutually exclusive. They analyze how to divide a timeslot for these two operations respectively. The energy obtained is assumed to be fully consumed during task processing. To simultaneously process tasks and harvest energy, Hu et al. [48] deploy specialized radio-frequency (RF) transmitters to provide a continuous but distance-sensitive energy supply. In [49], ENs are suggested to collect renewable energy and support IoT devices through WPT. Grid power supply can also be the backup option when renewable energy supply is insufficient. In [50], due to the marginal utility of WPT, a nonlinear energy harvesting circuit model is applied.

In some aerial-assisted edge computing systems, battery-based UAVs can act as both energy consumers and power suppliers. In this case, it is important while challenging to plan the activities based on battery level. For example, in [51], the battery of UAVs should support its own flight and charge end devices. Penalties are imposed when energy constraints are violated for end devices and UAVs. Zhou et al. [52] assume that the UAV is sufficiently powerful to continuously broadcast RF signals to charge ground devices. They focus on managing the energy costs of end devices. UAVs are used as energy middlewares in [53] to collect energy from the high altitude platform and provide energy to ground devices. It constrains the current energy supply to be greater than the energy overhead. Ke et al. [54] use UAVs and Macro Base Stations (MBS) to provide services. When sustainable energy supply is insufficient, UAVs stop servicing and MBSs switch to the grid energy supply.

3.2. Service provision & incentive

How to incentivize idle devices to participate in service provisioning is an important issue. Resource owners need to decide whether to provide services and join the edge computing system based on the potential benefit. The potential conflict of interest between service costs and gains is often the deciding factor.

It is important to estimate the willingness of idle resource holders to take an extra workload. In [55], the servers adjust the probability of task acceptance according to the remaining power of the vehicles, computing power, and wireless transmission rate. Shi et al. [56] model service availability as associated with service probability and V2V link duration. The former is determined based on vehicle state information to represent the probability of accepting offloaded tasks. The latter is the sustaining duration of the communication link, where vehicles that stay within the communication range for longer periods are more likely to be selected as service vehicles.

To compensate for the energy overheads of service providers, resources can be priced for renting. Pricing is non-trivial in dynamic environments. For determined resource prices, some works try to adjust the offloading policy to ensure the profits of ENs. Xue et al. [57] define the profit as the difference between resource renting gains and energy costs, thereby aiming to maximize the revenue while satisfying delay constraints. Zhou et al. [58] build an End-Edge-Cloud system in which the cloud service center assists in task processing by renting resources from ENs. The goal is to minimize resource renting costs while guaranteeing task completion. Shi et al. [59] let users pay for resources based on task loads. To avoid passive resource allocation of ENs, they design the service reliability metric of ENs as offloading guidance.

The game between ENs and end devices is widely studied, in which ENs adjust the prices to maximize profits whilst the end devices make offloading decisions to reduce costs. For example, Du et al. [60] try to maximize the profits of blockchain miners. Since more resources can increase the probability of getting mining rewards, miners need to adjust the rent amount and price of resources. The balance between the renting cost and the expected reward needs to be optimized. Seid et al. [61] formulate the game as a Stackelberg game in which ENs propose resource prices based on user demands, and then users make decisions to reduce the overhead. Zhang et al. [62] concern the trade-off between delay and energy costs from the user's perspective, and balance the service benefits and energy costs of the assisted vehicles. The Stackelberg equilibrium is achieved by dynamically pricing the resources with an iterative algorithm.

3.3. Mobility awareness & control

The mobility of end devices and ENs poses a great challenge to the design of computation offloading methods. End devices may move out of the coverage of the original edge server. It would cause communication link interruption, task processing failure, or additional data transmission overhead. Changes in the relative distance also affect the wireless communication state which induces transmission rates and costs.

The out-of-coverage problem has been most extensively studied in the field of vehicular edge networks, where vehicles can quickly move through the service area of ENs. To tackle it, Kazmi et al. [63] directly mask out the ENs currently outside the service range. Considering the movement of devices, Zhao et al. [64] further restrict tasks to be offloaded to ENs that are capable of finishing the task before the end device leaves their coverage. Geng et al. [65] construct service availability metrics of ENs and bound users to select ENs that have high service availability.

However, accurately estimating the vehicle speed and trajectory is difficult and computation-intensive. Therefore, some works attempt to explore services or data migration between ENs. When the communication link is interrupted, the service or data is allowed to be migrated to a second EN, as shown in Fig. 5. Such an operation helps ensure task completion at the expense of additional transmission costs. For example, Wu et al. [66] migrate the services when the offloading targets change. The service migration latency is simply modeled as the service data size divided by the bandwidth. Maleki et al. [67] adopt the handover approach where partial results are returned when vehicles leave the coverage area of the previously assigned EN. The result data is then uploaded to a newly selected EN for continuous processing. Tang et al. [68] discuss the offloading of sequential tasks in moving vehicles. If the posterior tasks are not processed at the same EN as the prior tasks, it is necessary to deliver the prior results via edge-edge communication, which introduces additional communication delay.

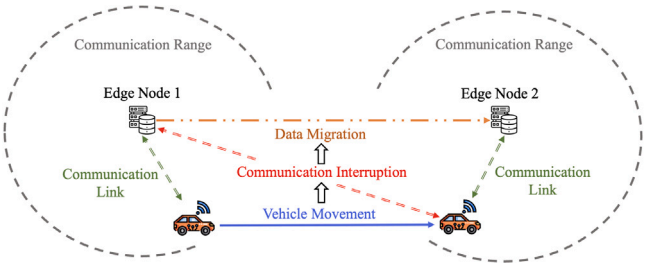

Fig. 5. An example of task data migration. When the vehicle moves out of the communication range of EN 1, the required data or services need to be migrated to EN 2 for a takeover. Additional data migration costs should be considered.

Besides, the movement of devices would change the distance to ENs, which consequentially impacts the path loss and the transmission rate [59,63]. Thus, it is important to incorporate geographic information of devices and ENs, including but not limited to location and relative distances [69], to optimize the selection of offloading targets.

In addition, to further reduce the overhead associated with transmission distances, mobility control has been extensively analyzed in recent works. Although it is impractical to directly control the movement of end devices, adjusting the flight trajectory of UAVs is feasible in aerial-assisted edge computing systems. The dynamic mobility control of UAVs could optimize the locations to match the random movement of ground devices. Dai et al. [70] attempt to adjust the UAV trajectory to approximate ground device clustering centroids to reduce the average communication distance. They also provide a propulsion-related energy consumption model for UAVs. Different from directly managing the target position, Zhao et al. [71] controls the direction and flight distance in each timeslot. In addition, Zhang et al. [72] combine the problem of UAV trajectory control with the problem of task offloading, i.e. determining which IoT device the UAV moves to for service provision.

3.4. Joint caching management

The caching mechanism for ENs plays a key role in reducing task processing costs. Remote data retrieval during offloading can be avoided by pre-storing relevant data on ENs. Much research effort has been devoted to jointly optimizing task offloading and caching placement. These caching modules typically perform the following types of functionality: essential data caching, result caching, and service caching.

Essential data refers to commonly accessed content such as video files and the original data of the tasks. In this context, Wang et al. [73] suggest ENs to cache requested content with limited storage capabilities. Cache hits can reduce the transmission delay of acquiring content from the cloud. To utilize the cooperation of multiple edge nodes, cooperative content retrieving is also applied [74]. They enable

content acquisition from nearby ENs instead of cloud servers to reduce transmission costs and increase efficiency.

Several works proposed to directly cache computation results for the tasks. When similar tasks are offloaded, results could be directly returned to avoid redundant data uploading and task processing. In [75], the local cache and nearby EN caches are queried sequentially when tasks are offloaded. Duplicate computations are avoided if the cache hits, otherwise the task processing is executed. However, since cache query overhead is non-negligible, improving the cache hit rate is required. Peng et al. [76] studied a shared offloading system. Tasks are split into multiple blocks and result sharing is available between similar blocks. It leads to lower latency and energy costs but results in extra costs for content searching. Whether to use shared offloading should be decided.

Service caching is also relevant to computation offloading since task processing requires the deployment of the corresponding service. Yu et al. [77] directly constrains that the valid offloading target should have proactively cached corresponding services. Xue et al. [78] divides the Cloud-Edge-End system into four layers. After selecting the offloading target, it needs to check the caching status from the target layer upward and actually offload to the nearest location for service caching. Li et al. [79] decide whether to replace the service caching, which follows the first-in-first-out policy. The tradeoff between the price of adjusting service caching and the reduced task processing costs is optimized. Zhang et al. [80] formulate the service migration overhead. When a task is offloaded to an EN without the corresponding service caching, additional software preparation costs are incurred, i.e., the service transmission latency via the backhaul link.

3.5. Interdependent task offloading

A complete task, e.g., a workflow, can be modeled as several subtasks with dependencies, which implies that succeeding subtasks need to rely on the results of the prior subtasks. Such a feature has been extensively considered in recent years. Depending on the context, a task can be formulated as a sequence or a directed acyclic graph (DAG) of subtasks.

Subtasks are assumed to be processed sequentially in [45,81]. If two succeeding tasks are offloaded to different ENs, it incurs a handover delay. Chen et al. [82] consider that the output of the preceding subtasks are necessary to be transmitted via the Edge-Edge channel when the handover occurs. The duration of delay is related to the data size and the transmit rate. Similarly, in [83] it is presumed that no transmission delay is incurred when offloading to the same target. Otherwise, the delay is estimated by using the allocated bandwidth to represent the transmit rate.

Describing a task as a directed acyclic graph (DAG) of subtasks is widely adopted since it can represent more complex dependencies. To simplify the problem, some works like [65] sort the tasks by some kind of priority and optimize the accumulated delays of all tasks. However, it ignores the efficiency of parallel processing, leading to a higher total delay. Instead, [84] directly optimizes the total time costs of the critical path, i.e. the path with maximum delay. Yuan et al. [85] focus on optimizing the earliest start time, which is reformulated as the maximum value between the previous subtask completion time and the node available time. Besides dependencies of subtasks, Zhou et al. [86] even includes the external dependencies between tasks, where subtasks in different DAGs can share data for cooperation and obtain better utility.

3.6. System security & reliability

The application of edge computing in industrial IoT and vehicular edge networks usually demands high reliability and security, which are threatened by unstable, insecure wireless communication channels and the unpredictable failure of devices. These have raised increasing concerns when it comes to service offloading.

Affected by the complex wireless channel state, transmission failures may occur during data uploading and downloading. Earlier methods such as [87,88] simply use a fixed value to represent the transmission failure probability. This model ignores the connection between system state and transmission failures. Lu et al. [89] point that the transmission failure rate is correlated with the probability of coding errors, which can be further expressed as a function of SINR, packet size and coding block size. Fang et al. [90] and Nguyen et al. [91] construct the transmission model under imperfect channel state information (CSI) where the transmission fails if the actual SINR is below the threshold. Based on these models, the DRL agent can learn to make reliable decisions based on its observations of the system.

Interference and eavesdropping by malicious attackers are serious threats to the system's security and reliability and have been discussed in many works. On the one hand, attackers could broadcast noise within the channel to interfere the normal data transmission. This interference is mostly considered as a strong noise signal that mathematically results in lower SINR, which in turn reduces the transmission rate [92]. Xu et al. [93] consider a harsh condition where the attacker could completely block all transmissions within the affected channel. On the other hand, some studies pay attention to the eavesdroppers who attempt to intercept data sent by end devices. Supported by the physical layer-based wiretap coding scheme, eavesdroppers cannot access confidential information at a secure rate, i.e., the difference between the transmission rate and the eavesdropping rate [94]. To address the eavesdropping problem, some studies have suggested that one can compress the eavesdroppers through auxiliary devices or the BS [95,96]. In addition, blockchain is also widely adopted to improve data integrity and security [61,97]. However, it could introduce extra computation requirements. Thus, many studies integrate task offloading with blockchain parameter optimization such as the block size and the block interval to balance computation cost and benefits of security [98,99].

During the computation phase, device failures may occur due to hardware or software malfunctioning. This breaks the task processing and affects system reliability. Such failures are usually unpredictable. For this reason, an effective approach is to treat device failures as random events following the Poisson distribution described by a failure rate parameter [100,101], which is calibrated on the fly according to the number of failure events in the past period [102]. In addition, some studies intuitively focus on reducing task failures [103] caused by response latency and energy shortage. In this paper, we discuss energy management issues in Section 3.1 and view latency as a constraint or a contributing factor in the design of rewards of the MDP model (Section 4).

4. MDP model construction

A fundamental step of developing DRL-based algorithms is the abstraction of the Markov Decision Process (MDP), in which the agent takes an action after observing the system state and receives the corresponding reward. Given the various scenarios and optimization goals, the construction of MDP models diverges significantly across studies. This section aims to shed light on the fundamental elements included in MDP models, particularly for task offloading. A simple summary of these elements is shown in Fig. 6.

4.1. State and observation

State or observations typically contain factors that are dependent on scenarios and objectives. Agents take them as input to sense the environment. For task offloading, existing studies typically consider network state, task state, and device state. Additional information

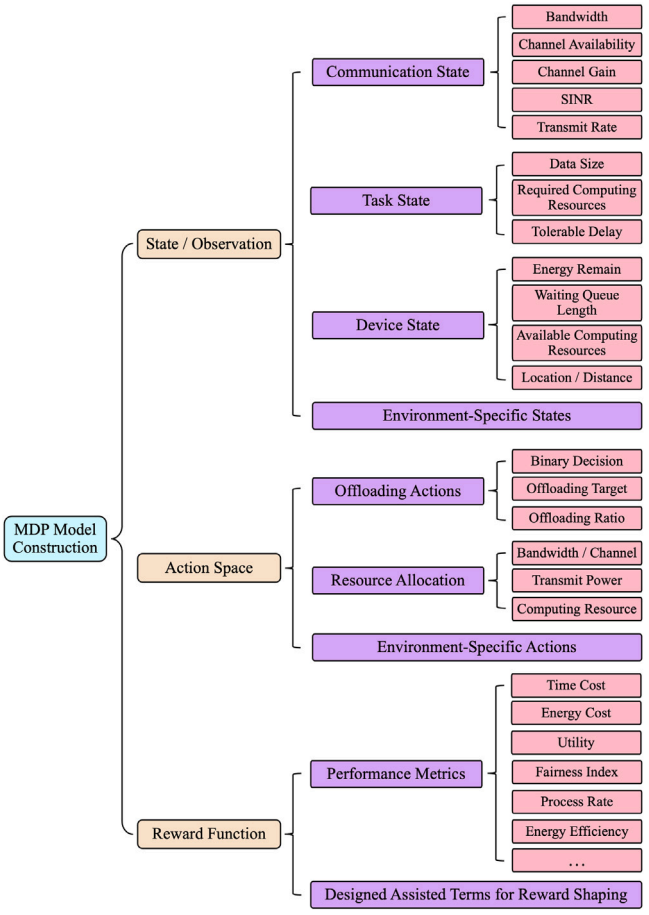

Fig. 6. A summary of elements considered in the literature regarding the MDP model for offloading.

can be exploited to expand the dimensionality of states to facilitate decision-making.

The network state is closely related to the transmission rate and costs. For example, it is very common to include channel gain in the state vector as it affects the transmission rate [104,105]. Some studies also include SINR to cover more comprehensive channel quality information related to background noise and interference [73,92]. In some contexts, it is feasible for the agent to obtain the actual transmission rate for any given channel [76,106]. In the case of variable spectrum frequency, the bandwidth is provided to the agents as part of the state information [43]. With orthogonal frequency division, the channel resource block (RB) availability needs to be integrated into the state information [69,77]. Otherwise, the historical channel load or interference intensity should be included to reflect spectral resource contention (e.g., in NOMA) [62,107].

Task states include data size and the level of computation load. In general, tasks are often assumed to be generated on user devices with uniformly distributed data size. The compute resource requirements could be linearly related to the data size or sampled independently. When the tasks are time-sensitive, the maximum tolerable delay will be included to guide policies safeguarding the deadline. Beyond these, depending on the scenario factors considered, some other information is also included, such as the popularity [92], and the finish reward [108].

Device states represent the attributes of user devices and ENs. For communication, position or distance is usually included as they relate to the path fading strength and the movement of devices. For computing, the available computing resources of devices are the most relevant. When tasks are difficult to complete in a single timeslot, the current waiting backlog queue length should be included. For devices that rely on batteries with limited power supply, the remaining available energy makes an important dimension of the state. In addition, the caching status and the remaining storage space should also be taken into account when jointly optimizing caching management [76,109].

Additional information could be useful for action selection. For instance, historical information is a good choice to explicitly perceive changes in the environment. For example, Tang and Wong [110] contain historical load levels, thereby facilitating agents to sense load patterns. Cui et al. [111] include all historical states to provide more comprehensive information but might lead to large state space dimensions. Chen et al. [112] include historical actions to perceive the decision patterns. Some approaches [82,113] also include historical responses such as time costs and energy costs to make corrections and improvements. In addition, environment-related information should also be considered. For example, Geng et al. [65] contain task information for the predecessor and successor subtasks when offloading dependent subtasks. Sun et al. [114] include the communication coverage graph as an adjacent matrix. Yu et al. [77] contain the large timescale actions as part of states for small timescale decision-making.

4.2. Action space

DRL agents determine actions to interact with the environment based on the acquired state. The actions mainly consist of offloading-related decisions and are usually jointly determined with resource allocation. In addition, many studies further combine it with special factors such as caching placement and movement control for joint decision-making.

For specific types of tasks, the offloading actions can be categorized into binary offloading and partial offloading. Binary offloading refers to having the task either completely offloaded or processed locally. It is typically adopted when the task is atomic and not divisible. For divisible tasks such as streaming data processing, partially offloaded can be taken for processing in parallel.

Based on the number of available target ENs, offloading actions can be categorized into two types: single-target and multi-target. In earlier work, it was common to divide the edge system into multiple small single-cell sub-systems that contain only one EN to be selected for binary offloading. It effectively reduces the action space and facilitates fast convergence. Recent researches pay attention to large-scale edge systems with multiple ENs, even together with the cloud center to provide services. In this case, one of the service providers needs to be selected, i.e., multi-target offloading. Each end device needs to select a one-hot vector as the offloading action. Such offloading mode is also called the user association from the perspective of ENs.

Using a special encoding scheme, Yin and Yu [115] provide decimal IDs for end devices and directly select devices by IDs to represent offloading actions. Sharma et al. [84] slice tasks into multiple atomic subtasks with separate binary offloading. Wei et al. [116] construct the offloading decision as a triplet decision including offloading, waiting, or dropping. In Yang et al. [117], UAV deployments have multiple models with different computing costs and data compression capabilities. Decisions need to be made on which model to use.

The decision on computation offloading is often made together with resource allocation, but their action space requires a different design. For network resources, the allocation of RB, bandwidth, and transmit power are the main factors. An RB is a unit of radio resources consisting of a certain number of subcarriers and/or a certain amount of time frame. Each offloading process should exclusively occupy one of the RBs in orthogonal transmission, hence a binary decision should be made. Some works allocate bandwidth directly to transmission processes. The action space is a continuous space that represents the allocation ratio and constrains the total ratio to be less than 1. Transmit power allocation forms a trade-off, where higher power increases

the data rate and brings higher energy costs. In addition, for non-orthogonal transmission, it is more necessary to adjust the transmit power to minimize co-channel interference. It has a continuous action space and is constrained lower than the maximum transmit power. Computing resources are normally measured by only CPU cycles. For arriving tasks, a certain percentage of CPU frequency can be allocated to each task or arranged as a queue to adjust the CPU frequency sequentially for computation based on Dynamic Voltage and Frequency Scaling (DVFS).

Environment-specific actions are designed to fit particular scenarios. For example, in the energy harvesting scenario, each timeslot needs to be properly split for WPT and task processing respectively to avoid energy shortage [103]. To encourage service providing, Shi et al. [56] jointly decide on the unit price paid per rent CPU cycles for processing, and Zhu et al. [118] make service pricing and offloading decisions by competitive agents. The end device or EN movement is commonly controlled by adjusting the direction and speed [68]. Some works also try to replace the speed with the moving distance or the target location [105], but the maximum flying length constraints during one timeslot should be considered. Caching decisions are typically taken with a binary action space, where each content should decide whether or not to be cached in each edge server [77,109].

4.3. Reward function design

The reward function, which is related to the objectives, specifies the immediate payoff that an agent can receive for taking an action. In computation offloading, the processing latency and energy costs are commonly used as the optimization objectives. Several derived metrics have been further proposed to guide the learning and decision-making process towards better system performance.

The weighted sum of time and energy costs is commonly adopted as the basic form of reward function. For example, the reserved time given the tolerable delay can be a viable replacement of time costs [119,120]. Jiao et al. [121] believe that optimizing energy costs is more important when the remaining energy is lower. Hence they leverage the sigmoid function to implement an adaptive weight function. Many studies also focus on optimizing one of them [122,123], while ignoring or only applying a maximum usage constraint on another one. This approach can more effectively focus on reducing system latency or energy costs but ignores the trade-off and tends to overspend and diminish returns.

Utility is a derived metric commonly defined as the difference between the benefits and costs of computation offloading. Zhan et al. [124] assume that the EN proportionally allocates resources according to the uploaded data size from multiple end devices, and more resources allocated leverage better performance. The utility is defined as the resources obtained minus the costs. For tasks with different priority levels, Shi et al. [56] represent the utility of high-priority tasks as a saved-time dependent function and gives a fixed penalty for timeouts. For low-priority tasks, the finish reward is constant but decays with the degree of timeout. From the perspective of the whole system, Zhao et al. [64] define utility as the saved rate of average delay and energy cost of current tasks compared to the previous timeslot. Wang et al. [73] divide the total utility into communication, computation, and caching utilities. Given the unit price of each resource, each utility is the difference between the processed data rate and the resource renting price.

Load balancing or service fairness is a critical aspect of task offloading. Imbalanced workloads can result in unnecessary resource contention, high processing latency, and low resource efficiency. The intensity of workload can be defined in various ways, e.g. the task resource demand [125], the estimated task completion time [126], or the amount of resource available [127]. Based on the distribution of workloads, it is common to add a particular term to the reward function to encourage service fairness. For instance, the Jain fairness index is widely employed in the literature [52,128]. Besides, Zhang et al. [129] use the mean square error to encourage workload levels around the average. Hao et al. [130] use the α-fair utility of the average time reduction as the optimization goal. Based on each user's downlink data rate, Yin and Yu [115] design a fairness metric based on entropy with the aim of balancing the throughput of each user.

When tasks are assumed to arrive in a stream and are processed continuously, the processing rate can be adapted to measure the system performance [131]. Energy efficiency, which is the ratio of processing rate to energy costs, is adopted in some studies to encourage higher processing rates per unit energy cost [132,133]. It is more compatible with the reality of energy-limited edge devices and can effectively avoid the diminishing returns situation. To avoid tasks expiring and failure, many works incorporate task completion rewards or task failure penalties [103,116]. [76,134,135] consider both benefits and costs and incorporate the price paid in their reward function.

The timing of reward is critical for guiding the DRL agent during training. For example, Kumar et al. [136] transform a long-term constraint into an instant penalty term in the reward function based on Lyapunov Optimization. In UAV movement control scenarios, a large constant penalty term is applied when UAVs collide or go out of bounds [137]. Tan et al. [24] argue that reducing the allocation of resources primarily contributes to energy savings. Therefore, they use the minimum amount of required resources to satisfy delay constraints as the reward function. For better energy management, Zhu et al. [138] design auxiliary rewards about energy reserved, policy fairness as well as the remaining battery with adaptive weights. Moreover, scenario-specific metrics including data security [94,139] and accuracy [140,141] can also be taken into account.

Reward shaping is a useful technique in DRL that introduces additional incentives to agents to guide the learning process. It has been applied by many methods to rationally organize the reward function. For example, Geng et al. [65] simultaneously consider the reward of a selected action as well as that of fully local processing. The ratio of these two is taken as the actual reward to prevent fluctuations caused by random task generation and to make the reward more informative. Xu et al. [142] employ a weighted sum of time costs and time-to-deadline as a reward function. This implies a soft constraint by the tolerable delay. Guo et al. [143] and Chen et al. [144] set a positive constant reward for performance improvement over the previous timeslot and penalize vice versa to encourage better decisions. Zhao et al. [145] design a multi-layer hierarchical reward function for different constraint violations and optimization objectives. It gradually provides information to motivate agents to satisfy each constraint. To motivate better resource utilization while avoiding excessive resource allocation, Peng et al. [146] introduce a bootstrap function related to the remaining resources. Zhan et al. [147] add penalties to discourage task queue overflow and illegal decisions. This helps the agent avoid unnecessary exploration and improves training efficiency.

To summarize, we found that most of the existing studies define the reward based on task processing gains, response latency, and energy cost. Some additional factors are included to meet specific requirements or to carry certain prior knowledge regarding the environment. Different metrics result in different optimizing tendencies and characteristics, which need to be properly selected and adjusted according to the actual scenarios.

5. Learning strategy improvements

Recent studies have implemented various DRL algorithms and attempted to achieve better performance by improving the learning strategy. Based on the difference in methodologies, we classify existing researches into the following categories: better information usage, improved exploration and exploitation, reduction of action space, and multi-agent cooperation and competition.

Table 2
Related works of basic DRL algorithms. CR is the abbreviation for Computing Resources.

	Offloading type	DRL Approach	State				Action	Reward
			Network state	Task state	Device state	Additional		
[148]	Partial Multi-Target	DQN		Task Size Required CR	Available CR	Offloading Matrix	Offloading Target Offloading Ratio	Time Cost Energy Cost
[149]	Binary Multi-Target	DQN	Transmit Rate	Required CR Number of Tasks	Available CR		Offloading Target Node Selection Block Interval Block Size	Task Finish Score Throughput
[76]	Binary Multi-Target	DDQN	Transmit Rate	Task ID	Available CR Waiting Queue Cached Task ID		Offloading Mode	Time cost Energy Cost Price Paid
[43]	Binary Single-Target	DDQN	Bandwidth		Available CR		Offloading Decision Spectrum Allocation CR Allocation	Energy Cost
[107]	Binary Multi-Target	DDQN	Bandwidth Network Load	Required CR Tolerable Delay	Current Load		Offloading Target	Task Finish
[150]	Partial Single-Target	DDPG		Task Size	Energy Remain Location or Distance		Agent Selection Flight Angle Flight Speed Offloading Ratio	Time Cost
[132]	Binary Multi-Target	Async AC	Transmit Rate Bandwidth		Available CR Waiting Queue Max Transmit Power		Offloading Target Bandwidth Transmit Power CR Allocation	Energy Efficiency
[92]	Partial Multi-Target	AC	Bandwidth SINR	Popularity Required Time	Energy consumption		Offloading Target Offloading Ratio Transmit Power	Time Cost Energy Cost SINR
[151]	Binary Single-Target	DDPG	Channel Gain SINR	Task Size	Waiting Queue		Offloading Decision Computing Power Transmit Power Bandwidth	Time Cost Energy Cost Bandwidth Usage
[81]	Binary Multi-Target	A3C		Task Size Required CR	Location or Distance Available MEC	Historical Decision Historical Delay Handover Delay Historical Energy Cost	Offloading Target	Utility
[69]	Binary Single-Target	TD3	RB Availability	Task Size	Available CR Location or Distance		Offloading Decision CR Allocation RB Allocation	Time Cost Energy Cost
[134]	Partial Multi-Target	TD3	Bandwidth Channel Gain	Task Size	Available CR	Historical Delay	Offloading Decision Offloading Ratio Bandwidth	Price Paid
[152]	Binary Multi-Target	SAC	SNR Link Duration	Task Size Required CR Tolerable Delay	Available CR Reliability		Offloading Target Unit Service Price	Utility
[56]	Binary Multi-Target	SAC	SNR	Task Size Required CR Tolerable Delay	Available CR Service Availability		Offloading Target Price Paid	Utility

5.1. Basic DRL algorithm

Most of the DRL methods are model-free, i.e. they do not require an explicit environment model. Following this rationale, basic DRL algorithms are widely adopted in the literature (see Table 2).

Value-based DRL algorithms are suitable for discrete action space because they perfectly match the binary offloading decision. For example, Li et al. [148] maintain a device-to-EN offloading ratio matrix. DQN is applied to select an action between $-1, 0, 1$, respectively indicating decreasing, maintaining, or increasing the offloading ratio for each element in each timeslot. In [149], the system's trustworthiness is ensured by introducing blockchain for data management. In addition to traditional task offloading decisions, DQN is used to optimize the parameters of the blockchain to reduce the additional overhead. To improve the over-estimation problem of DQN, DDQN has been proposed and has been widely used in the computation offloading of edge

systems. For example, DDQN is applied in [76] to potentially sense caching placement strategies. By intelligently deciding the offloading target and offloading mode selection, it can effectively promote collaboration with the cache. To support continuous action selection, Zhou et al. [43] discretize continuous resource allocation actions and adopt DDQN for decision-making. Since the reward is only received after the task is completed in reality, Yamansavascilar et al. [107] propose the delayed transitions-based DDQN, where the transition is recorded only after the reward for the action is returned).

The Actor-Critic (AC) framework is suitable for handling both continuous action space and discrete action space. Xiao et al. [92] consider a partial offloading scenario with non-orthogonal transmission interference. By adopting AC, the transmit power is fine-tuned to improve system utility. To facilitate data acquisition and narrow the gap between simulation and reality, a digital twin model is constructed in [132]. The Asynchronous Actor-Critic algorithm is applied, where multiple agents collect transitions into a global replay buffer for training. Gu et al. [81] focus on edge-edge collaboration where dependent subtasks can be switched between ENs for processing. They propose an A3C-based scheduling algorithm for offloading target selection.

DDPG was designed to learn precise and efficient deterministic policies under the AC framework. In Ke et al. [151], DDPG is applied and the Ornstein–Uhlenbeck noise parameter is fine-tuned for different actions to obtain better results. Wang et al. [150] analyzed the UAV-assisted partial offloading edge computing network and adopted DDPG to make decisions on UAV flight direction, speed, and offloading ratio. As an improved method of DDPG, TD3 is applied in [69]. In comparison to DQN and DDPG, TD3 achieves significantly higher performance in the constructed binary offloading and resource allocation environments. Huang et al. [134] conducted a similar study on partial offloading and bandwidth allocation with TD3. Their results show that both DDPG and TD3 outperformed DQN.

In recent years, Soft Actor-Critic (SAC) has achieved strong performance in many fields and has been applied to computation offloading. By incorporating entropy regularization, SAC achieves a better balance between exploration and exploitation, which results in a robust and efficient policy in multiple scenarios. Shi et al. [152] propose a SAC-based task allocation algorithm to decide the offloading target and the unit service price paid in blockchain-based vehicular networks. In [56], SAC is also applied to determine the offloading target and the price paid to the server, which is related to the computing resources the server is willing to allocate. Experiments show that SAC achieves better performance in comparison with other DRL algorithms.

5.2. Better information usage

To further strengthen the effectiveness of DRL-based scheduling algorithms, numerous studies have focused on optimizing information utilization. The goal is to enable agents to perceive the environment more comprehensively and thus make better decisions. Considering the heterogeneous nature of data structures, related approaches focus on the extraction and utilization of state information using special network structures (Fig. 7). In Table 3 we provide a summary of the related studies.

Historical information includes historical states, actions, and rewards. With this information, the agent can better understand the environment dynamics and long-term effects of actions. Xu et al. [133] include the historical QoS as part of the state to characterize the dynamic and periodically changing QoS. Cui et al. [111] focus on DNN inference tasks and adopt long-short Term Memory (LSTM)-assisted TD3 to extract historical state features. Instead of assembling historical data into states, Lu et al. [125] adopt DRQN, where a temporal feature is maintained. With LSTM, the current state and the temporal feature will be fused to provide historical information and to update the temporal feature. A similar idea can be found in [110], where the historical load level is processed by an LSTM model to produce temporal hidden features.

To deal with the incompleteness of information, some studies use historical data to predict unknown information about current or future states. For example, Chai et al. [153] use GRU to learn workload patterns and predict the current workload levels. They apply Dueling Double DQN (D3QN), which combines the dual network structure and target network to accurately evaluate the state and action value with lower overestimation. In [119], GRUs are utilized to predict the available CPU resources of ENs. Therefore, tasks that might exceed the future resource constraints are migrated to idle ENs. In [104], vehicle trajectories are predicted by LSTM. A task vehicle is clustered with multiple service vehicles by trajectory distances, and highly overlapping clusters can be further joined together. DQN is used for independent decision-making within each cluster. Xu et al. [154] employ a graph weighted convolution network (GWCN) for traffic flow prediction, which is related to the computation loads. Therefore, potential high loads can be sensed and rationally scheduled in advance to avoid overloading. Li et al. [105] assume that the states of ground devices are large and cannot be obtained immediately. They propose to predict the future state of ground devices using LSTM. Instead of relying on real-time acquisition of all states, UAVs make decisions based on the future states and fetch them with higher latency. Tian et al. [155] focus on the impact of content popularity on the joint optimization of task offloading and caching scheduling. They propose the OSTP algorithm to learn temporal features from three different granularities (closeness, period, trend) and use the DeepSTN+ network to predict content popularity.

Graph neural networks (GNNs) can serve as a powerful feature extraction model to assist DRL-based decision-making. Li et al. [156] encode the dependency between subtasks in the DAG task with GNN. The Seq2Seq paradigm with LSTM is adapted to encode the contextual representations of subtasks and decode the global features into offloading decisions according to the processing order. Meta-PPO is applied for better performance and adaptability. Sun et al. [114] utilize GNN to embed the device positions and communication coverage of ENs. The graph features held by each node will be used for offloading decision-making through MLP. Action quantization is applied for exploration. The policy network is trained to minimize cross-entropy loss between the output and the optimal action with maximum reward, which can be seen as a special policy-based DRL. Since instantly getting the network state is difficult, Li et al. [29] propose a GNN-A2C algorithm, in which GNN is adopted to predict network states and A2C makes decisions based on it. Zhou et al. [157] propose a caching placement and computation offloading scheduling algorithm. They utilize the Spatial–Temporal Graph Neural Network (STGNN) to predict future content requirements for content caching. TD3 is applied for offloading decision-making.

In addition, Hou et al. [158] follow the idea of ensemble learning and propose two strategies. The first strategy adopts LSTM-based PPO to aggregate historical states. The second one is a past average strategy based on supervised training with stored transitions. One of the generated actions of these strategies is selected based on a predefined probability. Liu et al. [159] design a double-net DDQN approach, where two DNNs separately evaluate the time cost and energy cost of state–action pairs. To combine these values for action selection, they propose to use the attention mechanism to adaptively compute the weighted sum of them. Supervised training of the attention module can be performed by collecting the most appropriate weights when interacting with the environment. Similarly, Li et al. [108] design two independent agents that focus on optimizing a single objective. Under the AC framework, one agent focuses on reducing system overhead while the other focuses on increasing service profit. This allows network managing operators to choose policies that meet different requirements.

Table 3
Related works of better information usage. CR is the abbreviation for Computing Resources.

	Category	Offloading type	DRL Approach	State				Action	Reward
				Network state	Task state	Device state	Additional		
[133]	Integrate Historical Info	Binary Multi-Target	DDPG			Waiting Queue Energy Remain	Historical QoS	Offloading Target Channel Allocation Transmit Power Allocation	Energy Efficiency
[111]	Integrate Historical Info	Binary Multi-Target	TD3		Task Size Required CR	Waiting Queue	Historical State	Offloading Target	Time Cost Inference Accuracy
[125]	Integrate Historical Info	Binary Multi-Target	DRQN	Bandwidth	Task Size Required CR	Available CR	Hidden Feature	Offloading Target	Energy Cost Load Balancing
[110]	Integrate Historical Info	Binary Multi-Target	AC		Task Size	Waiting Queue	Historical Load Level	Offloading Target	Time Cost
[153]	Predict Unknown States	Binary Multi-Target	D3QN		Task Size	Waiting Queue	Processed Data Volume Historical Workload	Offloading Target	Time Cost Energy Cost
[119]	Predict Unknown States	Binary Single-Target	DDPG			States of ES and Cloud Available CR		Offloading Decision	Time Cost
[70]	Predict Unknown States	Partial Multi-Target	DDPG		Task Size Required CR Tolerable Delay	Location or Distance	Previous Action	Task Partition	Energy Cost
[104]	Predict Unknown States	Binary Multi-Target	DQN	Channel Gain		Waiting Queue		Offloading Target	Throughput Queue Stability
[154]	Predict Unknown States	Partial Multi-Target	AC				Offloadable Data Amount Acceptable Data Amount	Task Partition	Time Cost Energy Cost
[105]	Predict Unknown States	Binary Multi-Target	DDPG	Channel Gain		Waiting Queue Energy Remain Location or Distance		Device Selection UAV Target Location UAV Speed	Dropped Tasks
[155]	Predict Unknown States	Binary Multi-Target	DDPG	Bandwidth	Task Size Content Popularity	Available CR Location or Distance Caching Status		Offloading Target Caching Placement	Time Cost
[156]	Graph Information Utilization	Binary Multi-Target	PPO		Task Size Time Required Dependencies			Offloading Target	Time Cost
[114]	Graph Information Utilization	Binary Single-Target	Policy-based	Transmit Rate		Available CR Waiting Queue	Communication Graph	Device for Offloading	Time Cost
[29]	Graph Information Utilization	Binary Single-Target	A2C	SNR		Waiting Queue Energy Remain Location or Distance	Number of Offloading Tasks	Offloading Decision UAV Target Location UAV Speed	
[157]	Graph Information Utilization	Partial Multi-Target	TD3		Task Size Popularity	Location or Distance Available CR Storage Capability		Offloading Target Offloading Ratio	Time Cost
[158]	Better Policy Structure	Binary Multi-Target	PPO		Task Size Required CR Tolerable Delay	Waiting Queue Available CR	Historical States	Offloading Target	Utility
[159]	Better Policy Structure	Binary Multi-Target	DDQN	Channel Gain	Task Size Required CR	Energy Remain	Last Target EN	Offloading Target Transmit Power CPU Frequency	Time Cost Energy Cost
[108]	Better Policy Structure	Binary Multi-Target	Actor-Critic	Bandwidth	Task Size Required CR Finish Reward	Available CR		Offloading Target	Time Cost Energy Cost Task Failure Rate

5.3. Improved exploration and exploitation

The trade-off between exploration and exploitation is a major challenge for DRL-based approaches. Making decisions by exploiting existing experience often results in relatively high rewards in the short term. By contrast, exploring new states and actions can lead to the discovery of potentially better decision trajectories. Therefore, how to balance the two during training has attracted much attention (see Table 4).

The ϵ-greedy method is a traditional solution to the exploration–exploitation dilemma. With a probability ϵ, the agent chooses a random action for exploration or otherwise takes a policy-guided action. The ϵ value is the key parameter for balancing exploration and exploitation, yet it is difficult to accurately and conveniently obtain an appropriate one. Responding to this issue, Xu et al. [142] suggest decreasing the ϵ value as training goes to encourage more exploration in the early stage of training and prefer to exploit existing experience in the later stage of training. However, this is not enough to encourage agents to explore a larger state–action space, and the exploitation of overestimated data collected in earlier iterations at a later stage can lead to sub-optimal policies. Hence, in [160], ϵ can adaptively take a large value to force the agents to explore unknown regions and avoid the exploitation of incorrect actions.

Many researchers figured out that randomly generating actions for exploration is inefficient since the exploration could be invalid or valueless. An intuitive way to solve this problem is to restrict random action selection within the feasible space. For example, in Chen et al. [112], based on the constraints, the valid action space is determined under the given previous actions. The greedy or random action selection can only be performed within that space. To ensure that tasks are fully allocated for processing in partial offloading scenarios, Ale et al. [161] propose Dirichlet DDPG which characterizes the offloading action with Dirichlet distribution. It ensures that the constraints are met and also maintains exploration ability.

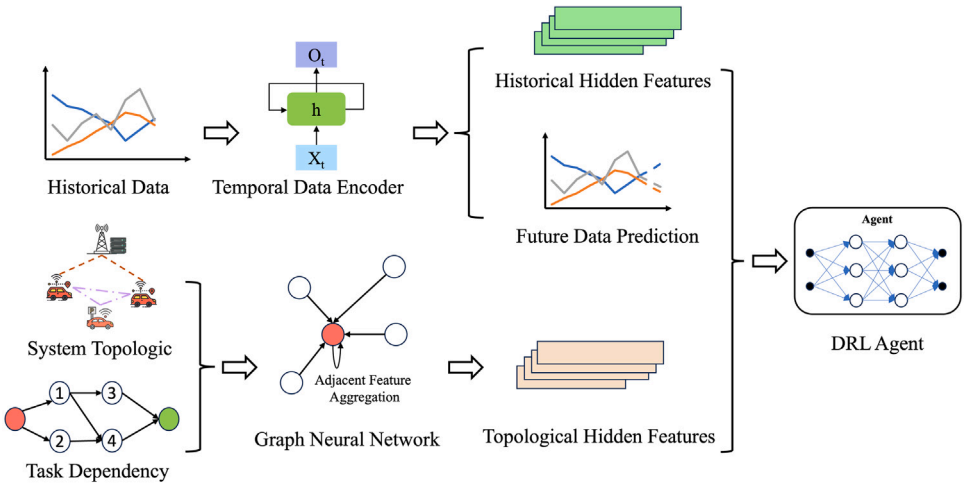

Fig. 7. A brief overview of better information usage. With a feature extractor, hidden features are extracted from various heterogeneous data to provide additional information for decision-making.

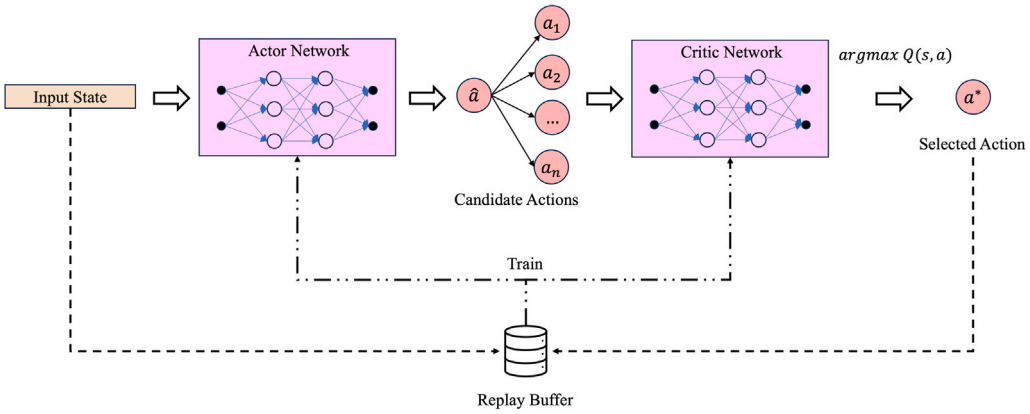

Fig. 8. The framework for action extension-based agent design. The generated action would be expanded into multiple candidate actions based on the designed algorithm. The optimal one would be selected to guide the agent training.

Furthermore, randomly generated actions may deviate far from the current policy. As a result, the information obtained does not provide appropriate guidance for policy updating. To address this problem, action expansion has been studied extensively, which expands the current actions to multiple candidate actions in the nearby action space, to find better-performing actions that can more effectively guide policy iterations, as shown in Fig. 8.

For continuous action space, some studies propose to sample a set of candidate actions that are distributed on both sides of the original actions. To find the optimal transmission duration, based on the selected action a, Qian et al. [162] generate a candidate action set $A = \{a - k\Delta, \dots, a - \Delta, a, a + \Delta, \dots, a + K\Delta\}$, where Δ is the exploration interval. The task partition decision and computing resources allocation are formulated as a strictly convex optimization problem and solved by conventional solvers like CVX. Hence, the current reward can be obtained, and the optimal action $\hat{a}$ in A is stored as an experience to guide the policy network update by gradient decent the mean square error $\mathcal{L} = (\pi(s) - \hat{a})^2$. Similarly, in [163], other actions can be optimally decided via the proposed Lagrangian duality-based convex optimization method. In [47], the first-order derivative of the function between the transmit energy proportion and the optimization objective is monotonically increasing and the optimal value can be obtained by bisection.

For discrete action space, the order-preserving quantization method (OPQ) for action generation is proposed by Huang et al. [164]. In this way, the policy network generates a continuous action and expands it into a candidate set of binary offloading actions by OPQ. The optimal offloading action is selected as the training label to induce the generated continuous actions approximating the optimal offloading action. They claim that the OPQ for action generation creates higher diversity to find the optimal action and has better convergence than the KNN method. Such a paradigm has been rapidly adopted by recent studies. For example, in [165] a DRL agent with OPQ makes the offloading decision, and the optimal value of the WPT duration and allocated computing time are obtained via the designed worst WD-adjusting approach. Jiao et al. [121] introduce crossover and mutation to further improve the diversity of the generated candidate offloading sets in the OPQ. They propose a feasibility analysis algorithm that selects only feasible actions for interaction and training. However, OPQ can only generate binary discrete actions. To address this issue, Chen et al. [166] assume that tasks are discretizable for offloading. They propose a new action generation method that is suitable for any discrete decisions, which helps with offloading ratio selection. Chen et al. [167] also provide an order-preserving quantization approach for any discrete actions. The action with the highest joint action value is stored to train the Q-Mix algorithm for higher training efficiency.

Jiang et al. [28] propose a Light Wolf Search algorithm for action refinement. They utilize the Gray Wolf Optimizer (GWO) and update the offloading part of the light wolf based on the channel gain information. Proportionally reduction is applied to the allocated resources to meet constraints. Liu et al. [122] discretize the continuous outputs to feasible BS selection and offloading decisions. They propose

Table 4
Related works with improved exploration and exploitation. CR is the abbreviation for Computing Resources.

	Category	Offloading type	DRL Approach	State				Action	Reward
				Network state	Task state	Device state	Additional		
[142]	Adjusting ϵ	Partial Multi-Target	Independent AC	Bandwidth Channel Gain	Task Size Required CR Tolerable Delay	Available CR		Offloading Ratio Resources Type Matching Transmit Power CR Allocation	Time Cost
[160]	Adjusting ϵ	Binary Multi-Target	DRQN		Number of Arrived Tasks Number of Waiting Tasks Number of Processing Tasks	Available CR		Offloading Target Number of Tasks Started	Task Finish Reward Task Drop Penalty
[112]	Restrict Exploration Space	Binary Multi-Target	DQN	Transmission Rate		Available CR	Previous Decisions	Scheduling Decision	Time Cost
[161]	Restrict Exploration Space	Partial Multi-Target	DDPG	Transmission Rate	Task Size Required CR Tolerable Delay	Available CR		Offloading Target Offloading Ratio Computing Resources	Task Finish Reward Time Cost Energy Cost
[162]	Action Generation	Partial Single-Target	Policy-based	Channel Gain				Transmit Duration	Energy Cost
[163]	Action Generation	Partial Single-Target	Policy-based	Channel Gain				WPT Duration	Process Rate
[47]	Action Generation	Partial Single-Target	Policy-based	Channel Gain				WPT Duration	Process Rate
[164]	Action Generation	Binary Single-Target	Policy-based	Channel Gain					Process Rate
[165]	Action Generation	Binary Single-Target	Policy-based	Channel Gain	Task Size			Offloading Decision	Time Cost
[121]	Action Generation	Binary Single-Target	Policy-based	Channel Gain	Task Size	Energy Remain		Offloading Decision	Time Cost Energy Cost
[166]	Action Generation	Partial Single-Target	DQN	Channel Gain	Task Size	Energy Remain		Offloading Ratio Channel Selection	Energy Cost
[167]	Action Generation	Binary Multi-Target	Q-Mix				Historical Time Cost Service Reputation	Offloading Target	Transmission Rate
[28]	Action Generation	Binary Multi-Target	AC	Channel Gain	Task Size Required CR			Offloading Target CR Allocation	Energy Cost
[122]	Action Generation	Binary Multi-Target	Policy-based	Channel Condition	Task Size Required CR			Offloading Target BS Selection	Time Cost
[168]	Action Generation	Binary Multi-Target	Policy-based	Channel Gain				Offloading Target	Time Cost Energy Cost
[138]	Improved Experience Replay	Binary Single-Target	DDPG			Energy Remain	Historical Decision	Offloading Decision CR Allocation Bandwidth Allocation	Number of Finish Task
[169]	Improved Experience Replay	Partial Multi-Target	DQN	Channel Gain	Task Size Required CR Transmission Delay QoS Type			CR Allocation Bandwidth Allocation Workload Control Factor	Spectral Efficiency Task Success Rate
[170]	Improved Experience Replay	Partial Multi-Target	SAC			Load Available CR		Offloading Target Offloading Ratio	Load Balancing Time Cost
[171]	Improved Experience Replay	Partial Single-Target	DDPG	Maximum Power Channel Gain	Task Arrival Time Slot Task Size Required CR Tolerable Delay	Available CR		Offloading Ratio CR Allocation Transmission Power	Time Cost Energy Cost
[65]	Training Optimization	Binary Multi-Target	AC		Task Size Required CR	Available CR Geography Info	Historical Actions Predecessor Task Info Successor Task Info	Offloading Target	Time Cost Energy Cost
[172]	Training Optimization	Binary Single-Target	AC	Channel Gain Noise Power	Task Size Required CR Tolerable Delay	Available CR Waiting Queue Geography Info		Offloading Decision Channel Selection	Energy Cost
[82]	Training Optimization	Binary Multi-Target	AC				Historical Time Cost Historical Energy Cost	Offloading Target Bandwidth Allocation CR Allocation	Reward for Better Performance

a Weighted Congestion Game-based Algorithm (WCGA) solver, which sacrifices precision for faster convergence and decision making. Jiang et al. [168] propose a Lévy Flight Search-based action generation method. To generate discrete offloading actions and utilize channel state information, the offloading action is randomly generated or kept unchanged by comparing the normalized channel gain with the Lévy flight step on each iteration. Thus, tasks are likely to be assigned to MEC with higher channel gain.

How to effectively utilize the collected high-quality data for updating also attracts much attention. Some works attempt to improve the experience replay pool to make it easier to sample high-quality data. Zhu et al. [138] design a reward-aware replay pool to store experiences that have higher rewards and meet constraints. The training data is sampled proportionally from the normal replay pool and the design one. Yun et al. [169] construct a distributed DQN for computation offloading where multiple agents interact in parallel to gather experience. They propose the Best Experience Push method that copies and saves high-reward transitions multiple times. Cui et al. [170] adopt SAC with priority experience replay (PER). They point out that the TD-error-based priority in PER might provide high-complexity samples in the early stage that cannot be understood by the model. Thus, they combine the past sampling frequency, reward, and TD error to formulate the priority. Chen et al. [171] argue that using only TD-error as a priority ignores the importance of the sample to the actor network. In response, they combine the TD-error and the gradient of the actor network as the score and divide samples into N ranks with predetermined intervals as the priority.

Efforts have been made to improve the training process. Geng et al. [65] design CNN, MLP, and LSTM branches respectively to encode task state matrix, vehicle state, and historical information. Meanwhile, they propose an adaptive n-step update method to effectively reduce the bias while avoiding the large variance caused by inconsistent strategies. Similarly, distributed Actor-Critic with n-step update is adopted in [172]. They also propose to combine the Deep Neuroevolution with policy gradient. First, multiple agents with different policy network parameters are initialized. Following the idea of genetic algorithms, a disturbance noise is added to the policy network parameters as a mutation process. After that, each agent updates the parameters based on the policy gradient. Based on the long-term reward, the master agent selects the best k solutions for updating, and the weaker agents will be discarded. Instead of learning to estimate the reward value, Chen et al. [82] follows distributed RL, where the distributed means to learn the probability of an approximate reward distribution. The reward space is discretized into multiple levels, and the critic network estimates the probability that the reward belongs to each level. The TD error is formulated as the KL divergence between the approximate distribution and the target distribution.

5.4. Reduction of action space

An enormous action space causes difficulties for efficient exploration by the DRL agent, which leads to slow convergence and suboptimal performance. In computation offloading, not only the offloading decisions should be made, but other actions should be jointly selected for higher system efficiency. Thus, a common solution is to decompose the complex optimization problems for action space reduction. Actions need to be made at different frequencies. For example, computation offloading should be decided in milliseconds while movement planning or cache placement might be decided in seconds. This is essentially the difference between unified-timescale and multiple-timescale methods. For unified-timescale approaches, related actions are taken sequentially and are selected at the same timeslot. For multiple-timescale approaches, actions are selected independently at different frequencies and will have impacts on the environment over multiple timeslots.

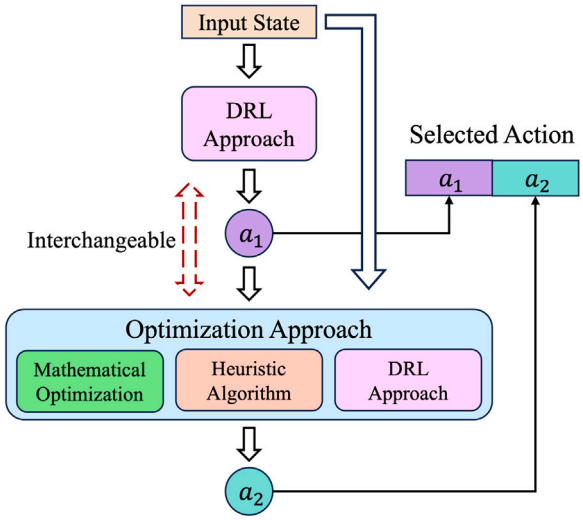

Fig. 9. The framework of unified-timescale methods. The original problem is decomposed and tackled with DRL and traditional optimization methods, respectively. The order is interchangeable according to the problem setting. The two components work in sequence and their results are combined to form the final decision.

The Unified-timescale methods typically employ a DRL agent to handle part of the action selection while using traditional optimization methods to solve the rest (see Fig. 9). For example, Van Dat Tuong et al. [173] adopt IQL for channel allocation and user association, with the computing resource allocation solved by the Lagrangian Dual Decomposition with Karush–Kuhn–Tucker (KKT) conditions. Chen et al. [144] use DQN for offloading ratio adjustment and construct the Lagrangian of the resource allocation problem. The closed-form optimal value is achieved by expanding it to the Taylor Series and setting the partial derivative equal to 0. Similarly, in [174], DQN is utilized for offloading. The CPU frequency allocation is proven to be convex, and solved by the gradient descent approach. Hu et al. [103] adopt TD3 with two branches to generate discrete-continuous hybrid actions. Besides, they propose an Alternating Direction Method of Multipliers (ADMM)-based method to solve the large-scale distributed convex problem of computing energy and resource allocation.

However, in most cases, it is too ideal to assume that the subproblem is convex. To handle such situations, Luong et al. [175] utilize DQN for UAV movement control. A Difference of Convex Algorithm (DCA) is then proposed to solve user association and transmit beamforming. To find the best number of subchannels, Yang et al. [117] relax the integrality of binary offloading decision and select the optimal one with the Lagrangian Duality. DQN is then used to select an offloading target. After deciding the offloading ratio by the Actor-Critic algorithm, Zhang et al. [176] handle the remaining geometric programming problem by the interior point algorithm. In [109], after making offloading decisions by DQN, they transform the non-convex problem into a convex one by variable conversion and solve it with the sequential quadratic programming method. In [177], the UAV trajectory is controlled by a DDPG agent. They propose a greedy-heuristic iterative method to allocate proper resources.

These methods are tightly coupled with the underlying theoretical models. Hence, it is difficult to adapt these methods to a different environment. In addition, the significant time delay of iterative solutions is unacceptable when the constructed problem is complex. By contrast, heuristic and meta-heuristic algorithms are more efficient solvers. Huang et al. [178] adopt a heuristic CPU resource management approach. The Worst-Case Execution Time (WCET) for each scheduled task is calculated upon their completion. When the WCET is within the tolerable delay, the CPU frequency is actively reduced to reserve energy. On this basis, they employ a partially-observable TD3-based

Table 5
Related works of action space reduction. CR is the abbreviation for Computing Resources. UT refers to Unified-Timescale, MT refers to the Multi-Timescale.

	Category	Approach		Actions taken by Non-DRL Algorithm	Offloading type	State				Action	Reward
						Network state	Task state	Device state	Additional		
[173]	UT	IQL	Lagrangian Duality	CR Allocation	Binary Multi-Target	Channel Gain	Task Size Required CR		Historical Channel Assignment	User Association	Time Cost
[144]	UT	DQN	Lagrangian Duality	Transmit Power CR Allocation	Partial Multi-Target		Task Size	Maximum Transmit Power Available CR	Offloading Ratio	Offloading Ratio	Time Cost
[174]	UT	DQN	Gradient Descent	CR Allocation	Binary Single-Target			Waiting Queue Location or Distance		Offloading Decision Task Scheduled Set	Time Cost Energy Cost
[103]	UT	AC	ADMM-based Optimization	Energy for Offloading CR Allocation	Binary Single-Target	Channel Gain	Task Size	Energy Remain Location or Distance		Offloading Decision Energy Transmit Duration	Time Cost Energy Cost
[175]	UT	DQN	DCA	User Association Power Allocation Transmit Beamformers	Binary Multi-Target	Channel Gain		Location or Distance		UAV Direction UAV Moving Distance	Achievable Rate
[117]	UT	DQN	Lagrangian Duality	Number of Subchannel	Binary Multi-Target	Channel Gain	Task Size Tolerable Delay	Location or Distance		Offloading Target	Time Cost Price Cost Task Finish Reward
[176]	UT	AC	Lagrangian Duality Interior Point Algorithm	CR Allocation Uplink Duration Downlink Duration	Partial Single-Target	Channel Gain				Offloading Ratio	Energy Cost
[109]	UT	DQN	SQP Optimization	Transmit Power CR Allocation	Binary Multi-Target			Available CR Cache Capacity	System Cost	Offloading Target Caching Decision	Time Cost Energy Cost
[177]	UT	DDPG	Greedy-Heuristic	User Association CR Allocation Bandwidth Allocation	Binary Multi-Target			Location or Distance Required Data Rate User Association		UAV Target Location	QoE
[178]	UT	TD3	Heuristic	CR Allocation	Binary Multi-Target	Transmit Rate		Workload Waiting Queue		Offloading Target	Time Cost Energy Cost Deadline-miss Penalty
[24]	UT	DQN	Ant Colony	Offloading Target	Binary Multi-Target				User ID Target ID	Channel Selection Transmit Power	Minimum Required CR
[179]	UT	DQN	Ant Colony & PSO	CDS Selection Offloading Ratio Resource Allocation	Partial Single-Target			Network Location		Network Routing	Time Cost Energy Cost
[73]	UT	Twin-Actor DDPG			Partial Multi-Target	SINR		Waiting Queue Caching Status		Resource Slicing RB Allocation	Utility
[113]	UT	Two-Actor SAC			Partial Single-Target	Channel Gain	Task Size Required CR Tolerable Delay	Energy Remain	Historical Success Rate	Offloading Ratio Channel Allocation Transmit Power Local CR Allocation Importance of Fairness Edge CR Allocation	Success Rate Balancing Time Cost
[180]	UT	IQL	Hungarian Algorithm	Offloading Target	Binary Multi-Target	Transmit Rate	Required CR	Available CR	Historical Decisions	Offloading Target Type	Utility
[181]	MT	DDPG	DQN		Binary Multi-Target		Task Size Required CR	Energy Remain Location or Distance		UAV Direction Offloading Target	Time Cost
[62]	MT	IQL	Stackelberg Equilibrium	Resource Pricing Resource Rent Amount	Binary Multi-Target	Channel Gain Sojourn Time Interference		Available CR		Offloading Mode CR Allocation RB Allocation	Time Cost Satisfaction Ratio
[77]	MT	DQN	DQN		Binary Multi-Target	Available Subcarriers Transmit Rate	Required Service	Available CR Waiting Queue	Large Timescale Action	Cache Placement Offloading Target Subcarriers	Time Cost Storage Space Usage
[182]	MT	DQN	Greedy-Heuristic	User Association	Binary Multi-Target	Channel Gain		Available CR Waiting Queue		Sub-band Allocation Transmit Power IRS phase shift Offloading Decision	Energy Cost
[183]	MT	AC	AC	Admission Control CR Allocation	Binary Multi-Target			Waiting Queue Data Arrival Rate Coverage Availability	Large Timescale Action	Offloading Target Transmit Power	Queue Delay Queue Stability
[184]	MT	Hierarchical PPO			Binary Multi-Target	Contact Time Channel Gain		Location or Distance Waiting Queue	Large Timescale Action	Offloading Target Bandwidth Allocation	Time Cost
[87]	MT	DQN	K-Means	User Clustering	Binary Multi-Target	Channel Gain		Available CR Waiting Queue		Transmit Power Level	Time Cost Energy Cost
[185]	MT	SAC	Greedy Clustering	User Clustering Master Node Selection	Binary Multi-Target	Transmit Rate Interference Channel Gain	Task Size Required CR	Available CR		Offloading Target CR Allocation	Time Cost Energy Cost
[186]	MT	DQN+DDPG	Fuzzy C-Means	User Clustering	Binary Multi-Target	Path Loss	Task Size	Energy Remain		Offloading Target Transmit Power Bandwidth Allocation	Process Rate
[70]	MT	DDPG	Clustering	User Clustering	Partial Multi-Target		Task Size Required CR Tolerable Delay	Location or Distance	Historical Action	Offloading Ratio	Energy Cost

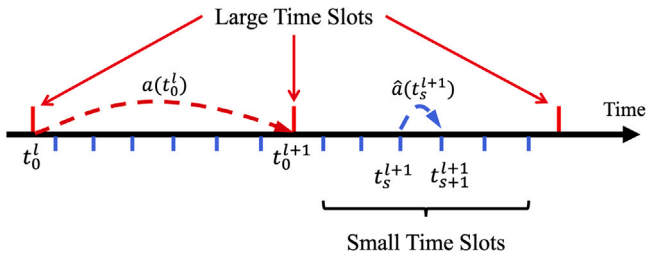

Fig. 10. Two-Timescale Paradigm. Stable actions are generated following the large timescale, while dynamic actions are generated following the small timescale.

algorithm for task offloading. Tan et al. [24] design a two-level alternation method, where the ant colony algorithm is adopted for task offloading decisions at the upper level, and a DQN agent is deployed for resource allocation at the lower level. Yi et al. [179] construct a multi-hop MEC system and a Connected Dominating Set (CDS) selection algorithm to select part of nodes to assist data transfer. Afterward, they introduce a two-layer algorithm, combined with ant colony and particle swarm optimization (PSO) for task offloading and resource allocation, and construct the network routing by DQN. Wang et al. [73] propose the twin-actor DDPG, to separately determine actions by two actor networks. Genetic Algorithm (GA) is used in [113] to search for feasible resource allocation. The remaining actions are decided by a two-actor SAC algorithm. In [180], multi-agent DQN is introduced to decide the type of offloading targets, and the Hungarian algorithm is then applied to match the exact target.

The multi-timescale paradigm has been adopted in several studies to deal with multi-level decision frequencies. They select different actions on different timescales and thus are more compatible with the system adjustment frequency in reality, as shown in Fig. 10. In [181], the moving of UAVs is controlled by DDPG at each large timeslot, and offloading decisions are made by DQN at each small timeslot. Zhang et al. [62] construct the service pricing into a Stackelberg Game. On large timescale, users rent computing resources and the controller adjusts the price. The Stackelberg Equilibrium can be found. On a small timescale, Multi-Agent DDQN is adopted for task offloading and resource allocation. To tackle the damage from outdated and low-reward state–action pairs, the lenient mechanism is introduced to omit policy updates with a given probability. Yu et al. [77] decompose service caching management and task scheduling. Two DQN agents following the federated learning paradigm are introduced for privacy protection. Wang et al. [182] decompose the user association and offloading decision. They greedily exchange the associated BS of two users for better performance on large time scales, and DQN is then adopted on small time scales.

Certain operations must be performed at a lower frequency due to environmental conditions, which necessitates multi-timescale decision-making. Since the UAV position and channel states are changed at different frequencies, Liao et al. [183] construct a multi-timescale scheduling algorithm. Task offloading targets are decided at each large timeslot, and the power control is optimized at each small timeslot. They design two federated Actor-Critic agents for action selection. Satellite-aided systems are considered in [184]. With hierarchical-PPO, adjustments of satellite link are made on a large time scale, whilst that of the terrestrial link are made on a small time scale.

Clustering users to reduce the dimensionality is of great necessity for large-scale ultra-dense systems. These methods can be regarded as multi-timescale approaches because clustering is commonly performed in advance or at a lower frequency. Liu et al. [87] propose to cluster end devices by k-means algorithm based on the relative distance and the probability of offloading. Devices in the same cluster follow the same offloading policy. Greedy clustering is used in [185]. Some vehicles are selected to be the cluster heads based on the link lifetimes, relative distances, and speeds. Decisions are made by cluster heads and executed by all vehicles in the cluster. Dai et al. [70] propose to group end devices into sub-systems by the fuzzy c-means algorithm. Each sub-system is served by a UAV, which keeps moving towards the geographic center to shorten the communication distance. In [186], sensors are aggregated into multiple groups where the group leader collects the generated data from the members and participates in offloading. Both DQN and DDPG are used for decision-making.

We summarize the abovementioned studies in Table 5.

5.5. Multi-agent cooperation and competition

Multi-agent DRL incorporates multiple agents that make independent decisions by their own. Decentralized action selection fits well with the autonomic nature of edge computing systems. It also reduces the dimensionality of the state and action space for large-scale environments. Depending on the optimization objectives, the relationship between agents can be classified into cooperative and competitive frameworks. In a cooperative framework, agents take actions to maximize global rewards. For a competitive framework, agents are selfish to achieve higher individual rewards, in which case the system eventually converges to a Nash equilibrium (NE).

Independent Learning represents an intuitive and simple way of multi-agent collaboration. For example, IQL is a common independent learning method that simply deploys an individual Q-network on each agent. Due to the disturbance of other agents' decisions, it is difficult to learn a stable individual Q-network for each agent. Providing additional information to facilitate perception and collaboration between agents could help. Ren et al. [42] follow IQL and additionally provide the previous action selection of other agents to facilitate cooperation. Ke et al. [54] utilize the GRU-assisted IQL, where GRU is used to extract features from the Historical global state over a short period to avoid environmental inconsistencies. Instead of IQL, Yu et al. [25] propose the heterogeneous distributed multi-agent DDPG algorithm. The end devices and edge nodes have heterogeneous actor networks respectively to select the corresponding actions. An Independent critic network for each agent is deployed for action evaluation based on local information.

However, with independent learning, an agent cannot perceive the intentions of other agents. It is also difficult to determine each agent's contribution to the global objective. To address this problem, value decomposition methods, such as Q-Mix, break down the global reward into individual components. Tan et al. [187] propose an RNN-assisted Q-Mix algorithm. Two RNN branches separately extract the communication information and the local features from historical observations. The neighboring agents would share the communication information to promote agent sensing and collaboration. [33] also adopts the Q-Mix algorithm with RNN for temporal feature extraction and applies GNN to aggregate network graph information for more accurate value decomposition weights. Yin and Yu [115] combines P-DQN and Q-Mix for discrete-continuous action selection. In each agent, the actor generates continuous action, while the critic combines it with all possible discrete actions for evaluation. The best combination is selected as the decision.

The utilization of global observations is vital to the learning process. Based on this idea, CTDE emerges as a promising paradigm. In centralized training, the critic network acquires global states for accurate value estimation, which can more efficiently encourage cooperation and guide actor network updates. Only decentralized actor networks are required for action selection with local observations at both the training and execution stages. Cai et al. [137] follows this paradigm, but critic networks with different parameters are deployed for multiple agents. They also design the attention module-based critic network to aggregate nearby agent states. Wei et al. [135] focus on service pricing scenarios, where the vehicles form a cooperative coalition for more favorable prices than RSUs. Multi-agent SAC is adopted, where the critic uses the attention module to aggregate the observations, and actors employ GRU to extract the temporal information. As one of the most well-known methods, MADDPG has been adopted in various works as

Table 6
Related works of multi-agent cooperation and competition. CR is the abbreviation for Computing Resources.

	Category	DRL Approach	Offloading type	State				Action	Reward
				Network state	Task state	Device state	Additional		
[42]	Independent RL	IQL	Binary Multi-Target	Channel Gain	Task Size Required CR		Last Offloading Target	Offloading Target	Energy Cost
[54]	Independent RL	IQL	Binary Multi-Target	SINR	Task Size Required CR Task Arrival Rate	Waiting Queue Energy Remain Energy Arrival Rate	Historical Global State	Offloading Target	Time Cost Energy Cost
[25]	Independent RL	Independent DDPG	Binary Single-Target	Transmit Rate		Waiting Queue Location or Distance		Offloading Decision Transmit Power CR Allocation IRS Reflection Coefficients	Energy Efficiency Queue Stability
[187]	Value Decomposition	Q-Mix	Binary Multi-Target		Number of Tasks	Location or Distance Number of Loads		UAV Direction UAV Speed	Service Fairness Load Fairness
[33]	Value Decomposition	Q-Mix	Binary Multi-Target	Channel Gain	Task Size Required CR Required Service Delay Thresholds	Caching Status		Offloading Target Caching Decisions Channel Allocation	Utility
[115]	Value Decomposition	Q-Mix	Binary Multi-Target	Received Signal Strength				Offloading Target Transmit Power	Throughput Service Fairness
[137]	CTDE	Multi-Agent AC	Partial Multi-Target	Bandwidth	Task Size Required CR	Available CR Location or Position		Offloading Ratio Offloading Target UAV Direction Moving Distance	Time Cost Energy Cost
[135]	CTDE	Multi-Agent SAC	Binary Multi-Target		Required CR Delay Sensitive Factor Utility Factor	Available CR Location or Position	Historical Service Price Historical CR Requirements	Offloading Target CR Purchase	QoS Service Provision Profit Service Usage Cost
[120]	CTDE	MADDPG	Partial Multi-Target		Number of Tasks Required CR Time to Live			Offloading Ratio CR Allocation	Time Cost
[188]	CTDE	MADDPG	Binary Multi-Target	Channel Gain Interference	Task Size Required CR Tolerable Delay	Waiting Queue Location or Distance		Offloading Target	Time Cost Energy Cost Task Finish Reward
[189]	Independent RL	IQL	Binary Multi-Target		Task Size Required CR	Available CR Waiting Queue Location or Distance	Coordinate Message	Offloading Target	Load Balancing Time Cost
[190]	Independent RL	IQL	Binary Single-Target		Data Size		Historical Actions	Bandwidth Allocation Transmit Power Allocation	Time Cost Energy Cost
[106]	CTDE	MADDPG	Binary Multi-Target	Transmit Rate	Task Size Required CR Tolerable Delay	Available CR Waiting Queue	States of Collaborators	Offloading Target	Time Cost
[191]	CTDE	MADDPG	Partial Multi-Target	Channel Gain	Workload	Remain Energy	Timeslot Length	Beamforming Strategy Offloading Time Offloading Target	Energy Efficiency Reliability
[124]	Competitive	Independent AC	Partial Single-Target	Bandwidth			Historical Bandwidth Historical Action	Offloading Ratio	Utility
[192]	Competitive	MADDPG	Binary Single-Target	Subchannel Availability	Task Size Required CR	Maximum Transmit Power Available CR	Transaction State	Offloading Decision Subchannel Selection Transmit Power Allocation CR Allocation	Utility
[118]	Competitive	DDPG MADDPG	Binary Single-Target		Task Size	Available CR Location or Position	Historical Offloading Decisions Historical Pricing Decisions Current Price	Number of Offloading Tasks	Profit Satisfaction Degree
[193]	Competitive	MADDPG	Binary Multi-Target	SINR		Location or Distance Cache State	Satisfaction Matrix	User Association Power Allocation Proactive Caching index UAV Position	Time Cost

the backbone. In [120], real-time status is assumed to be difficult to obtain instantly. Thus, they propose to adopt LSTM for state estimation. However, the sharing of local state sharing results in excessive amount of data exchange. To this end, they use the variational RNN to compress state information for communication. To avoid excessive global state dimensions, Gao et al. [188] employs the multi-head attention module to aggregate observations. They follow the idea of D3QN to construct the critic network.

However, agents in the above methods take actions based solely on their own observations. This potentially leads to sub-optimal offloading decisions. Therefore, several studies proposed to incorporate coded message communication. Such methods facilitate cooperation and offer improved performance with acceptable communication costs. For example, Zhang et al. [189] propose to deploy a coordinator at the edge to send coordination messages to agents to facilitate cooperation. The coordinator uses Q-Learning to observe the global state and

sends messages to the agents. The agent adopts GRU-based IQL and adds a variational regularization term to ensure the guidance of the coordination messages. In [190], a global information center collects observations and shares the encoded feature between agents. Each agent adopts a judger module to make a binary decision on whether to incorporate the shared feature. Li et al. [106] use DNNs to decide whether devices participate in offloading, including becoming initiators to seek help or providing services as collaborators. Each initiator with nearby collaborators forms a subsystem. When a collaborator is in multiple subsystems, it can aggregate and share features between subsystems. Instead of feature sharing, Zhou et al. [191] propose to additionally add a centralized model to estimate the offloading actions of device agents. All devices collaborate to generate actions with MADDPG based on observations and estimation of other agents' actions. The actual decisions will be sent back to the center to guide the center model updates.

When the participants of edge networks are selfish, the relationship between agents is competitive rather than cooperative. It means that end devices aim to maximize the local rewards without concerning the negative impact on the global rewards. In this case, the optimization algorithms are expected to converge to a Nash Equilibrium. In [124], the resource allocation among devices forms a competitive game. To ensure private protection, they adopt the independent actor-critic algorithm and take the differential neural computer (DNC) to construct the critic network for better state–action evaluation. In Nguyen et al. [192], each device makes decisions to maximize its own utility. They give the potential function to formulate the system into an exact potential game. MADDPG is then applied. Instead of a game between end devices, Zhu et al. [118] constructed the vehicle-against-server system in which vehicles make offloading decisions while the servers adjust the service pricing accordingly. MADDPG is utilized by multiple vehicles and DDPG is used by service providers. Zhang et al. [193] model the problem as a Stackelberg game, where the leader adopts DDPG to minimize the global delay, and users as followers adopt MADDPG to optimize their own latency. They propose a correction mechanism to compensate for the bias of the leader's reward estimation due to the follower's unstable decision, and the gradients from other agents are aggregated for model updates.

We summarize this line of works in Table 6.

6. Open challenges

Computation offloading is a key technique for today's edge computing systems, attracting extensive research effort in the field. DRL-based approaches have proven effective in improving system throughput, QoS, energy efficiency, and task execution latency. However, through investigation, we believe that there are still open problems and challenging issues that deserve further exploration.

6.1. Task partitioning and dependency

In recent years, the strategy of decomposing tasks into subtasks for offloading has garnered significant interest because it works on finer decision granularity. In this setting, tasks are usually assumed to be arbitrarily divisible and simultaneously executable when partial offloading, but the methodology for task division and the dependency between subtasks has not been thoroughly analyzed. Although recent studies have started to analyze dependent subtasks, they still ignore the task partitioning step. They also lack effective utilization of the dependency information that is typically only used for sub-task ordering.

We believe that the analysis of task partitioning and dependency information is critical. Firstly, while some studies have explored the combined optimization of model partitioning and offloading specifically for DNN inference tasks [194], a broader analysis for general tasks remains absent. Secondly, It is possible that integrating dependency features into decision-making, such as employing GNN for feature extraction, could substantially enhance system efficiency.

6.2. Event driven offloading

Existing work typically performs computation offloading across discrete timeslots, wherein actions are generated at the beginning of each timeslot. It leads to additional waiting time for tasks arriving midway through any timeslot. Although some existing works [68] have attempted the event-driven paradigm, which could instantly schedule tasks, neither of them effectively addresses the impact of subsequent rewards. Unlike the timeslot-based paradigm, random interval lengths exist between task arrival events and reward obtaining. The traditional reward decays are not adapted to the event-driven approach. How to properly design the reward decays and construct an event-driven offloading algorithm is a problem worthy of future research.

6.3. Heterogeneous computing architecture

With the development of AI accelerators, more applications now rely on heterogeneous computing power to attain desired performance. For instance, DNN-based applications achieve better performance on the Graph Processing Units (GPUs) or Tensor Processing Units (TPUs), and some applications require scalable memory space and IO resources. However, most current works simply use the frequency of the CPU to represent their processing speed, which is too coarse-grained in resource representation. While some studies like [30,31] begin to include heterogeneous resources, they still significantly differ from reality.

Therefore, a major problem in this direction is the absence of a computing model for heterogeneous resources. Heterogeneous devices and resources lack acknowledged mathematical models for quantifying the latency and energy overhead of task processing, which makes scheduling optimization difficult. We suggest that existing model-free DRL techniques could be easily migrated to heterogeneous resource scenarios, while a more critical demand is for a generic model to quantify available computing resources and relative computing costs on heterogeneous edge devices with different computing architectures.

6.4. High reliability guarantee

Edge computing is increasingly deployed in critical scenarios such as industrial IoT and vehicular edge networks, wherein high system reliability is crucial. Any task failures or system malfunctions could lead to catastrophic outcomes, causing significant property damage or posing serious threats to personal safety. The robust and reliable performance of edge computing systems under these scenarios is not just a matter of efficiency but of safety and security. Although several studies have paid attention to the problem (as discussed in Section 3.6), scenario-specific high reliability guarantee remains an open challenge deserving further research.

System reliability is typically guaranteed as constraints. However, we notice that almost all DRL-based works simply convert them into objectives by assigning penalties for violations to potentially incentivize agents. This, however, cannot theoretically guarantee to meet constraints and ensure high reliability. In addition, the trial-and-error nature of DRL inevitably leads to accessing invalid states or actions that violate constraints.

Safe RL emerges as a new concept where we need to prevent access to unsafe states or actions and to operate reliably within predefined constraints. It helps avoid harmful actions during both the training and deployment phases. It could be an essential direction to ensure decision feasibility and enhance system reliability [195]. Offline RL is also a great option on this point. It allows learning from pre-collected data without the necessity of fully exploring the critical environment [196]. Moreover, it is also necessary to explore effective fault-tolerant algorithms, such as task reassignment or redundant offloading, to ensure system reliability [197].

6.5. Data integrity and privacy

Data integrity and privacy are critical in real-world edge computing systems where large amounts of sensitive data are processed. For example, in the industrial IoT, data leakage could expose critical manufacturing details or corporate information. In vehicular edge networks, malicious eavesdropping and manipulation of vehicle behavior pose safety risks.

Defense strategies against intentional jamming and eavesdropping have received broad attention. While some works have proposed proactive anti-attack strategies, such as using idle devices to disturb eavesdroppers [95], most of them assume the invader follows a simple pattern. Omitting analysis of adversarial attacks would likely result in overfitting to the specified attack strategy and failing to address the real attack. For this reason, it is necessary to analyze the adversarial scenario where the competitive capabilities of DRL can be exploited to further improve data privacy.

Besides, operational data from end devices involves user privacy and the security of device operation. This concerns both DRL training and decision-making. It is important to ensure data consistency and avoid privacy leakage in the learning and decision-making process. Federated Learning (FL) is a promising direction that avoids raw data sharing by distributed local training and parameter or gradient aggregation. The training of multiple DRL agents can be performed within the FL framework, but the complexity of data sampling and the cost of trial-and-error steps are still a concern in real-world scenarios [198]. Meanwhile, Aono et al. [199] reported that sharing gradients can still risk the privacy of the original data. To address it, various FL-based methods are proposed [200], but whether a DRL process can converge smoothly in the FL setting is still an open question.

6.6. Environment dynamics and adaptability

How to adaptively transfer the algorithms to unknown environments while maintaining good performance is an open challenge. Besides, any edge systems can also evolve dynamically. The switch or change of environment causes distributional shift that may turn a good policy learned by the DNN-based actor network to a bad one. Therefore, deploying pre-trained DRL algorithms in actual systems requires a long period of re-training, which sometimes also brings about performance degradation. To fill this gap, techniques such as curriculum learning, meta-RL, and transfer reinforcement learning [84,201,202] have been used. Nonetheless, these methods commonly require additional training across different scenarios or domains. The required data acquisition process is complex and expensive. In addition, it also requires more effort for model fine-tuning.

Furthermore, it should be noted that environment dynamics in terms of dimensionality is not friendly to the underlying MDP model. When the numbers of end devices and ENs are not constant, the state and action dimensions may change, making traditional MLP-based DRL unsuitable. Existing approaches attempt to adopt the Seq2Seq paradigm [203] or group devices into a fixed number of clusters [186], but they introduce significant execution overhead while the system performance is greatly affected by the quality of clustering. To this end, it is worth investigating in future research on how to adapt DRL-based strategies to dynamic environments.

6.7. Interpretability of DRL

Unlike traditional offloading algorithms that have clear formulation and rules, the black-box nature of DRL makes the connection between input observations and output decisions fairly intangible. The lack of interpretability greatly affects the trustworthiness of DRL-based strategies and thus creates a gap between research and applying them to real-world scenarios. Although it is believed to be very challenging, recent developments of DRL explanation has shed some light in this direction [204]. By any means, providing the reasons behind the decisions would help understand and improve the policies for higher system efficiency. It also enhances the trustworthiness and security of the algorithms, which is always imperative for actual deployment in security-sensitive edge computing systems.

7. Conclusion

Computation offloading is a key technique for edge computing systems. How to reasonably determine the location of task processing and manage the transmission and computation resources related to it have received much attention in recent years. DRL demonstrates powerful perception and decision-making capabilities compared with traditional optimization algorithms. This has catalyzed a large number of DRL-based offloading algorithms in the domain. However, the complexity and diversity of application scenarios and system models make it difficult to understand the design principles behind these approaches and also hinder a wider practical deployment.

To this end, we present a novel view of DRL-based offloading strategies via the lens of key design elements. We first introduce environmental factors that are commonly considered and closely related to computation offloading. Then we discuss the technical design of state spaces, action spaces, and reward functions for MDP models. We also categorize and compare the existing studies based on their improvement of learning strategies to inspire subsequent applications and enhancements of DRL in computation offloading. Finally, we highlight possible research directions based on the open issues to provide insights for future studies. Although existing work has explored diverse edge scenarios and improved DRL's performance, the dynamic and heterogeneous nature of edge systems is still the major challenge. On this point, the insufficient adaptability, reliability, and interpretability of DRL-based methods somewhat hinder practical applications. We believe that DRL has great potential to enable intelligent computation offloading whilst these open issues require more attention in future research.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgments

This work is supported by Guangdong Major Project of Basic and Applied Basic Research (2019B030302002); National Natural Science Foundation of China (62072187); the Major Key Project of PCL, China under Grant PCL2023A09; NSFC-FDCT Grants 62361166662; National Key R&D Program of China 2023YFC3503400, 2022YFC3400400; The Innovative Research Group Project of Hunan Province 2024JJ1002; Key R&D Program of Hunan Province 2023GK2004, 2023SK2059, 2023SK2060; Top 10 Technical Key Project in Hunan Province 2023 GK1010, and Key Technologies R&D Program of Guangdong Province (2023B1111030004 to FFH).

References

[1] T. Alsop, Number of Edge Enabled Internet of Things (IoT) Devices Worldwide from 2020 to 2030, by Market, Technical Report, Statista, 2022.

[2] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: Vision and challenges, SIGCOMM Comput. Commun. Rev. 45 (5) (2015) 37–42.

[3] L. Kong, J. Tan, J. Huang, G. Chen, S. Wang, X. Jin, P. Zeng, M. Khan, S.K. Das, Edge-computing-driven Internet of Things: A survey, ACM Comput. Surv. 55 (8) (2022) 1–41.

[4] Z. Zabihi, A.M. Eftekhari Moghadam, M.H. Rezvani, Reinforcement learning methods for computation offloading: A systematic review, ACM Comput. Surv. 56 (1) (2024) 1–41.

[5] Q. Luo, S. Hu, C. Li, G. Li, W. Shi, Resource scheduling in edge computing: A survey, IEEE Commun. Surv. Tutor. 23 (4) (2021) 2131–2165.

[6] H. Tran-Dang, S. Bhardwaj, T. Rahim, A. Musaddiq, D.-S. Kim, Reinforcement learning based resource management for fog computing environment: Literature review, challenges, and open issues, J. Commun. Netw. 24 (1) (2022) 83–98.

[7] J. Liu, M. Ahmed, M.A. Mirza, W.U. Khan, D. Xu, J. Li, A. Aziz, Z. Han, RL/DRL meets vehicular task offloading using edge and vehicular cloudlet: A survey, IEEE Internet Things J. 9 (11) (2022) 8315–8338.

[8] A.M.A. Hamdi, F.K. Hussain, O.K. Hussain, Task offloading in vehicular fog computing: state-of-the-art and open issues, Future Gener. Comput. Syst. 133 (2022) 201–212.

[9] Z. Song, X. Qin, Y. Hao, T. Hou, J. Wang, X. Sun, A comprehensive survey on aerial mobile edge computing: challenges, state-of-the-art, and future directions, Comput. Commun. 191 (2022) 233–256.

[10] W. Chen, X. Qiu, T. Cai, H.-N. Dai, Z. Zheng, Y. Zhang, Deep reinforcement learning for Internet of Things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1659–1692.

[11] M.S. Frikha, S.M. Gammar, A. Lahmadi, L. Andrey, Reinforcement and deep reinforcement learning for wireless Internet of Things: A survey, Comput. Commun. 178 (2021) 98–113.

[12] T. Li, K. Zhu, N.C. Luong, D. Niyato, Q. Wu, Y. Zhang, B. Chen, Applications of multi-agent reinforcement learning in future internet: A comprehensive survey, IEEE Commun. Surv. Tutor. 24 (2) (2022) 1240–1279.

[13] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, L. Guo, Computation offloading in mobile edge computing networks: A survey, J. Netw. Comput. Appl. 202 (2022) 103366.

[14] N. Kumari, A. Yadav, P.K. Jana, Task offloading in fog computing: A survey of algorithms and optimization techniques, Comput. Netw. 214 (2022) 109137.

[15] A. Acheampong, Y. Zhang, X. Xu, D. Kumah, A review of the current task offloading algorithms, strategies and approach in edge computing systems, CMES Comput. Model. Eng. Sci. 134 (1) (2022) 35–88.

[16] S. Taheri-abed, A.M. Eftekhari Moghadam, M.H. Rezvani, Machine learning-based computation offloading in edge and fog: A systematic review, Cluster Comput. 26 (5) (2023) 3113–3144.

[17] K. Sadatdiynov, L. Cui, L. Zhang, J.Z. Huang, S. Salloum, M.S. Mahmud, A review of optimization methods for computation offloading in edge computing networks, Digit. Commun. Netw. 9 (2) (2023) 450–461.

[18] B. Kar, W. Yahya, Y.-D. Lin, A. Ali, Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: A survey, IEEE Commun. Surv. Tutor. 25 (2) (2023) 1199–1226.

[19] M.Y. Akhlaqi, Z.B. Mohd Hanapi, Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions, J. Netw. Comput. Appl. 212 (2023) 103568.

[20] D. Hortelano, I. de Miguel, R.J.D. Barroso, J.C. Aguado, N. Merayo, L. Ruiz, A. Asensio, X. Masip-Bruin, P. Fernández, R.M. Lorenzo, E.J. Abril, A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems, J. Netw. Comput. Appl. 216 (2023) 103669.

[21] D.H. Abdulazeez, S.K. Askar, Offloading mechanisms based on reinforcement learning and deep learning algorithms in the fog computing environment, IEEE Access 11 (2023) 12555–12586.

[22] S. Zhou, W. Jadoon, I.A. Khan, Computing offloading strategy in mobile edge computing environment: A comparison between adopted frameworks, challenges, and future directions, Electronics 12 (11) (2023) 2452.

[23] G. Chen, Q. Wu, W. Chen, D.W.K. Ng, L. Hanzo, IRS-aided wireless powered MEC systems: TDMA or NOMA for computation offloading? IEEE Trans. Wireless Commun. 22 (2) (2023) 1201–1218.

[24] L. Tan, Z. Kuang, L. Zhao, A. Liu, Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing, IEEE Trans. Wireless Commun. 21 (3) (2022) 1960–1972.

[25] J. Yu, Y. Li, X. Liu, B. Sun, Y. Wu, D. Hin-Kwok Tsang, IRS assisted NOMA aided mobile edge computing with queue stability: Heterogeneous multi-agent reinforcement learning, IEEE Trans. Wireless Commun. 22 (7) (2023) 4296–4312.

[26] W. Liu, B. Li, W. Xie, Y. Dai, Z. Fei, Energy efficient computation offloading in aerial edge networks with multi-agent cooperation, IEEE Trans. Wireless Commun. 22 (9) (2023) 5725–5739.

[27] X. Yuan, J. Chen, N. Zhang, J. Ni, F.R. Yu, V.C.M. Leung, Digital twin-driven vehicular task offloading and IRS configuration in the internet of vehicles, IEEE Trans. Intell. Transp. Syst. 23 (12) (2022) 24290–24304.

[28] F. Jiang, Y. Peng, K. Wang, L. Dong, K. Yang, MARS: A DRL-based multi-task resource scheduling framework for UAV with IRS-assisted mobile edge computing system, IEEE Trans. Cloud Comput. 11 (4) (2023) 3700–3712.

[29] K. Li, W. Ni, X. Yuan, A. Noor, A. Jamalipour, Deep-graph-based reinforcement learning for joint cruise control and task offloading for aerial edge Internet of Things (EdgeIoT), IEEE Internet Things J. 9 (21) (2022) 21676–21686.

[30] Y. Liu, Y. Mao, Z. Liu, F. Ye, Y. Yang, Joint task offloading and resource allocation in heterogeneous edge environments, IEEE Trans. Mob. Comput. (2024) 1–16.

[31] Z. Tong, J. Wang, J. Mei, K. Li, W. Li, K. Li, Multi-type task offloading for wireless Internet of Things by federated deep reinforcement learning, Future Gener. Comput. Syst. 145 (2023) 536–549.

[32] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.

[33] Z. Yao, S. Xia, Y. Li, G. Wu, Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach, IEEE J. Sel. Areas Commun. 41 (11) (2023) 3401–3413.

[34] C. Sun, W. Ni, X. Wang, Joint computation offloading and trajectory planning for UAV-assisted edge computing, IEEE Trans. Wireless Commun. 20 (8) (2021) 5343–5358.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with deep reinforcement learning, 2013, arXiv e-prints, arXiv:1312.5602.

[36] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, No. 1, 2016.

[37] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling network architectures for deep reinforcement learning, in: International Conference on Machine Learning, PMLR, 2016, pp. 1995–2003.

[38] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, PMLR, 2016, pp. 1928–1937.

[39] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2015, arXiv e-prints, arXiv:1509.02971.

[40] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International Conference on Machine Learning, PMLR, 2018, pp. 1587–1596.

[41] R. Lowe, Y.I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, Adv. Neural Inf. Process. Syst. 30 (2017).

[42] Y. Ren, Y. Sun, M. Peng, Deep reinforcement learning based computation offloading in fog enabled industrial Internet of Things, IEEE Trans. Ind. Inform. 17 (7) (2021) 4978–4987.

[43] H. Zhou, K. Jiang, X. Liu, X. Li, V.C. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing, IEEE Internet Things J. 9 (2) (2022) 1517–1530.

[44] L. Wang, G. Zhang, Deep reinforcement learning based joint partial computation offloading and resource allocation in mobility-aware MEC system, China Commun. 19 (8) (2022) 85–99.

[45] S. Shen, Y. Han, X. Wang, Y. Wang, Computation offloading with multiple agents in edge-computing-supported IoT, ACM Trans. Sensor Netw. 16 (1) (2020) 1–27.

[46] Z. Wei, R. He, Y. Li, Deep reinforcement learning based task offloading and resource allocation for MEC-enabled IoT networks, in: 2023 IEEE/CIC International Conference on Communications in China, ICCC Workshops, 2023, pp. 1–6.

[47] J. Niu, S. Zhang, K. Chi, G. Shen, W. Gao, Deep learning for online computation offloading and resource allocation in NOMA, Comput. Netw. 216 (2022) 109238.

[48] H. Hu, D. Wu, F. Zhou, X. Zhu, R.Q. Hu, H. Zhu, Intelligent resource allocation for edge-cloud collaborative networks: A hybrid DDPG-D3QN approach, IEEE Trans. Veh. Technol. 72 (8) (2023) 10696–10709.

[49] J.A. Ansere, E. Gyamfi, Y. Li, H. Shin, O.A. Dobre, T. Hoang, T.Q. Duong, Optimal computation resource allocation in energy-efficient edge IoT systems with deep reinforcement learning, IEEE Trans. Green Commun. Netw. 7 (4) (2023) 2130–2142.

[50] C. Wan, S. Guo, J. He, G. Liu, P. Zhou, iCOS: A deep reinforcement learning scheme for wireless-charged MEC networks, IEEE Trans. Veh. Technol. 71 (7) (2022) 7739–7750.

[51] B. Li, W. Liu, W. Xie, X. Li, Energy-efficient task offloading and trajectory planning in UAV-enabled mobile edge computing networks, Comput. Netw. 234 (2023) 109940.

[52] X. Zhou, L. Huang, T. Ye, W. Sun, Computation bits maximization in UAV-assisted MEC networks with fairness constraint, IEEE Internet Things J. 9 (21) (2022) 20997–21009.

[53] Z. Cheng, M. Liwang, N. Chen, L. Huang, X. Du, M. Guizani, Deep reinforcement learning-based joint task and energy offloading in UAV-aided 6G intelligent edge networks, Comput. Commun. 192 (2022) 234–244.

[54] H. Ke, H. Wang, W. Sun, H. Sun, Adaptive computation offloading policy for multi-access edge computing in heterogeneous wireless networks, IEEE Trans. Netw. Serv. Manag. 19 (1) (2022) 289–305.

[55] Y. Wang, K. Wang, H. Huang, T. Miyazaki, S. Guo, Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications, IEEE Trans. Ind. Inform. 15 (2) (2019) 976–986.

[56] J. Shi, J. Du, J. Wang, J. Wang, J. Yuan, Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning, IEEE Trans. Veh. Technol. 69 (12) (2020) 16067–16081.

[57] J. Xue, Q. Wu, H. Zhang, Cost optimization of UAV-MEC network calculation offloading: A multi-agent reinforcement learning method, Ad Hoc Netw. 136 (2022) 102981.

[58] H. Zhou, Z. Wang, H. Zheng, S. He, M. Dong, Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An A3C-based approach, IEEE Trans. Netw. Sci. Eng. 10 (3) (2023) 1326–1338.

[59] J. Shi, J. Du, Y. Shen, J. Wang, J. Yuan, Z. Han, DRL-based V2V computation offloading for blockchain-enabled vehicular networks, IEEE Trans. Mob. Comput. 22 (7) (2023) 3882–3897.

[60] J. Du, W. Cheng, G. Lu, H. Cao, X. Chu, Z. Zhang, J. Wang, Resource pricing and allocation in MEC enabled blockchain systems: An A3C deep reinforcement learning approach, IEEE Trans. Netw. Sci. Eng. 9 (1) (2022) 33–44.

[61] A.M. Seid, J. Lu, H.N. Abishu, T.A. Ayall, Blockchain-enabled task offloading with energy harvesting in multi-UAV-assisted IoT networks: A multi-agent DRL approach, IEEE J. Sel. Areas Commun. 40 (12) (2022) 3517–3532.

[62] X. Zhang, M. Peng, S. Yan, Y. Sun, Joint communication and computation resource allocation in fog-based vehicular networks, IEEE Internet Things J. 9 (15) (2022) 13195–13208.

[63] S.M.A. Kazmi, T.M. Ho, T.T. Nguyen, M. Fahim, A. Khan, M.J. Piran, G. Baye, Computing on wheels: A deep reinforcement learning-based approach, IEEE Trans. Intell. Transp. Syst. 23 (11) (2022) 22535–22548.

[64] L. Zhao, E. Zhang, S. Wan, A. Hawbani, A.Y. Al-Dubai, G. Min, A.Y. Zomaya, MESON: A mobility-aware dependent task offloading scheme for urban vehicular edge computing, IEEE Trans. Mob. Comput. (2023) 1–15.

[65] L. Geng, H. Zhao, J. Wang, A. Kaushik, S. Yuan, W. Feng, Deep-reinforcement-learning-based distributed computation offloading in vehicular edge computing networks, IEEE Internet Things J. 10 (14) (2023) 12416–12433.

[66] C.-L. Wu, T.-C. Chiu, C.-Y. Wang, A.-C. Pang, Mobility-aware deep reinforcement learning with seq2seq mobility prediction for offloading and allocation in edge computing, IEEE Trans. Mob. Comput. (2023) 1–17.

[67] H. Maleki, M. Başaran, L. Durak-Ata, Handover-enabled dynamic computation offloading for vehicular edge computing networks, IEEE Trans. Veh. Technol. 72 (7) (2023) 9394–9405.

[68] H. Tang, H. Wu, G. Qu, R. Li, Double deep Q-Network based dynamic framing offloading in vehicular edge computing, IEEE Trans. Netw. Sci. Eng. 10 (3) (2023) 1297–1310.

[69] L. Yao, X. Xu, M. Bilal, H. Wang, Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning, IEEE Trans. Intell. Transp. Syst. 24 (11) (2023) 12991–12999.

[70] B. Dai, J. Niu, T. Ren, Z. Hu, M. Atiquzzaman, Towards energy-efficient scheduling of UAV and base station hybrid enabled mobile edge computing, IEEE Trans. Veh. Technol. 71 (1) (2022) 915–930.

[71] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, D. Niyato, Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing, IEEE Trans. Wireless Commun. 21 (9) (2022) 6949–6960.

[72] L. Zhang, A. Celik, S. Dang, B. Shihada, Energy-efficient trajectory optimization for UAV-assisted IoT networks, IEEE Trans. Mob. Comput. 21 (12) (2022) 4323–4337.

[73] Z. Wang, Y. Wei, F.R. Yu, Z. Han, Utility optimization for resource allocation in multi-access edge network slicing: A twin-actor deep deterministic policy gradient approach, IEEE Trans. Wireless Commun. 21 (8) (2022) 5842–5856.

[74] X. Kong, G. Duan, M. Hou, G. Shen, H. Wang, X. Yan, M. Collotta, Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles, IEEE Trans. Ind. Inform. 18 (9) (2022) 6308–6316.

[75] S. Yang, J. Liu, F. Zhang, F. Li, X. Chen, X. Fu, Caching-enabled computation offloading in multi-region MEC network via deep reinforcement learning, IEEE Internet Things J. 9 (21) (2022) 21086–21098.

[76] X. Peng, Z. Han, W. Xie, C. Yu, P. Zhu, J. Xiao, J. Yang, Deep reinforcement learning for shared offloading strategy in vehicle edge computing, IEEE Syst. J. (2022) 1–12.

[77] S. Yu, X. Chen, Z. Zhou, X. Gong, D. Wu, When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network, IEEE Internet Things J. 8 (4) (2021) 2238–2251.

[78] Z. Xue, C. Liu, C. Liao, G. Han, Z. Sheng, Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems, IEEE Trans. Veh. Technol. 72 (5) (2023) 6709–6722.

[79] Z. Li, C. Yang, X. Huang, W. Zeng, S. Xie, CoOR: Collaborative task offloading and service caching replacement for vehicular edge computing networks, IEEE Trans. Veh. Technol. 72 (7) (2023) 9676–9681.

[80] J. Zhang, Y. Shen, Y. Wang, X. Zhang, J. Wang, Dual-timescale resource allocation for collaborative service caching and computation offloading in IoT systems, IEEE Trans. Ind. Inform. 19 (2) (2023) 1735–1746.

[81] B. Gu, M. Alazab, Z. Lin, X. Zhang, J. Huang, AI-enabled task offloading for improving quality of computational experience in ultra dense networks, ACM Trans. Internet Technol. (TOIT) 22 (3) (2022) 1–17.

[82] S. Chen, J. Chen, Y. Miao, Q. Wang, C. Zhao, Deep reinforcement learning-based cloud-edge collaborative mobile computation offloading in industrial networks, IEEE Trans. Signal Inf. Process. Netw. 8 (2022) 364–375.

[83] G. Qu, H. Wu, R. Li, P. Jiao, DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing, IEEE Trans. Netw. Serv. Manag. 18 (3) (2021) 3448–3459.

[84] N. Sharma, A. Ghosh, R. Misra, S.K. Das, Deep meta Q-learning based multi-task offloading in edge-cloud systems, IEEE Trans. Mob. Comput. (2023) 1–17.

[85] J. Yuan, H. Xiao, Z. Shen, T. Zhang, J. Jin, ELECT: Energy-efficient intelligent edge-cloud collaboration for remote IoT services, Future Gener. Comput. Syst. 147 (2023) 179–194.

[86] X. Zhou, S. Ge, P. Liu, T. Qiu, DAG-based dependent tasks offloading in MEC-enabled IoT with soft cooperation, IEEE Trans. Mob. Comput. (2023) 1–12.

[87] X. Liu, J. Yu, J. Wang, Y. Gao, Resource allocation with edge computing in IoT networks via machine learning, IEEE Internet Things J. 7 (4) (2020) 3415–3426.

[88] C. Xu, Y. Xie, X. Wang, H.H. Yang, D. Niyato, T.Q.S. Quek, Optimal status update for caching enabled IoT networks: A dueling deep R-network approach, IEEE Trans. Wireless Commun. 20 (12) (2021) 8438–8454.

[89] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, K. Wang, Edge QoE: Computation offloading with deep reinforcement learning for Internet of Things, IEEE Internet Things J. 7 (10) (2020) 9255–9265.

[90] F. Fang, K. Wang, Z. Ding, V.C.M. Leung, Energy-efficient resource allocation for NOMA-MEC networks with imperfect CSI, IEEE Trans. Commun. 69 (5) (2021) 3436–3449.

[91] T.T. Nguyen, L.B. Le, Q. Le-Trung, Computation offloading in MIMO based mobile edge computing systems under perfect and imperfect CSI estimation, IEEE Trans. Serv. Comput. 14 (6) (2021) 2011–2025.

[92] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji, Y. Zhang, Reinforcement learning-based mobile offloading for edge computing against jamming and interference, IEEE Trans. Commun. 68 (10) (2020) 6114–6126.

[93] Y. Xu, J. Chen, Y. Xu, F. Gu, K. Yao, L. Jia, D. Liu, X. Wang, Energy-efficient channel access and data offloading against dynamic jamming attacks, IEEE Trans. Green Commun. Netw. 5 (4) (2021) 1734–1746.

[94] Y. Ju, Y. Chen, Z. Cao, L. Liu, Q. Pei, M. Xiao, K. Ota, M. Dong, V.C.M. Leung, Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach, IEEE Trans. Intell. Transp. Syst. 24 (5) (2023) 5555–5569.

[95] S. Yoo, S. Jeong, J. Kang, Hybrid UAV-enabled secure offloading via deep reinforcement learning, IEEE Wirel. Commun. Lett. 12 (6) (2023) 972–976.

[96] M. Sun, X. Xu, S. Han, H. Zheng, X. Tao, P. Zhang, Secure computation offloading for device-collaborative MEC networks: A DRL-based approach, IEEE Trans. Veh. Technol. 72 (4) (2023) 4887–4903.

[97] Q. He, Z. Feng, H. Fang, X. Wang, L. Zhao, Y. Yao, K. Yu, A blockchain-based scheme for secure data offloading in healthcare with deep reinforcement learning, IEEE/ACM Trans. Netw. 32 (1) (2024) 65–80.

[98] L. Yang, M. Li, P. Si, R. Yang, E. Sun, Y. Zhang, Energy-efficient resource allocation for blockchain-enabled industrial Internet of Things with deep reinforcement learning, IEEE Internet Things J. 8 (4) (2021) 2318–2329.

[99] M. Li, P. Pei, F.R. Yu, P. Si, Y. Li, E. Sun, Y. Zhang, Cloud-edge collaborative resource allocation for blockchain-enabled Internet of Things: A collective reinforcement learning approach, IEEE Internet Things J. 9 (22) (2022) 23115–23129.

[100] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K.-C. Chen, H. Zhang, Reliable computation offloading for edge-computing-enabled software-defined IoV, IEEE Internet Things J. 7 (8) (2020) 7097–7111.

[101] T. Jing, X. Ma, X. Wang, X. Li, Enhancing soft AC based reliable offloading for IoV with edge computing, in: 2023 IEEE Wireless Communications and Networking Conference, WCNC, IEEE, Glasgow, United Kingdom, 2023, pp. 1–6.

[102] T. Long, Y. Ma, Y. Xia, X. Xiao, Q. Peng, J. Zhao, A mobility-aware and fault-tolerant service offloading method in mobile edge computing, in: 2022 IEEE International Conference on Web Services, ICWS, IEEE, Barcelona, Spain, 2022, pp. 67–72.

[103] Z. Hu, J. Niu, T. Ren, B. Dai, Q. Li, M. Xu, S.K. Das, An efficient online computation offloading approach for large-scale mobile edge computing via deep reinforcement learning, IEEE Trans. Serv. Comput. 15 (2) (2022) 669–683.

[104] G. Ma, X. Wang, M. Hu, W. Ouyang, X. Chen, Y. Li, DRL-based computation offloading with queue stability for vehicular-cloud-assisted mobile edge computing systems, IEEE Trans. Intell. Veh. 8 (4) (2023) 2797–2809.

[105] K. Li, W. Ni, F. Dressler, LSTM-characterized deep reinforcement learning for continuous flight control and resource allocation in UAV-assisted sensor network, IEEE Internet Things J. 9 (6) (2022) 4179–4189.

[106] K. Li, X. Wang, Q. He, M. Yang, M. Huang, S. Dustdar, Task computation offloading for multi-access edge computing via attention communication deep reinforcement learning, IEEE Trans. Serv. Comput. 16 (4) (2023) 2985–2999.

[107] B. Yamansavascilar, A.C. Baktir, C. Sonmez, A. Ozgovde, C. Ersoy, DeepEdge: A deep reinforcement learning based task orchestrator for edge computing, IEEE Trans. Netw. Sci. Eng. 10 (1) (2023) 538–552.

[108] K. Li, X. Wang, Q. He, B. Yi, A. Morichetta, M. Huang, Cooperative multiagent deep reinforcement learning for computation offloading: A mobile network operator perspective, IEEE Internet Things J. 9 (23) (2022) 24161–24173.
[109] Q. Chen, Z. Kuang, L. Zhao, Multiuser computation offloading and resource allocation for cloud-edge heterogeneous network, IEEE Internet Things J. 9 (5) (2022) 3799–3811.
[110] M. Tang, V.W. Wong, Deep reinforcement learning for task offloading in mobile edge computing systems, IEEE Trans. Mob. Comput. 21 (6) (2022) 1985–1997.
[111] E. Cui, D. Yang, H. Wang, W. Zhang, Learning-based deep neural network inference task offloading in multi-device and multi-server collaborative edge computing, Trans. Emerg. Telecommun. Technol. 33 (7) (2022) e4485.
[112] X. Chen, S. Hu, C. Yu, Z. Chen, G. Min, Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning, IEEE Trans. Parallel Distrib. Syst. (2024) 1–14.
[113] X. Li, Y. Qin, J. Huo, W. Huangfu, Heuristically assisted multiagent RL-based framework for computation offloading and resource allocation of mobile-edge computing, IEEE Internet Things J. 10 (17) (2023) 15477–15487.
[114] Z. Sun, Y. Mo, C. Yu, Graph-reinforcement-learning-based task offloading for multiaccess edge computing, IEEE Internet Things J. 10 (4) (2023) 3138–3150.
[115] S. Yin, F.R. Yu, Resource allocation and trajectory design in UAV-aided cellular networks based on multiagent reinforcement learning, IEEE Internet Things J. 9 (4) (2022) 2933–2943.
[116] Z. Wei, B. Zhao, J. Su, Event-driven computation offloading in IoT with edge computing, IEEE Trans. Wireless Commun. 21 (9) (2022) 6847–6860.
[117] C. Yang, B. Liu, H. Li, B. Li, K. Xie, S. Xie, Learning based channel allocation and task offloading in temporary UAV-assisted vehicular edge computing networks, IEEE Trans. Veh. Technol. 71 (9) (2022) 9884–9895.
[118] X. Zhu, Y. Luo, A. Liu, N.N. Xiong, M. Dong, S. Zhang, A deep reinforcement learning-based resource management game in vehicular edge computing, IEEE Trans. Intell. Transp. Syst. 23 (3) (2022) 2422–2433.
[119] Z. Sun, H. Yang, C. Li, Q. Yao, D. Wang, J. Zhang, A.V. Vasilakos, Cloud-edge collaboration in industrial Internet of Things: A joint offloading scheme based on resource prediction, IEEE Internet Things J. 9 (18) (2022) 17014–17025.
[120] J. Yang, Q. Yuan, S. Chen, H. He, X. Jiang, X. Tan, Cooperative task offloading for mobile edge computing based on multi-agent deep reinforcement learning, IEEE Trans. Netw. Serv. Manag. 20 (3) (2023) 3205–3219.
[121] X. Jiao, H. Ou, S. Chen, S. Guo, Y. Qu, C. Xiang, J. Shang, Deep reinforcement learning for time-energy tradeoff online offloading in MEC-enabled industrial Internet of Things, IEEE Trans. Netw. Sci. Eng. (2023) 1–14.
[122] Y. Liu, Y. Mao, Z. Liu, Y. Yang, Deep learning-assisted online task offloading for latency minimization in heterogeneous mobile edge, IEEE Trans. Mob. Comput. (2023) 1–14.
[123] J. Cai, H. Fu, Y. Liu, Multitask multiobjective deep reinforcement learning-based computation offloading method for industrial Internet of Things, IEEE Internet Things J. 10 (2) (2023) 1848–1859.
[124] Y. Zhan, S. Guo, P. Li, J. Zhang, A deep reinforcement learning based offloading game in edge computing, IEEE Trans. Comput. 69 (6) (2020) 883–893.
[125] H. Lu, C. Gu, F. Luo, W. Ding, X. Liu, Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning, Future Gener. Comput. Syst. 102 (2020) 847–861.
[126] P. Li, W. Xie, Y. Yuan, C. Chen, S. Wan, Deep reinforcement learning for load balancing of edge servers in IoV, Mob. Netw. Appl. 27 (4) (2022) 1461–1474.
[127] Y. Dong, G. Xu, M. Zhang, X. Meng, A high-efficient joint 'Cloud-Edge' aware strategy for task deployment and load balancing, IEEE Access 9 (2021) 12791–12802.
[128] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, A. Nallanathan, Deep reinforcement learning based dynamic trajectory control for UAV-assisted mobile edge computing, IEEE Trans. Mob. Comput. 21 (10) (2022) 3536–3550.
[129] Z. Zhang, C. Li, S. Peng, X. Pei, A new task offloading algorithm in edge computing, EURASIP J. Wireless Commun. Networking 2021 (1) (2021) 17.
[130] H. Hao, C. Xu, W. Zhang, S. Yang, G.-M. Muntean, Computing offloading with fairness guarantee: A deep reinforcement learning method, IEEE Trans. Circuits Syst. Video Technol. 33 (10) (2023) 6117–6130.
[131] J. Feng, J. Gong, Joint detection and computation offloading with age of information in mobile edge networks, IEEE Trans. Netw. Sci. Eng. 10 (3) (2023) 1417–1430.
[132] Y. Dai, K. Zhang, S. Maharjan, Y. Zhang, Deep reinforcement learning for stochastic computation offloading in digital twin networks, IEEE Trans. Ind. Inform. 17 (7) (2021) 4968–4977.
[133] Y. Xu, Q. Sun, W. Zhou, G. Yu, Resource allocation for UAV-aided energy harvesting-powered D2D communications: A reinforcement learning-based scheme, Ad Hoc Netw. 136 (2022) 102973.
[134] J. Huang, J. Wan, B. Lv, Q. Ye, Y. Chen, Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning, IEEE Syst. J. 17 (2) (2023) 2500–2511.
[135] Z. Wei, B. Li, R. Zhang, X. Cheng, L. Yang, Many-to-many task offloading in vehicular fog computing: A multi-agent deep reinforcement learning approach, IEEE Trans. Mob. Comput. (2023) 1–16.
[136] A.S. Kumar, L. Zhao, X. Fernando, Task offloading and resource allocation in vehicular networks: A Lyapunov-based deep reinforcement learning approach, IEEE Trans. Veh. Technol. 72 (10) (2023) 13360–13373.
[137] T. Cai, Z. Yang, Y. Chen, W. Chen, Z. Zheng, Y. Yu, H.-N. Dai, Cooperative data sensing and computation offloading in UAV-assisted crowdsensing with multi-agent deep reinforcement learning, IEEE Trans. Netw. Sci. Eng. 9 (5) (2022) 3197–3211.
[138] K. Zhu, Z. Zhang, M. Zhao, Auxiliary-task-based energy-efficient resource orchestration in mobile edge computing, IEEE Trans. Green Commun. Netw. 7 (1) (2023) 313–327.
[139] M. Wu, Q. Song, L. Guo, I. Lee, Energy-efficient secure computation offloading in wireless powered mobile edge computing systems, IEEE Trans. Veh. Technol. 72 (5) (2023) 6907–6912.
[140] W. Wu, P. Yang, W. Zhang, C. Zhou, X. Shen, Accuracy-guaranteed collaborative DNN inference in industrial IoT via deep reinforcement learning, IEEE Trans. Ind. Inform. 17 (7) (2021) 4988–4998.
[141] A. Fresa, J.P. Champati, Offloading algorithms for maximizing inference accuracy on edge device in an edge intelligence system, IEEE Trans. Parallel Distrib. Syst. 34 (7) (2023) 2025–2039.
[142] C. Xu, Z. Tang, H. Yu, P. Zeng, L. Kong, Digital twin-driven collaborative scheduling for heterogeneous task and edge-end resource via multi-agent deep reinforcement learning, IEEE J. Sel. Areas Commun. 41 (10) (2023) 3056–3069.
[143] Y. Guo, Z. Zhao, K. He, S. Lai, J. Xia, L. Fan, Efficient and flexible management for industrial Internet of Things: A federated learning approach, Comput. Netw. 192 (2021) 108122.
[144] L. Chen, S. Tang, V. Balasubramanian, J. Xia, F. Zhou, L. Fan, Physical-layer security based mobile edge computing for emerging cyber physical systems, Comput. Commun. 194 (2022) 180–188.
[145] T. Zhao, F. Li, L. He, Secure video offloading in MEC-enabled IIoT networks: A multicell federated deep reinforcement learning approach, IEEE Trans. Ind. Inform. 20 (2) (2024) 1618–1629.
[146] Z. Peng, G. Wang, W. Nong, Y. Qiu, S. Huang, Task offloading in multiple-services mobile edge computing: A deep reinforcement learning algorithm, Comput. Commun. 202 (2023) 1–12.
[147] W. Zhan, C. Luo, J. Wang, C. Wang, G. Min, H. Duan, Q. Zhu, Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing, IEEE Internet Things J. 7 (6) (2020) 5449–5465.
[148] C. Li, J. Xia, F. Liu, D. Li, L. Fan, G.K. Karagiannidis, A. Nallanathan, Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach, IEEE Trans. Veh. Technol. 70 (3) (2021) 2922–2927.
[149] X. Wang, Z. Lu, S. Sun, J. Wang, L. Song, M. Nicolas, Optimization scheme of trusted task offloading in IIoT scenario based on DQN, Comput. Mater. Continua 74 (1) (2023) 2055–2071.
[150] Y. Wang, W. Fang, Y. Ding, N. Xiong, Computation offloading optimization for UAV-assisted mobile edge computing: A deep deterministic policy gradient approach, Wirel. Netw. 27 (4) (2021) 2991–3006.
[151] H. Ke, J. Wang, L. Deng, Y. Ge, H. Wang, Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks, IEEE Trans. Veh. Technol. 69 (7) (2020) 7916–7929.
[152] J. Shi, J. Du, Y. Shen, J. Wang, J. Yuan, Z. Han, DRL-based V2V computation offloading for blockchain-enabled vehicular networks, IEEE Trans. Mob. Comput. 22 (7) (2023) 3882–3897.
[153] Z. Chai, H. Hou, Y. Li, A dynamic queuing model based distributed task offloading algorithm using deep reinforcement learning in mobile edge computing, Appl. Intell. 53 (23) (2023) 28832–28847.
[154] X. Xu, C. Yang, M. Bilal, W. Li, H. Wang, Computation offloading for energy and delay trade-offs with traffic flow prediction in edge computing-enabled IoV, IEEE Trans. Intell. Transp. Syst. (2022) 1–11.
[155] H. Tian, X. Xu, L. Qi, X. Zhang, W. Dou, S. Yu, Q. Ni, CoPace: Edge computation offloading and caching for self-driving with deep reinforcement learning, IEEE Trans. Veh. Technol. 70 (12) (2021) 13281–13293.
[156] Y. Li, J. Li, Z. Lv, H. Li, Y. Wang, Z. Xu, GASTO: A fast adaptive graph learning framework for edge computing empowered task offloading, IEEE Trans. Netw. Serv. Manag. 20 (2) (2023) 932–944.
[157] X. Zhou, M. Bilal, R. Dou, J.J.P.C. Rodrigues, Q. Zhao, J. Dai, X. Xu, Edge computation offloading with content caching in 6G-enabled IoV, IEEE Trans. Intell. Transp. Syst. (2023) 1–15.
[158] J. Hou, M. Chen, H. Geng, R. Li, J. Lu, GP-NFSP: Decentralized task offloading for mobile edge computing with independent reinforcement learning, Future Gener. Comput. Syst. 141 (2023) 205–217.
[159] T. Liu, Y. Zhang, Y. Zhu, W. Tong, Y. Yang, Online computation offloading and resource scheduling in mobile-edge computing, IEEE Internet Things J. 8 (8) (2021) 6649–6664.
[160] J. Baek, G. Kaddoum, Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks, IEEE Internet Things J. 8 (2) (2021) 1041–1056.
[161] L. Ale, S.A. King, N. Zhang, A.R. Sattar, J. Skandaraniyam, D3PG: Dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing, IEEE Internet Things J. 9 (19) (2022) 19260–19272.
[162] L. Qian, Y. Wu, F. Jiang, N. Yu, W. Lu, B. Lin, NOMA assisted multi-task multi-access mobile edge computing via deep reinforcement learning for industrial Internet of Things, IEEE Trans. Ind. Inform. 17 (8) (2021) 5688–5698.

[163] S. Zhang, H. Gu, K. Chi, L. Huang, K. Yu, S. Mumtaz, DRL-based partial offloading for maximizing sum computation rate of wireless powered mobile edge computing network, IEEE Trans. Wireless Commun. 21 (12) (2022) 10934–10948.
[164] L. Huang, S. Bi, Y.-J.A. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, IEEE Trans. Mob. Comput. 19 (11) (2020) 2581–2593.
[165] K. Zheng, G. Jiang, X. Liu, K. Chi, X. Yao, J. Liu, DRL-based offloading for computation delay minimization in wireless-powered multi-access edge computing, IEEE Trans. Commun. 71 (3) (2023) 1755–1770.
[166] X. Chen, W. Dai, W. Ni, X. Wang, S. Zhang, S. Xu, Y. Sun, Augmented deep reinforcement learning for online energy minimization of wireless powered mobile edge computing, IEEE Trans. Commun. 71 (5) (2023) 2698–2710.
[167] M. Chen, M. Yi, M. Huang, G. Huang, Y. Ren, A. Liu, A novel deep policy gradient action quantization for trusted collaborative computation in intelligent vehicle networks, Expert Syst. Appl. 221 (2023) 119743.
[168] F. Jiang, L. Dong, K. Wang, K. Yang, C. Pan, Distributed resource scheduling for large-scale MEC systems: A multiagent ensemble deep reinforcement learning with imitation acceleration, IEEE Internet Things J. 9 (9) (2022) 6597–6610.
[169] J. Yun, Y. Goh, W. Yoo, J.-M. Chung, 5G Multi-RAT URLLC and eMBB dynamic task offloading with MEC resource allocation using distributed deep reinforcement learning, IEEE Internet Things J. 9 (20) (2022) 20733–20749.
[170] Y. Cui, H. Li, D. Zhang, A. Zhu, Y. Li, H. Qiang, Multi-agent reinforcement learning-based cooperative multitype task offloading strategy for Internet of Vehicles in B5G/6G network, IEEE Internet Things J. 10 (14) (2023) 12248–12260.
[171] J. Chen, H. Xing, Z. Xiao, L. Xu, T. Tao, A DRL agent for jointly optimizing computation offloading and resource allocation in MEC, IEEE Internet Things J. 8 (24) (2021) 17508–17524.
[172] X. Qiu, W. Zhang, W. Chen, Z. Zheng, Distributed and collective deep reinforcement learning for computation offloading: A practical perspective, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2021) 1085–1101.
[173] Van Dat Tuong, W. Noh, S. Cho, Delay minimization for NOMA-enabled mobile edge computing in industrial Internet of Things, IEEE Trans. Ind. Inform. 18 (10) (2022) 7321–7331.
[174] J. Gao, Z. Kuang, J. Gao, L. Zhao, Joint offloading scheduling and resource allocation in vehicular edge computing: A two layer solution, IEEE Trans. Veh. Technol. 72 (3) (2023) 3999–4009.
[175] P. Luong, F. Gagnon, L.-N. Tran, F. Labeau, Deep reinforcement learning-based resource allocation in cooperative UAV-assisted wireless networks, IEEE Trans. Wireless Commun. 20 (11) (2021) 7610–7625.
[176] X. Zhang, X. Zhang, W. Yang, Joint offloading and resource allocation using deep reinforcement learning in mobile edge computing, IEEE Trans. Netw. Sci. Eng. 9 (5) (2022) 3454–3466.
[177] L. Zhang, B. Jabbari, N. Ansari, Deep reinforcement learning driven UAV-assisted edge computing, IEEE Internet Things J. 9 (24) (2022) 25449–25459.
[178] H. Huang, Q. Ye, Y. Zhou, Deadline-aware task offloading with partially-observable deep reinforcement learning for multi-access edge computing, IEEE Trans. Netw. Sci. Eng. 9 (6) (2022) 3870–3885.
[179] M. Yi, P. Yang, M. Chen, N.T. Loc, A DRL-driven intelligent joint optimization strategy for computation offloading and resource allocation in ubiquitous edge IoT systems, IEEE Trans. Emerg. Top. Comput. Intell. 7 (1) (2023) 39–54.
[180] M.Z. Alam, A. Jamalipour, Multi-agent DRL-based hungarian algorithm (MADRLHA) for task offloading in multi-access edge computing Internet of Vehicles (IoVs), IEEE Trans. Wireless Commun. 21 (9) (2022) 7641–7652.
[181] T. Ren, J. Niu, B. Dai, X. Liu, Z. Hu, M. Xu, M. Guizani, Enabling efficient scheduling in large-scale UAV-assisted mobile-edge computing via hierarchical reinforcement learning, IEEE Internet Things J. 9 (10) (2022) 7095–7109.
[182] Z. Wang, Y. Wei, Z. Feng, F.R. Yu, Z. Han, Resource management and reflection optimization for intelligent reflecting surface assisted multi-access edge computing using deep reinforcement learning, IEEE Trans. Wireless Commun. 22 (2) (2023) 1175–1186.
[183] H. Liao, Z. Jia, Z. Zhou, Y. Wang, H. Zhang, S. Mumtaz, Cloud-edge-end collaboration in air-ground integrated power IoT: A semidistributed learning approach, IEEE Trans. Ind. Inform. 18 (11) (2022) 8047–8057.
[184] D. Han, Q. Ye, H. Peng, W. Wu, H. Wu, W. Liao, X. Shen, Two-timescale learning-based task offloading for remote IoT in integrated satellite-terrestrial networks, IEEE Internet Things J. 10 (12) (2023) 10131–10145.
[185] I. Budhiraja, N. Kumar, H. Sharma, M. Elhoseny, Y. Lakys, J.J.P.C. Rodrigues, Latency-energy tradeoff in connected autonomous vehicles: A deep reinforcement learning scheme, IEEE Trans. Intell. Transp. Syst. 24 (11) (2023) 13296–13308.
[186] Y. Liu, J. Yan, X. Zhao, Deep-reinforcement-learning-based optimal transmission policies for opportunistic UAV-aided wireless sensor network, IEEE Internet Things J. 9 (15) (2022) 13823–13836.
[187] S. Tan, B. Chen, D. Liu, J. Zhang, L. Hanzo, Communication-assisted multi-agent reinforcement learning improves task-offloading in UAV-aided edge-computing networks, IEEE Wirel. Commun. Lett. 12 (12) (2023) 2233–2237.
[188] Z. Gao, L. Yang, Y. Dai, Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing, IEEE Trans. Mob. Comput. 22 (6) (2023) 3425–3443.
[189] B. Zhang, B. Tang, F. Xiao, Learning to coordinate in mobile-edge computing for decentralized task offloading, IEEE Internet Things J. 10 (1) (2023) 893–903.
[190] Y. Lyu, Z. Liu, R. Fan, C. Zhan, H. Hu, J. An, Optimal computation offloading in collaborative LEO-IoT enabled MEC: A multi-agent deep reinforcement learning approach, IEEE Trans. Green Commun. Netw. 7 (2) (2023) 996–1011.
[191] H. Zhou, Y. Long, S. Gong, K. Zhu, D.T. Hoang, D. Niyato, Hierarchical multi-agent deep reinforcement learning for energy-efficient hybrid computation offloading, IEEE Trans. Veh. Technol. 72 (1) (2023) 986–1001.
[192] D.C. Nguyen, M. Ding, P.N. Pathirana, A. Seneviratne, J. Li, H.V. Poor, Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning, IEEE Trans. Mob. Comput. 22 (4) (2023) 2021–2037.
[193] T. Zhang, Z. Wang, Y. Liu, W. Xu, A. Nallanathan, Joint resource, deployment, and caching optimization for AR applications in dynamic UAV NOMA networks, IEEE Trans. Wireless Commun. 21 (5) (2022) 3409–3422.
[194] C. Li, L. Chai, K. Jiang, Y. Zhang, J. Liu, S. Wan, DNN partition and offloading strategy with improved particle swarm genetic algorithm in VEC, IEEE Trans. Intell. Veh. (2024) 1–11.
[195] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, A. Knoll, A review of safe reinforcement learning: Methods, theory and applications, 2023, arXiv:2205.10330.
[196] R. Figueiredo Prudencio, M.R.O.A. Maximo, E. Luna Colombini, A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems, 2022, arXiv e-prints, arXiv:2203.01387.
[197] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, Y. Zhang, Deep learning empowered task offloading for mobile edge computing in urban informatics, IEEE Internet Things J. 6 (5) (2019) 7635–7647.
[198] J. Wang, J. Hu, J. Mills, G. Min, M. Xia, N. Georgalas, Federated ensemble model-based reinforcement learning in edge computing, IEEE Trans. Parallel Distrib. Syst. 34 (6) (2023) 1848–1859.
[199] Y. Aono, T. Hayashi, L. Wang, S. Moriai, et al., Privacy-preserving deep learning via additively homomorphic encryption, IEEE Trans. Inf. Forensics Secur. 13 (5) (2017) 1333–1345.
[200] X. Yin, Y. Zhu, J. Hu, A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions, ACM Comput. Surv. 54 (6) (2021) 1–36.
[201] Z. Gao, L. Yang, Y. Dai, Fast adaptive task offloading and resource allocation in large-scale MEC systems via multi-agent graph reinforcement learning, IEEE Internet Things J. 11 (1) (2024) 758–776.
[202] K. Shuai, Y. Miao, K. Hwang, Z. Li, Transfer reinforcement learning for adaptive task offloading over distributed edge clouds, IEEE Trans. Cloud Comput. 11 (2) (2023) 2175–2187.
[203] Y. Chen, Y. Sun, B. Yang, T. Taleb, Joint caching and computing service placement for edge-enabled IoT based on deep reinforcement learning, IEEE Internet Things J. 9 (19) (2022) 19501–19514.
[204] T. Hickling, A. Zenati, N. Aouf, P. Spencer, Explainability in deep reinforcement learning: A review into current methods and applications, ACM Comput. Surv. 56 (5) (2023) 125:1–125:35.