Alexandria Engineering Journal 108 (2024) 89–107

Contents lists available at ScienceDirect

Alexandria Engineering Journal

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

Review

Reinforcement learning-based computation offloading in edge computing: Principles, methods, challenges

Zhongqiang Luo [a,b,*,1], Xiang Dai [a,1]

[a] School of Automation and Information Engineering, Sichuan University of Science & Engineering, Yibin, 644000, Sichuan, China
[b] Artificial Intelligence Key Laboratory of Sichuan Province, Sichuan University of Science & Engineering, Yibin, 644000, Sichuan, China

ARTICLE INFO

Keywords:
Edge computing
Reinforcement learning
Computation offloading
Offloading decision
Edge caching
Resource allocation

ABSTRACT

With the rapid development of mobile communication technologies and Internet of Things (IoT) devices, Multi-Access Edge Computing (MEC) has become one of the most potential technologies for wireless communication. In MEC systems, faster and more reliable data processing can be provided to IoT devices through computation offloading, but edge servers have limited computing and storage resources. The prerequisite for whether an IoT device can offload a computation task to an edge server for processing is whether the edge server has enough remaining available resources and whether the edge server caches the services related to the task, followed by finding the best way to offload the task. Therefore, to process tasks efficiently, offloading decisions, resource allocation, and edge caching need to be jointly considered during offloading tasks to edge servers. Reinforcement Learning (RL) has recently emerged as a key technique for solving the computation offloading problem in MEC, and a large number of optimization methods have emerged. In this context, we provide a comprehensive survey of RL-based computation offloading fundamental principles and theories in MEC, including mechanisms for finding optimal offloading decisions, methods for joint resource allocation, and means for joint edge caching. In addition, we also discuss the challenges and future work of RL-based computation offloading methods.

1. Introduction

In recent years, the surge in Internet of Things (IoT) devices has ushered in a plethora of intelligent applications that seamlessly permeate our daily existence, such as real-time data processing [1], autonomous driving [2], smart healthcare [3], virtual reality (VR) [4], augmented reality (AR) [5], and industrial IoT [6]. Due to the escalating computing requirements and exponential growth of computation tasks generated by these applications, resource-constrained devices are unable to efficiently perform a large number of computing-intensive and latency-sensitive tasks in a short time [7].

Cloud computing has been implemented to address this challenge. IoT devices leverage computation offloading strategies to migrate computing-intensive tasks to cloud servers endowed with ample computing, storage, and other resources. This approach significantly diminishes the time to execute the tasks and reduces the energy consumption of the IoT devices [8]. However, since cloud servers are usually deployed far from IoT devices, offloading tasks to cloud servers incur large transmission latency, which is very unfriendly to latency-sensitive applications. To address the constraints inherent in cloud computing, the ETSI has introduced the Multi-Access Edge Computing (MEC) paradigm. MEC facilitates the relocation of task offloading positions in proximity to IoT devices through the deployment of computing, storage, and ancillary resources. This strategy allows for prompt responsiveness to user service requests and diminishing network traffic.

MEC serves as an adjunctive technology to cloud computing, but edge servers lack the significant computing and storage resources available on cloud servers and cannot cache the services required for all tasks. Therefore, to handle offloaded tasks successfully, it is necessary to ensure that servers have sufficient computing resources and have cached the services related to the tasks [9]. Additionally, the tasks themselves are interdependent, which compounds the difficulty of computation offloading.

In this scenario, the efficient execution of tasks necessitates judicious offloading decisions, incorporating considerations of the remaining resources and edge caches within the MEC system, in addition to

* Corresponding author.
E-mail addresses: luozhongqiang@suse.edu.cn (Z. Luo), 322085404128@stu.suse.edu.cn (X. Dai).
[1] Zhongqiang Luo and Xiang Dai contributed equally to this work.

https://doi.org/10.1016/j.aej.2024.07.049
Received 10 March 2024; Received in revised form 3 June 2024; Accepted 12 July 2024
Available online 23 July 2024

Table 1
Surveys of existing theoretical for computation offloading.

Ref.	Contributions
[16]	Concepts and challenges of computation offloading
[17]	Potential advantages of computation offloading, ideas based on ML methods, and comparisons with traditional methods
[18]	Three intelligent algorithms for optimizing computation offloading
[19]	Relevant concepts, requirements, paradigms, and techniques for edge ML

characteristics inherent in the tasks themselves. For this reason, numerous approaches have been proffered to address the computation offloading predicament, such as the Lyapunov optimization method [10], convex optimization methods [11], and game theory [12], heuristic techniques [13], have been suggested. Nevertheless, the heterogeneous characteristics of edge networks pose a challenge for these approaches in guaranteeing dependable and effective data transmission [14]. For instance, heuristic algorithms require many iterations, only consider one optimization [15], and suffer from low robustness. Henceforth, a substantial work of recent research has pivoted its objective towards resolving the computation offloading quandary through the employment of Reinforcement Learning (RL) techniques imbued with formidable cognitive and decision-making capabilities, especially the Deep Reinforcement Learning (DRL) methods, which are generated by combining the advantages of RL and Deep Learning (DL), have attracted much attention.

1.1. Existing surveys on computation offloading

This section introduces current surveys focused on computation offloading, classifying them into theoretical and methodological categories. Theoretical surveys predominantly scrutinize concepts, methodological principles, challenges, and associated concepts of computation offloading. Methodological surveys primarily explore techniques and methods of existing research endeavors in computation offloading. Table 1 delineates the theoretical surveys, while Table 2 encapsulates the methodological surveys.

1.1.1. Theoretical surveys

In [16], the authors focus on the computation offloading impact of the architecture, nature, granularity, and network connectivity on the performance of computation offloading and analyze the four challenges it faces: program partitioning, task allocation, resource management, and distributed execution. In [17], the authors introduce the concept and potential benefits of computation offloading, discuss the idea of Machine Learning (ML) based approaches, and compare ML-based methods with traditional methods. [18] focuses on three intelligent algorithms for computation offloading optimization: evolutionary, swarm intelligent, and ML algorithms. [19] provides a comprehensive overview of the concepts, requirements, paradigms, and techniques related to edge ML, while pointing out future research directions.

1.1.2. Methodological surveys

In Ref. [20], the authors furnish an exhaustive survey encompassing the architecture of MEC along with computation offloading techniques. They systematically classify research endeavors on computation offloading into three domains: offloading decisions, mobility management, and resource allocation and analyze the existing efforts in these three areas. In [21], the authors focus on computation offloading approaches for cloud–edge collaboration and heuristic offloading, analyzing and discussing related efforts. In [22], the authors mainly examine the computation offloading methods under different MLs. The authors of [23] focus on the modeling problem of computation offloading, which they classify as Markov Decision Process (MDP), convex and nonconvex optimization, game theory, Lyapunov optimization, and ML. They systematically analyze the merits and limitations of several methods. [24] surveys related efforts on computation offloading through three distinct lenses: computation perspective, decision-making perspective, and algorithmic paradigm. In [7], the authors systematically review computation offloading efforts, focusing on their approaches, objectives, and methodologies. [25] focus on a survey of computation offloading approaches for the unmanned aerial vehicle (UAV)-assisted MECs and compare the described methods in terms of design methodology, operational characteristics, optimization goals, and application areas. In [26], the authors discuss the objective functions, types of offloading methods, and application areas of six computation offloading optimization methods: game theory, convex optimization, heuristic techniques, Lyapunov optimization, and ML. The study conducted by [27] comprehensively examined efforts concerning the privacy-preserving aspects of the computation offloading process. The existing methods for privacy-preserving offloading were systematically categorized into three distinct classes, guided by their association with offloading phases: management of offloaded data and modes, secure transmission of offloaded data, and the selection of offload destinations. In [28], computation offloading under ML-based and traditional algorithms is mainly analyzed. In [29], the authors provide an overview of computation offloading methods based on network, use cases and MEC architectures, RL algorithms, computing offloading objectives, decision methods, and scenarios.

1.2. Motivation for this survey

Upon reviewing existing surveys, it is evident that the majority of those on computation offloading concentrate on traditional algorithms or provide a general overview of all algorithms. Few surveys offer a comprehensive survey of only RL-based computation offloading methods, leading to an incomplete overview of RL-based computation offloading. Furthermore, they tend to focus on only one problem, while in MEC, computation offloading is not a single process but often a combined issue of offloading decisions, resource allocation, and edge caching. Upon receiving a task request, the system determines if and how the task should be offloaded, as well as how many resources should be allocated based on what service programs and content the server has cached and whether there are still computing resources and storage resources. In light of the deficiencies identified in the aforementioned surveys, we introduce an exhaustive survey of the computation offloading problem within MEC from the vantage point of RL. Specifically, we categorize and discuss recent approaches that use RL techniques to address offloading decisions, joint resource allocation, and joint edge caching in MEC. In this paper, the term “joint resource allocation” denotes the amalgamation of both the computation offloading and resource allocation issues. Similarly, “joint edge caching” pertains to the integration of the computation offloading and edge caching problems. These combinations aim to optimize the efficient allocation of tasks generated by applications to the most appropriate servers.

1.3. Contributions of this survey

In difference to the existing work described in Tables 1 and 2, this survey focuses on RL-based joint computation offloading methods in MEC. The following are this paper’s primary contributions:

- Outline the basic network architecture of MEC and the key networking technologies.
- The paper delves into an analysis of the performance metrics employed in RL-based computation offloading within MEC, and several of the most commonly used RL algorithms are presented.
- A comprehensive survey of recent works used RL to address the computation offloading in MEC. Particularly, we categorize and discuss recent RL-based methods for computation offloading from three aspects: finding optimal offloading decisions, joint resource allocation, and joint edge caching.

Table 2
Surveys of existing methodological for computation offloading.

Ref.	Techniques of survey				Computation offloading aspects				Focus
	TT	DL	ML		OD	Joint RA	Joint EC	UN	
			ALL	RL					
[20]					✓				Analysis of efforts related to offloading decisions, resource allocation, and mobility management
[21]	✓							✓	Gaming and cooperation between edge and cloud, heuristic offloading
[22]			✓					✓	ML-based computation offloading
[23]	✓		✓					✓	Modeling techniques for computation offloading
[24]	✓		✓		✓			✓	Categorize and discuss computation offloading schemes
[7]	✓		✓					✓	Computation offloading modalities, objectives, and methods
[25]	✓	✓		✓	✓			✓	Computation offloading methods for UAV-assisted MEC
[26]	✓		✓					✓	Six Optimization Methods for computation offloading
[27]								✓	Privacy protection during computation offloading
[28]	✓	✓	✓					✓	Computation offloading with different algorithms
[29]				✓	✓			✓	Computation offloading in different scenarios
Our survey				✓	✓	✓	✓		RL-based computation offloading in MEC and its joint optimization approach

Note: TT: Traditional techniques; In the ML item, all represents all ML methods; OD: Offloading decisions; RA: resource allocation; EC: edge caching; UN: Focusing only on the offloading process without carefully categorizing whether the work in the literature reviewed considers both resource allocation and edge caching.

- Delves into the challenges RL-based in computation offloading within MEC, elucidating the pertinent issues that necessitate resolution.

The remainder of the paper is organized as follows: Section 2 outlines the basic network architecture of MEC, key network technologies, typical tasks, performance metrics, key factors affecting offloading performance, and optimization methods and RL algorithms for RL in MEC. Section 3 provides a comprehensive survey of RL-based computation offloading efforts. Section 4 explores RL-based computation offloading within MEC, elucidating the pertinent issues that necessitate resolution. Section 5 concludes the paper.

2. Overview

Within this section, we initially provide an overview of the network structure and two pivotal technologies of MEC. Subsequently, we present the offloading model, offloading metrics, and critical factors affecting the offloading performance for computation offloading. Lastly, a comprehensive analysis and comparison of various RL algorithms for computation offloading is conducted.

2.1. MEC

The advent of MEC endeavors to relocate functions and services from their original placement in the cloud to the network's edge so that users can obtain high broadband and ultra-low latency, thereby ensuring superior high-quality network service functions [30]. There are three definitions of MEC in academia, namely edge computing, mobile edge computing [31], and multi-access edge computing [32]. Edge computing was the first concept proposed, subsequently, the ETSI introduced the concept of mobile edge computing, aiming to incorporate edge computing into the framework of mobile networks. However, with the development of scientific research and considering the edges of non-mobility networks, ETSI extended mobile edge computing by advancing the notion of multi-access edge computing, thus further extending edge computing to wireless access networks.

As mentioned above, Edge Computing, Mobile Edge Computing, and Multi-Access Edge Computing are three different concepts. However, they all serve the same purpose: mitigating the computational burden on cloud computing, minimizing latency, and enhancing both Quality of Experience (QoE) and Quality of Service (QoS). Although ETSI has clarified the differences between the different terms, authors usually interchange these terms in numerous literatures [33]. In this survey, we use MEC to refer to Edge Computing, Mobile Edge Computing, and Multi-Access Edge Computing to facilitate a better survey.

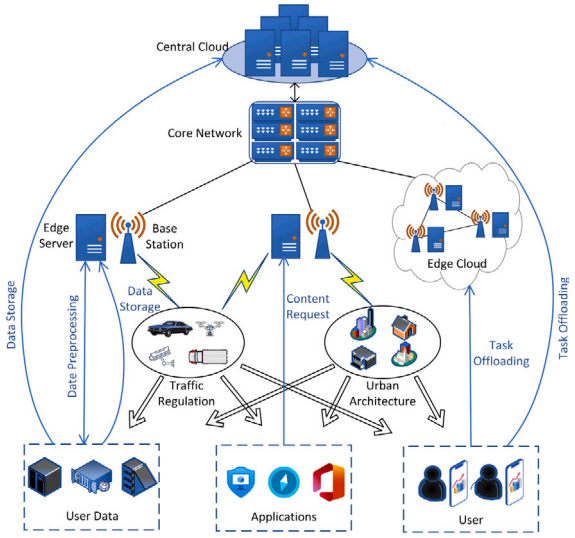

Fig. 1. MEC network architecture.

2.1.1. MEC network architecture

As depicted in Fig. 1, the network architecture of MEC consists of the cloud layer, edge layer, and end device layer. The cloud layer comprises sizable data centers that are geographically distant from users. As a result of the lengthy transmission distance, cloud-based tasks experience significant transmission latency. The edge layer consists of multiple edge servers located next to the base station (BS). Each edge server represents a miniature data center deployed to cater to specific service requests, including task offloading, data storage, and data preprocessing [7]. In contrast, task data is generated in the terminal layer, which comprises user equipment (UE), vehicles, UAVs, and other components.

2.1.2. Key networking technologies enabling MEC implementation

(1) Software Defined Networking

Software Defined Networking (SDN) is a new networking technology that separates the data plane from the control and management planes [34]. The introduction of SDN into MEC [35] enables centralized logical control of distributed computing nodes. In addition, SDN can use a rule-based forwarding policy to determine the most appropriate offloading path for the computation

offloading process. Aiming at the above characteristics, literature [36] presents an architecture integrating SDN with MEC to address the task offloading strategy for IoT devices within the MEC framework. Facing the complex edge network environment, [37] proposed an SDN-based SDBlockEdge approach to find efficient offloading decisions for MEC while considering traffic scheduling, load balancing, and controlling anomalous requests of IoT devices for MEC. The authors of [38] proposed an RL-based software-defined edge task assignment algorithm to reduce latency and energy consumption.

(2) Network Functions Virtualization
Network Functions Virtualization (NFV) by leveraging standard IT virtualization technologies and integrating proprietary hardware-based network functions into standard commercial equipment [39], aiming to provide a more efficient and scalable allocation of network resources through the development of virtualization technologies to manage network functions, shorten time-to-market, and reduce equipment costs [40]. MEC network functions are implemented as virtual network functions (VNFs) in an NFV environment [41]. However, the distributed and mobile nature of UEs can lead to uneven traffic distribution in MEC systems, resulting in VNFs placed on edge servers fluctuating according to UE fluctuations, making it crucial to study the placement of VNFs. To this end, [42] proposes a gaming approach for VNF placement in satellite MEC. This approach aims to mitigate the deployment cost per user request, leveraging the uninterrupted coverage and minimal transmission latency inherent in low-Earth-orbit satellite networks. In [43], the authors examine the resource allocation quandary concerning VNF placement within MEC environments. [44] introduced EdgeGym, a DRL framework for creating repeatable NFV environments to optimize resource allocation through RL.

SDN and NFV, as Key networking technologies for MEC, are highly complementary, so much work has focused on the convergence of NFV and SDN in MEC networks. In [45], capitalize on MEC and put forth a network architecture that integrates SDN and NFV. This framework offers a novel network solution, reaping the advantages of optimization, configuration, and automated provisioning of NFV. [46] proposed a network operating system called LlihtMANO that integrates SDN and NFV, for the management and orchestration of network services.

2.1.3. Typical tasks and applications in MEC

With the rapid development of MEC, the number of scenarios in which MEC is applied is also gradually increasing, and the types of tasks that need to be handled in the face of different scenarios tend to be different. This section introduces several common, typical tasks in MEC and their application scenarios.

(1) Computing-intensive tasks
Computing-intensive tasks are those that require a large amount of computational resources to process, and such tasks usually involve complex mathematical operations and large amounts of data processing. Examples include object detection [47] and facial recognition [48,49] in video surveillance and image recognition, and calculating optimal travel paths [50] and obstacle avoidance paths [51] in UAVs and autonomous driving.

(2) Data-intensive tasks
Data-intensive tasks are tasks that require processing of large amounts of data, which are more dependent on the transmission and storage of data than Computing-intensive tasks, but relatively less dependent on computational power. For example, sensor data processing in smart home and medical IoT [52,53], patient status monitoring [54], and environmental data analysis in environmental detection [55].

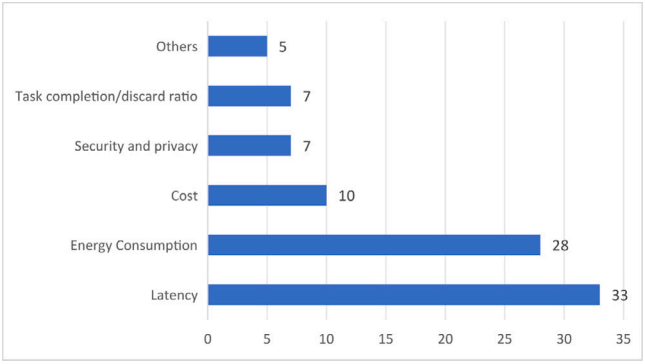

Fig. 2. Bar chart of RL-based computation offloading performance metrics.

(3) Latency-sensitive tasks
Latency-sensitive tasks with very high response time requirements, which usually require data processing and feedback in a short period of time, are critical in many scenarios. For example, image rendering [56], interaction processing [57] data synchronization [58] in AR and VR, and real-time monitoring [59] and fault response [60] in smart grids.

(4) Hybrid tasks
Hybrid tasks tend to be the most common tasks because MEC is expected to provide efficient, real-time, and reliable services, the tasks need to fulfill multiple requirements simultaneously. For example, in [61], different vehicles generate a large amount of data that needs to be transferred and processed between edge servers and cloud servers, while communication between vehicles needs to be processed instantly. In [62], the authors consider UAV joint trajectory optimization, directed acyclic graph task scheduling, and service function deployment as optimization problems, which combine computing-intensive, data-intensive, and latency-sensitive properties.

2.2. Computation offloading

Computation offloading is a practice whereby IoT devices transfer a portion or the entirety of their computation tasks to remote computing devices with abundant resources (e.g., cloud servers or edge servers) to address the deficiencies in computation performance and energy efficiency of IoT devices.

2.2.1. Performance metrics

To analyze the importance attached to each performance metric of RL-based computation offloading, we summarize and analyze various performance metrics in the surveyed literature. The number of times each metric is used is shown in the bar chart in Fig. 2. In order to observe the use of each metric more intuitively, we draw a pie chart, as shown in Fig. 3. From Fig. 3, we can see that latency and energy consumption account for 37% and 32% of the performance metrics used in the literature investigated, and the rest of the performance metrics account for much lower percentages than latency and energy consumption.

(1) Latency
For latency-sensitive applications such as VR, real-time data processing, related work tends to minimize latency as the first optimization goal [63–65]. Some work will sacrifice other performances to achieve this goal in order to minimize latency. For example, in [66], to avoid the task offloading process into a queuing latency due to insufficient computing resources, thus increasing the total latency, the authors added an idle server on top of the server system, which reduces the latency but

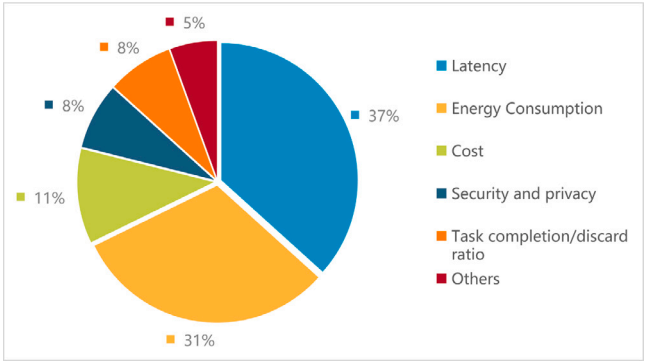

Fig. 3. Pie distribution of RL-based computation offloading performance metrics.

also wastes specific resources. In MEC systems, computing, communication, and storage resources are limited, and latency is not the only metric considered in many studies, such as [67] used latency and bandwidth cost as the optimization objective, and [68] used latency and task discard ratio as the optimization objective.

(2) Energy consumption
Reducing energy consumption saves limited resources and increases the lifetime of equipment. Many efforts are designed to reduce energy consumption only [69]; some add a latency constraint to ensure that the task minimizes energy consumption within the maximum tolerated latency [70–72]; and others consider both latency and energy reduction, and decide the weight of the two metrics by the degree of importance that different tasks place on latency and energy consumption [73–76].

(3) Cost
The cost of computation offloading is usually a composite performance metric, including latency cost [77,78], energy cost [77, 79], bandwidth resource cost [79], and computing resource cost [80]. Most of the current work on minimizing the offloading cost is focused on finding the optimal solution by building different cost models.

(4) Security and privacy
During computation offloading, the offloaded tasks are vulnerable to attacks from malicious attackers, such as private data leakage, data tampering, and data replication. After much research, the methods to protect offloaded data can now be categorized into three groups: secure transmission of offloaded data [81,82], management of offloaded patterns and data [83], and selection of offloading destinations [84]. For privacy protection during computation offloading, one can refer to [27], where the authors comprehensively analyze various approaches to privacy protection.

(5) Task completion/discard ratio
The task completion rate is the proportion of tasks that can be processed successfully compared to the total number of tasks generated in each time slot, while the task discard rate is vice versa. The task completion/discard rate in a system is influenced by several factors, such as the volume of tasks and latency; this is because the edge server and its computing resources are constant. If a lot of user tasks need to be processed at once, it might not have the resources to handle them all, and task dropping occurs if the latency is too high, so the task completion/discard rate is often studied along with latency. For example, [85] considered increasing the task completion ratio and decreasing the latency as the research objectives, [86,87] jointly consider the latency, energy consumption, and task discard ratio of the system, and [88] used the cost and task completion ratio as the optimization objectives.

2.2.2. Computation model

During the computation offloading process, tasks generated by UE can either be offloaded to an edge server or processed locally. From the analysis in the previous section, we know that the two most commonly used performance metrics in the computation offloading process are latency and energy consumption; therefore, we model these two metrics and their trade-off models in this section to analyze them in more depth.

It is posited that the system comprises a total I of UEs, denoted as $U = \{U_1, U_2, \dots U_I\}$, and each UE has J tasks, denoted as $D = \{D_1, D_2, \dots D_J\}$. Each computation task of the UE is defined as $M_{i,j} = (M^{in}_{i,j}, M^{c}_{i,j})$, where $M^{in}_{i,j}$ represents the size of the task input data, while $M^{c}_{i,j}$ denotes the size of the computation data required for the task, $i \in \{1, 2, \dots, I\}$, $j \in \{1, 2, \dots, J\}$. f^l and f^e symbolize the computational capabilities of the UE and the edge server. It is assumed that these computational capabilities remain constant throughout the computation process. The number of CPU cycles required to process 1 bit of data by the UE and the edge server are represented by C^l and C^e respectively. Define $a_{i,l} \in \{0, 1\}$ as the decision variable, when $a_{i,l} = 0$, the task is executed locally on the UE, and when $a_{i,l} = 1$, the task is offloaded to the edge server for processing.

(1) Local computing
When $a_{i,l} = 0$, the task is executed locally, and no data transfer will take place. Therefore, during the task execution, there is no incurred transmission latency or energy consumption related to transmission, and only the computation time and energy consumption are considered. So, the local execution latency and energy consumption of task j of UE i is defined as

$$T^{l}_{i,j} = \frac{M^{c}_{i,j} C^{l}_{i}}{f^{l}_{i}} \tag{1}$$

$$E^{l}_{i,j} = \delta M^{c}_{i,j} C^{l}_{i} (f^{l}_{i})^2 \tag{2}$$

where δ represents the effective capacitance factor, a variable contingent upon the chip architecture employed within the device.

(2) Offloading to the edge
When $a_{i,l} = 1$, the task is offloaded to the edge server for processing. During this process, the task data from the UE can be transferred to the edge server through a wireless communication link. The data transfer rate is denoted by R. According to Shannon's formula, we have:

$$R_i = W_i \log_2 \left(1 + \frac{P_i h}{N}\right) \tag{3}$$

where W_i is the channel bandwidth allocated to the user, $\frac{P_i h}{N}$ is the user signal-to-noise ratio, N is the Gaussian channel noise, P_i is the transmitted power, and h is the channel gain.
The latency incurred when a task is processed in the edge server consists of three components: task upload latency, task execution latency in the edge server, and result feedback latency. However, the data fed back from the edge server is much smaller compared to the data at the time of upload, and the downlink transmission rate is much higher than that of the uplink transmission rate, so the latency generated by the result feedback is often not considered in existing work. Therefore, the transmission latency and execution latency incurred by task offloading to the edge is

$$T^{eu}_{i,j} = \frac{M^{in}_{i,j}}{R_i} \tag{4}$$

$$T^{ec}_{i,j} = \frac{M^{c}_{i,j} C^{e}}{f^{e}_{i,j}} \tag{5}$$

The energy expenditure associated with task offloading to edge computing encompasses two primary components: the energy

required for data transmission and the energy utilized during task execution, denoted as:

$$E_{i,j}^{eu} = P_i T_{i,j}^{eu} = P_i \frac{M_{i,j}^{in}}{R_i} \tag{6}$$

$$E_{i,j}^{ec} = \delta M_{i,j}^{c} C^{e} (f_{i,j}^{e})^2 \tag{7}$$

Summarizing, the average delay incurred in executing the task can be expressed as:

$$T = \frac{1}{IJ} \sum_{i=1}^{I} \sum_{j=1}^{J} (1 - a_{i,l}) T_{i,j}^{l} + a_{i,l} (T_{i,j}^{eu} + T_{i,j}^{ec}) \tag{8}$$

The average energy consumption generated by performing the task is

$$E = \frac{1}{IJ} \sum_{i=1}^{I} \sum_{j=1}^{J} (1 - a_{i,l}) E_{i,j}^{l} + a_{i,l} (E_{i,j}^{eu} + E_{i,j}^{ec}) \tag{9}$$

Therefore, we can define a multi-objective optimization task, denoted as

$$\begin{aligned} &\min \{T, E\} \\ &s.t. \\ &C1 : a_{i,l} \in \{0, 1\} \\ &C2 : \sum_{i=1}^{I} \sum_{j=1}^{J} f_{i,j}^{e} \le f^{e} \end{aligned} \tag{10}$$

If consider that different types of tasks have different sensitivities to latency and energy consumption, e.g., latency-sensitive tasks tend to be more concerned about the impact of latency. We can define a weighting factor ∂, $\partial \in [0, 1]$, to capture the varying degrees of emphasis placed on latency and energy consumption by the task, the multi-objective optimization problem is reformulated into a single-objective optimization problem:

$$X = \partial T + (1 - \partial) E \tag{11}$$

In this regard, the optimization objective for computation offloading can be represented as

$$\begin{aligned} &\min \{X\} \\ &s.t. \\ &C1 : a_{i,l} \in \{0, 1\} \\ &C2 : \sum_{i=1}^{I} \sum_{j=1}^{J} f_{i,j}^{e} \le f^{e} \end{aligned} \tag{12}$$

In the optimization objective, constraint term $C1$ stipulates that the task can either be completed locally or offloaded to an edge server for processing; the constraint term $C2$ ensures that the computing resources allocated for the task do not surpass the server's total available resources.

2.2.3. Key factors affecting offloading performance

During the process of computation offloading, the performance of offloading is influenced by an interplay of multiple factors: offloading decisions, resource allocation, and edge caching. These elements collectively determine the efficiency and effectiveness of the offloading process.

(1) Offloading decision

The essence of computation offloading is formulating the optimal offloading decision [89]. This encompasses resolving key issues such as whether the task is offloaded, what is offloaded, and how much is offloaded. The results of offloading decisions are categorized into three types: (1) Local execution: the entire computation process is performed on the UE and does not need to be offloaded to the edge server. This generally occurs because the edge server computing resources are unavailable or offloading has no practical significance. (2) Complete offloading: that is, the entire computation process is offloaded and processed by the edge server. (3) Partial offloading: a portion of the computation process is executed locally, and the remainder is sent to the edge server to be executed. Fig. 4 illustrates the potential outcomes resulting from the offloading decision.

The offloading decision represents a multifaceted process, influenced by a range of factors including user preferences, radio conditions, UE functionality, and the availability of computing resources [90]. Developing an offloading decision also requires considering the offloadability of tasks and the interdependencies among tasks, determining the applicability in the offloading process.

- Offloadability of computation tasks: Tasks can be divided into three groups according to whether they are offloadable or not. The first type of computation task is not offloadable; this type of task can only be executed locally. The second type is a computation task fully offloadable, which consists of N offloadable parts. During the computation offloading process, decisions are made on which parts to execute locally and which parts to offload to the edge servers, depending on the amount of data in each part and the computation required, as shown in Fig. 5 UE1. The third category is the computation tasks, which are partially offloadable and partially non-offloadable; the non-offloadable component must be executed locally, while the offloadable part again decides which parts are executed locally and which parts are offloaded to the edge servers according to the difference of its data volume and the amount of computation required, as depicted in Fig. 5 UE2.
- Dependencies of computation tasks: Each part of a computation task can be independent or interdependent. When there are no dependencies between all computation tasks, how to offload the task is determined by its offloadability; when there is a dependency between the parts of the task, after determining the offloadability of the task, it becomes imperative to assess these interdependencies between the parts and to determine further which parts can be offloaded and which parts cannot be offloaded.

(2) Resource allocation

Once the offloading decision has been made, the next step is considering the rational resource allocation. Resource allocation in MEC mainly includes computing resource allocation, storage resource allocation, and communication resource allocation [91]. Communication resource allocation is mainly based on the wireless system environment to allocate communication resources reasonably. The allocation of storage and computing resources primarily hinges on the specific requirements of the task at hand, ensuring an effective distribution of these resources following those needs. When storage and computing resource allocation are performed, tasks can be offloaded to one or more edge servers according to the dependency of computation tasks. Suppose the computational tasks of a UE are non-splittable or splittable but the split parts are connected, the tasks can only be offloaded to the same edge server, as shown in Fig. 6 UE1. In the case of splittable and split parts that are not connected, it is possible to assign the tasks to several edge servers, as shown in Fig. 6 UE2. According to the above, the current resource allocation nodes are mainly categorized into single-node and multi-node allocations [92].

(3) Edge caching

Edge caching includes content caching and service caching (or service placement). Content caching denotes the storing of input or output data of computation tasks at the server or user side [93], and service caching denotes the pre-storage of essential service programs and their corresponding databases or

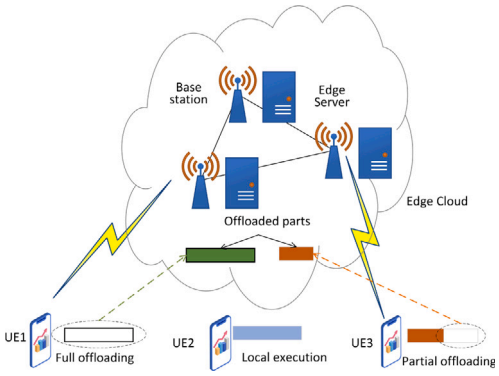

Fig. 4. Possible outcomes of offloading decision.

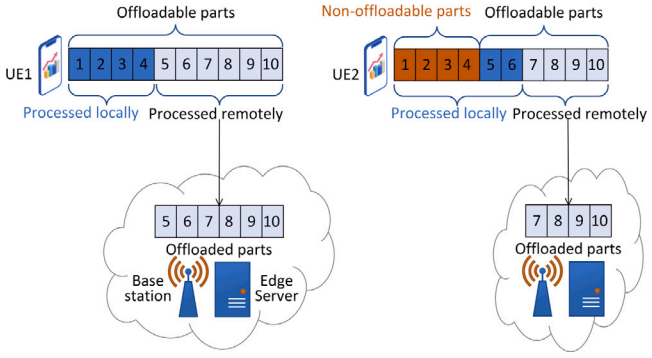

Fig. 5. Offloadability of tasks.

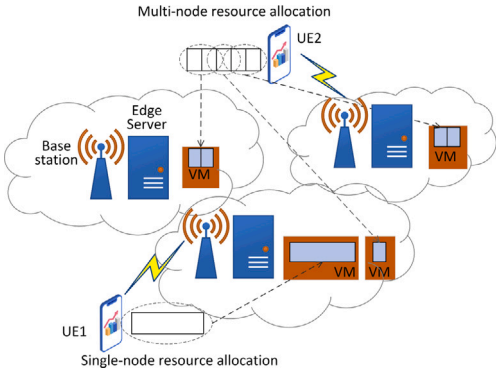

Fig. 6. Example of resource allocation.

libraries within edge servers, specifically to facilitate the execution of computational tasks on these servers [94]. Edge caching significantly mitigates latency, curtails energy expenditure, and decreases bandwidth costs associated with retrieving and initializing service applications during the offloading of computation tasks to edge servers [95]. However, due to the constrained storage and computing resources of edge servers, only a limited fraction of data and services can be cached at any given time. Therefore, which services and content are cached on which edge server is closely related to the offloading decision, and when an edge server caches the required services and content, it is preferred to offload tasks to that server for execution, which improves the performance of computation offloading [96].

From the above analysis, we can see that in the computation offloading process of MEC, offloading decision, resource allocation, and edge caching are closely related, so many recent researches jointly optimize them to improve the system performance. Numerous sophisticated algorithms have been developed for optimizing offloading decisions, resource allocation, and edge caching strategies, with the RL algorithm standing out due to its efficacy in these domains. In the ensuing discussion, we will delve into the RL algorithms frequently employed in the computation offloading process.

2.3. RL in MEC

The fundamental principle of RL is to test in an interactive environment using a created software agent that constantly interacts with the environment. Specifically, the agent determines the next course of action for the environment after assessing its current state. Upon executing a specified action, the environment provides feedback in the form of a reward to the agent, and through these interactions, the agent can decide on a better strategy. These features of RL make it well suited to handle complex and dynamic scenarios in MEC to effectively address the computation offloading problem. Its basic flow is shown in Fig. 7.

2.3.1. RL optimization techniques in MEC

(1) Knowledge distillation

Knowledge distillation [97,98] is an approach that significantly reduces the number of parameters and computational requirements of a model while maintaining high accuracy. It focuses on improving the performance of the student model by directing the lightweight student model to mimic the better-performing and more complex teacher model, as shown in Fig. 8. Due to its outstanding benefits, there are many studies that use it for model training in MEC systems, e.g., in [99], the authors proposed a method for efficiently transferring DRL-based control knowledge of edge devices in resource-constrained edge computing systems by utilizing knowledge distillation techniques to simultaneously perform knowledge transfer and policy model compression in a single training session on an edge device. [100] united RL and Supervised Learning (SL) to turn the teacher model into an edge-appropriate student model through an RL-based DNN knowledge distillation method, which determines the complexity of the data through an SL-based offloading strategy, where simple data is processed by the edge devices while complex data is processed by the cloud servers.

(2) Quantization

Quantization aims to reduce the number of bits required to store neural network weights [101,102]. Quantization can be categorized into post-training quantization and quantization-aware training [103], post-training quantization directly quantizes the trained model without complex fine-tuning, and quantization-aware training is to insert pseudo-quantization operators on the trained model to simulate the error generated by quantization. The fundamental goal of [104] is to find a mixed-accuracy quantization strategy associated with Deep Neural Network (DNN) partitioning to enhance privacy protection and reduce device energy consumption. [105] proposed a fixed-point quantized perception training scheme, which can halve the data precision after a certain training time without losing accuracy. However, due to the inconsistency between the computational complexity and the latency constraints of the associated problem, the structurally complex neural network cannot be realized in real-time operation.

(3) Pruning

Pruning [106] reduces the number of parameters and computations in the model without compromising its accuracy by eliminating unimportant weights, as shown in Fig. 9. [107] proposed a novel two-stage DRL framework based on the joint optimization of Convolutional Neural Network (CNN) using pruning and quantization. However, it does not consider the dynamic resource demands on edge devices. [108] proposed a DRL-based

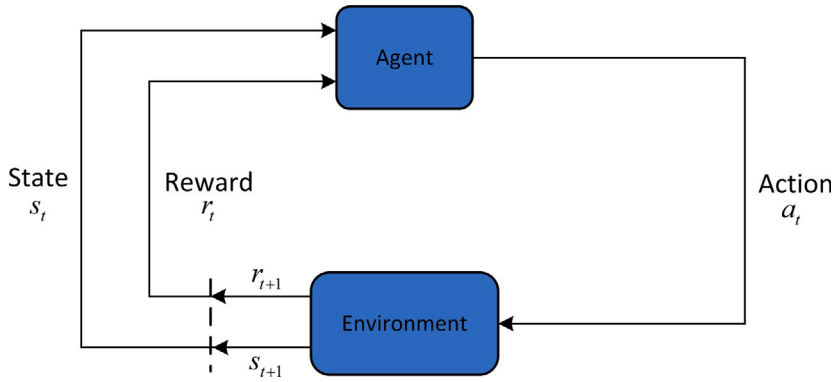

Fig. 7. Basic flow of RL.

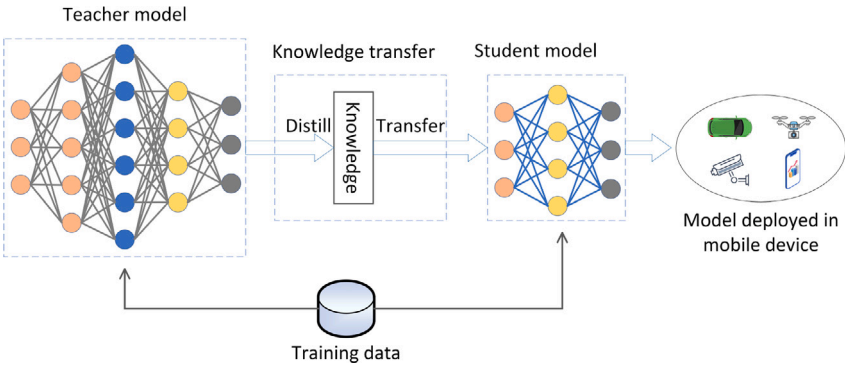

Fig. 8. Knowledge distillation.

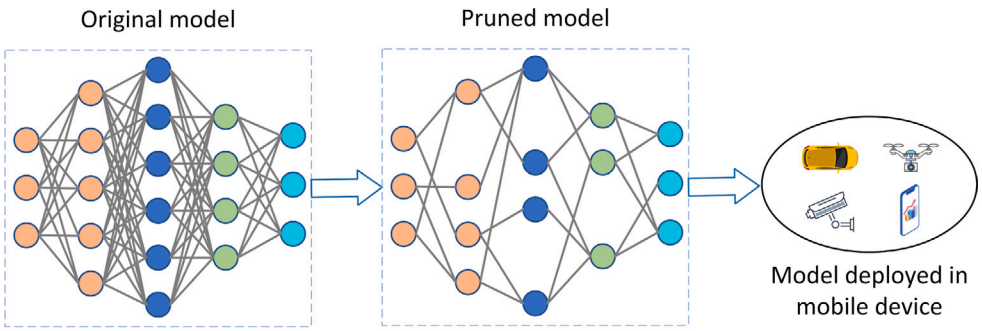

Fig. 9. Pruning.

auto-pruning scheme to solve the problem addressed by [107]. This scheme dynamically adjusts model compression based on energy availability, optimizing the trade-off between energy use and accuracy.

(4) Distributed RL

When training RL in MEC, the training task is distributed to multiple edge nodes to parallelize the learning process, which can effectively improve the training speed and efficiency. For example, [109] proposed a distributed multi-agent RL algorithm in which each agent adaptively learns the optimal policy. [110] extended the centralized RL approach to distributed RL by proposing a distributed RL algorithm with privacy protection.

2.3.2. *RL algorithms for computation offloading*

In RL, the action value function and the state value function are often used to assess how good or bad an agent is in a particular state under the action of a strategy π and how good or bad an action a is made while in state s. The collection of states within the environment is designated as the state space, denoted by S_t. In the computation offloading process, the state space S_t can be the available resources, transmission rate, and offloading decision, among others. The complete array of potential actions executable by the agent within a given environment constitutes the action space, represented as A_t. In the realm of computation offloading, this could include the offloading scheme, the allocation of computing resources to the task, and similar parameters, and is defined differently for different works. The state value function represents the anticipated payoff when the agent starts from state s and then acts according to policy π, defined as:

$$V_\pi(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s] \tag{13}$$

where r is the reward obtained for performing action a in state s. γ is a discount factor, meaning that it values near-term gains and weakens far-term gains. The expected payout of the agent beginning in state s, executing action a, and then adhering to the strategy is represented by

the action value function, defined as:

$$Q_\pi(s,a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, A_t = a] \tag{14}$$

The state value function can be articulated in relation to the action value function using the following expression:

$$\begin{aligned} V_\pi(s) &= E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s] \\ &= \sum_a \pi(a|s) E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, A_t = a] \\ &= \sum_a \pi(a|s) Q_\pi(s,a) \end{aligned} \tag{15}$$

The goal of the RL algorithm is to find the optimal policy, i.e., the policy that maximizes the state value function and the action value function [29], which can be expressed as:

$$V^*(s,a) = \max_\pi V_\pi(s) \tag{16}$$

$$Q^*(s,a) = \max_\pi Q_\pi(s,a) \tag{17}$$

Optimizing the computation offloading process is finding the optimal policy [111]. Several algorithms designed to identify the optimal policy in the computation offloading process are described below.

(1) Q-learning
Q-learning, a model-free, value-based approach, operates by calculating the Q-function for each state–action pair and subsequently updating the Q-values in the Q-Table following each interaction. Subsequently, the optimal offloading strategy is determined by choosing the action that possesses the highest Q-value in each respective state [70]. The formula for updating the Q value is as follows:

$$Q(s,a)^{new} = Q(s,a) + \partial \left[r + \gamma \max_a Q(s_t, a_t) - Q(s,a) \right] \tag{18}$$

where $\partial \in (0,1)$ is the learning rate, $Q(s,a)$ is the current Q value, is the Q value for the next state s_t of $Q(s_t, a_t)$, and the final $Q(s,a)^{new}$ obtained represents the best strategy.

(2) DQN
Deep Q Network (DQN) is a DRL algorithm formed by Q-learning combining it with a neural network, introducing a target network and experience playback. DQN transforms the Q-Table updating problem into a function-fitting problem, thus addressing the limitations inherent in Q-learning algorithms when applied to high-dimensional continuous problems [112]. In the DQN algorithm, the main network Q^m is used to predict the current Q value, and the target network Q^{tar} is used to find the target Q value, which can be expressed as:

$$t\arg et Q^{DQN} = r + \gamma \max_a Q^{tar}(s_t, a_t, \theta^{tar}) \tag{19}$$

Gradient descent can then be applied to minimize the discrepancy between its predicted value and the output generated by the target network (loss function):

$$loss(\theta) = E\left[\left(t\arg et Q^{DQN} - Q^m(s,a,\theta^m) \right)^2 \right] \tag{20}$$

where θ^{tar} is the target network parameter, and θ^m is the main network parameter. In the context of computation offloading, the DQN optimizes the neural network's parameters, achieved through minimizing the loss function. This optimization enhances the neural network's predictive accuracy, thereby enabling the agent to make more effective offloading decisions.

(3) Double DQN
The Double DQN is an algorithm designed to address the issue of Q-value overestimation inherent in the DQN. The fundamental distinction between these two algorithms resides in their respective methodologies for computing the target Q-value. From the target Q value of DQN as calculated above, we can see that DQN is directly selecting the largest Q value from the Q values of different actions in the target network. In contrast, Double DQN is obtaining the action $a^* = \arg\max_a Q^m(s_t, a_t, \theta^m)$ corresponding to the largest Q value through the main network and then calculating the target Q value in the target network by using the selected action. Therefore, the computation of the target Q-value of Double DQN can be expressed as:

$$t\arg et Q^{DDQN} = r + \gamma Q^{tar}(s_t, \arg\max_a Q^m(s_t, a_t, \theta^m), \theta^{tar}) \tag{21}$$

(4) AC
The Actor-Critic (AC) algorithm integrates the strengths of both policy-based and value-based methodologies, effectively combining their advantages [111]. The AC algorithm consists of a Critic network and an Actor network. The Actor network employs the Policy-Gradient method, which is responsible for generating actions. The Critic network can use the Q-value function described above [113], whose primary role is to assess the performance of the Actor and to guide the Actor's actions in the subsequent phase. Taking the method of [114] as an example, the authors take CPU frequency and channel gain as inputs to the Actor and output a slack offloading action a_t. To produce viable binary offloading decisions, the authors discretize a_t into B offloading actions. Subsequently, the efficacy of these offloading actions is assessed by a Critic, which selects the best-performing action as the output and uses that action to minimize the average cross-entropy loss function in updating the Actor parameters, which is expressed as:

$$loss(\theta_t) = -\frac{1}{\mathrm{T}_t} \sum_{\omega \in \mathrm{T}_t} ((a_\omega^*)^{\mathrm{T}} \log a_t + (1 - a_\omega^*)^{\mathrm{T}} \log(1 - a_t)) \tag{22}$$

where T_t is the set of selected times, a_ω^* is the optimal unloading action within that set, and the superscript T signifies the transpose operator.

(5) DDPG
The Deep Deterministic Policy Gradient (DDPG) algorithm is an improved algorithm for DQN, combining the idea of a deterministic policy gradient algorithm [115]. DDPG has four networks, namely Actor network $u(s|\theta^u)$, which is responsible for selecting the operation of calculating the offloading based on the state; Critic network $Q(s,a|\theta^Q)$, which is responsible for calculating the Q-value; and the target Actor network $u^t(s,a|\theta^{u^t})$ and the target Critic network $Q^t(s,a|\theta^{Q^t})$, which are used to evaluate the target Q-value [116]. Where θ^u, θ^Q, θ^{u^t}, and θ^{Q^t} are the network parameters.

In addition to the algorithms previously mentioned, some other RL algorithms are also employed in computation offloading, such as Dueling DQN, Synchronous Advantageous Actor-Critic (A2C), Asynchronous Advantageous Actor-Critic (A3C).

3. RL-based computation offloading in MEC

This section offers a comprehensive review of recent advancements in computation offloading employing RL in MEC, examined from three distinct perspectives: (1) RL-based methods for offloading decisions, (2) RL-based methods for joint resource allocation, (3) RL-based methods for joint edge caching.

3.1. RL-based methods for offloading decisions

Depending on the offloading method, we classify the methods in MEC that use RL to make offloading decisions into complete and partial offloading.

3.1.1. Complete offloading

The main objective of [117] is to address the unknown load dynamics of the edge nodes so that each UE can make offloading decisions without knowing the information of other UEs. To enhance the accuracy of the expected cost estimation in the proposed algorithm, the authors combined LSTM [118], dueling DQN [119], and Double DQN [120] techniques. The method proposed by the authors uses only local information for training the model, which might result in suboptimal convergence performance. This issue can be addressed by employing a hybrid of centralized and distributed approaches, as implemented in the study denoted by [121].

In order to swiftly arrive at near-optimal offloading decisions within wireless-powered (WP-MEC) networks, [63] proposed a DRL algorithm consisting of several key components, including the policy function, the exploration mechanism, and the policy update. In addition, the authors optimized the duration of wireless power transfer (WPT) to minimize the latency. Latency minimization for WP-MEC networks was also investigated in [122], where the wireless device in [63] offloads tasks to the access point (AP) via active transmission, while [122] uses both active transmission and backscatter communication.

The objectives of [123] are to achieve simultaneous optimization of geographic equity across all UEs, equitable distribution of UE load per UAV, and minimization of the energy consumption of the UEs. For this, the authors propose a multi-agent DDPG [124] to develop a solution. The proposed approach uses a framework based on centralized training and decentralized execution.

[125] A DRL-based computation offloading scheme is proposed to minimize energy consumption and latency to ascertain the most effective offloading decision. In the proposed scheme, the authors integrate the UAV with the ground BS to provide enhanced computation services. At the same time, different offloading methods are considered depending on the task's different latency and energy consumption requirements.

The interplay among various edge servers and the migration of services between them can potentially compromise the privacy integrity and security of the MEC system. This risk can be substantially mitigated through the implementation of blockchain technology within the MEC framework [126]. To this end, [127] integrates blockchain with MEC to reduce the system's offloading utility, achieving this by concurrently optimizing task offloading, privacy protection, and profit mining. However, the authors fail to consider the varying privacy requirements of distinct data sources, which may mean that the approach is unsuitable for all UE.

[81,127], and [84] all have the same goal. [81] established a QoS model and combined the QoS model with the DRL algorithm to derive the optimal offloading decision, using node state information and local link. This approach effectively addressed the issue posed by time-varying transmission channels. To address the privacy issue, the authors utilize FL for distributed training of multiple user data to improve model pervasiveness and protect data privacy. [84] formalized the computation offloading problem from the viewpoint of the privacy attacker with a priori knowledge and proposed a privacy-oriented computation offloading strategy using the DRL method, which has good generalization ability and convergence speed.

The objective of [128] is to achieve equilibrium between the energy expended in offloading and that in local computation, through the refinement of the offloading strategy. To achieve this, the authors propose a hybrid offloading model that utilizes active radio frequency (RF) communication and backscatter communication, similar to the approach of [122]. The authors view the wireless sensors as UEs that can transfer workloads and sensed data to a hybrid base station (HBS) situated at the same physical site as the edge server. The HBS allocates a time slot for offloading to each edge user, similar to the time-slot structure used in [129], but the two literatures adopt different approaches.

The method proposed in [130] combines the D3QN algorithm [131] and the C-D3QN algorithm. The method considers the offloading overhead, average latency, energy consumption, and task completion success rate of UEs, aiming to achieve the highest total revenue. [86] proposed an online predictive offloading algorithm by combining DRL with LSTM. During the model training phase, the algorithm utilizes LSTM to predict the load of edge servers, and this effectively increases the convergence speed and accuracy of the DRL algorithm in the context of the offloading procedure. During the offloading decision phase, the RL decision model receives prediction information from the LSTM network about the next task's characteristics. Based on this information, the RL decision model suggests an offloading scheme for the task that is predicted.

3.1.2. Partial offloading

[132] used an improved DDPG algorithm (NLDDPG) to iteratively train the model for optimal decision-making. NLDDPG is adapted from the DDPG algorithm [133], distinguishes itself primarily by its support for state normalization and enhanced extraction of task-specific features, functionalities that are not present in the standard DDPG algorithm. Relative to the algorithm presented in [133], the methodology proposed in [132] demonstrates superior stability and convergence characteristics. However, the method updates the network by randomly selecting a fixed number of samples from the empirical playback buffer during the training phase, disregarding the variability in sample significance and, at the same time, significantly affecting the final performance and training speed of the network.

Like [132], the authors of [134] also introduced LSTM in AC networks. The LSTM-based AC network architecture proposed in [134] is divided into three layers: representation network, Actor, and Critic network. The representation network contains an LSTM layer and a fully connected layer, tasked with discerning the temporal correlation among states.

In traditional AC, both the Actor and Critic networks predominantly consist of fully connected layers [135]. This architecture leads to an excessive number of parameters, thereby rendering the model training process inefficient. Furthermore, the Critic network is typically trained using one-step Temporal-Difference (TD) learning. This approach can result in a slower convergence rate for the Critic network [136], as described above in [132,134]. To address these aforementioned issues and the problem in [132] of disregarding the variability in sample significance. [89] designed an improved AC network (IACN) as well as a joint Adaptive n-step Learning (JMPA) and Prioritized Experience Replay (PER) mechanism, and finally integrated the IACN and JMPA through a distributed computation offloading algorithm based on multi-agent DRL to determine the optimal offloading approach.

The method proposed in [82] can solve the problems in [127]. Using the feature that local differential privacy mechanisms can generate different noise levels, [82] proposes a DRL method grounded in local differential privacy to fulfill various privacy requirements for subtask offloading decisions. The method is divided into three main phases: generation of personalized differential noise, exploration, and priority policy update.

Having the same goal as [86,87] decomposition of the computation offloading problem into two distinct problems: the lower-level problem of determining the offloading ratio and the higher-level problem of selecting appropriate offloading locations. Among them, the low-level problem and the high-level problem use continuous multi-agent PPO [137] (MAPPO) and discrete MAPPO to generate offloading ratio and server location selection for each task. While this approach simplifies the problem's complexity, it is less adaptable and requires manual decomposition of the problem.

Table 3
Comparison of studies on RL-based offloading decisions.

Ref.	Collaboration approach	RL methods	Performance metrics	Considerations
[117]	Terminal-Edge	Dueling DQN, Double DQN	Cost	Unknown load dynamics of edge nodes
[63]	Terminal-Edge	DRL-based	Latency	Wireless power transmission time
[123]	Terminal-UAV	DDPG	Energy consumption, Fairness index	Users covered by UAVs and assigned workloads
[125]	Terminal-UAV/Edge	Double DQN	Latency, Energy consumption	Collaboration between edge servers and Airborne servers
[127]	Terminal-Edge	Q-learning, DQN	Security and privacy, Latency, Energy consumption	Privacy protection during computation offloading
[81]	Terminal-Edge	Double DQN	Security and privacy, Latency, Energy consumption,	Dynamic changes in the channel, limited dataset for task modeling on a single device, privacy of local data
[84]	Terminal-Edge	PPO2	Security and privacy, Latency, Energy consumption	Protecting privacy when attackers have a priori knowledge
[128]	Terminal-Edge	RL-based	Energy consumption	Optimal transmission scheduling and workload distribution
[129]	Terminal-Edge	DDPG	Cost	Offloading decisions without a priori knowledge
[130]	Terminal-Edge	D3QN	Latency, Energy consumption, Task completion ratio, Offloading overhead	Dynamically adapting strategies in a dynamic environment,Slow convergence and poor exploration due to the Curse of Dimensionality
[86]	Terminal-Edge	DQN, Dueling DQN, Double DQN	Latency Energy consumption Task discard ratio	Reduce the cost of resources and increase the utilization of resource
[132]	Terminal-Edge	DDPG	Latency Energy consumption	Vehicle mobility, time-varying channels, and signal-blocking
[134]	Terminal-Edge-Cloud	AC	Energy consumption	Optimal allocation of computing between devices, edge, and cloud
[89]	Terminal-Terminal Terminal-Edge-Cloud	AC	Latency Energy consumption	Offloading decisions for fine-grained tasks
[82]	Terminal-Edge-Cloud	DRL-based	Security and privacy, Latency, Energy consumption, Task completion ratio	Privacy breaches on cloud platforms, different privacy requirements for different tasks
[87]	Terminal-Edge	MAPPO	Latency, Energy consumption, Task discard ratio	Offloading location and ratio

Note: Airborne servers refer to servers carried by UAVs.

3.1.3. Summary and analysis

Table 3 illustrates a summary of studies on RL-based offloading decisions in MEC. The most commonly used algorithms in these works are Double DQN and DDPG, which appear four times, but in [86,117], Double DQN is not the main algorithm studied but is used as part of the proposed algorithm. Furthermore, the most frequently employed performance metrics in the studies were latency and energy consumption, appearing eleven and thirteen times in the analyzed literature.

From the above works, we can see that the combination of multiple techniques can significantly improve the offloading decision algorithms, e.g., [86] obtained better performance than DQN, Dueling DQN, and Double DQN by combining Dueling DQN, Double DQN with LSTM; furthermore, considering the priority of tasks can significantly reduce the task discard rate and improve user QoE, this is because during the offloading of tasks to the edge servers, tasks may need to be queued for transmission and execution due to the shortage of communication resources as well as computing resources, and for latency-sensitive tasks, this may result in situations such as task loss.

3.2. RL-based methods for joint resource allocation

RL-based methods for offloading decisions exclusively consider how tasks offload to edge servers or execute locally. In particular, some works [138] proposed delegating tasks to MEC servers that maintain a direct connection with the device. The task will be routed to a remote cloud server if the MEC server to which the device is directly attached is not equipped with the required resources. This incurs a higher cost and ignores the utilization of resources available at nearby MEC servers [139]. For this reason, many works consider resource allocation jointly in the computation offloading processes to fully utilize the resources available at nearby MEC servers. In this section, we survey joint resource allocation methods, categorized into joint and segmented decisions depending on whether the methods for solving offloading decisions and resource allocation are solved as a whole or decomposed into different subproblems for solving the problem.

3.2.1. Joint decisions

To effectively tackle the issues related to limited communication resources in task transmission, [67] explores methods to reduce system latency within the limitations of bandwidth costs associated with task transmission, employing a decentralized, multi-agent algorithm for joint resource allocation. Within the limitations of computing resources and transmission bandwidth, the offloading of tasks to edge servers requires consideration of both the latency incurred during task transmission and the waiting time for the release of computing resources by ongoing tasks, which is not considered in [67], although the transmission bandwidth cost is taken into account. Taking the above problem as a research direction, [140] proposed a DRL-based joint resource allocation algorithm by considering the different priority levels of the tasks and the waiting time of the tasks in the communication and computation queues, to minimize the latency.

In order to cope with the challenges posed by dynamic environments to MEC, [70,141], and [68] take full advantage of the fact that DRL can be adapted by real-time adjustment of policies to cope with changes in the environment. [141] designed a joint resource allocation algorithm, which can efficiently obtain the optimal resource allocation and offloading decision to minimize the bandwidth leasing cost and computation cost under latency and transmission rate constraints. [70] explored a dynamic time-varying system to reduce energy consumption, taking into account the variable resource demands and latency constraints associated with diverse computation tasks. [68] proposed a DQN algorithm utilizing multi-branch networks to address the issue where a substantial increase in the number of UEs results in a combinatorial rise in the potential action space, which complicates

the learning process for the algorithm. The algorithm aims to use a multi-branch network so that the size of the output of the network increases linearly with the increase in the number of UEs instead of combinatorially; thus, the number of the outputs changes from n^N to $n \times N$ and avoids the problem of the explosion of the space of the actions causing the problem of difficult learning of the algorithm.

To optimize energy efficiency while adhering to latency constraints and resource limitations in IoT devices, [142] combined FL with DDPG and proposed a two-timescale federated DRL algorithm. In this, the small timescale is mainly responsible for training the resource allocation and offloading decisions on the edge server, and the large timescale is responsible for aggregating the trained model parameters. The small and large timescales are then executed alternately for better training performance. [88] proposed a transformer-based and policy-decoupled decentralized task offloading approach for multi-agent AC. The authors addressed the uncertain load problem by introducing a prediction network for predicting the queuing latency of edge servers. [143] proposed a multi-agent DRL-based joint resource allocation algorithm for small cell networks to minimize the overall energy consumption while meeting the computation task latency constraint. However, in the proposed method, each agent is restricted to local observation of the environment, lacking awareness of the resource allocation across the entire system. This constraint substantially impedes the efficacy of decision-making processes.

In [139,144], the authors describe the optimization problem of joint resource allocation in MEC systems as an MDP and then solve the MDP by a DDPG-based algorithm. [144] ignoring differences in the significance of the samples, which is avoided by the work in [139]. The methodology used in [139] is similar to that of [89], where the authors utilize the PER-assisted DDPG method to select metrics with absolute TD errors to describe the values of each sample accurately. To mitigate the risk of overfitting, a stochastic prioritization technique is adopted for empirical sample selection, and importance-sampling weights [145] were used to correct the state access bias from prioritized sampling.

[83,146,147] utilize different approaches to address privacy security in computation offloading. [83] proposes a DRL-based joint resource allocation scheme utilizing spectrum-sharing architecture and physical layer security techniques. Unlike the static eavesdropping scenario in [148], where only a single eavesdropper is considered, in [83], the authors consider a multi-eavesdropper scenario closer to reality. [146] is to enhance the reliability of data within the MEC system through the implementation of blockchain technology. In addition, the authors developed a joint resource allocation framework for blockchain-enabled MEC systems and have designed an algorithm based on the A3C. While [147] proposed a hierarchical joint resource allocation optimization algorithm based on the multi-agent DDPG algorithm and then used an FL algorithm to protect privacy and security. This approach involves exclusively sharing the trained model parameters, thereby eliminating the need for direct interaction with the local data, which protects privacy and at the same time achieves the centralized training effect.

3.2.2. Segmented decisions

[149,150] aim to maximize the computation rate of the system. [149] proposed a DRL-based computation offloading algorithm in UAV-assisted MEC, which, after solving the offloading decision of the user device using DRL, transforms the problem into a convex optimization problem to determine the resource allocation at each time frame. [150] proposed a joint resource allocation algorithm that can be quickly adapted under the changing wireless channel conditions. This method can speed up the computation rate. However, to optimize the computation rate, the strategy involves utilizing the entirety of the available energy for task execution in both offloaded and local computation modes. While effective for computational performance, this approach may result in significantly elevated energy consumption.

Both [72,151] investigated a two-tier optimization framework for collaborative computing to reduce the total energy consumption by jointly optimizing the offloading decision, the collaborative decision, the subcarrier allocation, the power allocation, and the computing resource allocation. [151] is a collaboration between edge and cloud, where the upper layer addresses the subproblems of offloading decisions, computation collaboration decisions, as well as power and subcarrier allocation, and the lower layer is dedicated to solving the subproblem of computing resource allocation. Unlike [72,151] is a collaboration between edge servers and mobile devices, where collaboration decisions and offloading decisions are solved at the upper layer, and the allocation of computing, subcarriers, and power resources are solved at the lower layer.

[64] proposed a two-stage distributed algorithm PG-MRL. During the initial stage, offloading decisions are ascertained at the beginning of each time slot utilizing a potential game approach. Subsequently, in the second stage, the strategy for resource allocation is formulated contingent upon the previously derived offloading decisions.

[114,152] aim to minimize system energy consumption and latency. [152] divided the joint resource allocation problem into two subproblems and used a DRL-based algorithm to address the offloading decision problem and the salp swarm algorithm [153] to address the resource allocation problem. In [114], an AC-based DRL algorithm is proposed to address the joint resource allocation and the strong coupling of interdependent tasks to optimize offloading decisions and resource allocation.

3.2.3. Summary and analysis

Table 4 illustrates an overview of RL-based joint resource allocation research within the context of MEC. Most of these works use latency and energy consumption as performance metrics and find the best decision by formulating the optimization problem as an MDP and then solving the MDP using the corresponding RL method. Both partial offloading and complete offloading are considered in [141,150], where the decision of offloading the UE to the edge server is complete offloading, while the decision of the edge server to collaborate with the rest of the servers based on the remaining available resources is partial offloading.

After a deeper survey of the work on joint resource allocation in MEC, it is evident that the performance of RL-based joint resource allocation algorithms tends to outperform that of algorithms that only consider offloading decisions due to the fact that when resource allocation is also considered, the computational task avoids the time spent waiting for the ongoing task to release the resources and the risk of the task being redirected to a remote cloud server. Secondly, centralized training tends to produce better results, this is because centralized training can train the network using the global information to get a more desirable performance. However, centralized training is prone to privacy leakage, a non-negligible problem, as in work done in [147]. The above research content also exposes the problem that most of the existing work is based on experiments conducted in simulated environments, and very little work has verified the effectiveness of the proposed strategies in real systems, which is considered in the work of [154].

3.3. RL-based methods for joint edge caching

Joint resource allocation can allow offloading tasks to use resources fully in resource-constrained environments. Nevertheless, it is implicitly assumed in these studies that edge servers can handle various tasks, which is difficult to achieve in practice. If the edge server does not cache the required services, the task cannot be executed even if sufficient computing resources are available at that time [155]. In addition, when the edge server caches reusable content, it can reduce repetitive execution and thus improve QoS [156]. Therefore, many recent works have investigated joint edge caching in MEC systems. The joint service caching and computation offloading model is shown in Fig. 10.

Table 4
Comparison of studies on RL-based joint resource allocation.

Ref.	Offloading mode	Resources considered	RL methods	Performance metrics	System model
[67]	Partial offloading	Bandwidth	Double DQN	Latency Cost	Cloud Multiple servers Multiple users
[140]	Complete offloading	Computing resource	A2C	Latency	Multiple servers Multiple users
[141]	Partial offloading	Bandwidth	TD3	Latency Cost	Cloud Multiple servers Multiple users
[70]	Complete offloading	Computing resource Spectrum resource	Q-learning Double DQN	Energy consumption	Single server Multiple users
[68]	Complete offloading	Computing resource	DQN	Latency Task discard ratio	Multiple servers Multiple users
[142]	Complete offloading	Bandwidth Computing resource	DDPG	Energy consumption	Multiple servers Multiple users
[88]	Complete offloading	Power Computing resource	AC	Cost Task completion ratio	Cloud Multiple servers Multiple users
Ref.	Collaboration approach	RL methods	Performance metrics	System model	Considerations
[143]	Partial offloading	Channel Power Computing resource	DDPG	Energy consumption	Multiple servers Multiple users
[144]	Partial offloading	Channel Computing resource	DDPG	Latency Energy consumption	Multiple servers Multiple users
[139]	Complete offloading	Bandwidth Power Computing resource Storage resource	DDPG	Latency Energy consumption	Multiple servers Multiple users
[83]	Complete offloading	Power Computing resource Spectrum resource	Double DQN	Security and privacy Latency	Single server Multiple users
[146]	Complete offloading	Power	A3C	Security and privacy Computation rate	Multiple servers Multiple users
[147]	Complete offloading	Computing resource Spectrum resource Storage resource	DDPG	Security and privacy Task completion ratio	Cloud Multiple servers Multiple users
[149]	Complete offloading	Power time frame	DRL-based	Computation rate	Single server Multiple users
[150]	Complete offloading	Time frame	DRL-based	Computation rate	Single server Multiple users
[151]	Partial offloading	Power Subcarrier Computing resource	Double DQN Dueling DQN	Energy consumption	Cloud Multiple servers Multiple users
[72]	Complete offloading	Power Subcarrier Computing resource	DQN	Energy consumption	Single server Multiple users
[64]	Complete offloading	Power Channel	DDPG	Latency	Multiple servers Multiple users
[152]	Complete offloading	Bandwidth Computing resource	DRL-based	Cost, Latency Energy consumption	Cloud Multiple servers Multiple users
[114]	Complete offloading	CPU frequency	AC	Latency Energy consumption	Single AP Single user

3.3.1. Joint service caching

[65,157,158] all aim at minimizing latency. [65,157] consider a system model containing a cloud–edge-end, wherein computation tasks have the option to be executed locally, edge, or cloud, and categorize the task offloading scenarios into six and three, respectively, based on the caching of the service. In [65] of the designed framework for joint service caching with cloud, edge servers, and vehicles collaborating, to minimize the latency, the authors propose a DDPG-based algorithm to make offloading decisions and caching decisions efficiently. In [157], the authors initially developed a virtual queuing model and decoupled the issue utilizing Lyapunov optimization methods, converting the issue into a single timeslot optimization issue. Then, they used the AC algorithm to find the best offloading and caching decisions for each timeslot. To ensure the trustworthiness of the system while reducing the latency, [158] introduced blockchain technology into the MEC system and proposed a joint service caching scheme. The authors divide the optimization problem into two subproblems and iteratively optimize the offloading and caching decisions by the DRL algorithm and greedy algorithm. While the combination of greedy algorithms and DRL can keep offloading and caching decisions optimal over time, given that caching decisions need to be based on statistical data about the state of the network over time and that the popularity of services changes over time, these algorithms have to be executed periodically to accommodate these changes.

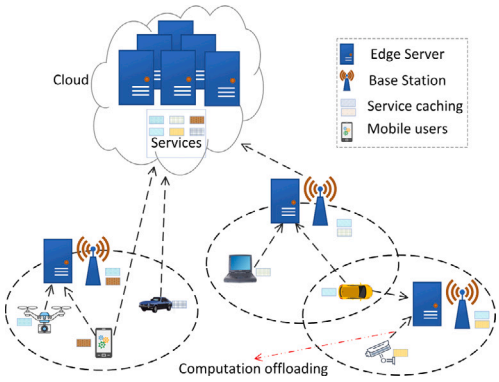

Fig. 10. Joint service caching and computation offloading model in MEC.

To minimize latency and cost within the system, [159,160] used A3C and DQN algorithms, respectively, to solve the problem. [159] proposed a DRL-based joint service caching scheme to optimize the policies with service caching, offloading decisions, and resource allocation. In their approach, the authors present the optimization challenge as a Mixed Integer Non-Linear Programming (MINLP) problem and introduce an A3C-based algorithm as a solution to this problem. [160] introduces a joint service caching scheme from a vehicular perspective, framing the minimization of system computation cost and latency as a MINLP problem and proposing an iterative algorithm that amalgamates Gibbs sampling with DQN techniques.

[161] used a Soft AC-based approach to concurrently determine continuous resource allocation decisions and discrete offloading and service caching decisions, transforming discrete action selection into a continuous space to cope with the challenges posed by continuous-discrete hybrid action spaces. [162] considers the collaboration between edge servers and proposes a DDPG-based algorithm for service caching, resource allocation, and computation offloading policy determination to minimize the system energy consumption.

To minimize the weighted sum of energy consumption and latency, [163] solved the problem by jointly optimizing service caching, resource allocation, and computation offloading and solved it by a Double DQN-based SCRACO algorithm. The algorithm dynamically matches competing resource allocations and individual offloading decisions among UEs while adapting to heterogeneous storage capacities and user requests of edge servers.

Both [164,165] utilize dual-timescale to solve the problem. [164] formulated the problem of joint service caching as a dual-timescale resource allocation problem and proposed a hierarchical DRL-based approach. On the large timescale, the server determines which services to cache, and on the small timescale, the IoT device selects the location for task execution. Unlike traditional approaches that use the total system overhead as a reward, this strategy provides intrinsic rewards based on whether or not the offloading task utilizes the caching service. The authors of [165] output latency-sensitive resource allocation decisions and application partitioning on fast-timescale and latency-insensitive service caching decisions on slow-timescale based on the fact that service caching, resource allocation, and application partitioning have different latency-sensitive. To safeguard the data privacy of UEs, implemented a distributed FL-based approach for model training to train the DRL agent.

3.3.2. Joint content caching

In [156], the authors propose a content popularity prediction algorithm OSTP and then utilize a DDPG-based algorithm to jointly optimize offloading and caching decisions and resource allocation policies to minimize the system latency. Inspired by blockchain incentive, A2C, and FL ideas, [166] proposed a BICC-FDRL algorithm, which mainly consists of two parts: contribution-based federated aggregation and blockchain-assisted local training.

[167] proposed a cache-assisted NOMA MEC framework. The authors first predict the task popularity using LSTM and then formulate a multi-agent Q-learning method based on Bayesian Learning Automata based on task popularity. The proposed algorithm does not need to solve complex joint optimization problems but learns from past experiences and automatically improves the assignment strategy. [168] addresses the problem of joint content caching in an ultra-dense network scenario, which the authors decompose to ask two subproblems and model them separately.

[169,170] aim at minimizing system energy consumption and latency. In the methodology presented in [169], the DDPG algorithm is initially applied for orchestrating offloading decisions for users within a solitary region without integrating caching strategies. Subsequently, as the system expands to encompass multiple regions, a collaborative caching algorithm is employed across these regions to enhance the system's overall cache hit rate. Ultimately, these two algorithms are amalgamated to generate conclusive offload decisions for all users. [170] formalized the cache-assisted resource allocation and collaborative task offloading problem as a QoE-perceived utility maximization problem based on the preference and execution cost at the user's end, as well as the task allocation state, task caching state, resources, and network state at the MEC end and decomposed the problem into two sub-problems, which are solved using different approaches.

3.3.3. Summary and analysis

Table 5 illustrates a summary of studies on RL-based joint edge caching in MEC. The vast majority of these efforts are aimed at minimizing latency, with little regard for other performance. The most commonly used algorithm continues to be DDPG, as it combines the strengths of several algorithms to improve the exploration and decision-making capabilities of the system greatly.

Through the above works, we can easily find that collaboratively considering the caching of nearby servers and devices when performing edge caching can effectively reduce the system cost. This is because when a nearby server or device caches the desired service or content, the task can be offloaded to the neighborhood for execution without having to offload to the cloud or re-cache, reducing the system overhead.

4. Challenges and future work

RL-based computation offloading is a crucial technique to achieve less system overhead and improve user QoE in MEC. Based on the research and analysis of the above work, there are still some challenges and pending issues for computation offloading in MEC.

(1) Fault tolerance

In the body of literature surveyed, none of the studies consider the system's fault tolerance. What would be the impact on the task if the system fails, and whether the proposed method is still applicable? Considering the fault tolerance of the system, we need to consider several issues.

If the server fails during task processing, how should the pending tasks be resolved? Can pending tasks tolerate the consequences of a server failure?

What will impact the system if the agent of RL fails during the decision-making process? If one of the agents fails in a multi-agent scenario, can the rest of the agents work normally? Or the rest of the agents can take over the work of the failed agent? As far as we know, the existing work has not studied how the system should regulate its work to continue service in case of system failure, so this issue has yet to be resolved.

Table 5
Comparison of studies on RL-based joint edge caching.

Ref.	Offloading mode	RL methods	Performance metrics	System model
[65]	Partial offloading	DDPG	Latency	Cloud Multiple servers Multiple users
[157]	Complete offloading	AC	Latency	Cloud Multiple servers Multiple users
[158]	Complete offloading	DRL-based	Latency	Multiple servers Multiple users
[159]	Complete offloading	A3C	Cost Latency	Single macro BS (MBS) Multiple servers Multiple users
[160]	Complete offloading	DQN	Cost Latency	Single MBS Multiple servers Multiple users
[161]	Complete offloading	SAC	Latency	Cloud Single server Multiple users
[162]	Partial offloading	DDPG	Energy consumption	Cloud Multiple servers Multiple users
[163]	Partial offloading	Double DQN	Latency Energy consumption	Cloud Multiple servers Multiple users
[164]	Complete offloading	DRL-based	Latency Energy consumption	Single server Multiple users
[165]	Partial offloading	DQN	Latency	Cloud Multiple servers Multiple users
[156]	Complete offloading	DDPG	Latency	Multiple servers Multiple users
[166]	Complete offloading	A2C	Cost	Cloud Multiple servers Multiple users
[167]	Complete offloading	Q-learning	Energy consumption	Single AP Multiple users
[168]	Complete offloading	DQN	Convergence	Single MBS Multiple small BS Multiple users
[169]	Complete offloading	DDPG	Latency Energy consumption	Multiple servers Multiple users
[170]	Complete offloading	RL-based	Latency Energy consumption	Cloud Multiple servers Multiple users

(2) Differences in empirical samples
When utilizing the RL decision offloading scheme, most of the work draws a small batch of samples randomly from the empirical pool for training. It ignores the difference in the importance of the samples, which can lead to inaccurate estimation of future rewards and thus is prone to large bias and low efficiency in the early stage of training. In this regard, one existing idea is to use the absolute time difference error to rank the importance of the samples, i.e., the larger the absolute time difference error, the higher the importance of the samples, so that the samples with high importance can be prioritized at the time of sampling, which improves the learning efficiency and performance [89]. In addition, uncertainty estimation methods can be used to assign weights to the samples, e.g., Bayesian [171] and Bootstrap [172] methods can be used to evaluate the uncertainty of each sample and prioritize the high-uncertainty samples during training to reduce the bias in the early stages of training.

(3) Blockchain and RL integration for computation offloading in MEC
Blockchain technology has been widely used, and in MEC, blockchain is commonly used to address privacy and security issues during computation offloading, such as [127,146, 166] mentioned above. Using the RL algorithm for centralized training during computation offloading requires global information to train the network, which results in more desirable performance. However, centralized training is prone to privacy leakage. Therefore, integrating blockchain technology with RL and using the consensus mechanism of blockchain technology ensures that each node reaches an agreement during the training process, avoids training bias caused by data inconsistency, and achieves decentralized decision-making by recording and managing the training and decision-making process of the agent through blockchain [173]. In addition, the blockchain can be used as a decentralized data-sharing platform, where the data to be shared are encrypted and stored on the blockchain. When performing RL training, data access control is performed using smart contracts to ensure that only authorized edge nodes and devices can access and use the data.

(4) Curse of dimensionality
The complexity and dynamics of the computation offloading environment make the state–action space of the RL agent too large, which results in a curse of dimensionality and makes the algorithms slow to converge and poor to explore. In this regard, the state space can be processed using state preprocessing [65]

and state space compression [174], or the action space can be processed using quantization to generate actions, which reduces the range of choices of actions by discretizing the continuous action space [175], and pre-classification can be used to limit the space by categorizing both the state space and the action space to limit the size of the space and make the learning process more focused on certain specific regions. In addition, can be used hierarchical learning to decompose a complex task into multiple subtasks [176] as a way to simplify the state space and action space of each subtask.

(5) Discrete and continuous control variables in the offloading process
The computation offloading process in MEC usually involves both discrete and continuous control variables, such as in the offloading decision process, if a computational task is to be offloaded in a partial offloading manner, this may require a decision on the task offloading ratio as well as server selection at the same time, which makes the decision-making process very difficult. Some existing works provide a staged or layered RL approach [64,152] to solve the problem, where different methods are used at different layers or stages to solve for both discrete and continuous variables. There is also an approach to convert the discrete into a continuous space, which is studied in [161].

(6) Evaluation of incentives in the offloading process
Agent for RL needs to be driven by real-time reward evaluation. Since the performance of MEC offloading is usually observed only after completing the workload, this makes the algorithm converge slowly and difficult to adapt quickly in the dynamically changing MEC environment. In addition, evaluating the goodness of an offloading decision often requires weighing multiple metrics, which increases the complexity of reward design and may lead to unclear or conflicting learning objectives, affecting the training effectiveness of RL. Therefore, to forecast rewards in the presence of imperfect network information, we need a more effective method that combines learning with model-based optimization [128].

5. Conclusion

In MEC, computation offloading is not a single problem but often involves both resource allocation and edge caching. In this paper, we provide a comprehensive survey of RL-based computation offloading methods in MEC, including optimal offloading decisions, joint resource allocation, and joint edge caching, as well as a classified comparison of the various approaches. From the analysis, we know that the most used performance metrics in computation offloading are latency and energy consumption, while the most used algorithms are DDPG and DQN and its variants. Finally, we explore the challenges of RL-based computation offloading in MEC and the problems to be solved from several perspectives.

CRediT authorship contribution statement

Zhongqiang Luo: Conceptualization, Funding acquisition, Investigation, Project administration, Resources, Supervision, Writing – original draft, Writing – review & editing. **Xiang Dai:** Data curation, Formal analysis, Funding acquisition, Methodology, Resources, Writing – original draft, Writing – review & editing.

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.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61801319, in part by the Sichuan Science and Technology Program under Grant 2020JDJQ0061 and 2021YFG0099, in part by the Opening Project of Artificial Intelligence Key Laboratory of Sichuan Province under Grant 2021RZJ01, in part by the Scientific Research and Innovation Team Program of Sichuan University of Science and Engineering under Grant SUSE652A011, in part by the Postgraduate Innovation Fund Project of Sichuan University of Science and Engineering under Grant Y2023280.

References

[1] N. Vipond, A. Kumar, J. James, F. Paige, R. Sarlo, Z. Xie, Real-time processing and visualization for smart infrastructure data, Autom. Constr. 154 (2023) 104998.
[2] M. Wu, F.R. Yu, P.X. Liu, Intelligence networking for autonomous driving in beyond 5 g networks with multi-access edge computing, IEEE Trans. Veh. Technol. 71 (6) (2022) 5853–5866.
[3] J. Srivastava, S. Routray, S. Ahmad, M.M. Waris, Internet of medical things (iomt)-based smart healthcare system: Trends and progress, Comput. Intell. Neurosci. 2022 (2022).
[4] S.M. LaValle, Virtual Reality, Cambridge University Press, 2023.
[5] F. Arena, M. Collotta, G. Pau, F. Termine, An overview of augmented reality, Computers 11 (2) (2022) 28.
[6] P.K. Malik, R. Sharma, R. Singh, A. Gehlot, S.C. Satapathy, W.S. Alnumay, D. Pelusi, U. Ghosh, J. Nayak, Industrial internet of things and its applications in industry 4.0: State of the art, Comput. Commun. 166 (2021) 125–139.
[7] 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.
[8] E. Mustafa, J. Shuja, S.K. uz Zaman, A.I. Jehangiri, S. Din, F. Rehman, S. Mustafa, T. Maqsood, A.N. Khan, Joint wireless power transfer and task offloading in mobile edge computing: a survey, Cluster Comput. 25 (4) (2022) 2429–2448.
[9] X. Liu, X. Zhao, G. Liu, F. Huang, T. Huang, Y. Wu, Collaborative task offloading and service caching strategy for mobile edge computing, Sensors 22 (18) (2022) 6760.
[10] Y. Jia, C. Zhang, Y. Huang, W. Zhang, Lyapunov optimization based mobile edge computing for internet of vehicles systems, IEEE Trans. Commun. 70 (11) (2022) 7418–7433.
[11] K. Tan, M. Feng, Lei, Decentralized convex optimization for joint task offloading and resource allocation of vehicular edge computing systems, IEEE Trans. Veh. Technol. 71 (12) (2022) 13226–13241.
[12] X. Xu, Q. Jiang, P. Zhang, X. Cao, M.R. Khosravi, L.T. Alex, L. Qi, W. Dou, Game theory for distributed iov task offloading with fuzzy neural network in edge computing, IEEE Trans. Fuzzy Syst. 30 (11) (2022) 4593–4604.
[13] X. Xu, D. Li, Z. Dai, S. Li, X. Chen, A heuristic offloading method for deep learning edge services in 5 g networks, IEEE Access 7 (2019) 67734–67744.
[14] 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.
[15] X. Yao, N. Chen, X. Yuan, P. Ou, Performance optimization of serverless edge computing function offloading based on deep reinforcement learning, Future Gener. Comput. Syst. 139 (2023) 74–86.
[16] L. Lin, X. Liao, H. Jin, P. Li, Computation offloading toward edge computing, Proc. IEEE 107 (8) (2019) 1584–1607.
[17] B. Cao, L. Zhang, Y. Li, D. Feng, W. Cao, Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework, IEEE Commun. Mag. 57 (3) (2019) 56–62.
[18] H. Jin, M.A. Gregory, S. Li, A review of intelligent computation offloading in multi-access edge computing, IEEE Access (2022).
[19] W. Li, H. Hacid, E. Almazrouei, M. Debbah, A comprehensive review and a taxonomy of edge machine learning: Requirements, paradigms, and techniques, Artif. Intell. 4 (3) (2023) 729–786.
[20] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutorials 19 (3) (2017) 1628–1656.
[21] C. Jiang, X. Cheng, H. Gao, X. Zhou, J. Wan, Toward computation offloading in edge computing: A survey, IEEE Access 7 (2019) 131543–131558.
[22] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective, Comput. Netw. 182 (2020) 107496.
[23] H. Lin, S. Zeadally, Z. Chen, H. Labiod, L. Wang, A survey on computation offloading modeling for edge computing, J. Netw. Comput. Appl. 169 (2020) 102781.

[24] A. Islam, A. Debnath, M. Ghose, S. Chakraborty, A survey on task offloading in multi-access edge computing, J. Syst. Archit. 118 (2021) 102225.
[25] S.A. Huda, S. Moh, Survey on computation offloading in UAV-enabled mobile edge computing, J. Netw. Comput. Appl. 201 (2022) 103341.
[26] 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. (2022).
[27] T. Li, X. He, S. Jiang, J. Liu, A survey of privacy-preserving offloading methods in mobile-edge computing, J. Netw. Comput. Appl. 203 (2022) 103395.
[28] A. Acheampong, Y. Zhang, X. Xu, D.A. Kumah, A review of the current task offloading algorithms, strategies and approach in edge computing systems, CMES Comput. Model. Eng. Sci. 134 (1) (2023) 35–88.
[29] 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, et al., A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems, J. Netw. Comput. Appl. 216 (2023) 103669.
[30] J. Lv, J. Zhang, Z. Zhang, C. Gan, Survey of mobile edge computing offloading strategies, J. Chin. Comput. Syst. 41 (9) (2020) 1866–1877.
[31] Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing—a key technology towards 5 g, ETSI White Pap. 11 (11) (2015) 1–16.
[32] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, D. Sabella, On multi-access edge computing: A survey of the emerging 5 g network edge cloud architecture and orchestration, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1657–1681.
[33] W. Shi, X. Zhang, Y. Wang, Q. Zhang, Edge computing: state-of-the-art and future directions, J. Comput. Res. Dev. 56 (1) (2019) 69–89.
[34] M. Hussain, N. Shah, R. Amin, S.S. Alshamrani, A. Alotaibi, S.M. Raza, Software-defined networking: Categories, analysis, and future directions, Sensors 22 (15) (2022) 5551.
[35] K. Gaur, J. Grover, Exploring vanet using edge computing and sdn, in: 2019 Second International Conference on Advanced Computational and Communication Paradigms, ICACCP, IEEE, 2019, pp. 1–4.
[36] C. Tang, C. Zhu, N. Zhang, M. Guizani, J.J. Rodrigues, Sdn-assisted mobile edge computing for collaborative computation offloading in industrial internet of things, IEEE Internet Things J. 9 (23) (2022) 24253–24263.
[37] Z. Latif, C. Lee, K. Sharif, S. Helal, Sdblockedge: Sdn-blockchain enabled multihop task offloading in collaborative edge computing, IEEE Sens. J. 22 (15) (2022) 15537–15548.
[38] T. Zhang, X. Zhu, C. Wu, Reinforcement-learning-based software-defined edge task allocation algorithm, Electronics 12 (3) (2023) 773.
[39] I. Cerrato, A. Palesandro, F. Risso, M. Suñé, V. Vercellone, H. Woesner, Toward dynamic virtualized network services in telecom operator networks, Comput. Netw. 92 (2015) 380–395.
[40] B. Yi, X. Wang, K. Li, M. Huang, et al., A comprehensive survey of network function virtualization, Comput. Netw. 133 (2018) 212–262.
[41] L.A. Haibeh, M.C. Yagoub, A. Jarray, A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches, IEEE Access 10 (2022) 27591–27610.
[42] X. Gao, R. Liu, A. Kaushik, Virtual network function placement in satellite edge computing with a potential game approach, IEEE Trans. Netw. Serv. Manag. 19 (2) (2022) 1243–1259.
[43] M. Liu, S.B. Alias, Cost-efficient virtual network function placement in an industrial edge system: A proposed method, IEEE Syst. Man Cybern. Mag. 9 (1) (2023) 10–17.
[44] J. Su, S. Nair, L. Popokh, Edgegym: A reinforcement learning environment for constraint-aware nfv resource allocation, in: 2023 IEEE 2nd International Conference on AI in Cybersecurity, ICAIC, IEEE, 2023, pp. 1–7.
[45] S. Garg, K. Kaur, G. Kaddoum, S. Guo, Sdn-nfv-aided edge-cloud interplay for 5g-envisioned energy internet ecosystem, IEEE Netw. 35 (1) (2021) 356–364.
[46] R. Riggio, S.N. Khan, T. Subramanya, I.G.B. Yahia, D. Lopez, Lightmano: Converging nfv and sdn at the edges of the network, in: NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2018, pp. 1–9.
[47] K. Wang, C.-M. Chen, M.S. Hossain, G. Muhammad, S. Kumar, S. Kumari, Transfer reinforcement learning-based road object detection in next generation iot domain, Comput. Netw. 193 (2021) 108078.
[48] H. Hu, H. Shan, C. Wang, T. Sun, X. Zhen, K. Yang, L. Yu, Z. Zhang, T.Q. Quek, Video surveillance on mobile edge networks—a reinforcement-learning-based approach, IEEE Internet Things J. 7 (6) (2020) 4746–4760.
[49] K. Yan, H. Shan, T. Sun, H. Hu, Y. Wu, L. Yu, Z. Zhang, T.Q. Quek, Reinforcement learning-based mobile edge computing and transmission scheduling for video surveillance, IEEE Trans. Emerg. Top. Comput. 10 (2) (2021) 1142–1156.
[50] Q. Liu, L. Shi, L. Sun, J. Li, M. Ding, F. Shu, Path planning for UAV-mounted mobile edge computing with deep reinforcement learning, IEEE Trans. Veh. Technol. 69 (5) (2020) 5723–5728.
[51] S. Yang, J. Tan, T. Lei, B. Linares-Barranco, Smart traffic navigation system for fault-tolerant edge computing of internet of vehicle in intelligent transportation gateway, IEEE Trans. Intell. Transp. Syst. (2023).
[52] K. Bajaj, B. Sharma, R. Singh, Implementation analysis of iot-based offloading frameworks on cloud/edge computing for sensor generated big data, Complex Intell. Syst. 8 (5) (2022) 3641–3658.
[53] R. Yadav, W. Zhang, I.A. Elgendy, G. Dong, M. Shafiq, A.A. Laghari, S. Prakash, Smart healthcare: Rl-based task offloading scheme for edge-enable sensor networks, IEEE Sens. J. 21 (22) (2021) 24910–24918.
[54] H. Yan, M. Bilal, X. Xu, S. Vimal, Edge server deployment for health monitoring with reinforcement learning in internet of medical things, IEEE Trans. Comput. Soc. Syst. (2022).
[55] Y. Yang, R. Elsinghorst, J.J. Martinez, H. Hou, J. Lu, Z.D. Deng, A real-time underwater acoustic telemetry receiver with edge computing for studying fish behavior and environmental sensing, IEEE Internet Things J. 9 (18) (2022) 17821–17831.
[56] A. Mehrabi, M. Siekkinen, T. Kämäräinen, Antti, multi-tier cloudvr: Leveraging edge computing in remote rendered virtual reality, ACM Trans. Multimedia Comput. Commun. Appl. (TOMM) 17 (2) (2021) 1–24.
[57] Z. Chen, H. Zhu, L. Song, D. He, B. Xia, Wireless multiplayer interactive virtual reality game systems with edge computing: Modeling and optimization, IEEE Trans. Wireless Commun. 21 (11) (2022) 9684–9699.
[58] P. Ren, X. Qiao, Y. Huang, L. Liu, C. Pu, S. Dustdar, J. Chen, Edge ar x5: An edge-assisted multi-user collaborative framework for mobile web augmented reality in 5 g and beyond, IEEE Trans. Cloud Comput. 10 (4) (2020) 2521–2537.
[59] H. Li, Y. Dong, C. Yin, J. Xi, L. Bai, Z. Hui, et al., A real-time monitoring and warning system for power grids based on edge computing, Math. Probl. Eng. 2022 (2022).
[60] W. Huo, F. Liu, L. Wang, Y. Jin, L. Wang, Research on distributed power distribution fault detection based on edge computing, IEEE Access 8 (2019) 24643–24652.
[61] L. Liu, J. Feng, X. Mu, Q. Pei, D. Lan, M. Xiao, Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing, IEEE Trans. Intell. Transp. Syst. (2023).
[62] X. Wei, L. Cai, N. Wei, P. Zou, J. Zhang, S. Subramaniam, Joint UAV trajectory planning, dag task scheduling, and service function deployment based on drl in UAV-empowered edge computing, IEEE Internet Things J. (2023).
[63] 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.
[64] S. Wang, X. Song, H. Xu, T. Song, G. Zhang, Y. Yang, Joint offloading decision and resource allocation in vehicular edge computing networks, Digit. Commun. Netw. (2023).
[65] 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. (2023).
[66] Y. Liu, J. Yan, X. Zhao, Deep reinforcement learning based latency minimization for mobile edge computing with virtualization in maritime UAV communication network, IEEE Trans. Veh. Technol. 71 (4) (2022) 4225–4236.
[67] H. Ke, H. Wang, H. Sun, Multi-agent deep reinforcement learning-based partial task offloading and resource allocation in edge computing environment, Electronics 11 (15) (2022) 2394.
[68] Y. Sun, Q. He, Joint task offloading and resource allocation for multi-user and multi-server mec networks: A deep reinforcement learning approach with multi-branch architecture, Eng. Appl. Artif. Intell. 126 (2023) 106790.
[69] 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) (2021) 3536–3550.
[70] 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) (2021) 1517–1530.
[71] G. Qiao, S. Leng, Y. Zhang, Online learning and optimization for computation offloading in d2d edge computing and networks, Mob. Netw. Appl. (2019) 1–12.
[72] 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) (2021) 1960–1972.
[73] Y. Wu, J. Xia, C. Gao, J. Ou, C. Fan, J. Ou, D. Fan, Task offloading for vehicular edge computing with imperfect csi: A deep reinforcement approach, Phys. Commun. 55 (2022) 101867.
[74] 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.
[75] J. Wang, J. Hu, G. Min, W. Zhan, A.Y. Zomaya, N. Georgalas, Dependent task offloading for edge computing based on deep reinforcement learning, IEEE Trans. Comput. 71 (10) (2021) 2449–2461.
[76] L. Liao, Y. Lai, F. Yang, W. Zeng, Online computation offloading with double reinforcement learning algorithm in mobile edge computing, J. Parallel Distrib. Comput. 171 (2023) 28–39.
[77] Z. Chen, L. Zhang, Y. Pei, C. Jiang, L. Yin, Noma-based multi-user mobile edge computation offloading via cooperative multi-agent deep reinforcement learning, IEEE Trans. Cognit. Commun. Netw. 8 (1) (2021) 350–364.
[78] M. Li, J. Gao, L. Zhao, X. Shen, Deep reinforcement learning for collaborative edge computing in vehicular networks, IEEE Trans. Cognit. Commun. Netw. 6 (4) (2020) 1122–1135.
[79] J. Wang, H. Ke, X. Liu, H. Wang, Optimization for computational offloading in multi-access edge computing: A deep reinforcement learning scheme, Comput. Netw. 204 (2022) 108690.

[80] K. Zhang, J. Cao, Y. Zhang, Adaptive digital twin and multiagent deep reinforcement learning for vehicular edge computing and networks, IEEE Trans. Ind. Inform. 18 (2) (2021) 1405–1413.
[81] J. Li, Z. Yang, X. Wang, Y. Xia, S. Ni, Task offloading mechanism based on federated reinforcement learning in mobile edge computing, Digit. Commun. Netw. 9 (2) (2023) 492–504.
[82] D. Wei, N. Xi, X. Ma, M. Shojafar, S. Kumari, J. Ma, Personalized privacy-aware task offloading for edge-cloud-assisted industrial internet of things in automated manufacturing, IEEE Trans. Ind. Inform. 18 (11) (2022) 7935–7945.
[83] Y. Ju, Y. Chen, Z. Cao, L. Liu, Q. Pei, M. Xiao, K. Ota, M. Dong, V.C. Leung, Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach, IEEE Trans. Intell. Transp. Syst. (2023).
[84] H. Gao, W. Huang, T. Liu, Y. Yin, Y. Li, Ppo2: Location privacy-oriented task offloading to edge computing using reinforcement learning for intelligent autonomous transport systems, IEEE Trans. Intell. Transp. Syst. (2022).
[85] T.M. Ho, K.-K. Nguyen, Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach, IEEE Trans. Mob. Comput. 21 (7) (2020) 2421–2435.
[86] Y. Tu, H. Chen, L. Yan, X. Zhou, Task offloading based on lstm prediction and deep reinforcement learning for efficient edge computing in iot, Future Internet 14 (2) (2022) 30.
[87] Y. Sun, Q. He, Computational offloading for mec networks with energy harvesting: a hierarchical multi-agent reinforcement learning approach, Electronics 12 (6) (2023) 1304.
[88] Z. Gao, L. Yang, Y. Dai, Fast adaptive task offloading and resource allocation via multiagent reinforcement learning in heterogeneous vehicular fog computing, IEEE Internet Things J. 10 (8) (2022) 6818–6835.
[89] 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. (2023).
[90] M. Othman, S.A. Madani, S.U. Khan, et al., A survey of mobile cloud computing application models, IEEE Commun. Surv. Tutorials 16 (1) (2013) 393–413.
[91] Z. Lushan, H. Xiaowen, Y. Jingmin, Z. Yifeng, Z. Guanglin, Z. Wenjie, Review on resources allocation and pricing methods in mobile edge computing, Telecommun. Sci. 38 (3) (2022).
[92] R. Xie, X. Lian, Q. Jia, T. Huang, Y. Liu, et al., Survey on computation offloading in mobile edge computing, J. Commun. 39 (11) (2018) 138–155.
[93] R. Aghazadeh, A. Shahidinejad, M. Ghobaei-Arani, Proactive content caching in edge computing environment: A review, Softw. - Pract. Exp. 53 (3) (2023) 811–855.
[94] H. Zhou, Z. Zhang, D. Li, Z. Su, Joint optimization of computing offloading and service caching in edge computing-based smart grid, IEEE Trans. Cloud Comput. (2022).
[95] S. Bi, L. Huang, Y.-J.A. Zhang, Joint optimization of service caching placement and computation offloading in mobile edge computing systems, IEEE Trans. Wireless Commun. 19 (7) (2020) 4947–4963.
[96] J. Xu, L. Chen, P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 207–215.
[97] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015, arXiv preprint arXiv:1503.02531.
[98] J. Gou, B. Yu, S.J. Maybank, D. Tao, Knowledge distillation: A survey, Int. J. Comput. Vis. 129 (6) (2021) 1789–1819.
[99] I. Jang, H. Kim, D. Lee, Y.-S. Son, S. Kim, Knowledge transfer for on-device deep reinforcement learning in resource constrained edge computing systems, IEEE Access 8 (2020) 146588–146597.
[100] T. Zhang, Z. Li, Y. Chen, K.-Y. Lam, J. Zhao, Edge-cloud cooperation for dnn inference via reinforcement learning and supervised learning, in: 2022 IEEE International Conferences on Internet of Things (IThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), IEEE, 2022, pp. 77–84.
[101] A. Gholami, S. Kim, Z. Dong, Z. Yao, M.W. Mahoney, K. Keutzer, A survey of quantization methods for efficient neural network inference, in: Low-Power Computer Vision, Chapman and Hall/CRC, 2022, pp. 291–326.
[102] A. Berthelier, T. Chateau, S. Duffner, C. Garcia, C. Blanc, Deep model compression and architecture optimization for embedded systems: A survey, J. Signal Process. Syst. 93 (8) (2021) 863–878.
[103] M. Nagel, M. Fournarakis, R.A. Amjad, Y. Bondarenko, M. Van Baalen, T. Blankevoort, A white paper on neural network quantization, 2021, arXiv preprint arXiv:2106.08295.
[104] G. Chaopeng, L. Zhengqing, S. Jie, A privacy protection approach in edge-computing based on maximized dnn partition strategy with energy saving, J. Cloud Comput. 12 (1) (2023) 29.
[105] J. Yang, S. Hong, J.-Y. Kim, Fixar: A fixed-point deep reinforcement learning platform with quantization-aware training and adaptive parallelism, in: 2021 58th ACM/IEEE Design Automation Conference, DAC, IEEE, 2021, pp. 259–264.
[106] S.-K. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K.-R. Müller, W. Samek, Pruning by explaining: A novel criterion for deep neural network pruning, Pattern Recognit. 115 (2021) 107899.
[107] H. Zhan, W.-M. Lin, Y. Cao, Deep model compression via two-stage deep reinforcement learning, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2021, pp. 238–254.
[108] M. Zawish, N. Ashraf, R.I. Ansari, S. Davy, Energy-aware ai-driven framework for edge-computing-based iot applications, IEEE Internet Things J. 10 (6) (2022) 5013–5023.
[109] H. Zhou, K. Jiang, S. He, G. Min, J. Wu, Distributed deep multi-agent reinforcement learning for cooperative edge caching in internet-of-vehicles, IEEE Trans. Wireless Commun. 22 (12) (2023) 9595–9609.
[110] S. Liu, C. Zheng, Y. Huang, T.Q. Quek, Distributed reinforcement learning for privacy-preserving dynamic edge caching, IEEE J. Sel. Areas Commun. 40 (3) (2022) 749–760.
[111] H. Zhang, T. Yu, Taxonomy of reinforcement learning algorithms, in: Deep Reinforcement Learning: Fundamentals, Research and Applications, 2020, pp. 125–133.
[112] 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.
[113] R. Huang, T. Yu, Z. Ding, S. Zhang, Policy gradient, in: Deep Reinforcement Learning: Fundamentals, Research and Applications, 2020, pp. 161–212.
[114] J. Yan, S. Bi, Y.J.A. Zhang, Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach, IEEE Trans. Wireless Commun. 19 (8) (2020) 5404–5419.
[115] 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 preprint arXiv:1509.02971.
[116] X. Deng, J. Yin, P. Guan, N.N. Xiong, L. Zhang, S. Mumtaz, Intelligent delay-aware partial computing task offloading for multi-user industrial internet of things through edge computing, IEEE Internet Things J. (2021).
[117] M. Tang, V.W. Wong, Deep reinforcement learning for task offloading in mobile edge computing systems, IEEE Trans. Mob. Comput. 21 (6) (2020) 1985–1997.
[118] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[119] 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.
[120] 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, 2016, pp. 2094–2100.
[121] S. Moon, Y. Lim, Federated deep reinforcement learning based task offloading with power control in vehicular edge computing, Sensors 22 (24) (2022) 9595.
[122] Y. Ye, L. Shi, X. Chu, D. Li, G. Lu, Delay minimization in wireless powered mobile edge computing with hybrid backcom and at, IEEE Wireless Commun. Lett. 10 (7) (2021) 1532–1536.
[123] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, L. Hanzo, Multi-agent deep reinforcement learning-based trajectory planning for multi-UAV assisted mobile edge computing, IEEE Trans. Cognit. Commun. Netw. 7 (1) (2020) 73–84.
[124] 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).
[125] S.A. Huda, S. Moh, Deep reinforcement learning-based computation offloading in UAV swarm-enabled edge computing for surveillance applications, IEEE Access (2023).
[126] R. Yang, F.R. Yu, P. Si, Z. Yang, Y. Zhang, Integrated blockchain and edge computing systems: A survey, some research issues and challenges, IEEE Commun. Surv. Tutor. 21 (2) (2019) 1508–1532.
[127] D.C. Nguyen, P.N. Pathirana, M. Ding, A. Seneviratne, Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2536–2549.
[128] S. Gong, Y. Xie, J. Xu, D. Niyato, Y.-C. Liang, Deep reinforcement learning for backscatter-aided data offloading in mobile edge computing, IEEE Netw. 34 (5) (2020) 106–113.
[129] Z. Chen, X. Wang, Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach, EURASIP J. Wireless Commun. Networking 2020 (1) (2020) 1–21.
[130] Z. Zhang, H. Li, Z. Tang, D. Gu, J. Zhang, A clustering offloading decision method for edge computing tasks based on deep reinforcement learning, New Gener. Comput. 41 (1) (2023) 85–108.
[131] L. Xie, S. Wang, A. Markham, N. Trigoni, Towards monocular vision based obstacle avoidance through deep reinforcement learning, 2017, arXiv preprint arXiv:1706.09829.
[132] X. Hu, Y. Huang, Deep reinforcement learning based offloading decision algorithm for vehicular edge computing, PeerJ Comput. Sci. 8 (2022) e1126.
[133] Y. Ren, A. Guo, C. Song, Y. Xing, Dynamic resource allocation scheme and deep deterministic policy gradient-based mobile edge computing slices system, IEEE Access 9 (2021) 86062–86073.
[134] B. Trinh, G.-M. Muntean, A deep reinforcement learning-based offloading scheme for multi-access edge computing-supported extended reality systems, IEEE Trans. Veh. Technol. 72 (1) (2022) 1254–1264.

[135] F. Fu, Y. Kang, Z. Zhang, F.R. Yu, T. Wu, Soft actor–critic drl for live transcoding and streaming in vehicular fog-computing-enabled iov, IEEE Internet Things J. 8 (3) (2020) 1308–1321.
[136] 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.
[137] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, Y. Wu, The surprising effectiveness of ppo in cooperative multi-agent games, Adv. Neural Inf. Process. Syst. 35 (2022) 24611–24624.
[138] S. Bebortta, D. Senapati, C.R. Panigrahi, B. Pati, Adaptive performance modeling framework for qos-aware offloading in mec-based iiot systems, IEEE Internet Things J. 9 (12) (2021) 10162–10171.
[139] H.M. Do, T.P. Tran, M. Yoo, Deep reinforcement learning-based task offloading and resource allocation for industrial iot in mec federation system, IEEE Access (2023).
[140] Z. Akhavan, M. Esmaeili, B. Badnava, M. Yousefi, X. Sun, M. Devetsikiotis, P. Zarkesh-Ha, Deep reinforcement learning for online latency aware workload offloading in mobile edge computing, in: GLOBECOM 2022-2022 IEEE Global Communications Conference, IEEE, 2022, pp. 2218–2223.
[141] 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. (2023).
[142] X. Chen, G. Liu, Federated deep reinforcement learning-based task offloading and resource allocation for smart cities in a mobile edge network, Sensors 22 (13) (2022) 4738.
[143] X. Huang, S. Leng, S. Maharjan, Y. Zhang, Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks, IEEE Trans. Veh. Technol. 70 (9) (2021) 9282–9293.
[144] Y. Chen, S. Han, G. Chen, J. Yin, K.N. Wang, J. Cao, A deep reinforcement learning-based wireless body area network offloading optimization strategy for healthcare services, Health Inf. Sci. Syst. 11 (1) (2023) 8.
[145] A.R. Mahmood, H.P. Van Hasselt, R.S. Sutton, Weighted importance sampling for off-policy learning with linear function approximation, Adv. Neural Inf. Process. Syst. 27 (2014).
[146] J. Feng, F.R. Yu, Q. Pei, X. Chu, J. Du, L. Zhu, Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach, IEEE Internet Things J. 7 (7) (2019) 6214–6228.
[147] W. Hou, H. Wen, H. Song, W. Lei, W. Zhang, Multiagent deep reinforcement learning for task offloading and resource allocation in cybertwin-based networks, IEEE Internet Things J. 8 (22) (2021) 16256–16268.
[148] Y. Liu, W. Wang, H.-H. Chen, F. Lyu, L. Wang, W. Meng, X. Shen, Physical layer security assisted computation offloading in intelligently connected vehicle networks, IEEE Trans. Wireless Commun. 20 (6) (2021) 3555–3570.
[149] P. Zhang, Y. Su, B. Li, L. Liu, C. Wang, W. Zhang, L. Tan, Deep reinforcement learning based computation offloading in UAV-assisted edge computing, Drones 7 (3) (2023) 213.
[150] F. Yu, D. Yang, F. Wu, Y. Wang, H. He, Resource optimization for UAV-assisted mobile edge computing system based on deep reinforcement learning, Phys. Commun. 59 (2023) 102107.
[151] L. Tan, Z. Kuang, J. Gao, L. Zhao, Energy-efficient collaborative multi-access edge computing via deep reinforcement learning, IEEE Trans. Ind. Inform. (2022).
[152] Z. Aghapour, S. Sharifian, H. Taheri, Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed ai execution tasks in iot edge computing environments, Comput. Netw. 223 (2023) 109577.
[153] Z. Aghapour, S. Sharifian, H. Taheri, An improved salp swarm algorithm for energy saving in iot to fog data communication, in: 2020 28th Iranian Conference on Electrical Engineering, ICEE, IEEE, 2020, pp. 1–5.
[154] J. Rosenberger, M. Urlaub, F. Rauterberg, T. Lutz, A. Selig, M. Bühren, D. Schramm, Deep reinforcement learning multi-agent system for resource allocation in industrial internet of things, Sensors 22 (11) (2022) 4099.
[155] S. Ke, Research on Service Caching and Task Offloading in Edge Computing (Master), Huazhong University of Science and Technology, 2022, http://dx.doi.org/10.27157/d.cnki.ghzku.2022.001528.
[156] 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.
[157] N. Li, X. Zhu, Y. Li, L. Wang, L. Zhai, Service caching and task offloading of internet of things devices guided by lyapunov optimization, in: 2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), IEEE, 2022, pp. 121–128.
[158] Y. Zhou, X. Li, H. Ji, H. Zhang, Blockchain-based trustworthy service caching and task offloading for intelligent edge computing, in: 2021 IEEE Global Communications Conference, GLOBECOM, IEEE, 2021, pp. 1–6.
[159] 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. (2023).
[160] 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. (2023).
[161] 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.
[162] H. Zhou, Z. Zhang, Y. Wu, M. Dong, V.C. Leung, Energy efficient joint computation offloading and service caching for mobile edge computing: A deep reinforcement learning approach, IEEE Trans. Green Commun. Netw. (2022).
[163] L. Wang, G. Zhang, Joint.service. caching, Resource allocation and computation offloading in three-tier cooperative mobile edge computing system, IEEE Trans. Netw. Sci. Eng. (2023).
[164] 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) (2022) 1735–1746.
[165] 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 5 g ultradense network, IEEE Internet Things J. 8 (4) (2020) 2238–2251.
[166] Q. Wang, S. Chen, M. Wu, Incentive-aware blockchain-assisted intelligent edge caching and computation offloading for iot, Engineering (2023).
[167] Z. Yang, Y. Liu, Y. Chen, N. Al-Dhahir, Cache-aided noma mobile edge computing: A reinforcement learning approach, IEEE Trans. Wireless Commun. 19 (10) (2020) 6899–6915.
[168] F. Li, C. Fang, M. Liu, N. Li, T. Sun, Intelligent computation offloading mechanism with content cache in mobile edge computing, Electronics 12 (5) (2023) 1254.
[169] 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.
[170] S. Chen, L. Rui, Z. Gao, W. Li, X. Qiu, Cache-assisted collaborative task offloading and resource allocation strategy: A metareinforcement learning approach, IEEE Internet Things J. 9 (20) (2022) 19823–19842.
[171] A. Asheralieva, D. Niyato, Bayesian reinforcement learning and bayesian deep learning for blockchains with mobile edge computing, IEEE Trans. Cognit. Commun. Netw. 7 (1) (2020) 319–335.
[172] J. Yang, J. Zhang, M. Xi, Y. Lei, Y. Sun, A deep reinforcement learning algorithm suitable for autonomous vehicles: Double bootstrapped soft-actor–critic-discrete, IEEE Trans. Cogn. Dev. Syst. 15 (4) (2021) 2041–2052.
[173] F. Guo, F.R. Yu, H. Zhang, H. Ji, M. Liu, V.C. Leung, Adaptive resource allocation in future wireless networks with blockchain and mobile edge computing, IEEE Trans. Wireless Commun. 19 (3) (2019) 1689–1703.
[174] N. Ma, H. Li, H. Liu, State-space compression for efficient policy learning in crude oil scheduling, Mathematics 12 (3) (2024) 393.
[175] R. Dadashi, L. Hussenot, D. Vincent, S. Girgin, A. Raichuk, M. Geist, O. Pietquin, Continuous control with action quantization from demonstrations, 2021, arXiv preprint arXiv:2110.10149.
[176] Y. He, Y. Wang, Q. Lin, J. Li, Meta-hierarchical reinforcement learning (mhrl)-based dynamic resource allocation for dynamic vehicular networks, IEEE Trans. Veh. Technol. 71 (4) (2022) 3495–3506.