Deep Learning based Collaborative Edge Caching for Mobile Edge Networks

Zhao, Ming

Awarding institution:
King's College London

Deep Learning based Collaborative Edge Caching for Mobile Edge Networks

Author : Ming Zhao

Supervisor : Dr. Mohammad Reza Nakhai

Department of Engineering
King's College London

The thesis is submitted for the degree of
Doctor of Philosophy

Dec 2023

Acknowledgment

This thesis would not have been possible without the assistance and guidance of several individuals. It is a pleasure to take this opportunity to express my sincere gratitude to all who in one way or another contributed to the completion of this study.

Firstly, please allow me to express greatest appreciation to my primary supervisor Dr. Mohammad Reza Nakhai, who guided me through my whole PhD study with professional experience and suggestion. Many thanks for his patient assistance and unwavering support in my research works carried out in the past four years, without which the thesis could not have been completed on my own. His exceptional research skills, deep enthusiasm and disciplined attitude in academia inspire me greatly, and I aspire to carry forward these qualities in my future career.

I would also like to express my thanks to China Scholarship Council for financially support my PhD study and King's College London for providing the advanced research conditions.

Special thanks go to all my friends across the world including Linyan Zhou, Yanhong Chen, Meng Wu and Dr. Dongzhu Liu, and my colleagues at King's College London including Dr. Zhenfeng Sun, Dr. Qian Shi, Hongying Zhou, Jian Jiang, Yujia Xu and Dr. Ming Chen, who supported me in various ways for the past four years. I would also like to acknowledge all members of staff in Department of Engineering including Dr. Yansha Deng, Dr. Hak-Keung Lam and Dr. Oya Celiktutan for their inspiring and valuable advice.

Last but not least, I would also like to thank my family for all their unconditional support and deepest love. I express my special appreciation, especially to my mother and father, for everything they have done for me and for accompanying me through such a long journey.

As a closing comment, I would like to remind myself that: take it easy, you can have your own answers.

Dedication

This thesis is dedicated to my grandma for her endless love.
You are still with me now, I know.
I always know.

Abstract

Currently, massive end devices connect to the wireless network to access various services and applications, e.g., content access, environmental monitoring and virtual management. This inevitably results in significant backbone network congestion, leading to a serious degradation in the quality of service/experience (QoS/QoE). To address these issues, mobile edge computing (MEC) has emerged as a promising paradigm, facilitating computation, caching and communication capabilities at network edges, e.g., base stations (BSs). Accordingly, edge caching is proposed as an efficient approach to lower retrieving latency and alleviate network congestion by caching popular content in proximity to users. Nonetheless, several critical challenges must be addressed before embracing the full potential of edge caching. First, the problem of how to precisely predict users' requests and cache the proper content in storage-constrained edge servers remains to be solved. Second, designing collaborative caching policies for distributed edge servers in unknown and dynamic environments remains a challenging issue. Conventional caching policies, such as least recently used (LRU) and least frequently used (LFU), cannot adapt to complex scenarios, particularly when the environment is dynamic or even unknown. In contrast, machine learning algorithms have the ability to learn and capture hidden features of massive request data in dynamic networks. Especially, reinforcement learning (RL) algorithms can iteratively optimize caching policies through repeated interaction with the environment by learning from either online or offline experiences.

Considering the aforementioned challenges and the advantages of machine learning algorithms shown in dealing with environment dynamics and request uncertainties, this thesis focuses on the design of optimal caching strategies for collaborative edge caching networks by leveraging advances in both optimization and machine learning.

The general aim of this thesis is to apply novel optimization and machine learning methods to collaborative edge caching systems, in order to enhance performance in resource utilization from a caching perspective, as well as in scalability and robustness from an algorithmic design perspective. Accordingly, this thesis introduces three innovative caching schemes to realize this

aim. The first work develops a novel caching policy optimization approach by integrating conventional RL techniques with evolutionary algorithms, to enhance collaboration in a MEC system with heterogeneous scenarios. The second work proposes a two-phase proactive caching scheme based on deep Q network (DQN) to overcome the curse of dimensionality in a computationally efficient manner. In the third contribution, we focus on the joint cache update and request delivery problem in scalable collaborative edge networks. An innovative unified federated DQN caching scheme is developed to minimize the system cost while ensuring QoS. The performance of these proposed caching schemes is compared and analyzed in complex and dynamic scenarios through comprehensive experiments, which highlights the advantages of the proposed schemes in terms of caching performance, scalability and robustness.

Contents

List of Figures

List of Tables

Notations and Acronyms

A3C	Asynchronous advantage actor-critic
AR	Augmented reality
AWS	Amazon web services
BS	Base station
CDN	Content delivery network
D2D	Device-to-device
DDPG	Deterministic policy gradient
DQN	Deep Q network
DRL	Deep reinforcement learning
EA	Evolutionary algorithm
eDQN	Evolutionary deep Q network
ETSI	European telecommunications standard institute
FC	Fully connected
FIFO	First in and first out
FL	Federated learning
F-RAN	Fog radio access network
HRL	Hierarchical reinforcement learning
IoV	Internet of vehicles
IoT	Internet of things
IRM	Independent reference model
LFU	Least frequently used
LRU	Least Recently used
MAB	Multi-armed bandit
MCC	Mobile cloud computing
MDP	Markov decision process
MEC	Mobile edge computing
MFEA	Multifactorial evolutionary algorithm
ODCR	Online distributed cache replacement
QoE	Quality-of-Experience
QoS	Quality-of-Service
RAN	Radio access network
RL	Reinforcement learning

SLAM	Simultaneous localization and mapping
SNM	Short noise model
TS	Thompson sampling
VR	Virtual reality

Chapter 1

Introduction

1.1 Background and Motivations

With the rapid development of intelligent mobile devices, e.g., smartphones and tablets, the demand of multimedia services has been growing dramatically, which imposes heavy burden on mobile cellular networks due to serious data traffic [4]. To deal with this issue, Mobile edge computing (MEC) is proposed as an innovative paradigm which enables the capabilities of computing, caching and communication at the network edges that are in close proximity to the end users [5, 6]. Specifically, this paradigm is capable of executing computation-intensive and latency-sensitive tasks at the resource-constrained mobile devices, by leveraging the idle computational resources and storage space at edge servers. In consequence, the burdens on backhaul networks are alleviated, and hence both the quality-of-experience (QoE) of users and the quality-of-service (QoS) of mobile network operators are improved [7, 8]. Particularly, edge caching [6, 9] has been observed as a promising technique over recent years, by which frequently requested contents are cached at the edge servers so that the redundant long transmission delays from the remote cloud platforms to end users can be greatly eliminated. Nonetheless, it is essential to tackle several challenges before fully harnessing the advantages of edge caching techniques. Hence, we point out three critical challenges within the realm of edge caching research fields, as follows.

Limited storage/computational capacities. The storage and computational capacities of MEC servers are considerably limited compared to those of cloud platforms. Therefore, optimal resource management in MEC systems, e.g., optimal content caching policy, is vital to achieve lower transmission delay and thus improve QoS/QoE.

Dynamic environment. In edge caching scenarios, the time-varying variables such as the channel conditions and cache status at edge servers may not be available when making decisions. Meanwhile, user requests are unknown

in advance and difficult to predict. Moreover, different users have different preferences and requesting models, which are generally highly heterogeneous. Even the same user may follow different content request patterns at different time periods, which further aggravates the request heterogeneity. Although these environment dynamics can potentially be predicted or modeled, the traditional techniques like Markov decision process (MDP) [10] require a complete knowledge of transition probability while the environment changes, which is impractical in real-world scenarios.

Efficient collaboration among distributed caching entities. Typically, edge caching techniques involve multiple entities, e.g., edge servers, remote cloud platforms, and end users, working together to optimize the storage and distribution of content. Even if it is possible to develop an optimal strategy using a centralized approach, the process still requires collecting a huge amount of information, such as aforementioned environment dynamics, and the signalling overheads may not be affordable. On the other hand, designing content caching and delivery strategies in a distributed manner is expected to be more effective, thanks to the inherent decentralization attribute of MEC systems. Nonetheless, the issue of efficient collaboration among multiple distributed caching entities is not fully addressed. Moreover, the challenge of designing collaborative strategies becomes significantly more complex in large-scale MEC systems due to the increased number of caching entities and highly heterogeneous user requests.

To tackle the above challenges, novel approaches based on optimization and machine learning techniques have been flourishing. Specially, reinforcement learning (RL) provides an efficient and effective solution for handling edge caching problems. The learning process of RL algorithms mimics the trial-and-error pattern observed in human behavior. The agent systematically refines its decision policy by interacting with the dynamic environment, relying solely on information derived from the reward signal [11]. In other words, RL algorithms can generate an optimal policy through interaction with the environment and receiving reward signals, without requiring knowledge of the dynamic models. This highlights the advantage of RL algorithms in revealing the "hidden insights" of the dynamic environment and massive data generated by MEC systems, enabling the generation of optimal caching policies for complex MEC systems. Furthermore, recent advanced algorithms based on the combination of optimization and RL, such as deep reinforcement learning (DRL) [12], have emerged to deal with content caching and delivery problems in edge networks. Nevertheless, the high-dimensional state and action spaces modeled for large-scale scenarios bring significant challenges to conventional DRL algorithms, which has not been fully addressed.

Therefore, these open issues motivate our exploration of novel approaches to overcome the challenges in collaborative edge caching systems, particularly in the context of large-scale systems.

1.2 Aims and Research Objectives

In this context, the research presented in this thesis focuses on proposing sophisticated approaches that utilize advanced optimization and machine learning techniques to address the difficulty in collaborative edge caching systems. Specifically, our aims mainly cover three key issues in edge caching systems, as specified below:

1) Enhancing collaboration among caching entities in an edge caching system with heterogeneous scenarios.

2) Enabling network scalability by addressing high dimensionality problem in large-scale systems with an increased number of edge servers, users, and files.

3) Solving the problem of joint cache update and request delivery for scalable collaborative edge networks.

Correspondingly, the objectives can be specified as follows,

1) To develop a caching policy optimization approach by utilizing deep Q network (DQN) and multifactorial evolutionary algorithm (MFEA), in order to simultaneously and collaboratively optimize the caching policies of distributed edge servers in different caching scenarios, thereby improving caching performance.

2) To propose a scalable DQN-based algorithm, which overcomes the large dimensionality of action space caused by the increased number of files, as well as minimizes the overall system cost while ensuring QoE for users.

3) To develop a caching scheme by integrating federated learning (FL) with the proposed scalable DQN-based algorithm, which will enhance the coordination among distributed servers, and thus improve the system performance in terms of joint cache update and request delivery. Besides, to evaluate the robustness of the proposed scheme against heterogeneous scenarios and request uncertainties.

1.3 Organization of the Works

- In Chapter 2, an overview of MEC is provided, covering the historical background, definitions, architecture and advantages, as well as a survey of issues on MEC research. Next, a comprehensive review on the state-of-the-art edge caching research area is conducted, with a focus on caching policy design based on optimization and machine learning techniques.

- In Chapter 3, the basic concepts of RL, FL and MFEA are introduced. Specifically, an overview of two novel algorithms in RL, Q-learning and DQN, is presented.

- In Chapter 4, a generic system model is presented for the sake of clarity and consistency. Based on this model, we introduce the basic concepts and specify essential assumptions in an collaborative edge caching system, which is applicable to subsequent chapters.

- In Chapter 5, a novel caching policy optimization approach based on DQN and MFEA, named as evolutionary DQN (eDQN) algorithm, is demonstrated. The design of the proposed eDQN algorithm and the training procedure are introduced, and the performance of the proposed eDQN algorithm is evaluated based on a real-world dataset from MovieLens.

- In Chapter 6, a DQN-based two-phase proactive caching scheme is presented. The output layer of the DQN neural network is specially designed to overcome the high dimensionality problem, and a two-phase action selection procedure is further devised to enhance exploration of actions. To verify the scalability and robustness of the proposed scheme, we conduct comparative experiments with other existing caching schemes on both small-scale and large-scale caching scenarios.

- In Chapter 7, an innovative unified federated DQN caching scheme for large-scale collaborative edge caching systems is introduced. The proposed scheme consists of a federated DQN algorithm and a greedy algorithm to address the joint cache update and request delivery problem, with the objective of minimizing the system cost while ensuring both users' QoE and the network operator's QoS. Comprehensive experiments are conducted to evaluate the performance of the proposed scheme.

- In Chapter 8, the final summary of the thesis is given and potential research directions for future work are discussed.

1.4 Contributions of the Thesis

The contributions of the thesis are summarized as follows:

1. A novel caching policy optimization approach based on DQN and MFEA (i.e., eDQN algorithm) is proposed for a MEC system with heterogeneous caching scenarios. To the best of our knowledge, we are the first to apply MFEA to solve the collaborative edge caching problem. The comparing experimental results demonstrate that our proposed algorithm can optimize caching policies in different scenarios collaboratively, and thus significantly improve cache hit rate and reduce content downloading latency. This work has been published in *2022 IEEE Globecom Workshops*.

2. A two-phase online proactive caching scheme based on DQN is proposed to deal with the exponentially growing dimension of the action space in a computationally efficient manner. Moreover, the performance of the developed caching scheme is significantly improved by introducing a request-driven exploration method during training. Numerical results in small-scale scenarios confirm the robustness and efficiency of the proposed scheme in terms of cache hit rate and runtime. Results in large-scale scenarios show the advantages of the proposed scheme over other existing caching methods, particularly in terms of average system cost and cache hit rate. The results have been accepted by *2024 IEEE Wireless Communications and Networking Conference (WCNC)* as a conference paper.

3. An innovative unified federated DQN caching scheme is proposed to tackle the joint cache update and request delivery problem in a collaborative edge caching network. In this scheme, each edge server optimizes its own caching policy while coordinating with its proximal servers to minimize the overall system cost and improve the cache hit rate. Furthermore, a Thompson sampling-based smart agent selection method is devised to reduce signaling overhead. The advantages of the proposed scheme over other three caching schemes are verified in comprehensive experiments. This work has been published in *IEEE Transactions on Mobile Computing*.

Correspondingly, our publications are listed as follows:

[1] **M. Zhao**, Z. Sun and M. R. Nakhai, "Evolutionary Deep Q Network for Collaborative Edge Caching," *2022 IEEE Globecom Workshops (GC Wkshps)*, Rio de Janeiro, Brazil, 2022, pp. 1840-1845.

[2] **M. Zhao** and M. R. Nakhai, "Deep Reinforcement Learning Based Two-phase Proactive Caching for Collaborative Edge Networks," *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, Dubai, UAE, 2024 (Accepted).

[3] **M. Zhao** and M. R. Nakhai, "A Unified Federated Deep Q Learning Caching Scheme for Scalable Collaborative Edge Networks," *IEEE Transactions on Mobile Computing*, doi: 10.1109/TMC.2024.3382824, pp.1-12, 2024.

[4] Z. Sun, **M. Zhao** and M. R. Nakhai, "Computation Offloading in Energy Harvesting Powered MEC Network," ICC 2021 - IEEE International Conference on Communications, Montreal, QC, Canada, 2021, pp. 1-6.

Chapter 2

Literature Review

In this chapter, we present a comprehensive review on the recent research works related to developing machine learning-based edge caching methods. We first give an overview of MEC in Section 2.1, including historical background, definitions, architecture and advantages, along with a survey of issues on MEC research. Next, in Section 2.2, we conduct a literature review on the state-of-the-art edge caching research area with a focus on caching policy design based on optimization and machine learning techniques.

2.1 Literature Review on Mobile Edge Computing

In the past several decades, the rapid development of mobile devices and the dramatic growth of mobile data traffic have been driving the manifold developments in computing and wireless communications. Particularly, the mobile applications such as mobile video streaming [13], wearable virtual reality (VR) steaming [14, 15], and mobile social media [16] require robust computing and data processing capability, which brings big challenges for resource-constrained mobile devices. To deal with these issues, mobile cloud computing (MCC) emerged as a new framework, by which computing and storage services are processed at centralized cloud platforms [17, 18]. As such, MCC can provide sufficient resources for mobile devices, as well as extend the battery lifetime of mobile devices.

Despite the advantages of MCC, the high backhaul bandwidth consumption and the long latency from the remote cloud to mobile devices remain inevitable problems within this architecture. Consequently, a new architecture called mobile edge computing (MEC, also known as mobile edge networks) is flourishing.

The concept of MEC was originally proposed by European Telecommunications Standard Institute (ETSI) in 2014, which was defined as a new paradigm

that "*provides IT and cloud computing capabilities within Radio Access Network (RAN) in close proximity to mobile subscribers*" [1, 5]. A typical architecture of MEC is illustrated in Fig. 2.1 [1], where MEC servers are co-located with base stations (BSs), and are connected to both the remote data center and content delivery network (CDN) via Internet through a gateway. Such an architecture brings advantages over MCC in various aspects, as will be elaborated in the following.

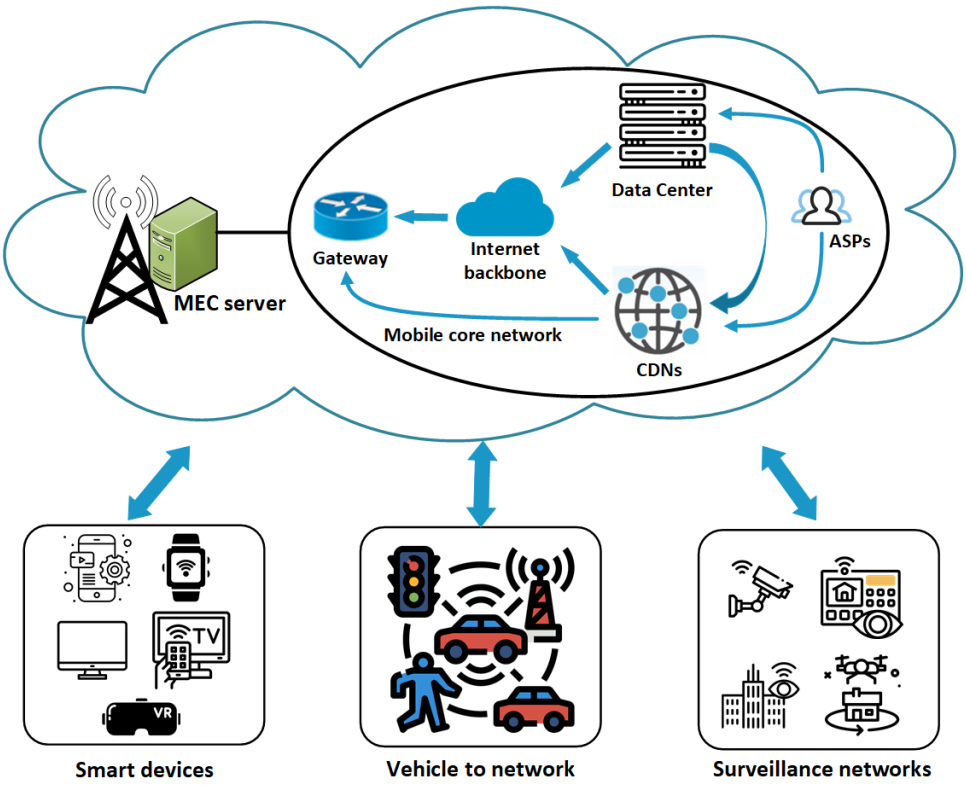

Figure 2.1: A typical architecture of MEC [1].

Low Latency: In general, experiments have indicated that the total latency for MCC falls within the range of 30-100ms [19], which is unacceptable for many latency-sensitive mobile applications. On the contrary, MEC makes computing and storage capabilities available at network edges, i.e., closer to end users. Consequently, the communication latency can be significantly reduced for a variety of mobile applications, including computation offloading and content delivery [20–22]. Zhang *et al.* [21] evaluate the performance of MEC in terms of local breakout and network end-to-end latency in various 4G LTE network scenarios, and confirm that MEC can support the low-latency services with a minimum delay requirement of 17ms in LTE networks. The work of [22] shows that computation offloading via edge computing platforms can greatly reduce latency for highly interactive mobile applications like mobile augmented reality (AR) and cognitive assistance, in both LTE and Wi-Fi networks.

High Energy Efficiency: MEC distinguishes itself as a promising solution for minimizing energy consumption for both network side [23, 24] and device side [25, 26], e.g., Internet of Things (IoT) devices [27–29], by effectively supporting computation offloading. Authors in [23] propose an energy-efficient computation offloading scheme, which jointly optimizes offloading and radio resource allocation to minimize system energy cost under the latency constraints. The results demonstrate a significant improvement in energy efficiency when compared to the non-offloading scheme. Substantial energy savings via computation offloading have been demonstrated in various experiments, e.g., the energy consumption reduction for different mobile applications in both LTE and Wi-Fi networks [22], the extension of battery life by 30-50% for different AR applications [28], or the decrease of energy consumption in a mobile device by up to 42% through computation offloading to edge servers compared to cloud offloading [30].

Context-Awareness: Since edge servers are deployed in close proximity to end users, they can track real-time context information such as users' behaviors and locations, as well as the network level information. With this information, edge servers can provide context-aware services to end users. In consequence, network resources can be allocated in a more efficient manner, leading to an enhanced user experience [31–33]. For instance, an AR application can be used in a museum video guide, to automatically deliver contents related to artworks or antiques, by predicting users' interests based on their locations within a museum [34]. Another example is the smart health system that leverages MEC techniques to implement predictive monitoring of high-risk patients [35–37].

Over the past few years, many efforts from both academia and industry have been dedicated to addressing MEC related issues. Subsequently, numerous survey articles have been published to offer comprehensive insights into the MEC area, each with its distinct focus [1, 6, 38–40]. Authors in [6] survey the state-of-the-art techniques of edge computing, caching and communications (3C). Furthermore, the potential synergies among 3C is explored. A taxonomy for MEC applications is introduced in [38], which also analyzes the limitations of the applications, along with potential directions such as content scaling and data aggregation, etc. Roman *et al.* [39] focus on the security issues on the edge paradigms including MCC, MEC and fog computing, and provides insights into potential synergies in the development of security mechanisms. Zhang and Letaief [40] survey the latest research efforts on MEC in Internet of Vehicles (IoV) from the perspective of edge caching, edge computing and edge AI. Furthermore, open research challenges in this scenario are identified. In [1], authors present an overview of MEC from the communication perspective, with a focus on joint radio-and-computational resource

management for MEC systems.

Considering the escalating complexity of mobile networks, an emerging trend is to integrate machine learning techniques into mobile edge systems for enabling *edge intelligence* [41, 42]. This has led to an overwhelming revolution in a wide range of MEC-related fields, including framework design [8, 43, 44], networking [45, 46], resource allocation [47–51], service placement and caching [52–54], content caching and delivery [55–58], and so on. In [8], a framework called "In-Edge AI" is proposed, whereby DRL techniques and FL framework are integrated to optimize mobile edge computing, caching and communication. In [45], a set of design principles for achieving intelligent MEC systems are introduced, including computation-oriented multiple access for ultra-fast data aggregation, importance-aware resource allocation for agile intelligence acquisition, and learning-driven signal encoding for high-speed data-feature transmission. The authors also provide illustrative examples to verify the effectiveness of these design principles, and identify the corresponding research opportunities. Authors in [59] survey the issues regarding FL implementation in MEC, and analyze the existing solutions to tackle these issues such as communication cost, resource allocation, data privacy and data security. The utilization of machine learning techniques for intelligent IoV is introduced in [40], together with several use cases including edge-assisted perception, mapping, and simultaneous localization and mapping (SLAM). Authors in [57] investigate related work of edge caching optimization from different stakeholder perspectives. They also discuss caching policies from different caching techniques, and highlight the challenges that need to be solved through comparative analysis.

As the research direction proposed in this thesis mainly focuses on the development of machine learning based edge caching strategies for MEC systems, we will conduct a comprehensive survey on the latest advancements in edge caching research fields in the next section.

2.2 Literature Review on Edge Caching

Edge caching is the practice of relocating memory storage, often referred to as caches, in proximity to the network's edge, instead of consolidating them at a distant central location. This approach is developed not only to alleviate the backhaul traffic caused by data transmissions between remote data center and edge servers, but also to reduce the latency and improve user experience. In view of the relevant studies, it has become evident that the key issues on edge caching technology can be categorized into the following three questions [9, 60]: Where to cache? What to cache? How to cache? In this section, we

will investigate the research efforts have been dedicated to advancing caching technology in MEC systems, from perspectives of caching entities, content attributes and caching policies, respectively.

2.2.1 Caching entities

In a MEC system, caching entities are components responsible for storing and managing cached data or content. They play a crucial role in improving the performance of MEC systems by reducing latency and optimizing data delivery. Several types of caching entities commonly found in MEC systems are discussed as follows.

Edge servers. Most of edge servers are equipped with caching units. They are strategically distributed at various geographic locations to bring content closer to end-users. Numerous studies have been conducted to assess the performance of caching at edge servers [61–64]. Zhang *et al.* [62] introduce a distributed edge caching framework for short video network, named AutoSight. It is deployed on edge servers to address issues related to non-stationary user access pattern and temporal/spatial video popularity. The effectiveness of AutoSight has been confirmed through experiments conducted using the real traces of more than 28 million videos with 100 million accesses from 488 edge servers located in 33 cities. In [63], a distributed edge caching framework is designed for individual edge server to learn its optimal policy locally and update caching decisions accordingly. Trace-driven simulation results have demonstrated that the proposed framework can significantly reduce average system cost compared to other existing caching algorithms.

End devices. Featuring enhanced computing and storage capabilities, current end devices like smartphones, laptops, tablets and IoT devices, can serve as caches to store content locally or share content with other end devices directly by leveraging Device-to-Device (D2D) communication technology [58, 65–67]. Zhang *et al.* [65] investigate user-side caching in D2D-enabled caching cellular networks. A heuristic multi-winner repeated auction based caching placement algorithm is proposed, with the aim of reducing traffic load and average content access delay. Two cooperative content replacement algorithms in D2D networks are proposed in [66] by taking account of the proximity factor and content priority of neighbor devices. Bai *et al.* [68] propose a cache-enabled D2D communication scheme considering mobile users' social relations and common interests. In [69], a D2D-assisted caching strategy is introduced to reduce the system energy cost.

Centralized caches. These refer to components where all data/content is stored and managed in a central location. For example, the *origin server* in CDN, where the original content is hosted, can be considered as a central-

ized cache [70]. Data stored in centralized caches is accessible to all edge servers within the network. Therefore, centralized caches can be useful in certain scenarios where edge servers need to retrieve content in response to user demands. In addition, centralized caching is particularly advantageous in scenarios with large datasets or high data turnover due to its ease of management and scalability, given that data is supervised within a centralized, controlled environment [71, 72]. However, the high latency in content retrieval poses a bottleneck for latency-sensitive edge applications. So it is essential to balance the advantages and disadvantages of centralized caching based on the specific requirements of a system or application. In [73], a hybrid content caching framework is proposed, where both central cloud units and BSs are adopted to optimize the caching strategy under the constraint of service latency.

2.2.2 Content attributes

We briefly discuss two attributes of content in the following, including content popularity and content types.

Content popularity is a pivotal attribute, which indicates the extent of interest or engagement that a specific content item has generated. It is closely interrelated to the question of what to cache in the edge networks. A lot of existing works have studied the issues regarding content popularity in caching systems. Next, we will discuss them from two aspects: static models and dynamic models.

Static popularity models. Many works address the caching issues under the assumption that the content popularity follows a static model. One commonly used popularity model is Zipf distribution model [74–77], also known as Zipf's law [78, 79]. It is a static distribution that describes the frequency distribution of elements in a dataset. Specifically, the frequency of an element is inversely proportional to its rank, which means the most frequent element occurs approximately twice as frequently as the second most frequent, three times as frequently as the third most frequent, and so forth. Another static model commonly adopted is independent reference model (IRM), in which the user requests follow an independent Poisson process with rate related to content popularity [80].

Dynamic popularity models. Usually, static models cannot reflect the time varying content popularity. Researchers have been devoted to modeling the content popularity by taking into account many factors such as temporal/spatial correlations of content, user preferences, user mobility, etc. A Gaussian process based Poisson regressor is developed to model the content requests and predict their popularity in [81]. Traverso *et al.* [82] propose a short noise model (SNM), which uses a pulse to model each content: the dura-

tion reflects the content lifespan and the height reflects its instantaneous popularity. In [83], a Markovian model is introduced to capture the long-term popularity as well as the temporal correlations of content. The effectiveness of the proposed model is confirmed by comparing the cache hit rate with a real-traced model under various caching algorithms. In addition to these synthetic models, real-world datasets are also adopted in some studies, such as MovieLens datasets [84–86], YouTube datasets [87], Netflix datasets [88] and data extracted from other social network platforms [89].

It is noted that the different content types will influence the caching design. Multimedia data, like videos and audio files, are the most frequently cached file types. The caching design for multimedia data is studied in many existing works [87, 90, 91].

Additionally, with the development of IoT, a huge amount of monitoring, measurement and automation data are generated by various IoT applications. It is beneficial to cache these IoT data to reduce the frequency of data requesting/retrieval and hence alleviating the network traffic. However, IoT data has much shorter lifetimes compared to multimedia data. As a result, researchers have designed efficient caching strategies to deal with this challenge [92–94]. Authors in [94] propose a caching policy that considers many factors including the lifetime and popularity of IoT data, the storage capacities of the nodes and the bandwidth available in network links. A remarkable performance improvement in terms of latency and cache hit rate can be seen from the simulation results.

2.2.3 Caching policies

Various caching policies have been developed for edge caching. In this subsection, we will classify them into two categories: conventional caching policies and learning based caching policies. Particularly, we will review the learning based caching policies from the perspective of key techniques and patterns of cooperation, respectively. A taxonomy of caching polices in MEC systems is shown in Fig. 2.2.

1) **Conventional caching policies**

Examples of conventional caching policies are Least Recently Used (LRU) [95], Least Frequently Used (LFU) [96] and First In and First Out (FIFO) [97]. These methods have been widely used in wired networks which have enough storage and computing resources. Though they are of low complexity and easy to implement, these conventional approaches may suffer performance degradation in MEC systems, due to the fact that it is difficult for them to predict the future content popularity.

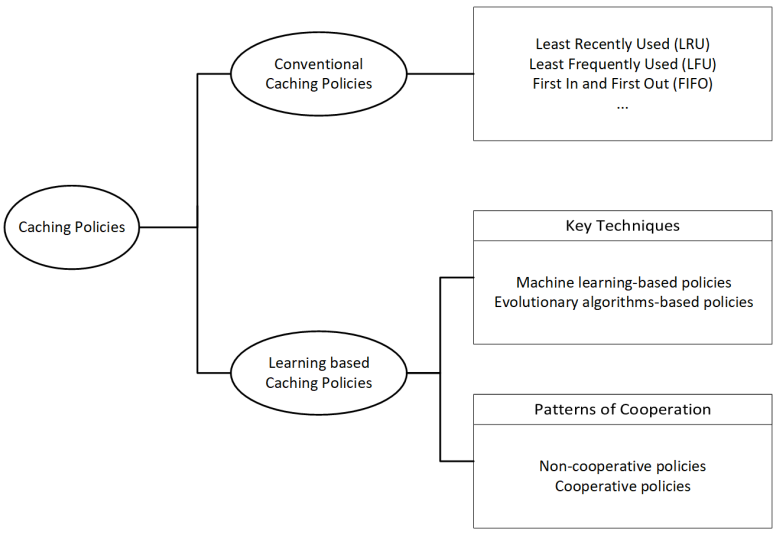

Figure 2.2: A taxonomy of caching polices in MEC systems.

2) **Learning based caching policies**

Compared to the conventional policies, learning based caching policies showcase the superiority in dealing with complex environmental dynamics of MEC systems [9]. In the following, we will discuss the state-of-the-art learning based caching policies from two aspects. First, the related research works are analysed according to the adopted key techniques, i.e., machine learning based policies and evolutionary algorithm (EA) based policies. Next, we categorise the previous works into non-cooperative policies and cooperative policies, and evaluate them respectively. The limitations in existing works are summarized at the end of this section, which emphasizes the research objectives proposed in Section 1.2.

- **Key techniques**

  Many existing works adopt machine learning techniques, especially RL [3, 85, 98, 99] and FL [63, 100, 101], to design caching policies.

  Authors in [98] develop a RL based user-assisted caching algorithm with a dynamic content library. In addition, a three-phase procedure at different time scales for joint cache placement and content delivery is introduced for small cell networks. To maximize the cache hit ratio, authors in [3] propose an online distributed cache replacement algorithm based on Thompson sampling (TS). Although both [98] and [3] show better performance in simulations compared to other benchmarks, these algorithms are developed under the assumption that user requests follow homogeneous distributions, which is less realistic in real-world scenarios, thereby limiting the practical applicability.

  In [100], an FL based mobility-aware proactive edge caching scheme

is proposed for vehicular networks. This scheme utilises a context-aware adversarial autoencoder model to estimate content popularity and proactively update the contents cached at the edge of vehicular networks. Besides, a mobility-aware cache replacement policy is integrated into the scheme to allow the network edges to add/evict contents in response to the mobility patterns and preferences of vehicles. [101] introduces an FL based proactive caching scheme to enhance user data privacy. This scheme adopts a hierarchical architecture in which each user trains a stacked autoencoder model using local data, and the server aggregates user-side model parameters to update a global model. Both [100] and [101] achieve satisfactory caching performance. However, such an FL-based training process inevitably involves a large amount of information exchange, which incurs intense computational cost. This situation will be more challenging for large-scale scenarios with an increased number of edge servers and end devices.

In addition to machine learning based policies, several articles leverage EAs to optimize the caching strategies [102, 103]. While EAs are not typically considered traditional "learning" algorithms in the sense of supervised or reinforcement learning, they inherently involve adaptation and learning to solve optimization problems. Specifically, EAs encompass the generation and evolution of a population of potential solutions across successive generations. By executing processes such as selection, mutation and crossover, these algorithms identify and enhance more effective solutions over time.

In [102], authors present an EA-based cooperative content caching and sharing scheme for a D2D communication system. The optimization objective focuses on enhancing the QoE for end users by identifying optimal D2D communication links with high mean opinion score, a metric indicating users' satisfaction levels. Subsequently, EA is adopted to solve this optimization problem. Simulation results demonstrate that the proposed scheme surpasses conventional D2D content dissemination approaches. Tang *et al.* [103] investigate the task caching strategy and fine-grained task migration strategy on the edge server. To achieve the goal of minimizing both task execution delay and energy consumption, the authors develop a method based on genetic algorithm. Though these research works indicate EA is an alternative approach to optimizing caching policies, the potential of EAs has not been fully explored in terms of collaborative edge caching. For example, the feasibility of combining EAs and machine

learning techniques to design caching strategies should be carefully studied.

- **Patterns of cooperation**

  According to the patterns of cooperation, we can divide the previous works into two categories, one of which designs the caching policies in a distributed manner, while the other enables the cooperation among edge servers to improve the caching performance. In the following, we will first discuss the previous works related to non-cooperative caching policies, and then investigate the works related to cooperative caching policies.

  Research [104] and [105] both develop caching strategies in a distributed manner, without considering additional information sharing among the servers. Although these approaches show superiority in terms of reducing the communication latency between local servers and users, the synergy of multiple servers within a system is not fully explored. Authors in [99] decompose the caching problem in MEC system into user-side cache optimization problem and BS-side cache optimization problem, and then develop a joint cache and delivery policy under the assumption that the policy on one side (either user or BS) is fixed while optimizing the policy on the other side. Nonetheless, the BS-side caching policy in this study is designed for each individual BS, with no exploration of inter-BS cooperation. [63] introduces a decentralized recommendation-enabled edge caching framework by leveraging soft actor-critic and FL. The proposed framework empowers individual edge server to learn its optimal policy locally and make caching decisions independently, without content sharing with other proximal servers. Besides, the caching policy designed for each server can only replace at most one content in each time slot, which imposes a substantial constraint on caching efficiency.

  Authors in [85] adopt multi-agent Multi-Armed Bandit (MAB) to design a cooperative content caching scheme for MEC system under the assumption that users' preference is unknown and only the content request history is observed. Similarly, [106] introduces an online collaborative caching strategy based on multi-agent MAB for small cell networks, optimizing the cache strategy in real-time under both stationary and non-stationary conditions. Both works indicate that the caching performance of MEC systems can be greatly improved through effective cooperation among edge servers.

  In addition to multi-agent MAB methods, deep learning techniques, such as DRL, have been well-exploited in collaborative edge caching

[99, 107–109].

In [107], a multi-agent actor-critic RL framework is introduced to tackle the edge caching problem where BSs have no knowledge of the content popularity distribution. Each BS, as one actor, makes its own caching decision based on a partially observable MDP while competing for chance to serve the covered users. Meanwhile, each BS cooperates with others to minimize the overall transmission delay. Deep Q Network (DQN) is applied on BS to optimize the BS-side caching policy in [99]. In [110], authors introduce a cooperative edge caching framework based on double DQN to minimize the long-term average content fetching delay. With the aim of maximizing the cache hit rate and minimizing the transmission delay, authors in [108] propose deep actor-critic RL based policies for both centralized and decentralized content caching. Qiao *et al.* [109] focus on the joint content placement and delivery in the vehicular edge networks. A cooperative caching scheme based on deep deterministic policy gradient (DDPG) is proposed to optimize the joint policy of content updating, vehicle scheduling, and bandwidth allocation.

Nonetheless, conventional DRL algorithms (e.g., DQN and actor-critic) face challenges in complex scenarios with high-dimensional state and action spaces. As a result, the works discussed above may experience performance degradation due to *the curse of dimensionality*, leading to scalability issues.

In [111], authors consider a two-level network caching scenario, where a parent node is connected to multiple distributed leaf nodes to jointly serve users. Specially, a scalable DQN approach is introduced to enable the parent node to learn-and-adapt to unknown policies of leaf nodes as well as dynamic evolution of file requests. Wu *et al.* [112] develop a soft actor-critic based algorithm to tackle the exponentially large action space in IoT sensing networks. The simulation results show that the proposed algorithm outperforms traditional DQN and actor-critic methods, especially when the number of edge nodes/sensors increases. However, the frequent model parameter exchanges between the cloud server and edge servers during model training incur significant transmission cost.

To summarize, the researches stated above reveal the advantages of using machine learning techniques, especially DRL and FL, to develop intelligent caching strategies for MEC systems, particularly for collaborative edge caching systems. On one hand, by integrating deep neural networks into RL, DRL enables caching entities to learn optimal caching decisions through trial

and error in complex, dynamic caching scenarios. On the other hand, limitations exist in DRL-based caching algorithms, as high-dimensional state and action spaces modeled for complex environments pose challenges to the scalability of deep neural networks. The utilization of FL in edge caching ensures the data privacy at distributed end devices, and enhances the cooperation among multiple servers through implicit communication via hierarchical model updates. Nonetheless, the problem of substantial signaling overheads caused by frequent information exchanges between the remote cloud and edge servers in the FL structure remains inadequately addressed. Furthermore, the integration of DRL and FL for developing sophisticated caching algorithms in collaborative edge caching is not fully explored. In addition to these machine learning based approaches, several articles have demonstrated the effectiveness of applying EAs to optimize caching strategies. However, to the best of our knowledge, there has been no comprehensive study on integrating EAs with machine learning techniques to develop effective caching strategies for collaborative edge caching systems.

Based on these open issues, the overarching goal of this thesis is to propose sophisticated approaches that leverage advanced EAs, DRL and/or FL techniques to address collaborative content caching and delivery problem in MEC systems. The detailed aims and objectives of the thesis are listed in Section 1.2.

Chapter 3

Preliminaries

This chapter provides an introduction to preliminary knowledge related to research objectives. A general overview of reinforcement learning (RL) is firstly introduced in Section 3.1, including the basic concepts of RL, Q-learning algorithm and deep Q network algorithm. Followed by the brief introduction of federated learning (FL) in Section 3.2, as well as multifactorial evolutionary algorithm (MFEA) in Section 3.3.

3.1 Reinforcement Learning

3.1.1 Basic concepts

Markov decision process

A Markov decision process (MDP) is a mathematical framework used to formulate RL problem of learning through interaction to achieve a goal. Typically, a MDP can be denoted as $(\mathcal{S}, \mathcal{A}, \gamma, P, r)$, where $\mathcal{S}$ denotes the state space of the environment, $\mathcal{A}$ denotes the set of action space, γ is the discount factor, P represents the state-transition probability that refers to the probability of transitioning between states given a specific action and is defined as $P(s'|s,a) = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$, and r denotes the reward function. A general agent-environment interaction process in RL problems is illustrated in Fig. 3.1 [11]. At each time slot $t \in \{0, 1, 2, 3, ...\}$, agent observes current state $s_t \in \mathcal{S}$ and takes action $a_t \in \mathcal{A}$ according to policy function $\pi(\cdot|s_t)$. After taking action a_t, the environment transits to the next state s_{t+1} based on the transition probability P. The agent receives a numerical reward r_t and finds itself in a new state s_{t+1}. The expected discounted return at time slot t is defined as

$$G_t = \mathbb{E}\left[\sum_{t_k=0}^{\infty} \gamma^{t_k} r_{t+t_k}\right], \tag{3.1}$$

where $\gamma \in [0, 1]$ determines the extent to which future rewards are taken into account for the current evaluation.

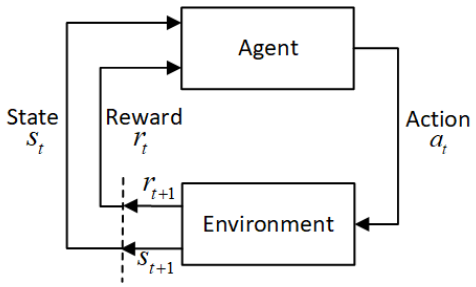

Figure 3.1: Illustration of agent-environment interaction in RL.

Policies and value functions

A policy π is a mapping from state s to the probabilities of selecting each potential action $a \in \mathcal{A}$ [11], which gives

$$a = \pi(s) \tag{3.2}$$

for deterministic policy and

$$a \sim \pi(\cdot|s) \tag{3.3}$$

for stochastic policy.

Value functions are defined with respect to these policies to help the agent evaluate the current situation. Two common value functions are state-value function $V_\pi(s)$ and action-value function $Q_\pi(s, a)$.

State-value function $V_\pi(s)$ is the expected return value when starting from s and subsequently following policy π, which is given by

$$V_\pi(s) = \mathbb{E}_\pi\big[G_t|s_t = s\big] = \mathbb{E}_\pi\left[\sum_{t_k=0}^{\infty} \gamma^{t_k} r_{t+t_k} \middle| s_t = s\right], \forall s \in \mathcal{S}, \tag{3.4}$$

where $\mathbb{E}_\pi[\cdot]$ represents the expected value of random variables under policy π, and t denotes any time slot.

The optimal state-value function is defined as

$$V^*(s) = \max_\pi V_\pi(s). \tag{3.5}$$

Similarly, the action-value function is defined as the expected return value for taking an action a in state s, and subsequently following policy π, which

is given by

$$Q_\pi(s,a) = \mathbb{E}_\pi\big[G_t | s_t = s, a_t = a\big] = \mathbb{E}_\pi\left[\sum_{t_k=0}^{\infty} \gamma^{t_k} r_{t+t_k} \middle| s_t = s, a_t = a\right]. \tag{3.6}$$

The optimal action-value function is defined as

$$Q^*(s,a) = \max_\pi Q_\pi(s,a). \tag{3.7}$$

3.1.2 Q-learning algorithm

Q-learning is recognized as a breakthrough in RL that was firstly proposed by Chris Watkins in 1989 [113]. It is a model-free, off-policy algorithm which evaluates each state-action pair using the action-value function, also known as Q-function. The Q-function is updated according to the Bellman equation, given by

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)], \tag{3.8}$$

where α is the learning rate.

Accordingly, all the Q-values for the state-action pair are stored in the Q-table, and the agent uses this table to decide which action to take in a given state. Furthermore, ϵ-greedy method is adopted in the process of action selection, as shown in Eq. 3.9, in order to balance exploration and exploitation.

$$a_{t+1} = \begin{cases} \arg\max_a Q(s_{t+1},a), & \text{with probability } 1-\epsilon, \\ \text{random } a \in \mathcal{A}, & \text{with probability } \epsilon, \end{cases} \tag{3.9}$$

where $\epsilon \in [0,1]$ determines the trade-off between exploration and exploitation. Specifically, a high ϵ value encourages exploration, while a low ϵ value

Algorithm 1: Q-learning Algorithm

```
Initialize Q(s,a), ∀a ∈ A, ∀s ∈ S, arbitrarily;
for every episode do
    repeat
        Initialize state s_t;
        repeat
            Choose and perform action a_t following ε-greedy method;
            Obtain r_t and observe s_{t+1} ;
            Update Q values according to Eq. (3.8);
            s_t ← s_{t+1}
        until s_t is terminal;
    until episode ends;
end
```

promotes exploitation. The general procedure of Q-learning algorithm is presented in Algorithm 1, adapted from [114]. Q-learning is notable for its simplicity and effectiveness in solving MDPs, making it a foundational RL algorithm and a basis for many subsequent developments in RL, such as deep Q network, which will be introduced in the next section.

3.1.3 Deep Q network

Deep Q network (DQN) is another crucial milestone in the history of RL that was introduced by Volodymyr Mnih and his colleagues in 2015 [115]. While Q-learning employs a Q-table to store Q-values for each visited state-action pair, DQN combines Q-learning with deep neural networks and uses a deep neural network to approximate Q-values. This enables DQN to handle high-dimensional or continuous state spaces, which are common in real-world applications like playing video games or controlling robots.

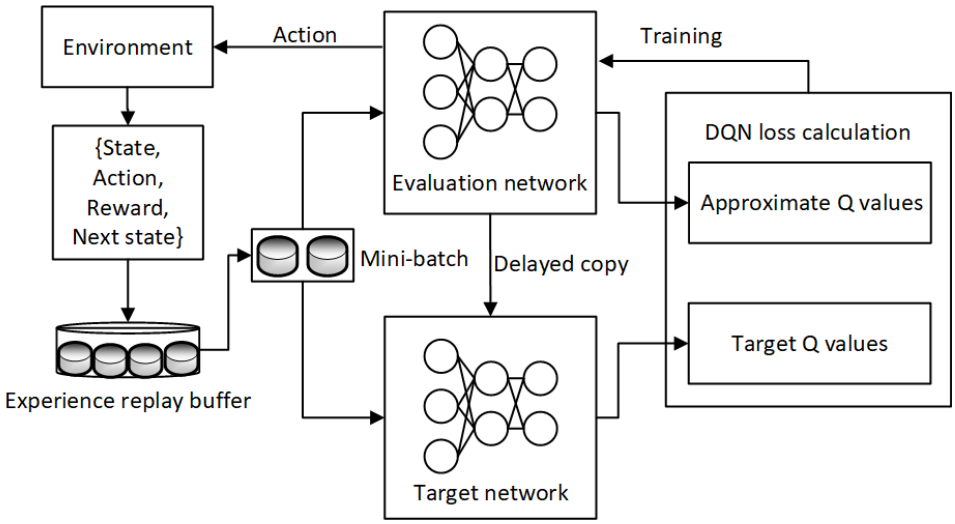

Figure 3.2: A diagram of DQN algorithm.

A diagram of DQN is shown in Fig. 3.2. There are two key components in DQN: deep neural networks and experience replay. Specifically, DQN employs two deep neural networks: the evaluation network and the target network. Both networks share the same structure but have different neural parameters. They take states as input and output Q-value estimates for all possible actions. The evaluation network, parameterized by $\boldsymbol{\theta}$, is trained to approximate Q value, denoted as $Q(s, a; \boldsymbol{\theta})$. The target network, parameterized by $\hat{\boldsymbol{\theta}}$, is used to obtain the target Q value for training purposes, denoted as $Q(s, a; \hat{\boldsymbol{\theta}})$. The experience replay buffer, denoted as $\mathcal{B}$, stores the agent's past experiences, denoted as (s, a, r, s'), where s represents the current state, a is the action taken in state s, r denotes the received instantaneous reward, and s' denotes the next state. During each training process, the agent randomly samples a mini-batch

of experiences from the replay buffer, i.e., $(s, a, r, s') \sim U(\mathcal{B})$, as the training data.

Accordingly, the loss function $L_i(\boldsymbol{\theta}_i)$ at each iteration i is defined as

$$L_i(\boldsymbol{\theta}_i) = \mathbb{E}\Big[(y_i - Q(s, a; \boldsymbol{\theta}_i))^2\Big], \tag{3.10}$$

where $y_i = \mathbb{E}[r+\gamma \max_{a'} Q(s', a'; \hat{\boldsymbol{\theta}}_i)]$ represents the target Q value at iteration i. Differentiating the loss function with respect to $\boldsymbol{\theta}$, one can write

$$\nabla_{\boldsymbol{\theta}_i} L_i(\boldsymbol{\theta}_i) = \mathbb{E}\Big[\big(r + \gamma \max_{a'} Q(s', a'; \hat{\boldsymbol{\theta}}_i) - Q(s, a; \boldsymbol{\theta}_i)\big)\nabla_{\boldsymbol{\theta}_i} Q(s, a; \boldsymbol{\theta}_i)\Big]. \tag{3.11}$$

Subsequently, the evaluation network uses stochastic gradient descent algorithm to update its parameter $\boldsymbol{\theta}$, as follows

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \nabla_{\boldsymbol{\theta}_i} L_i(\boldsymbol{\theta}_i). \tag{3.12}$$

Note that the target network parameter $\hat{\boldsymbol{\theta}}$, is periodically synchronized with the evaluation network parameter $\boldsymbol{\theta}$, to improve learning stability. The detailed DQN procedure is presented in Algorithm 2, adapted from [115].

Algorithm 2: Deep Q Network with experience replay

Initialize replay buffer $\mathcal{B}$;
Initialize the evaluation network $Q(s, a; \boldsymbol{\theta})$ with random weights $\boldsymbol{\theta}$;
Initialize the target network $Q(s, a; \hat{\boldsymbol{\theta}})$ with weights $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$;
for *t=1,2,...,T* **do**
    Choose and perform action a_t following ϵ-greedy method;
    Obtain r_t and observe s_{t+1} ;
    Store transition (s_t, a_t, r_t, s_{t+1}) into $\mathcal{B}$;
    Sample random mini-batch of transitions (s_j,a_j,r_j,s_{j+1}) from $\mathcal{B}$;
    Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \hat{\boldsymbol{\theta}})$;
    Update the evaluation network parameter $\boldsymbol{\theta}$ by performing Eq. (3.12) ;
    if $t \mod K = 0$ **then**
        Replace the parameters of target network $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$;
    end
end

3.2 Federated Learning

The concept of FL was proposed by Google in 2017 [116]. The key idea is to build machine learning models directly on distributed devices while ensuring the privacy of local data. FL is particularly well-suited for the following scenarios: 1) When dealing with large and privacy-sensitive data, avoiding the

upload of data to a centralized data center for model training is preferable. 2) In cases where the data generated by individual users varies, and a specific user's data may not be representative of the overall population distribution. This type of data is often referred to as Non-IID and unbalanced data, making it necessary to train models locally to provide more personalized and intelligent services for individual users.

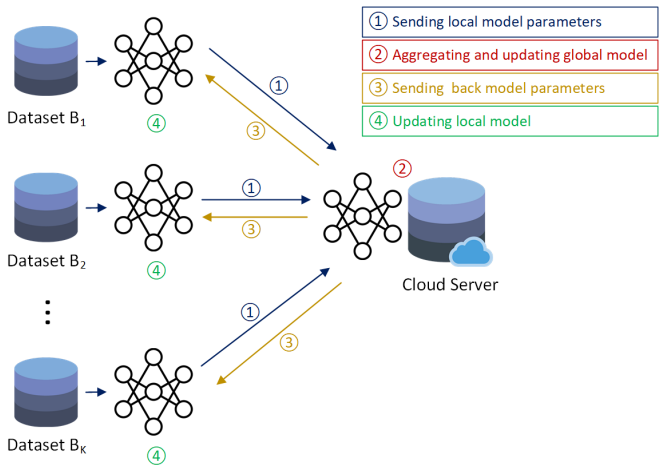

Figure 3.3: A typical architecture for FL system [2].

A typical architecture for FL system is illustrated in Fig. 3.3 [2]. In this system, K clients, sharing the same data structure, collaboratively learn a machine learning model with the assistance of a cloud server. The training process usually contains the following four steps:

- Step 1: Clients compute the training gradients based on their local data, and then send local model parameters to cloud server;
- Step 2: Server performs model aggregation on the global model;
- Step 3: Server sends back the aggregated model parameters to clients;
- Step 4: Clients update their individual models with the aggregated model parameters.

The above process repeats until the loss function converges. This architecture is independent of specific machine learning algorithms and the final model parameters will be shared among all clients.

3.3 Multifactorial Evolutionary Algorithm

In general, multifactorial evolutionary algorithm (MFEA) is designed as a cloud-based multitasking solver, which can solve multiple optimization problems simultaneously. Whereas, the traditional evolutionary algorithms solves a single optimization problem at a time [117, 118].

Consider a scenario with M optimization tasks to be performed simultaneously. Each task, denoted as $m \in \{1, 2, ..., M\}$, is defined over a search space $\mathcal{X}^m$ with dimension $D^m = |\mathcal{X}^m|$. MFEA is developed to simultaneously find the solutions:

$$\mathbf{x}^{m^*} = \arg_{\mathbf{x} \in \mathcal{X}^m} \min J^m(\mathbf{x}), \forall m \in \{1, 2, ..., M\}, \tag{3.13}$$

where $J^m : \mathcal{X}^m \mapsto \mathbb{R}$ is the objective of task m.

Firstly, according to evolutionary computation principles, a population of candidates $\{\mathbf{x}^i\}_{i=1}^I$ is defined. Note that the individuals $\mathbf{x}^i$ are encoded in a unified search space $\mathcal{X}^U$, and can be decoded to a solution $\mathbf{x}^{i,m} \in \mathcal{X}^m$ related to task m. The dimensionality of the unified search space is defined as $|\mathcal{X}^U| = \max_m\{D^m\}$, i.e., the maximum among the dimensions of considered tasks. In order to find task-specific solutions in a multitasking environment, several definitions need to be specified [118, 119]:

1. *Factorial Cost* ($\Psi^{i,m}$): For a given task m, the factorial cost of a candidate $\mathbf{x}^i$ is given by $\Psi^{i,m} = \lambda \cdot \delta^{i,m} + J^m(\mathbf{x}^{i,m})$, where λ is a penalizing multiplier, $\delta^{i,m}$ is the total constraint violation of $\mathbf{x}^i$ at task m, and $J^m(\mathbf{x}^{i,m})$ is the objective value of $\mathbf{x}^i$ with respect to task m. Accordingly, we have $\Psi^{i,m} = J^m(\mathbf{x}^{i,m})$ if candidate $\mathbf{x}^i$ is feasible with respect to task m.

2. *Factorial Rank* ($\rho^{i,m}$): It is simply the rank of candidate $\mathbf{x}^i$ in task m after sorting the population in ascending order with respect to $\Psi^{i,m}$.

3. *Skill Factor* (κ^i): It is the task in which the individual $\mathbf{x}^i$ performs best, i.e., $\kappa^i = \arg_{m \in \{1,2,...,M\}} \min \rho^{i,m}$.

4. *Scalar Fitness* (ζ^i): The scalar fitness of $\mathbf{x}^i$ is given by $\zeta^i = 1/\rho^i_j$, where $j = \kappa^i$.

Based on the scalar fitness of each individual, performance comparison can be carried out by assuming that individual $\mathbf{x}^i$ surpasses individual $\mathbf{x}^{i'}$ in multitasking sense if $\zeta^i > \zeta^{i'}$. Accordingly, the basic structure of MFEA is drawn based on the classic evolutionary algorithms, as shown in Algorithm 3 [118].

The significant deviations of MFEA from traditional evolutionary algorithms lie in the off-spring evaluation process (i.e., selection based on scalar fitness and skill factors), as well as the knowledge exchange among tasks, namely, *assortative mating* and *vertical cultural transmission* [118], refer to Lines 5-6 of Algorithm 3. *Assortative mating* indicates that the parent candidates prefer to mate with those have the same skill factor. In contrast, if two candidates

Algorithm 3: Multifactorial Evolutionary Algorithm (MFEA)

Generate an initial population of candidates $\{\mathbf{x}^i\}_{i=1}^{I}$;
Compute factorial costs $\Psi^{i,m}, \forall m, i$ of every individual with respect to each task in the multitasking environment;
Compute skill factor κ^i of every individual;
while *termination criterion not satisfied* **do**
    Apply genetic operators (crossover and mutation) on $\{\mathbf{x}^i\}_{i=1}^{I}$ to generate an off-spring population $\{\mathbf{x}^{\tilde{i}}\}_{\tilde{i}=1}^{\tilde{I}}$;
    Evaluate each $\{\mathbf{x}^{\tilde{i}}\}$ on its' parents skill factor;
    Concatenate $\{\mathbf{x}^i\}_{i=1}^{I}$ and $\{\mathbf{x}^{\tilde{i}}\}_{\tilde{i}=1}^{\tilde{I}}$ to form a merged candidates;
    Update the skill factor κ^i and scalar fitness ζ^i of merged candidates;
    Select I fitness individuals from merged candidates to form the next generation of population;
end

have different skill factors, crossover (mating) only occurs as per a predefined random mating probability, or else mutation arises. *Vertical cultural transmission* states a phenomenon of genetic inheritance such that the phenotype of an off-spring will be directly affected by the phenotype of its parents. In MFEA, this inheritance is realized by enforcing the off-spring to be evaluated only on one of its parents' skill factors, i.e., one optimization task. Such an operation makes MFEA more competitive in terms of computational complexity. Once the evaluation of the off-spring are done, they will be combined with those from previous generation. Then selection based on scalar fitness ζ^i is carried out to obtain the best I individuals as the population of the next generation.

Chapter 4

A Generic System Model

In this chapter, we will present a generic system model, which is applicable to subsequent chapters.

As depicted in Fig. 4.1, we focus on a collaborative edge caching system consisting of one cloud center that stores a total number of F files, and M edge servers. Let $\mathcal{F} = \{1, 2, ..., F\}$ and $\mathcal{M} = \{1, 2, ..., M\}$, respectively, denote the set of files and the set of edge servers. All edge servers are connected to the cloud center via backhaul links. Each edge server is co-located with one BS and equipped with a limited cache storage of size $N^m, m \in \mathcal{M}$. Furthermore, each edge server equipped with a learning model acts as an intelligent agent to make caching decisions according to the limited storage capacity as well as dynamic user requests and content popularity.

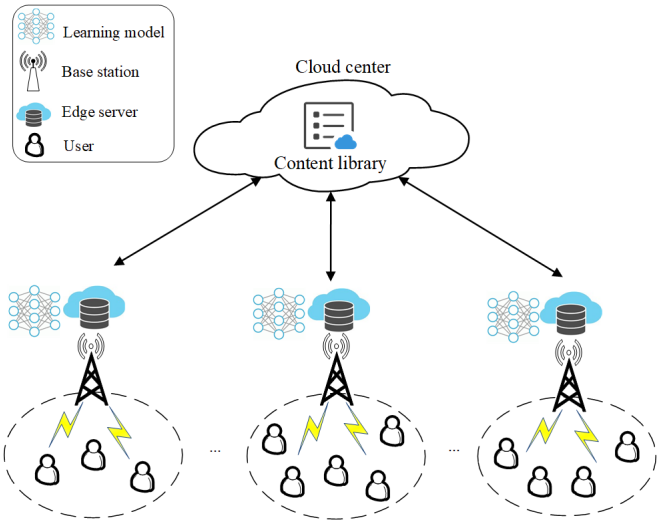

Figure 4.1: The generic system model.

In our collaborative edge caching system, each edge server is capable of sharing cached files to the neighbouring servers. This means that user requests can be satisfied by either the associated server or by other neighbouring servers, given that requested files are cached in the servers. We refer to this situation as a *cache hit*. Otherwise, the requested files need to be fetched from

the remote cloud center, which is referred to as a *cache miss*.

Building upon this generic model, we will focus on three key issues of collaborative edge caching system in the subsequent chapters, including collaboration in heterogeneous scenarios, network scalability, as well as joint cache update and request delivery problem in scalable edge networks.

We note here that there exists variations in this generic model across the following three chapters, including different ways of cooperation (e.g., content sharing). We will explicitly specify these variations in subsequent chapters. Unless stated otherwise, the definitions and notations of variables remain consistent with this generic model.

Chapter 5

Evolutionary Deep Q Network for Collaborative Edge Caching

5.1 Overview

In this chapter, we focus on reaching the first aim of our research, i.e., design a collaborative caching strategy for a MEC system with heterogeneous caching scenarios. Accordingly, we model the collaborative caching problem as a RL-based problem at edge servers, and define a latency-sensitive reward function, aiming at maximizing the accumulated reward over a long-term horizon. Instead of training these RL models at distributed edge servers independently, we decompose the formulated problem into multiple optimization tasks, and develop a novel algorithm based on MFEA to jointly solve these multiple optimization tasks. The main contributions of this work are summarized as follows.

- We introduce an evolutionary DQN (eDQN) algorithm for a MEC system with heterogeneous caching scenarios. By integrating MFEA with DQN, the proposed eDQN algorithm can collaboratively optimize the caching policies by simultaneously solving multiple optimization tasks at distributed edge servers. To the best of our knowledge, we are the first to apply MFEA-based method to solve the collaborative edge caching problem.

- We numerically evaluate our proposed algorithm based on a real-world dataset from MovieLens. Experimental results show that our proposed algorithm can optimize caching policies in different scenarios collaboratively, and thus improve cache hit rate and reduce content downloading latency significantly, compared to other popular caching schemes.

The structure of rest of this chapter is organized as follows. The system model is introduced in Section 5.2, followed by the problem formulation in

Section 5.3 and technical details of the proposed eDQN algorithm in Section 5.4. Next, the experimental results are discussed in Section 5.5 and final conclusions are drawn in Section 5.6.

5.2 System Model

Building upon the generic system model in Chapter 4, we focus on the collaborative caching domain and heterogeneous caching scenarios, as shown in Fig. 5.1.

A set of edge servers with limited storage are deployed in different scenarios. For instance, a park with relatively sparse crowds and high user mobility, a city center with dense crowds and moderate user mobility, or a concert hall with ultra-dense crowds and low user mobility. As a result of different user density and mobility in different scenarios, the content popularity distributions among these scenarios will be various. In this system, edge servers in different scenarios form a collaborative caching domain to provide contents for associated users. Within the caching domain, edge servers can collaboratively update their caching policies by means of knowledge transfer among their learning models. Besides, apart from being the content provider, cloud center also acts as a general solver to perform model evolving from those built on each edge server. Details of the general solver will be presented in Section 5.4.

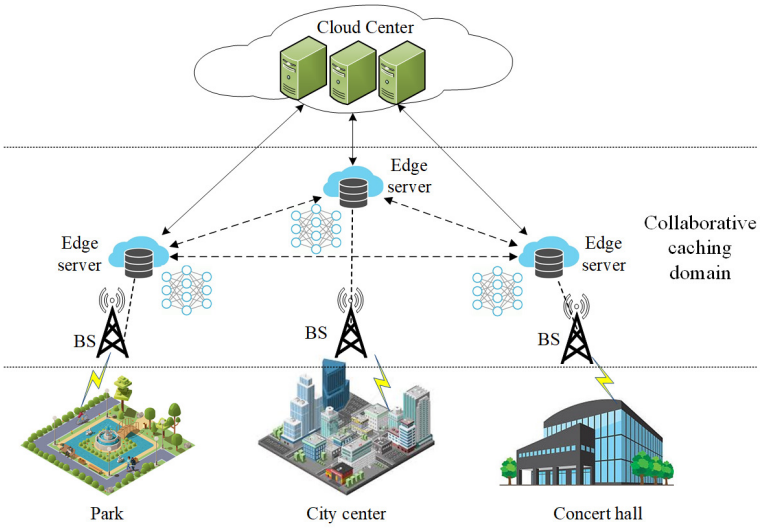

Figure 5.1: A MEC system with heterogeneous caching scenarios.

We assume that all the content have the same size, each edge server has the same storage capacity and can store up to N content files, the set of cached files at server m is denoted by $\mathcal{N}^m = \{1, 2, ..., N\}$.

Consider a finite time horizon of T with $t = 1, 2, ..., T$ denoting the indices for the discrete-time slots, and a total number of U users in a collaborative

caching domain. Define the set of U users in a domain as $\mathcal{U} = \{1, 2, ..., U\}$ and let $\mathcal{U}_t^m \subset \mathcal{U}$ denote the set of those users served by the edge server m at time slot t. We assume that $\mathcal{U}_t^m$ remains unchanged during each considered time slot.

As stated above, content popularity in different scenarios are varying due to the various user density and mobility. Moreover, content popularity in each scenario is temporally dynamic. Accordingly, we define content popularity of file f at time slot t in a specific scenario as

$$v_t^f = \frac{1}{|\mathcal{U}_t^m|} \sum_{u=1}^{|\mathcal{U}_t^m|} p_{f|u} \cdot \phi_t^f, \tag{5.1}$$

where $p_{f|u} \in [0, 1]$ denotes user's preference, i.e., the conditional probability that file f is requested by user u given that this user has a request, ϕ_t^f denotes the the number of requested times of file f at time slot t, and $|\mathcal{U}_t^m|$ is the number of users connected to edge server m at time slot t. We consider a realistic scenario that $p_{f|u}$ and ϕ_t^f are unknown in advance, while only the content request history is observed.

We denote $Y_t^{m,u,f}$ as the downloading latency of user $u \in \mathcal{U}_t^m$ fetching file f from the associated server m, $Y_t^{n,u,f}$ as the downloading latency of user $u \in \mathcal{U}_{m,t}$ fetching file f from a server $n, n \in \mathcal{M}, n \neq m$, $Y_t^{c,u,f}$ as the downloading latency of user u fetching file f from the cloud center. It is assumed that $Y_t^{m,u,f} < Y_t^{n,u,f} < Y_t^{c,u,f}$. Accordingly, the downloading latency incurred for user u fetching file f can be expressed as

$$Y_t^{u,f} = \begin{cases} Y_t^{m,u,f}, \text{ if } f \in \mathcal{N}_m, \\ Y_t^{n,u,f}, \text{ if } f \notin \mathcal{N}_m \text{ and } f \in \mathcal{N}_n, \\ Y_t^{c,u,f}, \text{ otherwise.} \end{cases} \tag{5.2}$$

5.3 Problem Formulation

With the system model introduced above, we focus on the collaborative edge caching problem in a specific domain with M edge servers, each serving a corresponding scenario.

For each scenario, we define a RL model which comprises an agent, i.e., the edge sever, the environment denoted by E where a task is defined (here the task corresponds to content caching problem), the state space $\mathcal{S}$, the action space $\mathcal{A}$ and the reward function denoted by $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Accordingly, the state, the action and the reward for the agent are defined as follows.

State: The state $\mathbf{s}_t \in \mathcal{S}$ for the edge server at time slot t is defined as

$$\mathbf{s}_t = \left[\mathbf{v}_t^{Req}, \mathbf{v}_t^{Cache}\right]^\top, \tag{5.3}$$

where $\mathbf{v}_t^{Req} = \left[v_t^{Req,1}, v_t^{Req,2}, ..., v_t^{Req,I}\right]$ denotes the feature vector of the received requests at time slot t, with elements $v_t^{Req,i}, i = 1, 2, ..., I$ representing the temporary content popularity of requested files. Similarly, $\mathbf{v}_t^{Cache} = \left[v_t^1, v_t^2, ..., v_t^N\right]$ is the feature vector of cached files at time slot t.

Action: Let $\mathbf{a}_t \in \mathcal{A}$ denote the cache update action, expressed as

$$\mathbf{a}_t = \left\{\Delta\mathcal{N}^{m,f} \in \{0,1\} \,\middle|\, f \in \mathcal{N}^m, m \in \mathcal{M}\right\}, \tag{5.4}$$

where $\Delta\mathcal{N}^{m,f} = 0$ means replacing the file f with a new request, and $\Delta\mathcal{N}^{m,f} = 1$ means maintaining the file f.

Reward: We use the negative value of downloading latency as the instantaneous reward for the agent m, given by

$$r_t^m \triangleq -\sum_{u \in \mathcal{U}^m} \sum_{f \in \mathcal{F}} Y_t^{u,f}. \tag{5.5}$$

The aim is to find an optimal edge caching policy π^m for each server $m \in \mathcal{M}$ within a collaborative caching domain by maximizing the total rewards $r_t^m, m \in \mathcal{M}$, over a finite time horizon of T, simultaneously and taking into account that each server is associated with a different caching scenario. To this end, we formulate the problem as

$$\begin{aligned} &\max_{\pi^m, m \in \mathcal{M}} \sum_{t=1}^{T} \sum_{m \in \mathcal{M}} r_t^m, \\ &\text{s. t.} \sum_{f \in \mathcal{F}} \Delta\mathcal{N}^{m,f} \leq N, \forall m \in \mathcal{M}, \\ &\qquad \Delta\mathcal{N}^{m,f} \in \{0,1\}, \forall m \in \mathcal{M}, f \in \mathcal{F}. \end{aligned} \tag{5.6}$$

Due to the look-ahead variables over a finite time horizon of T, which implies unavailability of some future information at present and the presence of integer-valued constrains, the optimization problem (5.6) cannot be solved via conventional methods. At this point, the well-known DQN algorithm can be applied on each edge server m, in which the policy π^m is encoded in the weights and biases of a deep neural network. In the traditional DQN, an optimal policy specialized for one agent will be learned over several training epochs from interacting with environment. However, such an approach imposes limitations on the situation wherein multiple DQN agents involve and need to communicate and learn polices collaboratively, as in our case. There-

fore, we opt to employ an evolutionary multitasking optimization algorithm to jointly evolve multiple DQN models, aiming at finding effective policies for edge servers and thereby solve problem (5.6).

5.4 Algorithm Design

In this section, we introduce an evolutionary DQN (eDQN) algorithm that benefits from MFEA as a multitasking solver and DQN modelling to account for the foresighted nature of the problem. The detailed procedural steps of the proposed eDQN algorithm at the edge servers and the cloud center are shown in Algorithm 4 and Algorithm 5, respectively.

Specifically, we apply the classical DQN algorithm to constructing DQN models for distributed edge servers. Meanwhile, MFEA is exploited to collaboratively optimize caching policies for multiple servers. For the details of the DQN modelling, we refer to Section 3.1.3. In the sequel, we provide a more detailed explanation of our MFEA-based optimization approach to solving problem (5.6).

First, we decompose the problem (5.6) into M optimization tasks $\{K^m\}_{m=1}^M$. The tasks are defined on the same environment E with different configurations, corresponding to varying content popularity and users' preferences in diverse caching scenarios. For each task K^m, a DQN model $M_{\boldsymbol{\theta}^m}$ with neural parameters $\boldsymbol{\theta}^m$ is constructed over N_{test} test episodes, expressed as

$$\text{Task } K^m : \max_{\boldsymbol{\theta}^m \in \mathbb{R}^{D^m}} \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} r\big(M_{\boldsymbol{\theta}^m}; E, n\big), \tag{5.7}$$

where the $r(M; E, n)$ represents the final reward obtained by model M on environment E at n-th test episode, $D^m = |\mathcal{X}^m|$ is the dimension of the search space $\mathcal{X}^m$ in which all possible solutions $\boldsymbol{\theta}^m$ to task K^m are embedded. Then the constructed task K^m is uploaded to the cloud center.

Second, we propose an *evaluate-and-evolve* strategy based on MFEA to efficiently execute the tasks $\{K^m\}_{m=1}^M$ at both the cloud center and edge server sides. A block diagram of the *evaluate-and-evolve* process is illustrated in Fig. 5.2.

Let us define a unified search space $\mathcal{X}^U$ at the cloud center consisting of the shared layers $\mathcal{X}^{U_{sh}}$ and the individual layers $\{\mathcal{X}^{U_m}\}_{m=1}^M$ of the uploaded DQN models $\{M_{\boldsymbol{\theta}^m}\}_{m=1}^M$ from the edge servers, as depicted in Fig. 5.3. This design criterion is inspired by transfer learning, which reuses a fraction of the parameter values of a pre-trained model to some layers of the model of target task, whereas the parameters of the rest of layers of the target task are trained accordingly via back-propagation [120]. A similar architecture has been used

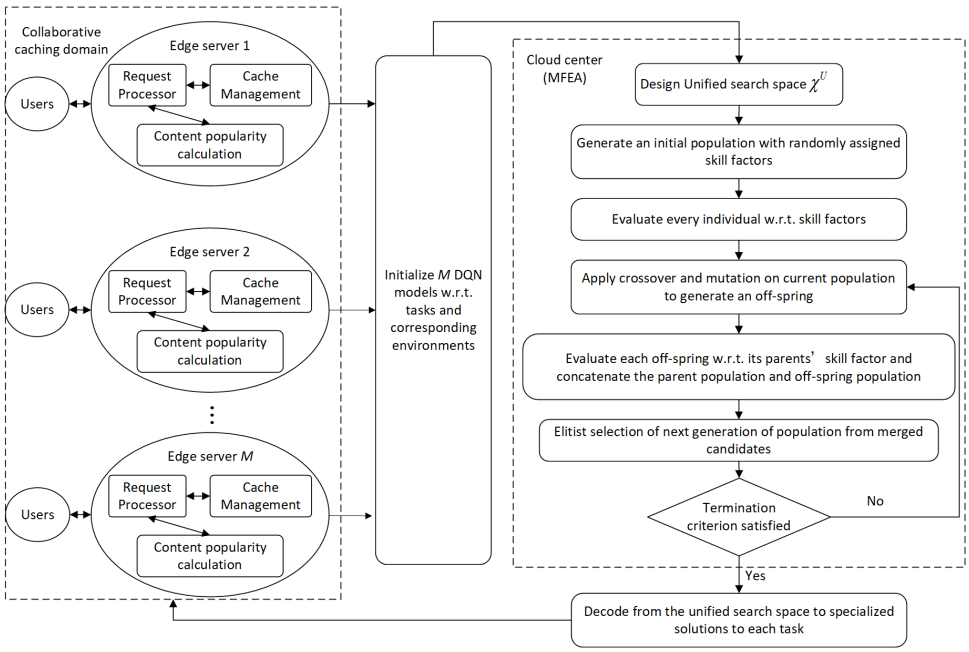

Figure 5.2: A block diagram of the *evaluate-and-evolve* process.

in [121] for a different scenario. The first part $\mathcal{X}^{U_{sh}}$ consists of the neural parameters of the L^{sh} first layers of all models $\{M_{\boldsymbol{\theta}^m}\}_{m=1}^{M}$, meaning that the value of these parameters will be the same among all models; the second part $\{\mathcal{X}^{U_m}\}_{m=1}^{M}$ consists of M elements, each embedding the last layer of model $M_{\boldsymbol{\theta}^m}$ which is specialized for task K^m and not shared among different tasks. Therefore, the dimensionality of the unified search space $\mathcal{X}^U$ is given by

$$\left|\mathcal{X}^U\right| = \sum_{l=1}^{L^{sh}} \max_m \left|\boldsymbol{\theta}^m\right| + \sum_{m=1}^{M} \sum_{l=L^{sh}+1}^{L^m} \left|\boldsymbol{\theta}^m\right|, \tag{5.8}$$
$$\boldsymbol{\theta}^m \in \text{layer } l \text{ of } M_{\boldsymbol{\theta}^m},$$

where L^m is the number of layers of $M_{\boldsymbol{\theta}^m}$.

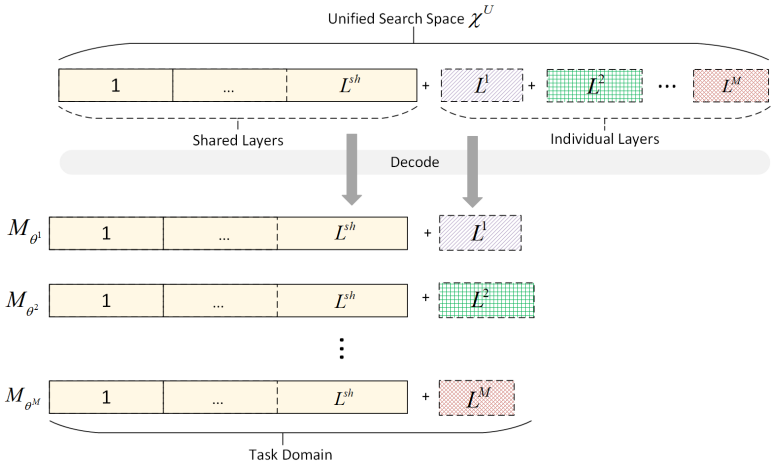

Figure 5.3: Illustration of unified search space.

Algorithm 4: Procedure at each edge server m

Initialize a DQN model $M_{\boldsymbol{\theta}^m}$ with neural parameters $\boldsymbol{\theta}^m$ with respect to task K^m and the corresponding environment E according to each caching scenario;
Upload the model $M_{\boldsymbol{\theta}^m}$ to the cloud center;
Obtain a well-trained DQN model $M_{\boldsymbol{\theta}^{m*}}$ representing the optimal policy π^{m^*} from the cloud center ;
while $t = 1, 2, ..., T$ **do**
    Receive the requests from users $\forall u \in \mathcal{U}_t^m$, and observe the current state $\mathbf{s}_t$;
    Compute the downloading latency $Y_t^{u,f}$ using Eq. (5.2), and obtain the reward r_t^m using Eq. (5.5);
    Take action $\mathbf{a}_t = \pi^{m^*}(\mathbf{s}_t)$ using policy π^{m^*} ;
    Update the cache status and observe the next state $\mathbf{s}_{t+1}$;
    $t \leftarrow t + 1$
end

Algorithm 5: Procedure at the cloud center

Receive all models $\{M_{\boldsymbol{\theta}^m}\}_{m=1}^M$ from edge servers, and establish a unified search space $\mathcal{X}^U$ accordingly ;
Set the population size I, the number of generations G, the stopping flag $g = 1$;
Generate an initial population of candidates $\{\mathbf{x}^i\}_{i=1}^I$;
Evaluate every individual with respect to each task $\{K^m\}_{m=1}^M$;
Compute skill factor κ^i of each individual;
while $g \leq G$ **do**
    Apply crossover and mutation on $\{\mathbf{x}^i\}_{i=1}^I$ to generate an off-spring population $\{\mathbf{x}^{\tilde{i}}\}_{\tilde{i}=1}^{\tilde{I}}$;
    Evaluate each $\{\mathbf{x}^{\tilde{i}}\}$ on its' parents skill factor;
    Concatenate $\{\mathbf{x}^i\}_{i=1}^I$ and $\{\mathbf{x}^{\tilde{i}}\}_{\tilde{i}=1}^{\tilde{I}}$ to form a merged candidates;
    Update the skill factor $\kappa^i, \forall i$ and scalar fitness $\zeta^i, \forall i$ of merged candidates;
    Select I fitness individuals from merged candidates to form the next generation of population;
    $g \leftarrow g + 1$
end
Generate optimal task-specific solutions $\{\boldsymbol{\theta}^{m^*}\}_{m=1}^M$ by decoding from $\mathcal{X}^U$ and return each solution to the corresponding edge server;

Next, an initial population of candidates $\{\mathbf{x}^i\}_{i=1}^I$ is generated on $\mathcal{X}^U$. Note here that the candidates $\{\mathbf{x}^i\}_{i=1}^I$ can be decoded to a solution $\mathbf{x}^{i,m} \in \mathcal{X}^m$ related to task K^m, i.e., $\mathbf{x}^{i,m} \equiv \boldsymbol{\theta}^m$, by concatenating the shared parameters in $\mathcal{X}^{U_{sh}}$ and those in $\mathcal{X}^{U_m}$, as shown in Fig. 5.3. Once decoded, these initial parent candidates will be evaluated on each task to obtain the factorial cost $\Psi^{i,m}, \forall i, m$ and the scalar fitness $\zeta^i, \forall i$, see Line 4 of Algorithm 5. The evaluation of candidates with respect to each task K^m is executed on each edge

server m by performing Eq. (5.7). Likewise, the same evaluation process will be executed when evaluating the off-spring candidates, i.e., refer to line 8 of Algorithm 5. Particularly, the averaged reward obtained by performing Eq. (5.7) is interpreted as the scalar fitness of the candidate, which is used to compare each candidate with respect to task K^m afterwards.

Following the principles of *assortative mating*, we apply crossover and mutation operators on these parent candidates to generate an off-spring population $\{\mathbf{x}^{\tilde{i}}\}_{\tilde{i}=1}^{\tilde{I}}$. Then each off-spring $\mathbf{x}^{\tilde{i}}$ will be evaluated on a specific task, namely, the task in which its parents perform the best, according to the criterion of *vertical cultural transmission*. After evaluation, the off-spring candidates are combined with the parent candidates to form a group of candidates. Finally, a selection based on scalar fitness $\zeta^i, \forall i$ of the merged candidates is executed, from which the best I candidates will be retained as the next generation of population. We refer to the above process as one round of *evaluate-and-evolve*, as shown in lines 6-13 of Algorithm 5. The fitness candidate in terms of each of the M tasks will finally be obtained after several rounds of *evaluate-and-evolve*.

Once the fitness candidates are obtained, we can decode the task-specific solutions $\{\boldsymbol{\theta}^{m^*}\}_{m=1}^{M}$ from the unified search space $\mathcal{X}^U$. Accordingly, DQN model with parameters $\boldsymbol{\theta}^{m^*}, \forall m \in \mathcal{M}$ is applied on each related edge server to solve the edge caching problems in the collaborative caching domain, as shown in lines 3-10 of Algorithm 4. Note here that the optimized policy π^{m^*} will act as a greedy policy once applied to the edge server.

A key feature of the proposed algorithm, i.e., the knowledge transfer among tasks $\{K^m\}_{m=1}^{M}$, is implicitly achieved via the shared partition $\mathcal{X}^{U_{sh}}$ of the unified search space and the evolutionary operators (crossover and mutation) at the cloud center side. Such operations enable a DQN model to gain knowledge from other models so as to improve the related policies cooperatively. Ideally, effective knowledge transfer will occur with higher possibility among tasks defined on the environment with similar configurations.

5.5 Simulation Results

We use a real-world dataset, MovieLens 100K Dataset [84], to evaluate our proposed eDQN algorithm. This dataset records 100000 ratings (scored on a scale from 1 to 5) by 943 users on 1682 movies in the form of (User ID, Movie ID, Rating, Timestamp). Besides, the demographic information about users is provided with format (User ID, Gender, Age, Occupation, Zip-code).

We assume a movie rating at a certain time in the dataset corresponds to a user's content request at that time. In our simulations, we only use the data

related to the movies that have been requested more than 180 times, i.e., $F = 137$. We set the number of edge servers $M = 3$, and then classify the users into 3 different groups according to their zip-code, so as to simulate 3 different scenarios within a caching domain. We further normalize the movies' rating scores to $[0, 1]$ and use these normalized values as users' preference $p_{f|u}, \forall f, u$. Accordingly, the content popularity of each file is computed using Eq. (5.1).

The DQN models for edge servers are constructed with an identical architecture: one input layer, two hidden layers (i.e., $L^{sh} = 2$ shared layers, the first with 256 units, the second with 128 units, both layers use ReLu activations), and one output layer with sigmoid activation. Besides, we set discount factor $\gamma = 0.9$, the downloading latency $Y_t^{m,u,f} = 1, Y_t^{n,u,f} = 2, Y_t^{c,u,f} = 10$. The configuration of MFEA is fixed as: the population size $I = 20$, the number of generations $G = 6$. The scalar fitness of candidate is computed as the averaged final reward obtained by the candidate over $N_{test} = 5$ episodes on corresponding environment.

We compare the performance of proposed algorithm with other three baselines: 1) *Thompson Sampling (TS)* [122], which is widely used in multi-armed bandit problem. It assumes that a value for each file is sampled from beta distribution with two parameters: successes and fails. In each trail, the beta distribution parameters are updated based on cache hit or cache miss, and the file with highest value is selected to be cached. 2) *Least Frequently Used (LFU)* [96], which will replace the cached item that has lowest request frequency with the new request. 3) *Least Recently Used (LRU)* [95], which will replace the cached item that is requested least recently with the new request.

In the following experiments, we employ two performance metrics: cache hit rate, defined as the ratio of cache hits to the number of content requests received by a server, and average downloading latency, defined as the ratio of the sum of downloading latency for all content requests to the total number of requests within a domain.

We first evaluate the cache hit rate for each caching scenario with varying cache sizes, as shown in Fig. 5.4, Fig. 5.5, Fig. 5.6, respectively. As evident from these results, the cache hit rate increases as the cache size of each server grows. The proposed algorithm achieves varying degrees of cache hit rates under different scenarios, consistently outperforming the other three baselines across all caching scenarios. This superiority can be attributed to its consideration of both the dynamic content request frequency and users' preferences while learning the policy. Additionally, the proposed algorithm takes advantage of collaborations among edge servers through joint model evolution and contents sharing, which also indicates the effectiveness of the proposed algorithm in dealing with heterogeneous scenarios.

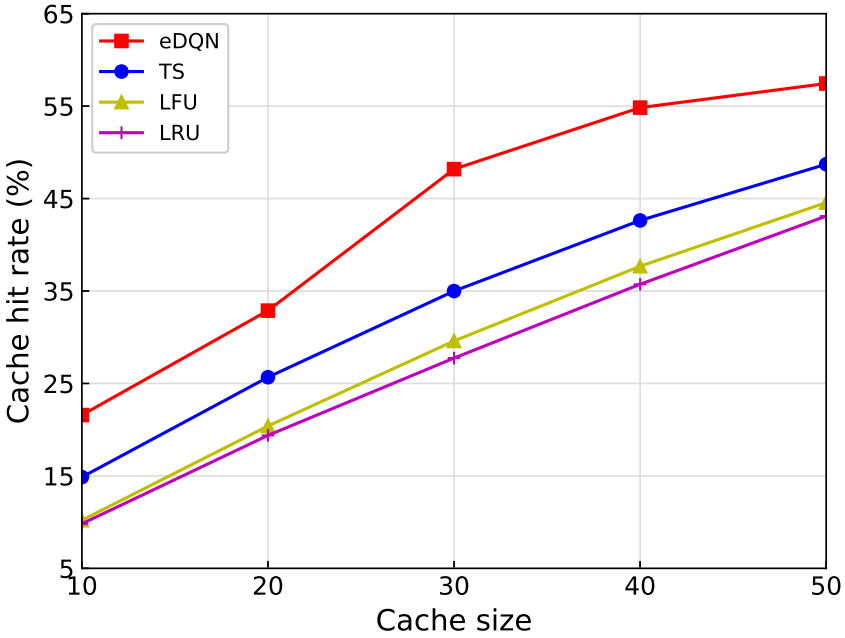

Figure 5.4: Cache hit rate of the proposed eDQN algorithm, TS, LFU and LRU in scenario served by server 1.

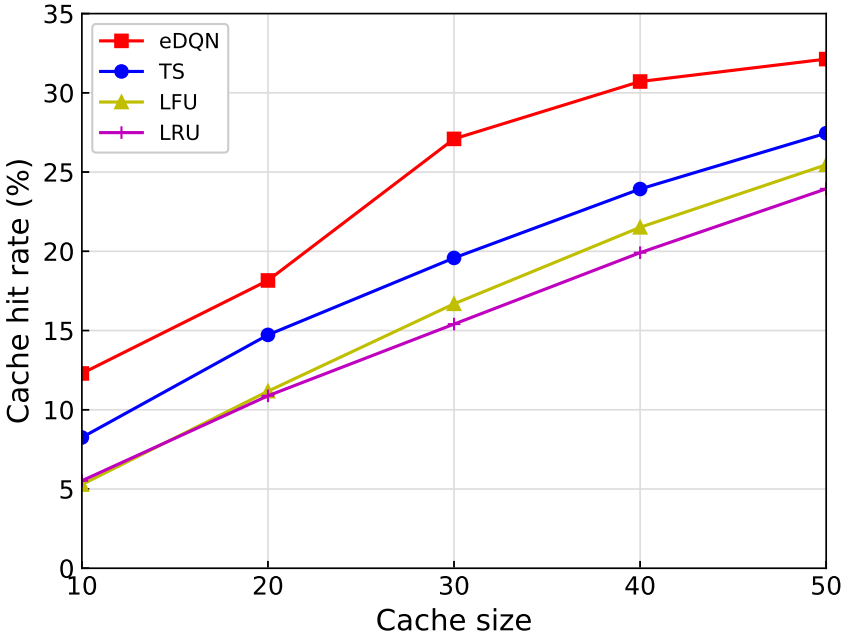

Figure 5.5: Cache hit rate of the proposed eDQN algorithm, TS, LFU and LRU in scenario served by server 2.

In Fig. 5.7, we compare the average downloading latency of the proposed algorithm with that of the other three baselines. As expected, the average downloading latency for all caching schemes decreases as the cache size of each server increases. Although the proposed eDQN algorithm performs sim-

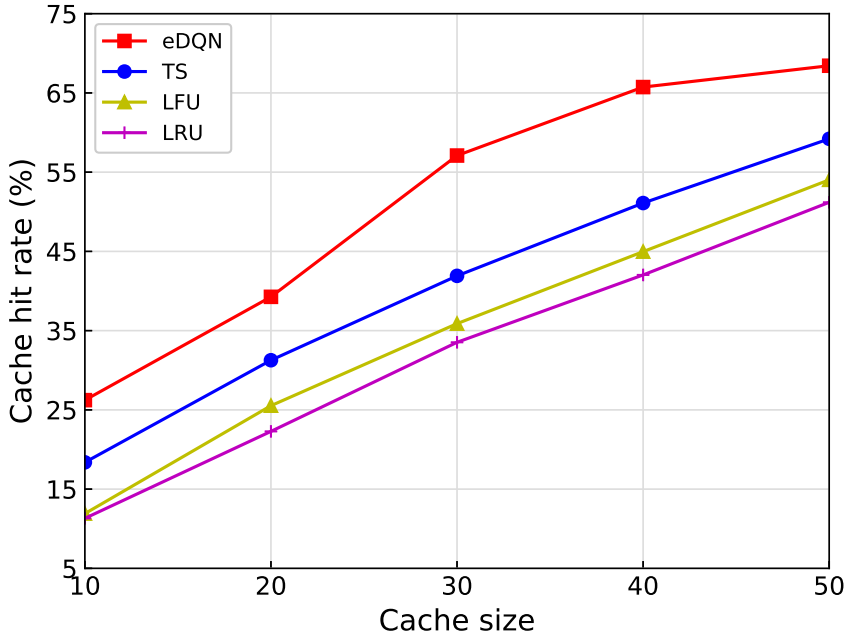

Figure 5.6: Cache hit rate of the proposed eDQN algorithm, TS, LFU and LRU in scenario served by server 3.

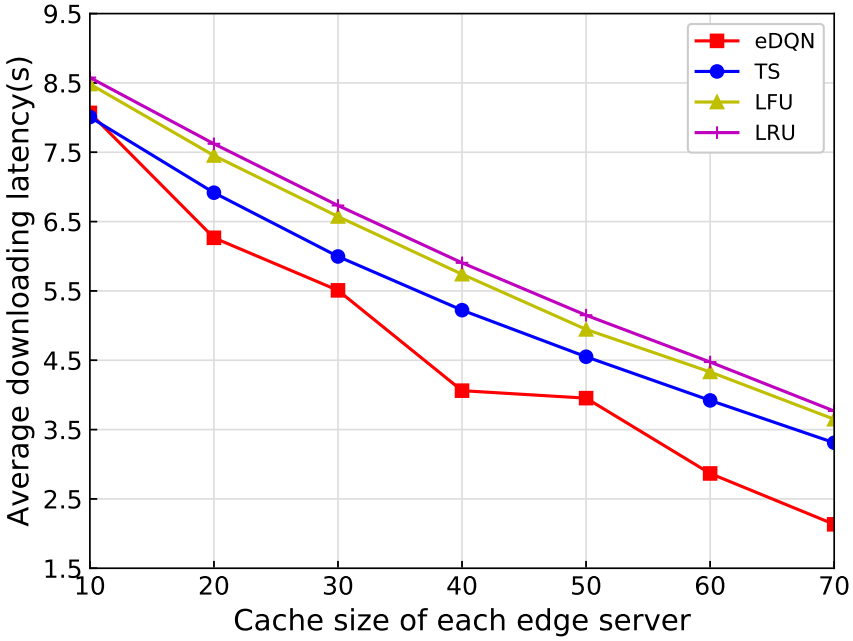

Figure 5.7: The average downloading latency of the proposed eDQN algorithm, TS, LFU and LRU under different cache sizes.

ilarly to its counterparts when the cache size is 10 units, it significantly reduces the average downloading latency and surpasses the other three baselines when the cache size exceeds 15 units. Particularly, eDQN outperforms TS, LFU, and LRU with improvements in average downloading latency of approximately

35.5%, 41.5%, 43.4%, respectively, when the cache size of each server reaches 70 units.

5.6 Conclusion

In Chapter 5, we addressed a collaborative content caching problem in a MEC system with heterogeneous caching scenarios. Initially, we formulated the problem as a RL problem with the objective of minimizing overall downloading latency over a long-term horizon. To optimize collaborative caching policies among multiple edge servers, we decomposed the problem into multiple optimization tasks. We then proposed an evolutionary DQN algorithm, integrating MFEA into DQN, to jointly evolve multiple DQN models for collaborative problem-solving. Through experiments conducted on a real-world dataset, MovieLens 100K, the simulation results demonstrate that our proposed algorithm significantly reduces downloading latency and enhances caching performance compared to other existing caching schemes.

Chapter 6

Deep Reinforcement Learning Based Two-phase Proactive Caching for Collaborative Edge Networks

6.1 Overview

In this chapter, we aim to develop a scalable DQN-based algorithm to be executed on distributed edge servers to solve the high-dimensional problem in large-scale systems, meanwhile collaboratively improve overall caching performance. To this end, we model the QoS of content providers as a time-varying system cost, and formulate the collaborative content caching problem as the minimization of long-term average system cost under the dynamic environment. Consequently, a two-phase online proactive caching scheme is proposed. This scheme successfully overcomes the curse of high dimensionality by re-designing the output layer of the neural network, and adaptively updates caching decisions for edge servers. Experimental results demonstrate that our proposed approach is robust to large-scale caching scenarios, able to predict the users' future requests with a high accuracy and collaboratively update the caching policies. Compared to other well-known caching schemes, the proposed approach outperforms in various performance metrics, including the average system cost and overall cache hit rate.

The remainder of this chapter is organized as follows. Section 6.2 presents the system model, followed by the problem formulation in Section 6.3. The technical details of the proposed two-phase proactive caching algorithm is described in Section 6.4 and the simulation results are illustrated in Section 6.5. Final conclusions are drawn in Section 6.6.

6.2 System Model

As illustrated in Fig. 6.1, we further introduce a central infostation to our generic model. The central infostation collects the caching status from individual servers, and it is not required to store all those raw files. As such, the central infostation can find the nearest neighbouring server which has the requested file in order to transmit a file copy to the requested server. If the requested file is not cached at any edge server, it needs to be fetched from the cloud center.

We assume that a server $m \in \mathcal{M}$ can provide content for a set of users, denoted as $\mathcal{U}^m = \{1, 2, ..., U^m\}, m \in \mathcal{M}$. Each file is of size n_f. Let $\mathcal{T} = \{1, 2, ..., T\}$ denote the discrete time slots, where T represents the finite time horizon. Within a time slot t, each user requests for at most one file, which must be served before the end of current time slot. At each time slot, the server reactively transmits any file that is requested by a user and not available in its local cache, and proactively stores some files that are not currently requested by any user.

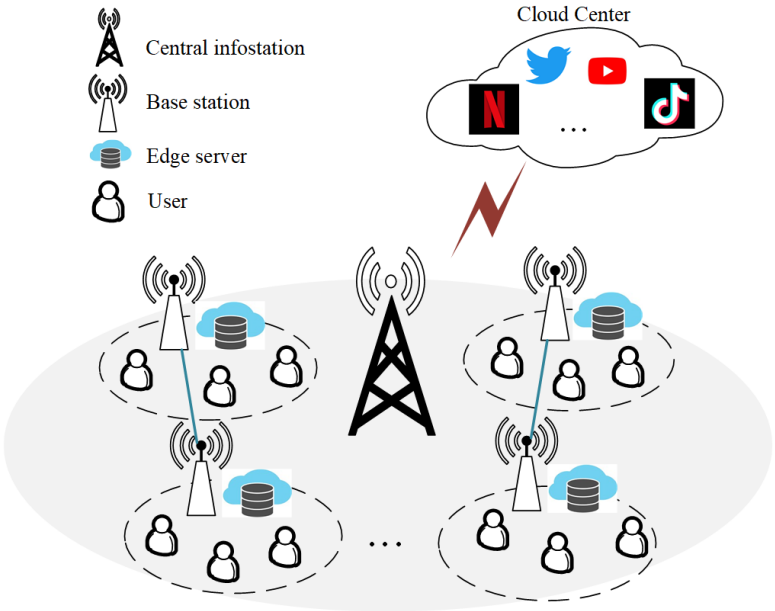

Figure 6.1: A collaborative edge caching system.

Let $D_t^{m,u} = f \in \bar{\mathcal{F}}$ denote the request of user u associated with server m at time slot t, where $\bar{\mathcal{F}} = \mathcal{F} \bigcup \{0\}$ and $D_t^{m,u} = 0$ indicates that there is no request by user u. We assume that each user's demand follows a Markov chain, which captures both the file popularity and temporal file correlations [83]. Specifically, we denote the transition probability as $p_{i,j}^{m,u}$, indicating the probability of requesting file $j \in \bar{\mathcal{F}}$ at time slot $t+1$ given that user u has requested file $i \in \bar{\mathcal{F}}$ at time slot t. We assume a practical scenario where both of the file popularity and temporal correlations are unknown in advance,

namely, the transition probability matrix $\left(p_{i,j}^{m,u}\right)_{i,j\in\bar{\mathcal{F}},u\in\mathcal{U}^m,m\in\mathcal{M}}$ is unknown *a priori*.

We measure the latency of file migration using the number of migration hops. Furthermore, we denote the source of fetched files at time slot t by $Z_t^m(f) \in \{0,1\}$, where $Z_t^m(f) = 1$ indicates the file f is fetched from the cloud center, and $Z_t^m(f) = 0$ indicates that the file f is fetched from the m'-th edge server, i.e.,

$$Z_t^m(f) = \begin{cases} 0, \text{ if } H_t^{mm'}(f) \leq H_T, \\ 1, \text{ if } H_t^{mm'}(f) > H_T, \end{cases} \tag{6.1}$$

where $H_t^{mm'}(f)$ denotes the minimum number of migration hops for file f from the m'-th edge server to server m, and H_T is the maximum tolerable neighborhood distance in number of hops.

From the perspective of content providers, a key performance indicator is QoS, which can be evaluated by the overall system cost. This cost encompasses caching cost, migration cost and penalty cost, as outlined below.

1) Caching cost: The caching cost is calculated based on the cache space occupied by the files. So the caching cost of server m storing file f at time slot t is given by

$$c_t^m(f) = \tau^{\text{cache}} n_f, \tag{6.2}$$

where $\tau^{\text{cache}} > 0$ is the unit cost of occupying the cache space of a server.

2) Migration cost: The migration cost is defined as the cost of migrating files from the neighbouring servers to the local server. So the migration cost of server m fetching file f from a neighbouring server m' at time slot t is calculated as

$$c_t^{m,m'}(f) = \tau^{\text{mig}} n_f d_t^m(f) H_t^{mm'}(f)(1 - Z_t^m(f)), \tag{6.3}$$

where $\tau^{\text{mig}} > 0$ is the unit cost of migrating a file from a neighbouring server m' to the local server m, $d_t^m(f) = \sum_{u\in\mathcal{U}^m} \mathbb{1}(D_t^{m,u} = f)$ represents the number of requested times for file f, received by server m at time slot t, and $\mathbb{1}(\cdot)$ denotes the indicator function throughout the thesis, which returns one if its argument holds true and returns zero, otherwise. As can be seen from Eq. (6.3), the migration cost will be higher if the same file f is requested by multiple users within a time slot.

3) Penalty cost: The QoS will degrade considerably if users cannot obtain the requested file from any accessible edge servers within their tolerable latency period. Consequently, the penalty cost will incur when a server needs to retrieve files from the cloud center. The penalty cost of server m retrieving

file f from remote cloud center at time slot t is given by

$$c_t^{m,c}(f) = \tau^{\text{pe}} d_t^m(f) H_c Z_t^m(f), \tag{6.4}$$

where $\tau^{\text{pe}} > 0$ is the unit cost of retrieving a file from the remote cloud center to the local server m, and H_c is the number of hops from the remote cloud to the edge servers. It is assumed that $H_c >> H_T$, implying that retrieving files from the remote cloud always endures greater latency.

6.3 Problem Formulation

Using the costs developed in Eq. (6.2), Eq. (6.3) and Eq. (6.4), we define the total system cost at time slot t as

$$c_t^{\text{total}} = \sum_{m \in \mathcal{M}} \sum_{f \in \mathcal{F}} c_t^m(f) + c_t^{m,m'}(f) + c_t^{m,c}(f). \tag{6.5}$$

Let $\mathbf{a}_t^m = \left(a_t^m(f)\right)_{f \in \mathcal{F}}$ denote the proactive caching decision vector of server m at time slot t, where $a_t^m(f) = 1$ if it is determined to store file f, and $a_t^m(f) = 0$ otherwise. In the sequel, we formulate the collaborative edge caching problem as minimizing the average system cost over a time horizon $T \in \mathcal{T}$, as

$$\min_{\mathbf{a}_t^m, \forall t} \frac{1}{T} \sum_{t \in \mathcal{T}} c_t^{\text{total}} \tag{6.6}$$

$$\text{s. t.} \quad \sum_{f \in \mathcal{F}} a_t^m(f) \leq N^m, \forall m \in \mathcal{M}, t \in \mathcal{T}, \tag{6.7}$$

$$a_t^m(f) \in \{0, 1\}, \forall m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}, \tag{6.8}$$

where constraint (6.7) is the cache size constraint.

The problem in (6.6) is a binary optimization problem and the objective is minimization over the long-term system cost without any prior knowledge of the statistics of the system dynamics, e.g., user request transition probabilities. The long-term optimization implies consideration of look-ahead system parameters, i.e., the proactive decisions in caching context, at current instance of optimization. Hence, in the sequel, we introduce a RL approach to solving the problem in (6.6) and develop an algorithmic solutions for finding the instantaneous decision variables $\mathbf{a}_t^m, \forall t$.

6.4 Algorithm Design

6.4.1 RL model

We first define the RL model for each server m, i.e., the agent, along with a set of potential states $\mathcal{S}^m$, a set of feasible actions $\mathcal{A}^m$ and a reward function $r_t^m(f)$ in terms of each file $f \in \mathcal{F}$.

States: We formulate the states as

$$\mathbf{s}_t^m = \left[\mathbf{d}_t^m, \mathbf{a}_{t-1}^m\right], \tag{6.9}$$

where $\mathbf{d}_t^m = [d_t^m(1), \ldots, d_t^m(F)]$ denotes users' request state at time slot t for edge server m, and $\mathbf{a}_{t-1}^m = \left[a_{t-1}^m(1), \ldots, a_{t-1}^m(F)\right]$ denotes the corresponding cache state of files at time slot $t-1$.

Remark 6.1. *Notice that the order of user requests has no effect on caching policy optimization. For instance, user requests* $\{D_{t_1}^{m,u}\}_{m\in\mathcal{M},u\in\mathcal{U}^m} = \{2,3,3\}$ *and* $\{D_{t_2}^{m,u}\}_{m\in\mathcal{M},u\in\mathcal{U}^m} = \{3,2,3\}$ *should yield the same optimal caching policy for all edge servers. In other words, an optimal caching policy for each server is expected to cache those files with the highest number of requesting times. Thus, we formulate the state with the requested number of files* $\mathbf{d}_t^m$.

Actions: We use the proactive caching decision vector to represent the m-th edge server's proactive caching action at time-slot t, i.e.,

$$\mathbf{a}_t^m = \left[a_t^m(1), \ldots, a_t^m(F)\right]. \tag{6.10}$$

Accordingly, the action dimension size is computed as $|\mathcal{A}^m| = 2^F$.

Reward of each requested file: The instantaneous reward of each requested file f in server m at time-slot t is defined as the negative value of the cost of serving a file f, either by the edge servers, or by the remote cloud center, expressed as

$$r_t^m(f) = -\begin{cases} c_t^m(f), & \text{if } a_t^m(f) = 1, \\ c_t^{m,m'}(f), & \text{if } a_t^m(f) = 0 \text{ and } Z_t^m(f) = 0, \\ c_t^{m,c}(f), & \text{otherwise.} \end{cases} \tag{6.11}$$

With the RL formulation, the optimization problem in (6.6) can be interpreted as finding the optimal binary decision vector $\mathbf{a}_t^{m^*}$ for each and every

server m by maximizing the long-term cumulative reward, i.e.,

$$\max_{\mathbf{a}_t^m} \sum_{t\in\mathcal{T}} \sum_{f\in\mathcal{F}} r_t^m(f) \tag{6.12}$$
$$\text{s. t. } (6.7), (6.8).$$

Due to the exponential growth of the action space and in order to break the resulting curse of dimensionality, we propose a new DQN-based two-phase proactive caching algorithm to solve problem (6.12). More specifically, we re-design the output layer of the neural network in the classical DQN to cope with the exponentially growing dimension of the action space in a computationally efficient manner. Furthermore, we develop a two-phase procedure for the action selection process, which will be introduced in the next subsection.

6.4.2 Proposed two-phase proactive caching scheme

The dimension of the search space, i.e., the actions $\mathbf{a}_t^m = \left[a_t^m(1), \ldots, a_t^m(F)\right]$, for an optimal solution to the problem in (6.12) is 2^F, which indicates an exponential growth with the number of files. Thus, adopting the conventional DQN structure wherein each neuron of the output layer of the network represents a potential action [123], would require a highly complex deep neural network for fitting and the training process would be time-consuming and difficult to converge.

To break the curse of dimensionality and to prepare for the scalability of the network, we redesign the output layer of the standard DQN so that each neuron therein represents the *soft Q value* of a specific file. In this way, the actions $\mathbf{a}_t^m$ are excluded from the DQN outputs and the number of the output neurons is significantly decreased from 2^F to F. In the proposed DQN architecture, as illustrated in Fig. 6.2, an output neuron indicates the *'predicted importance'* of a specific file. Thus a larger soft Q value of a file indicates its higher popularity in terms of being more likely to be requested by the users. The proposed restructured DQN model prepares for the scalability of the edge caching network by allowing for a significantly greater size of the content library.

Let the outputs of the proposed neural network for server m at time slot t be denoted by

$$\mathbf{Q}^m\left(\mathbf{s}_t^m; \boldsymbol{\theta}^m\right) = \left[Q^{m,1}\left(\mathbf{s}_t^m; \boldsymbol{\theta}^m\right), \ldots, Q^{m,F}\left(\mathbf{s}_t^m; \boldsymbol{\theta}^m\right)\right], \tag{6.13}$$

where the weights of the deep neural network are stacked in vector $\boldsymbol{\theta}^m$ and each element $Q^{m,f}\left(\mathbf{s}_t^m; \boldsymbol{\theta}^m\right), \forall f \in \mathcal{F}$ represents the corresponding soft Q value of file f at time slot t.

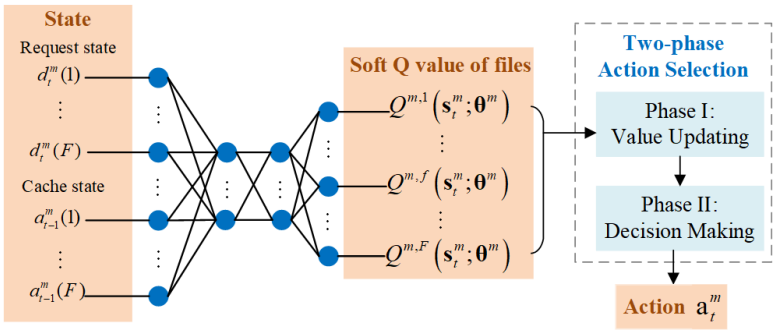

Figure 6.2: The proposed deep neural network architecture, where each output neuron represents the soft Q value of a specific file.

Since, in contrast to the conventional DQN algorithm, each output neuron in the proposed architecture no longer indicates a potential action, it is infeasible to directly choose an action (i.e., a neuron) from the output layer of the network. Next, we introduce a two-phase procedure for the action selection, namely, Phase I: value updating phase; and Phase II: decision making phase, as follows.

Phase I (Value Updating Phase): The soft Q value of a file f at time slot t is updated according to

$$Q^{m,f}\left(\mathbf{s}_t^m;\boldsymbol{\theta}^m\right) \leftarrow (1-\beta)Q^{m,f}\left(\mathbf{s}_t^m;\boldsymbol{\theta}^m\right) + \beta Q^{m,f}\left(\mathbf{s}_{t-1}^m;\boldsymbol{\theta}^m\right), \tag{6.14}$$

where $\beta \in (0,1)$ determines the present value of the soft Q value at time slot $t-1$. This value-updating operation is applied to smooth out the potentially large fluctuations in soft Q value of file f due to the variations of the state $\mathbf{s}^m$ at the input of the proposed deep neural network.

Phase II (Decision Making Phase): In RL, ϵ-greedy algorithm is commonly used for action selection process in order to balance the exploitation and exploration. In the following, we adopt a similar ϵ-greedy idea for action selection, while with an improvement on the exploration, i.e., request-driven exploration.

Exploitation: The greedy action for exploitation is to select the files with highest updated soft Q values. Specifically, we sort all files by their corresponding soft Q values in a descending order. Let the first N^m files with largest Q values at time slot t denote by $\mathcal{F}_t^{\text{cache}} = \left\{\tilde{f}_1, \tilde{f}_2, \ldots, \tilde{f}_{N^m}\right\}$. Accordingly, the greedy action for the server m at time slot t is to cache the files in $\mathcal{F}_t^{\text{cache}}$, i.e.,

$$\mathbf{a}_t^g = \left\{ a_t^m(f) = 1 \middle| f \in \mathcal{F}_t^{\text{cache}}, a_t^m(f) = 0 \middle| f \in \mathcal{F} - \mathcal{F}_t^{\text{cache}} \right\}. \tag{6.15}$$

Request-driven Exploration: In the classical DQN, exploration is usu-

ally performed by selecting the non-greedy actions with an equal probability, which greatly limits the effectiveness of exploration. Consequently, we propose an exploration approach based on accumulated requesting times of files, which are updated by the historical request information stored in states.

We denote the accumulated requesting times of all files stored at server m by $\mathbf{v}_t^m = \left\{v_t^m(f) = \sum_{i=0}^{t} d_t^m(f) | f \in \mathcal{F}\right\}$. All elements in $\mathbf{v}_t^m$ are sorted in a descending order, and then the corresponding first N^m files with highest values are collected as a set and denoted by $\mathcal{F}_t^{\mathbf{v}}$. Accordingly, the non-greedy action $\mathbf{a}_t^{\mathbf{v}}$ (exploration) for the server m at time slot t is to cache the files in $\mathcal{F}_t^{\mathbf{v}}$, i.e.,

$$\mathbf{a}_t^{\mathbf{v}} = \left\{a_t^m(f) = 1 \middle| f \in \mathcal{F}_t^{\mathbf{v}}, a_t^m(f) = 0 \middle| f \in \mathcal{F} - \mathcal{F}_t^{\mathbf{v}}\right\}. \tag{6.16}$$

In a word, at each time slot t, the agent m selects the action according to the following policy with a likelihood of $1-\epsilon$ or ϵ, i.e.,

$$\mathbf{a}_t^m = \begin{cases} \mathbf{a}_t^g, & \text{with } 1-\epsilon, \\ \mathbf{a}_t^{\mathbf{v}}, & \text{with } \epsilon. \end{cases} \tag{6.17}$$

In addition to the two-phase action selection procedure introduced above, the proposed algorithm contains other three key modules: evaluation neural network, target neural network and experience replay buffer, as illustrated in Fig. 6.3.

The evaluation neural network with parameters $\boldsymbol{\theta}^m$ are trained to approximate the soft Q values of files, while the target neural network with parameters $\hat{\boldsymbol{\theta}}^m$ is used for obtaining the target Q values. Both of them are constructed with the same architecture, where the input of the network is the state defined in Eq. (6.9), and the outputs are the soft Q values of the files defined in Eq. (6.13).

The target Q values are updated according to

$$\mathbf{y}_t^m = \mathbf{r}_t^m + \gamma \mathbf{Q}^m\left(\mathbf{s}_{t+1}^m; \hat{\boldsymbol{\theta}}^m\right), \tag{6.18}$$

where $\gamma \in [0,1]$ is the discount factor, the vector $\mathbf{r}_t^m = \left[r_t^m(1), r_t^m(2), ..., r_t^m(F)\right]$ contains the reward values of each content file at time slot t as its F dimensions, and $\mathbf{Q}^m\left(\mathbf{s}_{t+1}^m; \hat{\boldsymbol{\theta}}^m\right)$ is the output of the target neural network.

Let the temporal-difference (TD) error as a function of $\boldsymbol{\theta}^m$ be defined as

$$\boldsymbol{\delta}(\boldsymbol{\theta}^m) = \left[\mathbf{y}_t^m - \mathbf{Q}^m(\mathbf{s}_t^m; \boldsymbol{\theta}^m)\right] \odot \mathbf{a}_t^m, \tag{6.19}$$

where $\odot$ denotes element-wise vector multiplication. Note that this vector contains non-zero values only for files that are cached at time slot t. In other words, only the error with respect to the selected action will be backpropa-

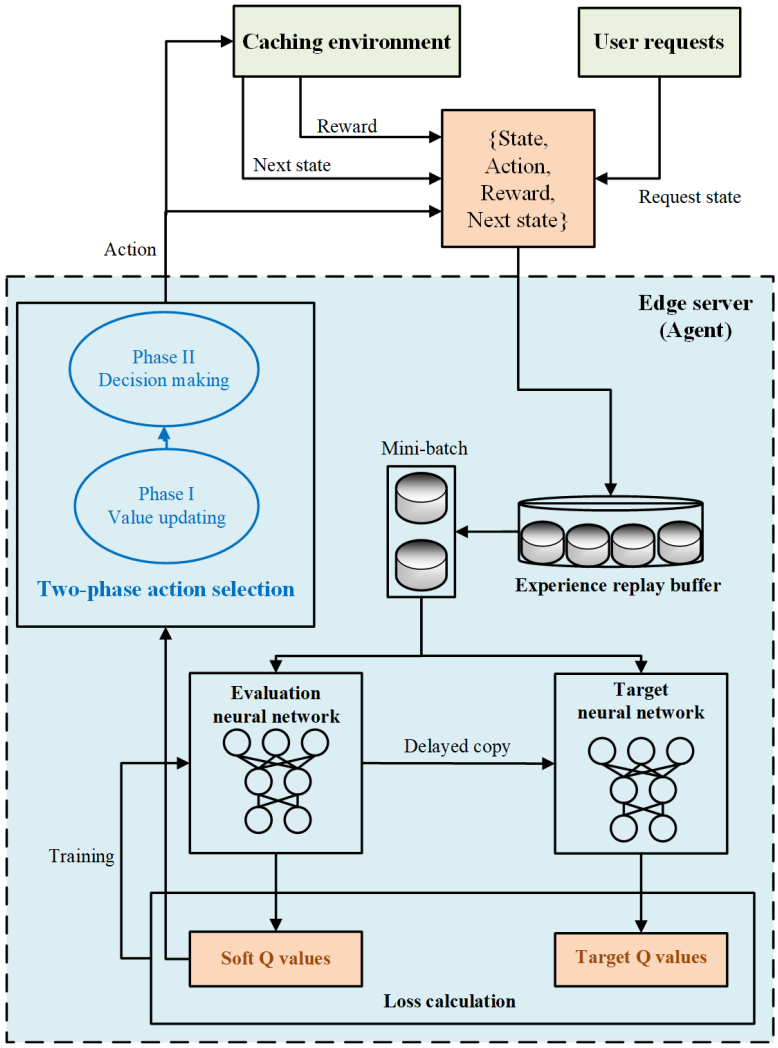

Figure 6.3: The framework of the proposed two-phase proactive caching scheme.

gated.

Subsequently, the evaluation neural network parameters $\boldsymbol{\theta}^m$ are trained using a mini-batch of data sampled from the experience replay buffer $\mathcal{B}^m$, where the server's past experiences $\{\mathbf{s}_t^m, \mathbf{a}_t^m, \mathbf{r}_t^m, \mathbf{s}_{t+1}^m\}$ are preserved. The training process is executed via the stochastic gradient descent iterations, i.e.,

$$\boldsymbol{\theta}_{t+1}^m = \boldsymbol{\theta}_t^m - \alpha \nabla_{\boldsymbol{\theta}^m} L(\boldsymbol{\theta}^m), \tag{6.20}$$

where $\alpha \in [0, 1]$ is the learning rate and the loss function $L(\boldsymbol{\theta}^m)$ is given by

$$L(\boldsymbol{\theta}^m) = \mathbb{E}_{(\mathbf{s}_j^m, \mathbf{a}_j^m, \mathbf{r}_j^m, \mathbf{s}_{j+1}^m\}) \sim U(\mathcal{B}^m)} \left\| \boldsymbol{\delta}(\boldsymbol{\theta}^m) \right\|_2^2. \tag{6.21}$$

Additionally, the target neural network parameters $\hat{\boldsymbol{\theta}}^m$ are periodically synchronized with the evaluation network parameters $\boldsymbol{\theta}^m$ per K time slots. The utilization of both the experience replay and the target neural network contributes to the stabilization of DQN updates. Incorporating the remedies in-

Algorithm 6: The Proposed DQN-based two-phase proactive caching scheme

Initialize $\boldsymbol{\theta}^m, \hat{\boldsymbol{\theta}}^m, \mathcal{B}^m, \forall m \in \mathcal{M}$;
for $t = 1, 2, 3, ..., T$ **do**
  for *Each edge server* $m \in \mathcal{M}$ **do**
    Receive the requests from associated users and observe the current state $\mathbf{s}_t^m$;
    Update the soft Q value of each file in the evaluation neural network according to $Q^{m,f}\left(\mathbf{s}_t^m; \boldsymbol{\theta}^m\right) \leftarrow (1-\beta) Q^{m,f}\left(\mathbf{s}_t^m; \boldsymbol{\theta}^m\right) + \beta Q^{m,f}\left(\mathbf{s}_{t-1}^m; \boldsymbol{\theta}^m\right)$;
    Take proactive caching action

$$\mathbf{a}_t^m = \begin{cases} \mathbf{a}_t^g, & \text{with } 1-\epsilon \\ \mathbf{a}_t^{\mathbf{v}}, & \text{with } \epsilon \end{cases}$$

    Receive the reward of each file $r_t^m(f)$ as per Eq. (6.11) and observe the next state $\mathbf{s}_{t+1}^m$;
    Store the transition $\{\mathbf{s}_t^m, \mathbf{a}_t^m, \mathbf{r}_t^m, \mathbf{s}_{t+1}^m\}$ into the buffer $\mathcal{B}^m$;
    Randomly sample a mini-batch of transitions $\{\mathbf{s}_j^m, \mathbf{a}_j^m, \mathbf{r}_j^m, \mathbf{s}_{j+1}^m\}$ from $\mathcal{B}^m$;
    Update the evaluation neural network parameters $\boldsymbol{\theta}_t^m$ using stochastic gradient descent algorithm as per Eq. (6.20);
    if $t \mod K = 0$ **then**
      Update the target neural network parameters $\hat{\boldsymbol{\theta}}_t^m \leftarrow \boldsymbol{\theta}_t^m$;
    end
  end
end

troduced above, Algorithm 6 outlines our DQN-based two-phase proactive caching scheme for the collaborative edge caching network.

6.5 Simulation Results

6.5.1 Setup and baselines

We use the same user request model as in [124], where the transition probability $p_{i,j}$ is given by

$$p_{i,j} = \begin{cases} P_0, & i \in \bar{\mathcal{F}}, j = 0, \\ (1-P_0)\frac{\frac{1}{j^{\lambda}}}{\sum_{j'=1}^{F}\frac{1}{j'^{\lambda}}}, & i = 0, j \in \mathcal{F}, \\ (1-P_0)\frac{1}{G}, & i \in \mathcal{F}, j = (i+g)\text{mod}(F+1),\ g \in \{1,2,\ldots,G\}, \\ 0, & \text{otherwise.} \end{cases} \tag{6.22}$$

Note that the user request model is parameterized by $\{P_0, \lambda, G\}$, which takes both the file popularity and temporal file correlations into account. Specifically, P_0 is the transition probability to no request by a user from its any current request state. For $i = 0$, i.e., the user does not have a request at current time slot, the probability of requesting any file $f \in \mathcal{F}$ follows a Zipf-like distribution with parameter λ. Additionally, we assign a set of G neighbouring files to each file $i \in \mathcal{F}$, i.e., $\mathcal{G}_i \triangleq \{f \in \mathcal{F} : (i+g)\text{mod}(F+1), g \in \{1,2,\ldots,G\}\}$. Then, the transition probability from any file $i \in \mathcal{F}$ to a neighbouring file $j \in \mathcal{G}_i$ follows a uniform distribution. The transition probability from a file $i \in \mathcal{F}$ to a non-neighbouring file $j \notin \mathcal{G}_i$ is set to zero. The specific values for the parameters of user request models will be specified in the context of different experiments.

To realistically simulate the results, we adopt pricing for the edge storage from Amazon Web Services (AWS) [125], and set $\tau^{\text{cache}} = \$0.05$, $\tau^{\text{mig}} = \$0.01$, $\tau^{\text{pe}} = \$0.02$, the maximum tolerant migration hops $H_T = 2$, the number of hops from the cloud center to the edge server $H_c = 20$. Besides, we assign $n_f = 1$ for all files $f \in \mathcal{F}$. We construct the neural network for each edge server with an identical architecture, as follows: an input layer with $2F$ neurons, two hidden layers (the first consisting of 256 neurons, the second of 128 neurons, both adopting ReLU activation), and an output layer with F neurons and sigmoid activation. Other main simulation parameters are given in Table 6.1.

Table 6.1: Parameter settings

Parameter	Value
Learning rate (α)	0.01
Discount factor (γ)	0.9
Q value discount rate (β)	0.95
Replay buffer size	500
Mini-batch size	32
Optimizer	Adam
Target network updating frequency (K)	300

We emphasize that the parameter settings used in the simulation serve as illustrative examples for evaluating our proposed caching algorithm. The selection of these parameters does not impact the effectiveness of our proposed algorithm.

We compare the performance of the proposed algorithm against four baseline algorithms, as follows:

1) Hierarchical reinforcement learning (HRL) based cache scheme proposed in [99], which adopts classical DQN to optimize the caching policies at BSs. Particularly, the number of neural network outputs, representing the dimensionality of the action space, is designed as the combination number of N^m (cache capacity of server m) in F (total number of files), denoted as $\binom{F}{N^m}$. Besides, the caching policy implemented on the BS side is designed for each individual server, and it does not involve exploration of inter-server cooperation.

2) Online Distributed Cache Replacement (ODCR) algorithm proposed in [3], which presents a TS based algorithm to optimize the caching policy for each edge server individually.

3) LFU.

4) LRU.

6.5.2 Experiments on small-scale scenarios

In this subsection, we compare the performance of the proposed algorithm with HRL in [99] under relatively small-scale scenarios with $M = 2$ edge servers, where each server covers $U^1 = U^2 = 10$ users. We assume that the request model for users covered by server 1 is parameterized by $\{P_0 = 0.2, \lambda = 0.8, G = 5\}$, and the request model for users covered by server 2 is parameterized by $\{P_0 = 0.1, \lambda = 0.6, G = 3\}$.

For the sake of fairness and clarity, we implement the HRL algorithm following the design in [99], where the number of output neurons is determined by the combination number of N^m in F (i.e., $\binom{F}{N^m}$). Given the absence of parameter setting details in [99], we maintain the consistency of other neural network parameters and training hyperparameters of HRL with the settings presented in 6.5.1. It is important to emphasize that, in the HRL cases, edge servers operate independently because HRL does not consider cooperation among edge servers.

We use cache hit rate as a performance metric, wherein the count of cache hits is calculated as the number of requests fulfilled by any of the edge servers within the system. Hence, the cache hit rate is defined as the proportion of the total cache hits to the overall number of content requests. The runtime of each algorithm is used as another performance criterion.

We first conduct an experiment using the same values for the content library size $F = 10$ and the cache size of each server $N^m = 3, \forall m \in \mathcal{M}$, as in [99]. The comparative results are presented in Table 6.2. Although HRL consumes less time than our proposed algorithm, its cache hit rate is lower due to the lack of consideration for the cooperation of edge servers. Besides, it is noted that the number of output neurons for HRL is twelve times larger than that of the proposed algorithm, leading to higher computational overhead during training. This higher computational overhead is expected to result in significant performance degradation as the content library size and cache storage gradually increase, as will be demonstrated in the following experiments.

Table 6.2: Comparison of the proposed algorithm and the baseline HRL algorithm with $F = 10, N^m = 3, \forall m \in \mathcal{M}$.

Algorithm	F	N^m	#Output neurons	Cache hit rate	Runtime/s
Proposed	10	3	10	50.36%	9.60
HRL	10	3	120	32.13%	7.36

Table 6.3: Comparison of the proposed algorithm and the baseline HRL algorithm under various sizes of content library, with a fixed number of server cache capacity $N^m = 5, \forall m \in \mathcal{M}$. (The exact number of output neurons for the HRL algorithm is $\binom{F}{N^m}$, while approximate values are provided in this table for better comparison.)

Algorithm	F	N^m	#Output neurons	Cache hit rate	Runtime/s
Proposed	10	5	10	80.66%	7.33
HRL	10	5	10×2.5	52.41%	4.61
Proposed	20	5	20	46.78%	7.92
HRL	20	5	$20 \times 7.8 \times 10^2$	29.02%	6.51
Proposed	30	5	30	28.02%	8.12
HRL	30	5	$30 \times 4.8 \times 10^3$	20.24%	22.40
Proposed	40	5	40	22.50%	8.68
HRL	40	5	$40 \times 1.6 \times 10^4$	16.06%	104.52
Proposed	50	5	50	19.21%	8.64
HRL	50	5	$50 \times 4.2 \times 10^4$	13.61%	353.35
Proposed	60	5	60	15.25%	10.04
HRL	60	5	$60 \times 9.1 \times 10^4$	11.88%	1117.13

Next, we compare the performance of the proposed algorithm with the baseline HRL under more varied scenarios involving different sizes of content library and cache storage for each server. Specifically, the content library size F ranges from 10 to 60, and the cache storage for each server is set to be 5 and 10, respectively. The comparison results are presented in Table 6.3 and Table 6.4, as elaborated below. Note that in these tables, the number of output

Table 6.4: Comparison of the proposed algorithm and the baseline HRL algorithm under various sizes of content library, with a fixed number of server cache capacity $N^m = 10, \forall m \in \mathcal{M}$. (The exact number of output neurons for the HRL algorithm is $\binom{F}{N^m}$, while approximate values are provided in this table for better comparison.)

Algorithm	F	N^m	#Output neurons	Cache hit rate	Runtime/s
Proposed	20	10	20	74.48%	24.01
HRL	20	10	$20 \times 9.2 \times 10^3$	54.46%	42.19
Proposed	30	10	30	57.38%	8.22
HRL	30	10	$30 \times 1.0 \times 10^6$	39.30%	6114.29
Proposed	40	10	40	44.83%	21.79
HRL	40	10	$40 \times 2.1 \times 10^7$	–	–
Proposed	50	10	50	37.00%	23.24
HRL	50	10	$50 \times 2.1 \times 10^8$	–	–
Proposed	60	10	60	31.78%	26.88
HRL	60	10	$60 \times 1.3 \times 10^9$	–	–

neurons for HRL is computed as approximate values for the ease of comparison. For example, if $F = 20, N^m = 5$, the exact number of output neurons for HRL is calculated as the combination number $\binom{20}{5} = 15504$. Whereas, we approximate the number of output neurons for HRL based on multiples of the content library size $F = 20$, resulting in the rounded value of $20 \times 7.8 \times 10^2$.

As depicted in both Table 6.3 and Table 6.4, the cache hit rate of the proposed algorithm surpasses that of HRL across various settings, as the former makes full use of cooperation among the edge servers.

From a runtime perspective, Table 6.3 demonstrates a rapid increase in the runtime of the HRL algorithm as the size of the content library and server storage grows. In contrast, the growth in the size of the content library and cache space does not significantly impact the runtime of our proposed algorithm. It maintains at a relatively low level, despite consuming more time relative to HRL when $F \leq 20$.

Table 6.4 reveals the challenges faced by HRL as parameters F and N^m increase. Notably, when $F = 30$ and $N^m = 10$, the runtime of the HRL is surprisingly high — approximately 744 times longer than that of our proposed algorithm. Moreover, HRL fails to accomplish the task when F and N^m keep increasing, as a result of the sharply growing number of output neurons in the neural networks. In contrast, our proposed algorithm, with an output layer consisting of only F neurons, remains robust. A surge in the size of the content library and server storage does not lead to significant performance degradation for our proposed algorithm. The robustness of our proposed algorithm is further verified by experiments conducted in the next subsection, where scenarios with a larger content library $F = 1000$ is considered. Since HRL cannot be ex-

ecuted normally under scenarios with larger sizes of content library and cache storage, we will not consider it as a baseline in the subsequent experiments.

6.5.3 Experiments on large-scale scenarios

In this subsection, we evaluate the performance of our proposed algorithm, ODCR in [3], LFU and LRU under large-scale scenarios. In specific, we consider $M = 6$ edge servers, each covering 20 users. There are $F = 1000$ files in the content library. Particularly, we consider time-varying user request models to achieve diverse user demand scenarios with dynamic file popularity. The parameters of user request models are aperiodically and randomly drawn from $P_0 \in \{0.1, 0.2, 0.3\}$, $\lambda \in \{0.8, 1.0, 1.2\}$, $G \in \{25, 50, 75\}$. In addition to cache hit rate, we also use time-averaged system cost as a performance criterion.

Fig. 6.4 illustrates a comparison of the overall cache hit rates for four caching schemes. As expected, the cache hit rate increases with the growing cache size of each server, ranging from 50 to 150 units. While the proposed algorithm performs similarly to its counterpart ODCR when the cache size is 50 units, it outperforms the other three baselines when the cache size exceeds 50 units. The proposed algorithm demonstrates superior performance compared to ODCR, LFU, and LRU when the cache size exceeds 100 units. Notably, it achieves approximate gains of 10.75%, 17.45% and 23.27%, respectively, when each server's cache size is 150 units. These results further confirm the robustness and scalability of our proposed algorithm in terms of the increasing

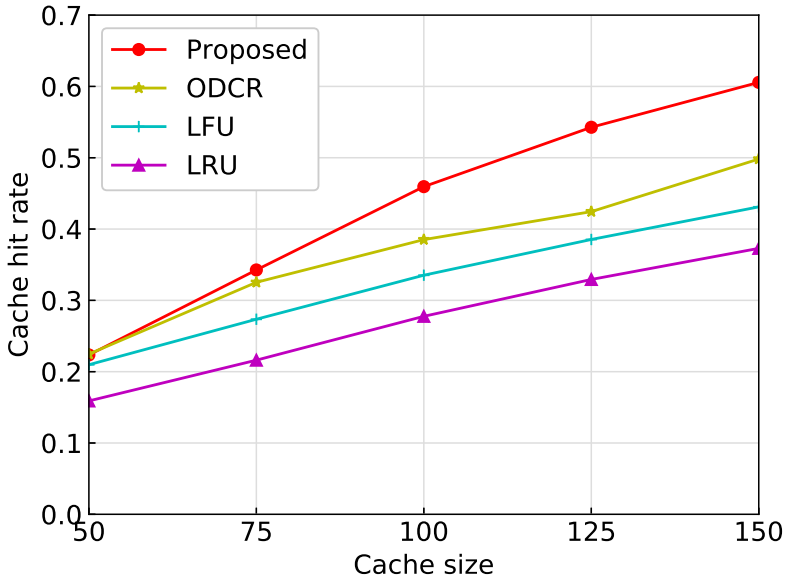

Figure 6.4: Comparison of cache hit rate of the proposed algorithm, ODCR in [3], LFU and LRU across various cache sizes at $F = 1000$.

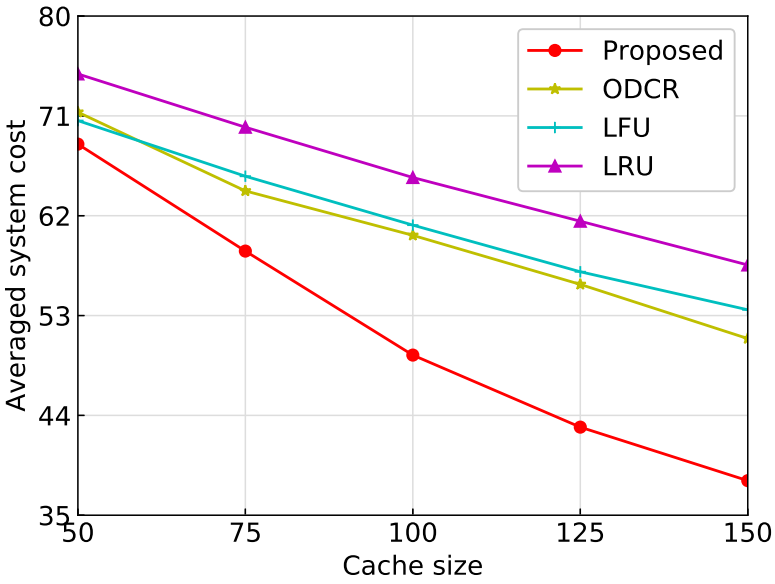

Figure 6.5: Comparison of averaged system cost of the proposed algorithm, ODCR in [3], LFU and LRU across various cache sizes at $F = 1000$.

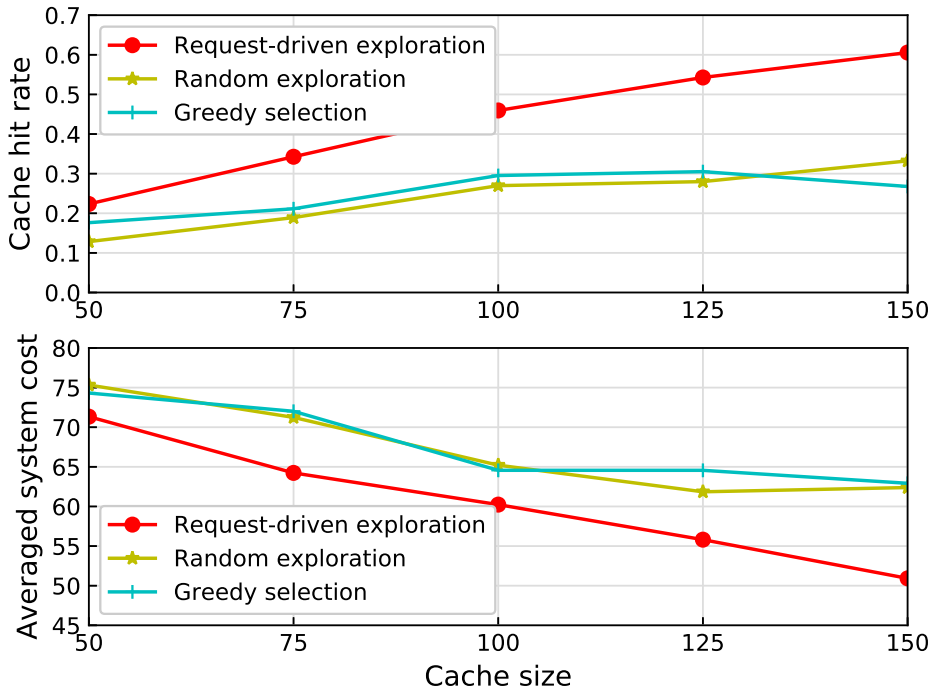

Figure 6.6: Effects of request-driven exploration in the proposed algorithm.

number of edge servers, files and heterogeneous users.

In Fig. 6.5, the averaged system cost versus cache size is compared among the proposed algorithm and three other baselines. It is evident that the proposed algorithm consistently achieves the lowest averaged system cost under different cache sizes. This improvement can be primarily attributed to a more

accurate forecast of user requests and more effective collaborations, namely content migration between edge servers.

Next, we assess the performance of the proposed algorithm using request-driven exploration, random exploration (i.e., each non-greedy action is selected with a uniform probability for exploration), and a greedy action selection scheme (i.e., only the files with the largest soft Q values are cached). As shown in Fig. 6.6, the proposed request-driven exploration method outperforms the other two schemes across various cache sizes in terms of both cache hit rate and averaged system cost. This performance gain is attributed to the exploitation of historical information, allowing the agent to cache files with higher popularity, rather than relying on blind exploration.

6.6 Conclusion

In Chapter 6, we tackled the high dimensionality problem in large-scale edge caching systems from the perspective of scalable DRL algorithm. We formulated a long-term optimization problem to minimize the system cost, while improving cache hit rate and ensuring QoE for users. To solve this problem without any prior information, we developed a DQN-based two-phase proactive caching scheme. By re-designing the output layer of the standard DQN, our proposed scheme successfully overcomes the curse of dimensionality, enabling efficient network scalability in terms of the number of both edge servers and files. Furthermore, we devised a two-phase action selection procedure to enhance exploration, thereby improving the cache hit rate as well as reducing the average system cost. We conducted experiments in both small-scale and large-scale scenarios to evaluate the performance of our proposed scheme. Numerical results for small-scale scenarios confirmed the robustness and efficiency of our proposed scheme in terms of cache hit rate and runtime. In large-scale scenarios, our proposed scheme demonstrated the lowest average system cost and the highest cache hit rate under various settings, outperforming other existing caching methods. Furthermore, we validated the effectiveness of the proposed request-driven exploration method by comparing it with both a random exploration method and a greedy selection method.

Chapter 7

A Unified Federated Deep Q Network Caching Scheme for Scalable Collaborative Edge Networks

7.1 Overview

The research of this chapter extends the scalable DQN algorithm proposed in Chapter 6, to address the joint cache update and request delivery problem in an edge caching system, which collectively considers the challenges of heterogeneous user requests, coordination among multiple edge servers, as well as network scalability. Specifically, we propose a unified federated DQN caching scheme, in which each edge server optimizes its own caching policy while coordinating with its proximal servers to minimize the overall system cost and improve the cache hit rate, under the assumption that the prior knowledge of the dynamic content popularity and the preferences of the heterogeneous users are unknown in advance. In particular, the proposed scheme integrates the actions of DQN and FL to jointly enable coordinated intelligent caching at distributed servers.

Our distinct contributions are outlined as follows:

- **Decomposition of optimization problem:** We decompose the joint cache update and request delivery problem into two subproblems: coordinated proactive cache updating and local request processing. Then, we propose a unified federated DQN caching scheme to tackle and coordinate these two subproblems.

- **Integration of FL and DQN:** We employ the scalable DQN algorithm developed in Chapter 6 to address the large dimensionality problem and

to learn the heterogeneous users' requests locally at distributed servers in an online manner. To coordinate training among DQN models at a global server, we adopt FL concept and integrate it into local DQN training processes to further enhance the proposed distributed caching scheme.

- **Signaling overhead reduction:** To reduce the signaling overhead among edge servers and the global server, a Thompson sampling (TS)-based agent selection method is introduced to predictively search for the optimal server set in the global model training process. Moreover, the proposed DQN model is partitioned into a common-network part and the output layer part, and only the common-network parameters are sent to the global server for the global model training, while the output layer parameters are trained locally to preserve the users' preference at individual servers.

The remainder of this chapter is organized as follows. Section 7.2 presents the system architecture and problem formulation, followed by the problem decomposition in Section 7.3. Next, we introduce the technical details of the proposed unified federated DQN edge caching scheme in Section 7.4. We evaluate the performance of the proposed scheme in Section 7.5, and conclude the chapter in Section 7.6.

7.2 Problem Formulation

In this section, we first introduce the system architecture in subsection 7.2.1. Next, we define the variables and cost functions to measure the QoS in subsection 7.2.2. Finally, we formulate the collaborative cache update and request delivery problem in subsection 7.2.3.

7.2.1 System architecture

Building upon the generic system model in Chapter 4, we further deploy a global server at the cloud center to enhance coordinated training among distributed edge servers, as illustrated in Fig. 7.1. The assumption and notations for file size n_f, the set of users $\mathcal{U}^m = \{1, 2, ..., U^m\}, m \in \mathcal{M}$, and user requests $D_t^{m,u}, \forall u \in \mathcal{U}_m, \forall m \in \mathcal{M}$ remain consistent with those in Chapter 6. Related details can be found in Section 6.2.

Additionally, in this proposed model, each edge server can only exchange information with its proximal edge servers via communications between the associated BSs, which can be either an exchange of the respective caching status or sharing files with one another. We denote the set of the proximal edge

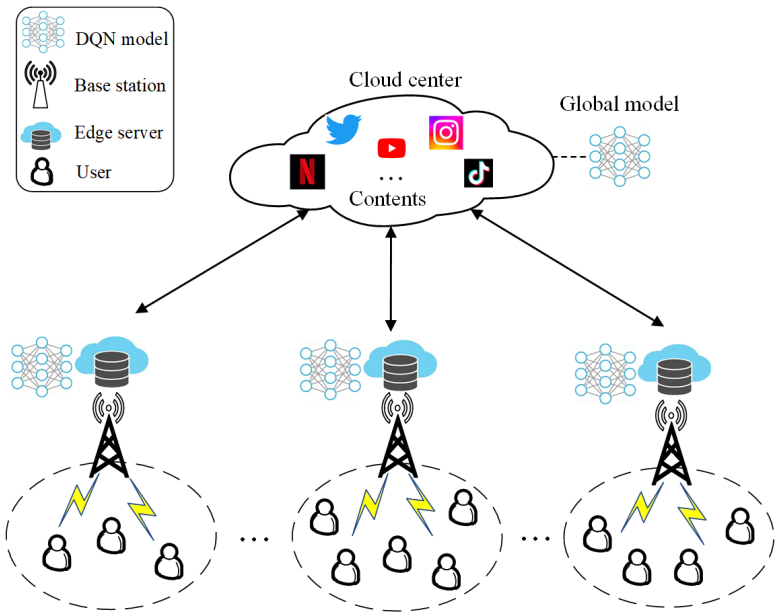

Figure 7.1: System architecture.

servers with respect to the local edge server m by $\mathcal{M}^m$, where $|\mathcal{M}^m| < |\mathcal{M}|$, and $|\mathcal{M}^m|, |\mathcal{M}|$ denote the number of edge servers in the sets $\mathcal{M}^m$ and $\mathcal{M}$, respectively.

Let the binary variable $a_t^{m,f} \in \{0,1\}$ indicate the caching state of file f, where $a_t^{m,f} = 1$ and $a_t^{m,f} = 0$, respectively, stand for the cached and otherwise states of the file f in server m at time slot t. Similarly, we let the binary variable $h_t^{m,m',f} \in \{0,1\}, \forall m' \neq m, m' \in \mathcal{M}^m$, indicate the fetching state of file f, where $h_t^{m,m',f} = 1$ and $h_t^{m,m',f} = 0$, respectively, stand for the fetched and otherwise states of the file f by server m from server m' at time slot t. Furthermore, the binary variable $h_t^{m,c,f} \in \{0,1\}$ indicates that the file f is retrieved by server m from the cloud, if $h_t^{m,c,f} = 1$, and otherwise, if $h_t^{m,c,f} = 0$.

7.2.2 System cost model

As introduced above, edge servers in close proximity can collaborate in content caching and sharing. Although collaborative caching may be resource-demanding, it has the potential to improve cache hit rate and reduce transmission cost compared to retrieving files from the remote cloud center. From the perspective of the mobile network operator, the key indicator for measuring QoS is the system serving cost, consisting of proactive caching cost, reactive fetching cost, and remote retrieval cost.

1) Proactive caching cost: The proactive content caching will incur a corresponding cost, which is calculated based on the caching space occupied by

the files. So the cost of server m storing file f at time slot t is given by

$$P_t^{m,f} = \frac{\tau^{\text{cache}} n_f}{d_t^{m,f}}, \forall m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}, \tag{7.1}$$

where $\tau^{\text{cache}} > 0$ is the caching cost per unit file size, $d_t^{m,f} = \sum_{u \in \mathcal{U}^m} \mathbb{1}(D_t^{m,u} = f)$ represents the number of requested times for file f, received by server m at time slot t. The factor $1/d_t^{m,f}$ in Eq. (7.1) indicates that the caching cost of a specific file will be lower if the file is repeatedly requested.

2) Reactive fetching cost: The cost of server m fetching file f from a neighbouring server m' at time slot t is given by

$$P_t^{m,m',f} = \frac{\tau^{mm'} n_f}{d_t^{m,f}}, \forall m' \neq m, m \in \mathcal{M}, m' \in \mathcal{M}^m, f \in \mathcal{F}, t \in \mathcal{T}, \tag{7.2}$$

where $\tau^{mm'} > 0$ is the unit fetching cost of file migration from a neighbouring server m' to the local server m. We further assume that $\tau^{mm'_1} \neq \tau^{mm'_2}, \forall m'_1 \neq m'_2, m'_1, m'_2 \in \mathcal{M}^m$. Similar to the proactive caching cost, the factor $1/d_t^{m,f}$ in Eq. (7.2) is applied to reduce the fetching cost if file f is requested by multiple users associated to server m.

3) Remote retrieving cost: Obviously, the QoS will be significantly affected by users who cannot obtain the requested file from any available edge servers. Let $P_t^{m,c,f}$ denote the cost of server m retrieving file f from remote cloud center at time slot t, expressed as

$$P_t^{m,c,f} = \tau^{\text{ret}} n_f d_t^{m,f}, \forall m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}, \tag{7.3}$$

where $\tau^{\text{ret}} > 0$ is the retrieving cost per unit file size. The factor $d_t^{m,f}$ in Eq. (7.3) is applied as a penalty, which means the retrieving cost will be higher in case multiple users request the same file f which is not cached locally. Given the fact that the corresponding cost is the largest for retrieving files from remote cloud center, we have $\tau^{\text{ret}} \gg \tau^{mm'} > \tau^{\text{cache}}$.

7.2.3 Joint cache update and request delivery problem

Using the definitions and the cost functions developed above, we formulate the problem of the optimal cache update and request delivery as a long-term foresighted minimization of the overall system cost over a time horizon $t \in \mathcal{T}$ by jointly finding the binary decision vector $\mathbf{a}_t^m, \mathbf{h}_t^m$ for each and every server m as

$$\min_{\mathbf{a}_t^m, \mathbf{h}_t^m} \sum_{t\in\mathcal{T}} \sum_{f\in\mathcal{F}} P_t^{m,f} a_t^{m,f} + P_t^{m,m',f} h_t^{m,m',f} + P_t^{m,c,f} h_t^{m,c,f} \tag{7.4}$$

$$\text{s. t.} \sum_{f\in\mathcal{F}} n_f a_t^{m,f} \leq N^m, \ \ \forall m \in \mathcal{M}, t \in \mathcal{T}, \tag{7.5}$$

$$a_t^{m,f} + h_t^{m,m',f} + h_t^{m,c,f} = 1, \ \forall m \in \mathcal{M}, t \in \mathcal{T}, f \in \mathcal{F}, \tag{7.6}$$

where $\mathbf{a}_t^m = \left[a_t^{m,1}, a_t^{m,2}, \ldots, a_t^{m,F}\right]$ denotes the proactive cache updating decision vector, $\mathbf{h}_t^m = \left[h_t^{m,m',1}, \ldots, h_t^{m,m',F}, h_t^{m,c,1}, \ldots, h_t^{m,c,F}\right]$ denotes the content migration decision vector, (7.5) is the cache size constraint, and constraint (7.6) indicates that a request can be satisfied by one place only, either an edge server or the remote cloud.

The problem in (7.4) heavily involves binary variables, and hence falls within the class of binary optimization problems, which is combinatorial and hard to solve by conventional means. It also involves evolution over time and aims at minimizing a long-term cost function without any prior knowledge of the statistical dynamics of the network, such as the unknown user request transition probabilities. To learn these environmental dynamics, we introduce RL to capture the features of heterogeneous user requests and derive an optimal content caching and delivery policy, as elaborated in the next section.

7.3 Problem Decomposition

In this section, we first define the RL model for each edge server in terms of state, action and reward. Based on the formulated RL model, we cast the optimization problem in (7.4) into two subproblems: coordinated proactive cache updating and local request processing.

7.3.1 RL model

Let us define the RL model for each server $m \in \mathcal{M}$ (i.e., an agent), which is composed of a server-specific state vector $\mathbf{s}_t^m$, a collaborative action vector $\mathbf{A}_t^m \in \mathcal{A}^m$ and a server-specific reward r_t^m, at each time-slot t, as follows.

State: Let the state vector m at time slot t, $\mathbf{s}_t^m \in \mathcal{S}_m$, be defined as

$$\mathbf{s}_t^m = \left[\mathbf{d}_t^m, \mathbf{a}_{t-1}^m\right]^\top, \tag{7.7}$$

where $\mathbf{d}_t^m = \left[d_t^{m,1}, d_t^{m,2}, \ldots, d_t^{m,F}\right]$ denotes the number of requests for each file in edge server m at the current time slot t, and $\mathbf{a}_{t-1}^m = \left[a_{t-1}^{m,1}, a_{t-1}^{m,2}, \ldots, a_{t-1}^{m,F}\right]$ indicates the corresponding caching states of the files at the previous time slot

$t-1$.

Action: The action vector $\mathbf{A}_t^m = \left[\mathbf{a}_t^m, \mathbf{h}_t^m\right]^\top$ for each edge server contains two component vectors. Namely, the proactive cache updating action for server m,

$$\mathbf{a}_t^m = \left[a_t^{m,1}, a_t^{m,2}, \ldots, a_t^{m,F}\right], \tag{7.8}$$

and the content migration action by server m,

$$\mathbf{h}_t^m = \left[h_t^{m,m',1}, \ldots, h_t^{m,m',F}, h_t^{m,c,1}, \ldots, h_t^{m,c,F}\right], \tag{7.9}$$

where the elements $\left\{h_t^{m,m',f}\right\}_{f\in\mathcal{F}}$ denote the content fetching action, and the elements $\left\{h_t^{m,c,f}\right\}_{f\in\mathcal{F}}$ denote the content retrieving action.

Reward: According to problem (7.4), our optimization objective is to minimize the overall system serving cost in a long run. Let us denote the instantaneous reward of each requested file f in server m at time slot t as

$$r_t^{m,f} = P_t^{m,f} a_t^{m,f} + P_t^{m,m',f} h_t^{m,m',f} + P_t^{m,c,f} h_t^{m,c,f}. \tag{7.10}$$

Hence, the instantaneous reward for the agent m at time slot t is given as a sum of serving cost, i.e.,

$$r_t^m = \sum_{f\in\mathcal{F}} r_t^{m,f}. \tag{7.11}$$

Consequently, the formulation in (7.4) can be achieved by minimizing the cumulative reward which depends on a sequence of actions made over the time horizon T.

7.3.2 Decomposition into two subproblems

According to the RL model formulated in subsection 7.3.1, the action space dimension is notably large, leading to extensive exploration of the action-state space in RL. As a result, it will take a substantial amount of time to achieve convergence. Moreover, the direct application of RL for solving the optimization problem in (7.4) becomes impractical with the growth of cache size, number of users, and number of files. To address the issue, we decompose the joint content cache and delivery problem into two subproblems, as follows.

P1 (Coordinated proactive cache updating): The subproblem P1 determines the proactive cache updating decisions according to the following foresighted optimization problem over a time horizon $t \in \mathcal{T}$

$$\min_{\mathbf{a}_t^m} \sum_{t\in\mathcal{T}} \sum_{f\in\mathcal{F}} P_t^{m,f} a_t^{m,f}, \quad \text{s. t. (7.5).} \tag{7.12}$$

Since the optimal solution to the problem in (7.12) requires look-ahead

knowledge of the caching parameters, the assumption of obtaining them instantaneously violates the causality principle. Hence, the myopic optimization per time slot would compromise the optimality in the long run due to the coupling amongst the optimization variables across time. Furthermore, the decision variables $\mathbf{a}_t^m$ should be determined coordinately among the distributed servers in order to minimize the overall system cost.

P2 (Local request processing): Subproblem P2 determines a source for server m, either another server in the neighborhood or the cloud, from which a missing content can be fetched cost-efficiently, by making a decision on $\mathbf{h}_t^m$ according to the following optimization problem,

$$\min_{\mathbf{h}_t^m} \sum_{f \in \mathcal{F}} P_t^{m,m',f} h_t^{m,m',f} + P_t^{m,c,f} h_t^{m,c,f}, \quad \text{s. t. (7.6).} \tag{7.13}$$

Remark 7.1. *At the beginning of time slot t, edge server m receives a set of user requests $\{D_t^{m,u}\}_{u \in \mathcal{U}^m}$. Any requests must first be satisfied by the local cache if they are currently cached. Otherwise, the edge server needs to take a content migration action to fetch the requested files, referred to as a cache miss. In other words, cache misses may occur during the coordinated proactive caching iterations within each time slot. Therefore, decisions on content migration action $\mathbf{h}_t^m$ need to be made reactively within each time slot.*

7.4 Algorithm Design

As shown in Fig. 7.2, we develop a unified federated DQN caching scheme to handle and coordinate the aforementioned two subproblems. As revealed in Remark 7.1, changing the cache update policy at distributed edge servers necessitates a comprehensive adjustment across all servers' cache statuses; thus, it concurrently affect the content migration actions of all servers. To address this, in each time slot, we first optimize the cache update policy at distributed edge servers, as stated in subproblem P1, (7.12). We introduce a federated DQN caching algorithm designed to solve the subproblem (7.12), detailed in subsection 7.4.1. After determining the cache update policy at each server, we proceed to optimize subproblem P2, (7.13) with the cache update policy fixed. Particularly, we apply a greedy algorithm to find the optimal $\mathbf{h}_t^{m^*}$ as a solution to subproblem (7.13), discussed in subsection 7.4.2. Finally, we conduct an analysis of the computational complexity of the proposed unified federated DQN caching scheme in subsection 7.4.3.

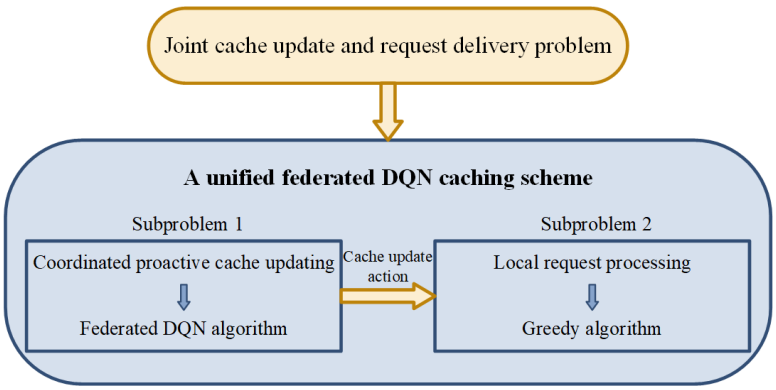

Figure 7.2: The framework of the unified federated DQN caching scheme.

7.4.1 A federated DQN caching algorithm

In this subsection, we develop a coordinated learning-based algorithmic solution to subproblem P1, (7.12). Specifically, we introduce a federated DQN caching algorithm composed of a scalable DQL approach, to learn the heterogeneous users' content demands and the FL [116] to coordinate among multiple servers by aggregating their local DQN model parameters.

Generally, the proposed federated DQN algorithm involves three functional stages. In the first stage, each edge server trains its local DQN model according to local observations. In the second stage, the global server, which is deployed in the cloud center, chooses a number of edge servers to acquire their local model parameters for training the global DQN model. In the third stage, the trained global model parameters are distributed to all edge servers for further refining their local DQL models. Thereafter, each edge server makes caching decisions based on the globally updated local model.

The scalable DQN algorithm presented in Chapter 6 has been confirmed to be adaptive to dynamic and heterogeneous user requests, as well as robust in large-scale caching scenarios. Consequently, we apply this approach here to construct DQN models for distributed edge servers. For the details of the scalable DQN algorithm, we refer to Section 6.4. The remiander of this subsection will thus be on introducing the FL-empowered training process for the global DQN model.

Overall, the structure of FL provides two key technical merits: 1) The distributed nature of data storage and processing in FL substantially diminishes the complexity associated with centralized data storage and processing. 2) The global model server only collects local model parameters from edge servers, rather than their complete data. This can efficiently reduce the signaling overhead.

From the perspective of our system, utilizing FL can further enhance the performance of training DQN models at distributed server through global coordination. In particular, our proposed approach notably alleviates the exces-

sive signaling overhead required by collecting all the experiences from edge severs in a centralized unit. This is achieved by developing a RL algorithm based on TS [122]. Additionally, we decompose the DQN parameters of edge servers, and only the parameters of the common-network part are exchanged with the global model server, in order to further facilitate convergence of the proposed scheme. In the sequel, we will introduce the two phases of FL-empowered training process: 1) TS-based smart agent selection; 2) Model aggregation and distribution.

TS-based Smart agent selection

In collaborative caching system, edge servers repeatedly perform cache updating while interacting with the environment. Naturally, there exists some edge servers achieving better caching performance compared to the others. Therefore, smart selection of participating edge servers in FL is central for better performance and reduced network complexity.

Let $L\big(\boldsymbol{\theta}^G\big)$ define the global loss function as

$$L\big(\boldsymbol{\theta}^G\big) = \sum_{m\in\mathcal{M}} \frac{|\mathcal{B}^m|}{B} L(\boldsymbol{\theta}^m), \tag{7.14}$$

where $\mathcal{B}^m, \forall m \in \mathcal{M}$ is the local dataset of server m, $B = \sum_{m\in\mathcal{M}} |\mathcal{B}^m|$, $L(\boldsymbol{\theta}^m) = \mathbb{E}_{(\mathbf{s}_j^m, \mathbf{a}_j^m, \mathbf{r}_j^m, \mathbf{s}_{j+1}^m\}) \sim U(\mathcal{B}^m)} \big\| \boldsymbol{\delta}(\boldsymbol{\theta}^m) \big\|_2^2$ is the loss function for server m. Unless specified otherwise, the symbols and functions associated with the DQN models used hereafter are aligned with those defined in Chapter 6.

The aim of the proposed smart agent selection is to minimize the global loss function Eq. (7.14) in the long run, which is equivalent to the minimization of TD errors of all servers, defined in Eq. (6.19), i.e., $\boldsymbol{\delta}(\boldsymbol{\theta}^m) = \big[\mathbf{y}_t^m - \mathbf{Q}^m(\mathbf{s}_t^m; \boldsymbol{\theta}^m)\big] \odot \mathbf{a}_t^m$.

The TD error is the difference between the target Q value and the approximated soft Q value, where the target Q value is actually the expected reward. Thus, the minimization of TD errors of all servers can be regarded as the maximization of the long-term accumulated rewards of all servers. Consequently, above global loss function minimization problem can be re-formulated as a long-term reward maximization problem, expressed as,

$$\max_{\mathbf{x}_t} \sum_{t\in\mathcal{T}} \sum_{m\in\mathcal{M}} x_t^m \cdot r_t^m, \tag{7.15}$$

where $\mathbf{x}_t = \{x_t^1, x_t^2, ..., x_t^M\}$, and each element $x_t^m \in \{0, 1\}$ is a binary variable to indicate whether server m is selected at time slot t. Specifically, $x_t^m = 1$ if server m is selected at time slot t, and $x_t^m = 0$, otherwise. Let $\mathcal{A}_x$ denote the set of all possible solutions for agent selection problem in Eq. (7.15). Hence,

the number of possible solutions for $\mathbf{x}_t$ is $|\mathcal{A}_x| = 2^M$.

To avoid biased selection of servers in which only the servers with current best performance are chosen, we propose a smart agent selection method based on TS. Typically, TS is a RL algorithm for online decision making, where actions are taken sequentially to better balance exploitation and exploration, so as to maximize the achievable accumulated rewards.

Supposing that the global server performs agent selection every T_s time slots, we calculate the sum of accumulated rewards obtained by all servers in T_s consecutive time slots as

$$R_{\mathbf{x}_t} = \sum_{i=t-T_s+1}^{t} \sum_{m \in \mathcal{M}} r_i^m. \tag{7.16}$$

Furthermore, we normalize $R_{\mathbf{x}_t}$ as $\hat{R}_{\mathbf{x}_t} \in [0, 1]$ and interpret the normalized value $\hat{R}_{\mathbf{x}_t}$ as the success probability of selecting action $\mathbf{x}_t$, which will be used in the sequential operations of TS.

In TS algorithm, a Beta distribution with parameters $a_{\mathbf{x}_t}$ and $b_{\mathbf{x}_t}$ is generated for each action $\mathbf{x}_t \in \mathcal{A}_x$, to estimate the probability of selecting the action $\mathbf{x}_t$, denoted as $\eta_{\mathbf{x}_t} \in [0, 1]$. Recall that the Beta distribution is a probability distribution on probabilities; hence, its domain is bounded between 0 and 1. Specifically, $\eta_{\mathbf{x}_t}$ is a probability random variable drawn from the Beta distribution, i.e., $\eta_{\mathbf{x}_t} \sim \text{Beta}(a_{\mathbf{x}_t}, b_{\mathbf{x}_t})$.

Once the action $\mathbf{x}_t$ is selected, the parameters of corresponding Beta distribution will be updated according to the following rule

$$\begin{cases} a_{\mathbf{x}_{t+1}} \leftarrow a_{\mathbf{x}_t} + u_{\mathbf{x}_t} \\ b_{\mathbf{x}_{t+1}} \leftarrow a_{\mathbf{x}_t} + 1 - u_{\mathbf{x}_t}, \end{cases} \tag{7.17}$$

where $u_{\mathbf{x}_t}$ is an independent sample drawn from the Bernoulli distribution with success probability $\hat{R}_{\mathbf{x}_t}$. If $\hat{R}_{\mathbf{x}_t}$ is close to 1, which implies that the action $\mathbf{x}_t$ could potentially improve the overall caching performance, there is a high probability that $a_{\mathbf{x}_{t+1}}$ increases by 1 while $b_{\mathbf{x}_{t+1}}$ keeps unchanged, and the mean of Beta distribution increases accordingly. On the contrary, if $\hat{R}_{\mathbf{x}_t}$ is close to 0, which implies that the action $\mathbf{x}_t$ may not benefit the overall caching performance, there will be a high probability that $b_{\mathbf{x}_{t+1}}$ increases by 1 while $a_{\mathbf{x}_{t+1}}$ keeps unchanged, and the mean of Beta distribution decreases accordingly.

Note that only the parameters of a selected action are updated via Eq. (7.17), and the proposed smart agent selection is performed by the global server according to

$$\mathbf{x}_t = \arg \max_{\mathbf{x}' \in \mathcal{A}_x} \eta_{\mathbf{x}'}. \tag{7.18}$$

Consequently, the signaling overhead is effectively reduced as only a fraction of edge servers are selected to upload their model parameters to the global server. Moreover, potential biased selection is alleviated by enabling better exploration among edge servers.

Model aggregation and distribution

Different from the existing FL algorithms, such as the FedAvg algorithm [116], which aggregates all local model parameters to the global model, we preserve the output layer of neural networks locally. This is driven by the fact that the output layer in our proposed framework is designed to encapsulate the *predicted importance* of files, which can be interpreted as the specific users' preference of each file at the individual server. Hence, the parameters of the output layer need to be trained individually by each server to accommodate the heterogeneous requests from covered users.

The early layers of neural network are referred to as the common-network part. We denote the common-network parameters and the parameters of the output layer as $\boldsymbol{\theta}^{m_c}$ and $\boldsymbol{\theta}^{m_o}, m \in \mathcal{M}$, respectively. As depicted in Fig.7.3, during each training round of FL, the selected servers only upload their parameters of the common-network part to the global server. Then the global common-network parameters $\boldsymbol{\theta}_t^{G_c}$ are updated according to the following rule

$$\boldsymbol{\theta}_t^{G_c} = \frac{1}{\sum_{m\in\mathcal{M}_t^G} |\mathcal{B}^m|} \sum_{m\in\mathcal{M}_t^G} |\mathcal{B}^m| \boldsymbol{\theta}_t^{m_c}, \tag{7.19}$$

where $\mathcal{M}_t^G$ is the set of selected agents at time slot t.

After aggregation, the updated global parameters $\boldsymbol{\theta}_t^{G_c}$ are distributed to all the edge servers. Next, each server combines the newly obtained common-

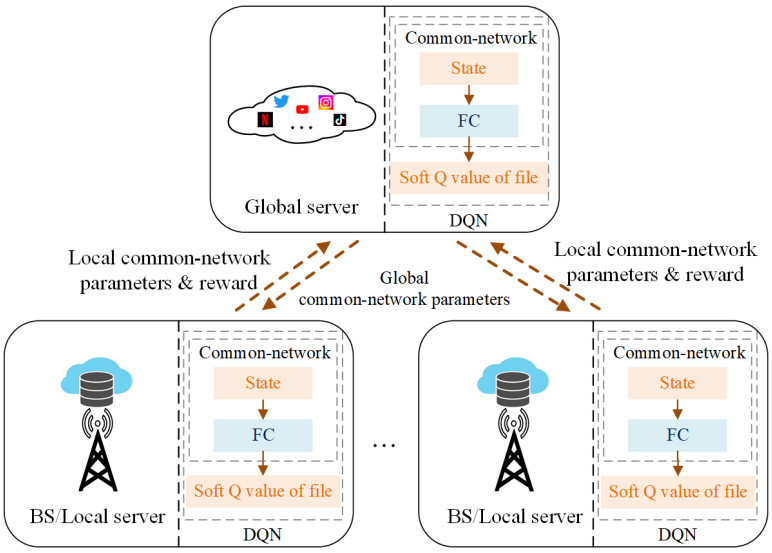

Figure 7.3: Procedure for exchanging parameters in the proposed DQN model.

network parameters with the locally trained output layer parameters to form the updated parameters of its local DQN model, i.e., $\boldsymbol{\theta}_{t+1}^{m} \leftarrow \left\{\boldsymbol{\theta}_{t}^{G_c}, \boldsymbol{\theta}_{t}^{m_o}\right\}$. Based on this newly updated local model, each server makes caching decisions while performing local model training in the subsequent time slots.

7.4.2 Local request processing

As aforementioned, during each time slot, content migration decisions are made reactively during each time slot when edge servers encounter cache misses. Considering that the number of user requests received at each time slot is finite and the edge servers aim to fetch files with the lowest cost, we introduce a greedy algorithm to find the optimal content migration action and solve problem in (7.13), as described below.

It is noted that the content migration action comprises the fetching action $\left\{h_t^{m,m',f} \middle| m' \in \mathcal{M}^m, f \in \mathcal{F}\right\}$ and the retrieving action $\left\{h_t^{m,c,f} \middle| f \in \mathcal{F}\right\}$. The retrieving action makes sense and can only be determined after the fetching action is determined. In our file fetching protocol, the local server consistently fetches the file f with the lowest fetching cost. Thus, the optimal fetching action is given by

$$h_t^{m,m',f*} = \begin{cases} 1, \text{ if } \prod_{j=1}^{i-1} \left(1 - a_t^{(j)^m,f}\right) a_t^{(i)^m,f} = 1, \\ 0, \text{ otherwise }, \end{cases} \\ \forall m' \neq m, m \in \mathcal{M}, m' \in \mathcal{M}^m, f \in \mathcal{F}, t \in \mathcal{T}, \tag{7.20}$$

where $m' = (i)^m$ is the server with i-th lowest fetching cost in terms of server m.

Once the optimal fetching action has been decided, the optimal retrieving action can be sequentially calculated as

$$h_t^{m,c,f*} = \begin{cases} 1, \text{ if } a_t^{m,f} + \sum_{m' \in \mathcal{M}^m} h_t^{m,m',f} = 0, \\ 0, \text{ otherwise }, \end{cases} \\ m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}. \tag{7.21}$$

In a nutshell, the proposed unified federated DQN caching scheme is summarized in Algorithm 7.

7.4.3 Algorithm analysis

We adopt the fully connected (FC) networks to construct the DQN models. Let Z and n_z denote the number of hidden layers of the DQN model, and the

Algorithm 7: The Unified Federated DQN Caching Scheme

Initialize $\{\boldsymbol{\theta}^m, \hat{\boldsymbol{\theta}}^m, \mathcal{B}^m\}_{m\in\mathcal{M}}$, $\boldsymbol{\theta}_t^G$;
for $t \in \mathcal{T}$ **do**
  for *Each edge server* $m \in \mathcal{M}$ ***in parallel*** **do**
    Receive the requests from associated users and observe the current state $\mathbf{s}_t^m$;
    Update the soft Q value of each file in the evaluation neural network according to
    $Q^{m,f}\left(\mathbf{s}_t^m;\boldsymbol{\theta}^m\right) \leftarrow (1-\beta)Q^{m,f}\left(\mathbf{s}_t^m;\boldsymbol{\theta}^m\right) + \beta Q^{m,f}\left(\mathbf{s}_{t-1}^m;\boldsymbol{\theta}^m\right)$;
    Take proactive cache updating action

$$\mathbf{a}_t^m = \begin{cases} \mathbf{a}_t^g, & \text{with } 1-\epsilon \\ \mathbf{a}_t^{\mathbf{v}}, & \text{with } \epsilon \end{cases}$$

    if *any cache miss occurs* **then**
      Take content migration action $\mathbf{h}_t^m$ according to Eq. (7.20) and Eq. (7.21);
    end
    Receive the reward of each file $r_t^m(f)$ as per Eq.(7.10) and observe the next state $\mathbf{s}_{t+1}^m$;
    Store the transition $\{\mathbf{s}_t^m, \mathbf{a}_t^m, \mathbf{r}_t^m, \mathbf{s}_{t+1}^m\}$ into the buffer $\mathcal{B}^m$;
    Randomly sample a mini-batch of transitions $\{\mathbf{s}_j^m, \mathbf{a}_j^m, \mathbf{r}_j^m, \mathbf{s}_{j+1}^m\}$ from $\mathcal{B}^m$;
    Update the evaluation neural network parameters $\boldsymbol{\theta}_t^m$ using stochastic gradient descent algorithm as per Eq. (6.20);
    if $t \mod K = 0$ **then**
      Update the target neural network parameters $\hat{\boldsymbol{\theta}}_t^m \leftarrow \boldsymbol{\theta}_t^m$;
    end
  end
  if $(t \mod T_s) = 0$ **then**
    The global server selects a set of servers $\mathcal{M}_t^G$ according to TS-based smart agent selection, i.e.,

$$\mathbf{x}_t = \arg\max_{\mathbf{x}'\in\mathcal{A}_x} \eta_{\mathbf{x}'}$$

    Each selected server uploads its common-network parameters $\boldsymbol{\theta}_t^{m_c}, m \in \mathcal{M}_t^G$ to the global server;
    The global server updates global parameters according to

$$\boldsymbol{\theta}_t^{G_c} = \frac{1}{\sum_{m\in\mathcal{M}_t^G}\left|\mathcal{B}^m\right|}\sum_{m\in\mathcal{M}_t^G}\left|\mathcal{B}^m\right|\boldsymbol{\theta}_t^{m_c}$$

    The global server send $\boldsymbol{\theta}_t^{G_c}$ back to all edge servers;
    Each server $m \in \mathcal{M}$ updates its local model with newly obtained parameters as $\boldsymbol{\theta}_{t+1}^m \leftarrow \left\{\boldsymbol{\theta}_t^{G_c}, \boldsymbol{\theta}_t^{m_o}\right\}$;
  end
end

number of neurons in the z-th hidden layer, respectively. The time complexity of the adopted DQN can be expressed as $\Phi = J(2F \cdot n_1 + \sum_{z=1}^{Z-1} n_z \cdot n_{z+1} + F \cdot n_Z)$, where $J(\cdot)$ is the time complexity of updating the parameters of fully connected layers. Thus, the complexity of backpropagation for training a DQN model in a single transition is $\mathcal{O}(\Phi)$. Since the local DQN models are trained in parallel, we only consider the complexity of one local model. Therefore, the time complexity of the proposed scheme is $\mathcal{O}(\Phi \times N \times T)$, where N is the number of transitions sampled in each training round.

7.5 Simulation Results

In this section, we set up simulation scenarios to evaluate the performance of our developed caching scheme under various settings and compare the results against some of the well-known existing caching approaches.

7.5.1 Simulation settings

We use the same user request model as in Chapter 6. The details of this request model can be found in Section 6.5. The specific values for the parameters of user request models will be specified in the context of different experiments. We construct the neural network for each edge server with an identical architecture, as follows: one input layer, two hidden layers with ReLu activation and one output layer with sigmoid activation. Unless specified otherwise, other main simulation parameters are given in Table 7.1.

Table 7.1: Parameter settings

Parameter	Value
Learning rate (α)	0.01
Discount factor (γ)	0.9
Q value discount rate (β)	0.95
Replay buffer size	2000
Mini-batch size	128
Optimizer	Adam
Common-network part	$2F \times 1024 \times 512$
Output layer	$512 \times F$
Target network updating frequency (K)	300
Agent selection frequency (T_s)	100

We compare the performance of the proposed scheme against three baseline algorithms: 1) Distributed training scheme, where each server trains its own DQN model with local experiences independently. There is no global server that facilitates information exchange among edge servers, and all servers

make caching decisions based on their individually trained models. Alternatively, we can refer to this scheme as the proposed method without FL. 2) LFU. 3) LRU.

7.5.2 Experiments with the number of servers $M = 6$ and the number of files $F = 1000$

In this subsection, we conduct comprehensive experiments with a scenario consisting of $M = 6$ edge servers. Particularly, each server covers different number of users, i.e., $U^1 = 20, U^2 = 30, U^3 = 40, U^4 = 50, U^5 = 60, U^6 = 70$. Each server is assumed to have $|\mathcal{M}^m| = 4$ proximal servers, and the unit fetching costs are set as $\tau^{12} = \tau^{23} = 3, \tau^{34} = \tau^{45} = 4, \tau^{56} = \tau^{61} = 5, \tau^{13} = 6, \tau^{24} = 7, \tau^{26} = \tau^{35} = 8, \tau^{46} = 9, \tau^{15} = 10$. Besides, the unit caching cost and the unit retrieving cost are set to be $\tau^{\text{cache}} = 1, \tau^{\text{ret}} = 20$, respectively. There are $F = 1000$ files in the cloud center, and we assign $n_f = 1$ for all $f \in \mathcal{F}$. The parameters of user request models are aperiodically and randomly drawn from $P_0 \in \{0.1, 0.2, 0.3\}$, $\lambda \in \{0.8, 1.0, 1.2\}$, $G \in \{25, 50, 75\}$.

1) **Performance with various exploration methods**

In Fig. 7.4, we evaluate the performance of the proposed algorithm using request-driven exploration, random exploration (i.e., each non-greedy action is selected with a uniform probability for exploration), and a greedy action selection scheme (i.e., only the files with the highest soft Q values are cached). Aligned with the results of the experiments conducted in subsection 6.5.3, the proposed request-driven exploration method achieves the lowest averaged system cost compared to the other two schemes across various cache sizes. This again confirms that our proposed request-driven exploration method effectively improves the caching performance by leveraging the historical request information.

2) **Performance with different DQN model parameter exchange methods**

As mentioned in subsection 7.4.1, we individually train the output layer parameters of each server to preserve users' preferences at different servers, instead of aggregating all the local model parameters into the global model. To justify this design, we investigate the averaged system cost of the proposed algorithm with different DQN model parameter exchange methods in Fig. 7.5. It is observed that the overall averaged system cost of the proposed method is significantly lower than that of the complete parameter exchange method over time. This improvement is attributed to the fact that the proposed scheme learns cache updating policies more effectively,

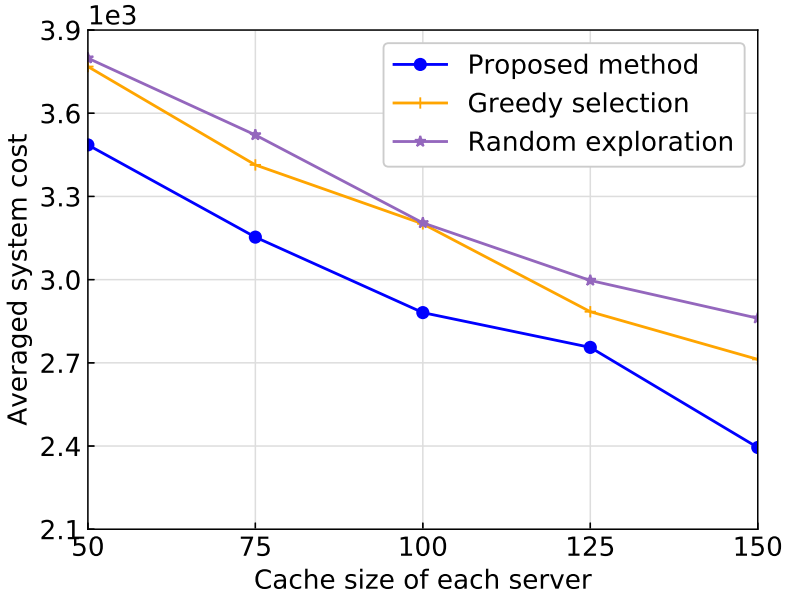

Figure 7.4: Averaged system cost with various exploration methods.

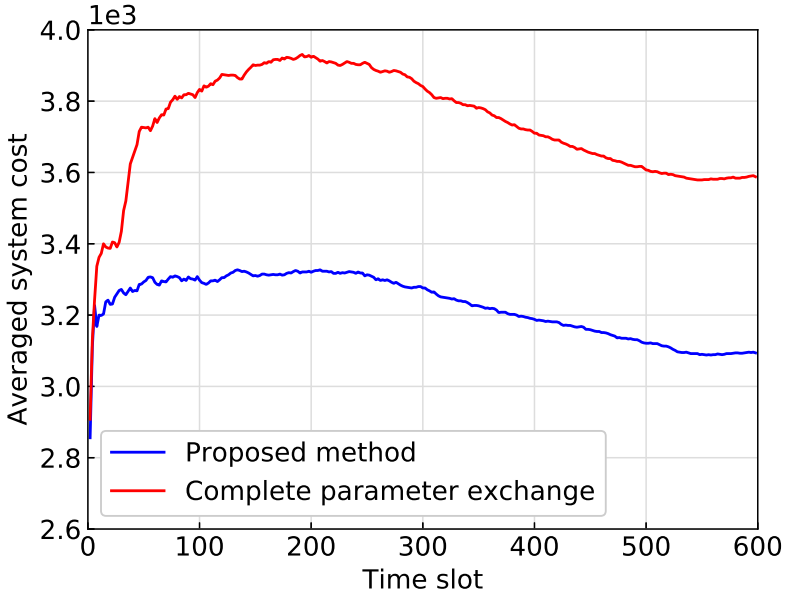

Figure 7.5: Averaged system cost with different DQN model parameter exchange methods, where the cache size of each server is set to $N^m = 100, \forall m \in \mathcal{M}$.

benefiting from well-preserved local users' preferences and more efficient global coordination among the DQN models at distributed edge servers.

3) **Performance with various agent selection methods**

Next, we evaluate the performance of the proposed smart agent selection method with various parameter configurations, as illustrated in Fig. 7.6. Furthermore, we compare the proposed smart agent selection method with greedy selection method, random selection method and all-inclusive method.

In the greedy method, edge servers with the currently highest accumulated rewards are selected to upload their local parameters for global model training. In the random method, edge servers are randomly selected for global model training. In the all-inclusive method, the local parameters of all edge servers are aggregated to update the global model parameters.

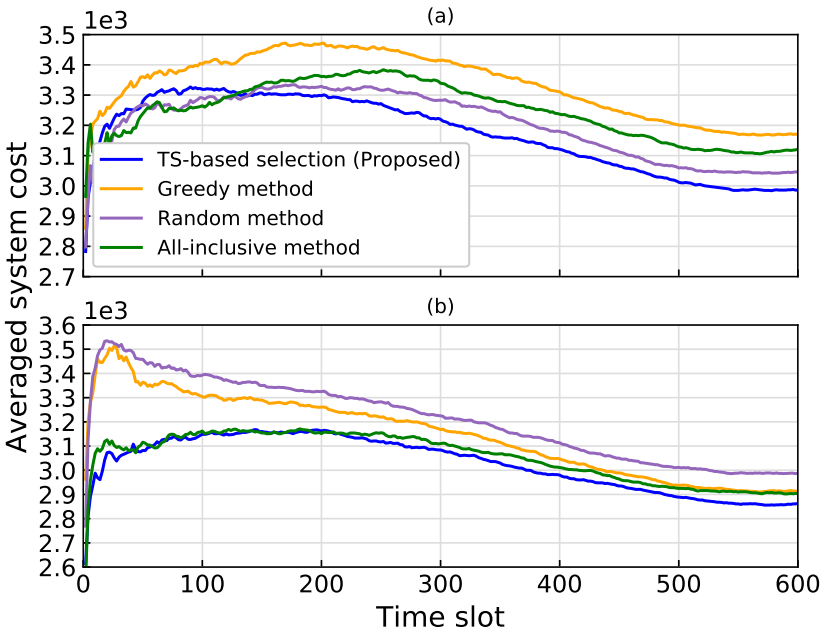

Figure 7.6: Averaged system cost with various agent selection methods under different parameter configurations. (a) Identical cache size for each server: $N^m = 100, \forall m \in \mathcal{M}$. (b) Different cache sizes for edge servers: $N^1 = 60, N^2 = 80, N^3 = 100, N^4 = 120, N^5 = 140, N^6 = 160$.

Fig. 7.6(a) shows the averaged system cost of all four selection methods under the configuration where each server has an identical cache size, i.e., $N^m = 100, \forall m \in \mathcal{M}$. In contrast, the cache sizes of edge servers are set to be different as $N^1 = 60, N^2 = 80, N^3 = 100, N^4 = 120, N^5 = 140, N^6 = 160$ in Fig. 7.6(b). Since the aim of the proposed smart agent selection is to minimize the system cost in the long run, it incurs slight performance losses due to exploration at the beginning, as shown in both Fig. 7.6(a) and Fig. 7.6(b). Nevertheless, the overall performance of the proposed smart agent selection outperforms the other three methods. These results suggest the effectiveness of the proposed method under various parameter configurations.

4) **Comparison of overall cache hit rate**

As demonstrated in Fig. 7.7, we compare the cache hit rate of our proposed algorithm with three baselines, i.e., distributed training scheme, LFU and LRU, under different cache sizes. It shows that the cache hit rate of

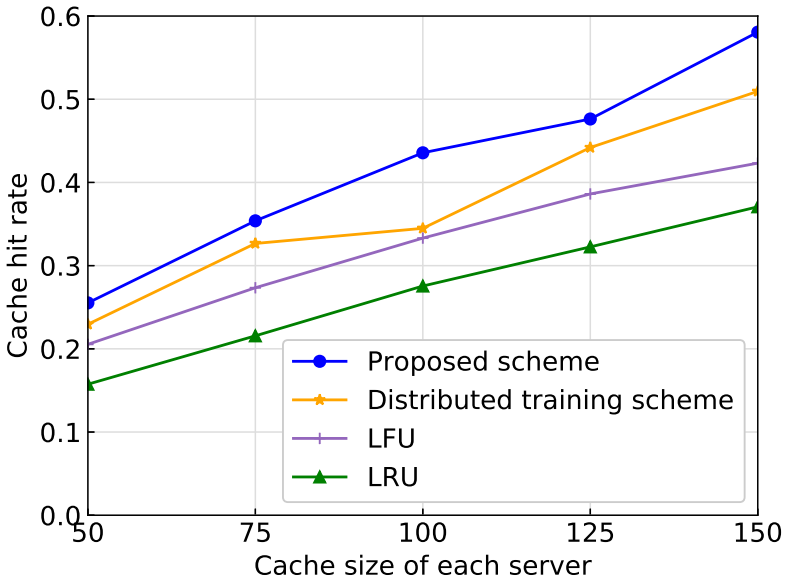

Figure 7.7: Comparison of cache hit rate of the proposed algorithm, distributed training scheme, LFU and LRU under different cache sizes with $M = 6, F = 1000$.

all schemes increases with the growing cache size of each server, ranging from 50 to 150 units. In particular, the proposed algorithm outperforms the distributed training scheme, LFU and LRU across all parameter configurations. This improvement is mainly attributed to a more precise prediction of heterogeneous user requests, as well as more effective coordination among edge servers via the FL-empowered training process.

7.5.3 Experiments with the number of servers $M = 25$ and the number of files $F = 1500$

In this subsection, we conduct experiments to investigate the robustness and scalability of our proposed caching scheme in a larger-scale scenario involving $M = 25$ edge servers and $F = 1500$ files.

In contrast to the experiments conducted in subsection 7.5.2, we adopt more flexible and varied parameter settings, particularly regarding the number of users, user request models, as well as sets of proximal servers. Specifically, the number of users covered by each server is randomly drawn from $\{20, 30, 40, 50, 60\}$. The parameters of user request models are aperiodically and randomly drawn from $P_0 \in \{0.1, 0.15, 0.2, 0.25\}$, $\lambda \in \{0.9, 1.0, 1.1, 1.2\}$, and $G \in \{50, 75, 100, 125\}$. Each server is assumed to have a minimum of 2 proximal servers and a maximum of 4 proximal servers. The unit fetching costs are set as $\{\tau^{mm'} = 3 \big| |m' - m| \leq 1, \tau^{mm'} = 6 \big| 1 < |m' - m| \leq 2\}$. Besides, we assign $n_f = 1$ for all $f \in \mathcal{F}$. The unit caching cost and the unit

retrieving cost are set to be $\tau^{\text{cache}} = 1, \tau^{\text{ret}} = 20$, respectively.

1) **Comparison of overall cache hit rate**

As observed in Fig. 7.8, the proposed scheme outperforms the other three baselines across various cache sizes, even with a higher number of edge servers and content files, i.e., $M = 25$ and $F = 1500$. Particularly, the performance gap between the proposed scheme and distributed training scheme indicates the benefits of the FL-empowered training process. In other words, effective information exchange is achieved during the FL-empowered training process, and hence enhancing the caching performance at distributed edge servers.

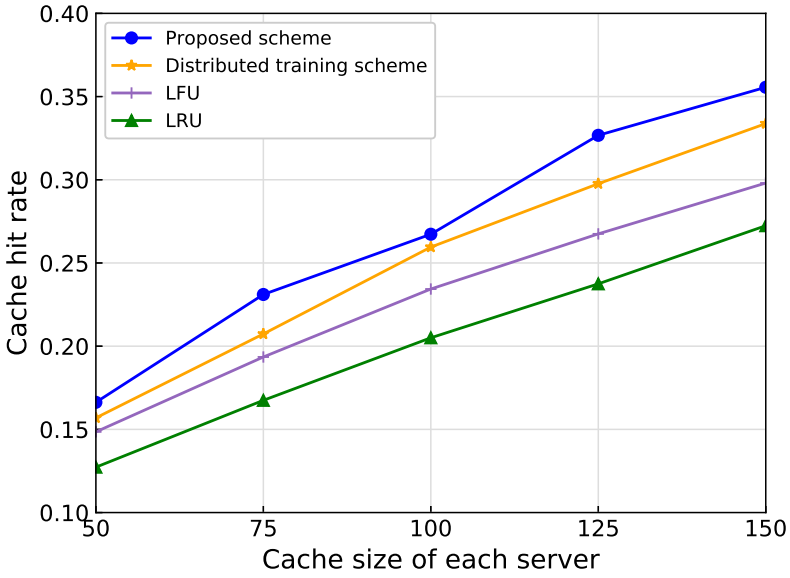

Figure 7.8: Comparison of cache hit rate of the proposed algorithm, distributed training scheme, LFU and LRU under different cache sizes with $M = 25, F = 1500$.

2) **Comparison of averaged system cost and remote retrieving cost**

In the sequel, we compare the performance of four caching schemes in terms of averaged system cost and remote retrieving cost in the large-scale scenario with $M = 25, F = 1500$. Particularly, we conduct experiments under a varied parameter setting, where the cache sizes of each server are sampled at intervals of 5, ranging from 80 to 200. The results in Fig. 7.9 demonstrate the advantages of our proposed scheme over other three baselines in terms of both averaged system cost and remote retrieving cost. Moreover, these results in both Fig. 7.8 and Fig. 7.9 confirm the scalability and effectiveness of our proposed scheme.

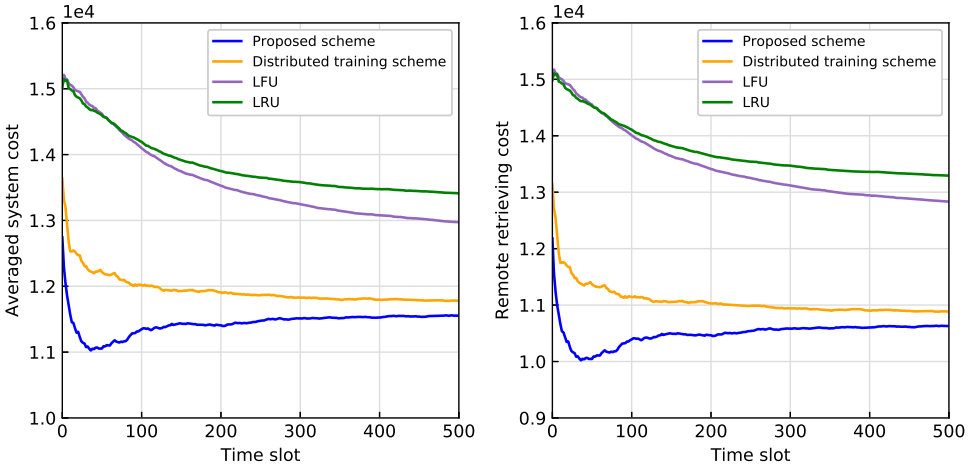

Figure 7.9: Comparison of the proposed algorithm, distributed training scheme, LFU and LRU in terms of averaged system cost and remote retrieving cost with $M = 25, F = 1500$. Besides, the cache sizes of each server are sampled at intervals of 5 from 80 to 200.

7.6 Conclusion

In Chapter 7, we focused on the joint cache update and request delivery problem in an edge caching system with multiple edge servers and heterogeneous users. We first formulated the problem as a long-term system cost minimization problem. To learn the time-varying file popularity and capture the features of heterogeneous user requests, we restated the problem in terms of RL, thereby decomposing the problem into two subproblems in order to deal with the large dimensionality of the action space. A unified federated DQN caching scheme is therefore developed, consisting of a federated DQN algorithm and a greedy algorithm to tackle these two subproblems respectively. In particular, the proposed federated DQN algorithm leverages FL and the scalable DQN algorithm developed in Chapter 6 to train the distributed DQN models coordinately via the FL-empowered training process. Furthermore, a TS-based smart agent selection method is introduced to not only search for optimal server set to participate in the training process, but also reduce signaling overhead among edge servers and the global server. Additionally, to deal with the request heterogeneity and to preserve the users' preferences at each server, we decompose the DQN parameters while the selected servers only exchange the common-network parameters with the global server. Through comprehensive experiments, the results have demonstrated that our proposed scheme outperforms three baseline algorithms in terms of average system cost and cache hit rate. Moreover, the results have confirmed that the proposed unified federated DQN caching scheme can easily fit into the large-scale caching scenarios with highly heterogeneous scenarios and user requests.

Chapter 8

Summary and Future Work

8.1 Thesis Summary

In alignment with the objectives outlined in Section 1.2, three novel caching strategies, which utilize advanced optimization and machine learning techniques, have been proposed in this thesis. These proposed methods successfully overcome the challenges faced by collaborative edge caching systems with high uncertainty and heterogeneity, and improve caching performance as well as QoE for end users. Particularly, two of them demonstrate advantages in network scalability and robustness. These research results are individually summarized as follows.

In Chapter 5, an eDQN algorithm is proposed for MEC systems with heterogeneous caching scenarios. By integrating MFEA into the DQN model training, the joint evolution of multiple DQN models is enabled, allowing multiple servers to collaboratively optimize their caching policies. Experiments are conducted on a real-world dataset, Movielens 100K, and the results confirm that the proposed algorithm significantly reduces downloading latency and enhances caching performance compared to other existing caching schemes. In summary, the proposed algorithm provides an effective caching policy optimization method for multiple edge servers in a collaborative system, which achieves the first objective set in the thesis.

In Chapter 6, a DQN-based two-phase proactive caching scheme is proposed. The specially designed neural network architecture successfully breaks the curse of dimensionality, enabling the scalability of the network. Additionally, a request-driven exploration method is devised to enhance caching performance. The proposed algorithm is compared with classical DQN-based algorithm on small-scale scenarios, demonstrating the priority of the proposed algorithm in terms of robustness and efficiency. Comparative results on large-scale scenarios further highlight the scalability of the proposed algorithm. In summary, the proposed algorithm addresses the challenge of large dimension-

ality, releases the training difficulties, and consequently improves cache hit rate and reduces average system cost, which fulfils the second objective outlined in the thesis.

The research in Chapter 7 extends the scalable DQN algorithm developed in Chapter 6 to a federated DQN algorithm, enhancing the coordination among distributed edge servers through a FL-empowered training process. Moreover, the training process effectively alleviates the heavy burden on signaling overhead through a TS-based smart agent selection procedure. A unified federated DQN caching scheme is designed by coordinating the federated DQN algorithm with a greedy algorithm, efficiently addressing the joint cache update and request delivery problem in a large-scale edge caching network. Furthermore, the robustness of the proposed scheme is evaluated through comprehensive experiments. These achievements fulfil the expectation of the third objective outlined in the thesis.

In a nutshell, efficient caching strategies combining optimization and machine learning techniques have been developed for collaborative edge caching networks, and all the objectives outlined in this thesis can be considered successfully accomplished.

8.2 Future Work

Moving forward, I plan to make further enhancement in developing more adaptive learning-based approaches for edge intelligent systems. Several promising research directions are outlined as follows.

- Further improvements can be made to the research presented in Chapter 7, where user requests are modeled based on file popularity and temporal interfile correlation. More generic scenarios could be explored, considering the possible correlation or dependency between user requests and cached files.

- Investigate the potential of applying asynchronous advantage actor-critic (A3C) [126] to develop collaborative caching strategies for distributed edge servers. A3C generally incorporates parallel actors operating on different threads, enhancing both exploration and stability. In a collaborative edge caching network, multiple edge servers cover various caching scenarios, aligning with the multi-actor structure of A3C. Each edge server can be considered as a local worker in the A3C training scheme. Consequently, this approach will be beneficial to boosting training efficiency and improving stability during the training process.

- Study on the combination of edge caching and recommendation systems. Recommendation systems have significantly influenced users' behavior while browsing the Internet, potentially impacting caching decisions for popular contents [127]. As a result, recommendation-enabled edge caching is expected to be a systematic and efficient way to capture heterogeneous user preferences and improve performance gain, e.g., cache hit rate, or resource utilization efficiency.

Bibliography

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

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[3] Z. Sun and M. R. Nakhai, "Distributed learning-based cache replacement in collaborative edge networks," *IEEE Communications Letters*, vol. 25, no. 8, pp. 2669–2672, 2021.

[4] "Cisco Annual Internet Report (2018–2023) White Paper," January 2022.

[5] ETSI, "Mobile-edge computing-introductory echnical white paper," September 2014.

[6] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[7] Y. Chen, K. Wu, and Q. Zhang, "From qos to qoe: A tutorial on video quality assessment," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 1126–1165, 2014.

[8] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[9] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2525–2553, 2019.

[10] E. A. Feinberg and A. Shwartz, *Handbook of Markov decision processes: methods and applications*, vol. 40. Springer Science & Business Media, 2012.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[12] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[13] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, "Understanding performance of edge content caching for mobile video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1076–1089, 2017.

[14] J. W. Murray, *Building virtual reality with Unity and Steam VR*. Crc Press, 2017.

[15] J. Tham, A. H. Duin, L. Gee, N. Ernst, B. Abdelqader, and M. McGrath, "Understanding virtual reality: Presence, embodiment, and professional practice," *IEEE Transactions on Professional Communication*, vol. 61, no. 2, pp. 178–195, 2018.

[16] T. Taleb, A. Ksentini, M. Chen, and R. Jantti, "Coping with emerging mobile social media applications through dynamic service function chaining," *IEEE Transactions on Wireless Communications*, vol. 15, no. 4, pp. 2859–2871, 2015.

[17] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[18] M. Satyanarayanan, "Mobile computing: the next decade," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 15, no. 2, pp. 2–10, 2011.

[19] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[20] B. P. Rimal, D. P. Van, and M. Maier, "Mobile-edge computing vs. centralized cloud computing in fiber-wireless access networks," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 991–996, IEEE, 2016.

[21] J. Zhang, W. Xie, F. Yang, and Q. Bi, "Mobile edge computing and field trial results for 5g low latency scenario," *China Communications*, vol. 13, no. 2, pp. 174–182, 2016.

[22] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the impact of edge computing on mobile applications," in *Proceedings of the 7th ACM SIGOPS Asia-Pacific workshop on systems*, pp. 1–8, 2016.

[23] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE access*, vol. 4, pp. 5896–5907, 2016.

[24] D.-G. Zhang, L. Chen, J. Zhang, J. Chen, T. Zhang, Y.-M. Tang, and J.-N. Qiu, "A multi-path routing protocol based on link lifetime and energy consumption prediction for mobile edge computing," *IEEE Access*, vol. 8, pp. 69058–69071, 2020.

[25] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[26] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[27] A. Somov and R. Giaffreda, "Powering iot devices: Technologies and opportunities," *IEEE IoT Newsletter*, vol. 367, p. 368, 2015.

[28] B. Shi, J. Yang, Z. Huang, and P. Hui, "Offloading guidelines for augmented reality applications on wearable devices," in *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 1271–1274, 2015.

[29] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud," *IEEE Internet of Things Journal*, vol. 3, no. 3, pp. 327–338, 2015.

[30] Y. Gao, W. Hu, K. Ha, B. Amos, P. Pillai, and M. Satyanarayanan, "Are cloudlets necessary?," *School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-139*, p. 8, 2015.

[31] S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner, and A. Schneider, "Enabling real-time context-aware collaboration through 5g and mobile edge computing," in *2015 12th In-*

ternational Conference on Information Technology-New Generations, pp. 601–605, IEEE, 2015.

[32] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 414–454, 2013.

[33] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, 2017.

[34] X. Luo, "From augmented reality to augmented computing: A look at cloud-mobile convergence," in *2009 International Symposium on Ubiquitous Virtual Reality*, pp. 29–32, IEEE, 2009.

[35] A. A. Abdellatif, A. Mohamed, C. F. Chiasserini, M. Tlili, and A. Erbad, "Edge computing for smart health: Context-aware approaches, opportunities, and challenges," *IEEE Network*, vol. 33, no. 3, pp. 196–203, 2019.

[36] A. A. Abdellatif, A. Mohamed, C. F. Chiasserini, A. Erbad, and M. Guizani, "Edge computing for energy-efficient smart health systems: Data and application-specific approaches," in *Energy Efficiency of Medical Devices and Healthcare Applications*, pp. 53–67, Elsevier, 2020.

[37] M. Hartmann, U. S. Hashmi, and A. Imran, "Edge computing in smart health care systems: Review, challenges, and research directions," *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 3, p. e3710, 2022.

[38] M. T. Beck, M. Werner, S. Feld, and T. Schimper, "Mobile edge computing: A taxonomy," in *2014 6th International Conference on Advances in Future Internet*, pp. 48–54, 2014.

[39] R. Roman, J. López, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *CoRR*, vol. abs/1602.00484, 2016.

[40] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the internet of vehicles," *CoRR*, vol. abs/1906.00400, 2019.

[41] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[42] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[43] L. A. Haibeh, M. C. Yagoub, and A. Jarray, "A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches," *IEEE Access*, vol. 10, pp. 27591–27610, 2022.

[44] H. Zhou, G. Yang, H. Dai, and G. Liu, "Pflf: Privacy-preserving federated learning framework for edge computing," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1905–1918, 2022.

[45] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Towards an intelligent edge: Wireless communication meets machine learning," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[46] W. Xu, Z. Yang, D. W. K. Ng, M. Levorato, Y. C. Eldar, and M. Debbah, "Edge learning for b5g networks with distributed signal processing: Semantic communication, edge computing, and wireless sensing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 17, no. 1, pp. 9–39, 2023.

[47] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *2018 IEEE wireless communications and networking conference (WCNC)*, pp. 1–6, IEEE, 2018.

[48] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1529–1541, 2021.

[49] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1517–1530, 2021.

[50] Y. Chen, Z. Liu, Y. Zhang, Y. Wu, X. Chen, and L. Zhao, "Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4925–4934, 2021.

[51] B. Liu, C. Liu, and M. Peng, "Resource allocation for energy-efficient mec in noma-enabled massive iot networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 4, pp. 1015–1027, 2021.

[52] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[53] D. Ren, X. Gui, and K. Zhang, "Adaptive request scheduling and service caching for mec-assisted iot networks: An online learning approach," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17372–17386, 2022.

[54] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[55] T. Hou, G. Feng, S. Qin, and W. Jiang, "Proactive content caching by exploiting transfer learning for mobile edge computing," *International Journal of Communication Systems*, vol. 31, no. 11, p. e3706, 2018.

[56] S. Safavat, N. N. Sapavath, and D. B. Rawat, "Recent advances in mobile edge computing and content caching," *Digital Communications and Networks*, vol. 6, no. 2, pp. 189–194, 2020.

[57] H. Wu, Y. Fan, Y. Wang, H. Ma, and L. Xing, "A comprehensive review on edge caching from the perspective of total process: Placement, policy and delivery," *Sensors*, vol. 21, no. 15, p. 5033, 2021.

[58] R. Zhang, F. R. Yu, J. Liu, T. Huang, and Y. Liu, "Deep reinforcement learning (drl)-based device-to-device (d2d) caching with blockchain and mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6469–6485, 2020.

[59] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[60] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[61] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, "Understanding performance of edge content caching for mobile video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1076–1089, 2017.

[62] Y. Zhang, P. Li, Z. Zhang, B. Bai, G. Zhang, W. Wang, B. Lian, and K. Xu, "Autosight: Distributed edge caching in short video network," *IEEE Network*, vol. 34, no. 3, pp. 194–199, 2020.

[63] C. Sun, X. Li, J. Wen, X. Wang, Z. Han, and V. C. M. Leung, "Federated deep reinforcement learning for recommendation-enabled edge caching in mobile edge-cloud computing networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 3, pp. 690–705, 2023.

[64] E. Ramadan, P. Babaie, and Z.-L. Zhang, "Performance estimation and evaluation framework for caching policies in hierarchical caches," *Computer Communications*, vol. 144, pp. 44–56, 2019.

[65] T. Zhang, X. Fang, Y. Liu, G. Y. Li, and W. Xu, "D2d-enabled mobile user edge caching: A multi-winner auction approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12314–12328, 2019.

[66] S. S. Kafıloğlu, G. Gür, and F. Alagöz, "Cooperative caching and video characteristics in d2d edge networks," *IEEE Communications Letters*, vol. 24, no. 11, pp. 2647–2651, 2020.

[67] R. Karasik, O. Simeone, and S. S. Shitz, "How much can d2d communication reduce content delivery latency in fog networks with edge caching?," *IEEE Transactions on Communications*, vol. 68, no. 4, pp. 2308–2323, 2019.

[68] B. Bai, L. Wang, Z. Han, W. Chen, and T. Svensson, "Caching based socially-aware d2d communications in wireless content delivery networks: A hypergraph framework," *IEEE Wireless Communications*, vol. 23, no. 4, pp. 74–81, 2016.

[69] J. Tang, H. Tang, X. Zhang, K. Cumanan, G. Chen, K.-K. Wong, and J. A. Chambers, "Energy minimization in d2d-assisted cache-enabled internet of things: A deep reinforcement learning approach," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5412–5423, 2020.

[70] R. Buyya, M. Pathan, and A. Vakali, *Content delivery networks*, vol. 9. Springer Science & Business Media, 2008.

[71] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on information theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[72] J. Liu, B. Bai, J. Zhang, and K. B. Letaief, "Cache placement in fog-rans: From centralized to distributed algorithms," *IEEE Transactions on Wireless Communications*, vol. 16, no. 11, pp. 7039–7051, 2017.

[73] J. Kwak, Y. Kim, L. B. Le, and S. Chong, "Hybrid content caching in 5g wireless networks: Cloud versus edge caching," *IEEE Transactions on Wireless Communications*, vol. 17, no. 5, pp. 3030–3045, 2018.

[74] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 1, pp. 126–134, IEEE, 1999.

[75] R. Li, Y. Zhao, C. Wang, X. Wang, V. C. Leung, X. Li, and T. Taleb, "Edge caching replacement optimization for d2d wireless networks via weighted distributed dqn," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, IEEE, 2020.

[76] D. Malak, M. Al-Shalash, and J. G. Andrews, "Optimizing the spatial content caching distribution for device-to-device communications," in *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 280–284, 2016.

[77] J. Song, M. Sheng, T. Q. S. Quek, C. Xu, and X. Wang, "Learning-based content caching and sharing for wireless networks," *IEEE Transactions on Communications*, vol. 65, no. 10, pp. 4309–4324, 2017.

[78] G. K. Zipf, *The psycho-biology of language: An introduction to dynamic philology*. Routledge, 2013.

[79] S. T. Piantadosi, "Zipf's word frequency law in natural language: A critical review and future directions," *Psychonomic bulletin & review*, vol. 21, pp. 1112–1130, 2014.

[80] G. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah, "Wireless caching: Technical misconceptions and business barriers," *IEEE Communications Magazine*, vol. 54, no. 8, pp. 16–22, 2016.

[81] S. Mehrizi, A. Tsakmalis, S. Chatzinotas, and B. Ottersten, "A bayesian poisson–gaussian process model for popularity learning in edge-caching networks," *IEEE Access*, vol. 7, pp. 92341–92354, 2019.

[82] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, "Temporal locality in today's content caching: Why it matters and how to model it," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 5, pp. 5–12, 2013.

[83] K. Psounis, A. Zhu, B. Prabhakar, and R. Motwani, "Modeling correlations in web traces and implications for designing replacement policies," *Computer Networks*, vol. 45, pp. 379–398, 07 2004.

[84] F. Harper and J. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2015.

[85] W. Jiang, G. Feng, S. Qin, and Y. Liu, "Multi-agent reinforcement learning based cooperative content caching for mobile edge networks," *IEEE Access*, vol. 7, pp. 61856–61867, 2019.

[86] Y. Jiang, M. Ma, M. Bennis, F. Zheng, and X. You, "User preference learning-based edge caching for fog radio access network," *IEEE Transactions on Communications*, vol. 67, no. 2, pp. 1268–1283, 2019.

[87] S. S. Tanzil, W. Hoiles, and V. Krishnamurthy, "Adaptive scheme for caching youtube content in a cellular network: Machine learning approach," *Ieee Access*, vol. 5, pp. 5870–5881, 2017.

[88] Y.-T. Lin, C.-C. Yen, and J.-S. Wang, "Video popularity prediction: An autoencoder approach with clustering," *IEEE Access*, vol. 8, pp. 129285–129299, 2020.

[89] S. Li, J. Xu, M. Van Der Schaar, and W. Li, "Popularity-driven content caching," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, 2016.

[90] Q. Cheng, H. Shan, W. Zhuang, L. Yu, Z. Zhang, and T. Q. Quek, "Design and analysis of mec-and proactive caching-based 360° mobile vr video streaming," *IEEE Transactions on Multimedia*, vol. 24, pp. 1529–1544, 2021.

[91] P. Dai, F. Song, K. Liu, Y. Dai, P. Zhou, and S. Guo, "Edge intelligence for adaptive multimedia streaming in heterogeneous internet of vehicles," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1464–1478, 2023.

[92] J. Yao and N. Ansari, "Caching in energy harvesting aided internet of things: A game-theoretic approach," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3194–3201, 2018.

[93] J. Yao and N. Ansari, "Caching in dynamic iot networks by deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3268–3275, 2021.

[94] G. Ruggeri, M. Amadeo, C. Campolo, A. Molinaro, and A. Iera, "Caching popular transient iot contents in an sdn-based edge infrastructure," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3432–3447, 2021.

[95] J. Wang, "A survey of web caching schemes for the internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, pp. 36–46, 1999.

[96] M. Bilal and S.-G. Kang, "A cache management scheme for efficient content eviction and replication in cache networks," *IEEE Access*, vol. 5, pp. 1692–1701, 2017.

[97] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 94–107, 1999.

[98] X. Zhang, G. Zheng, S. Lambotharan, M. R. Nakhai, and K. Wong, "A reinforcement learning-based user-assisted caching strategy for dynamic content library in small cell networks," *IEEE Transactions on Communications*, vol. 68, no. 6, pp. 3627–3639, 2020.

[99] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, "Reinforcement learning-based optimal computing and caching in mobile edge network," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2343–2355, 2020.

[100] Z. Yu, J. Hu, G. Min, Z. Zhao, W. Miao, and M. S. Hossain, "Mobility-aware proactive edge caching for connected vehicles using federated learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5341–5351, 2021.

[101] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas, "Federated learning based proactive content caching in edge computing," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2018.

[102] M. Deressa, M. Sheng, M. Wimmers, J. Liu, and M. Mekonnen, "Maximizing quality of experience in device-to-device communication using an evolutionary algorithm based on users' behavior," *IEEE Access*, vol. 5, pp. 3878–3888, 2017.

[103] L. Tang, B. Tang, L. Kang, and L. Zhang, "A novel task caching and migration strategy in multi-access edge computing based on the genetic algorithm," *Future Internet*, vol. 11, no. 8, p. 181, 2019.

[104] J. Gu, W. Wang, A. Huang, H. Shan, and Z. Zhang, "Distributed cache replacement for caching-enable base stations in cellular networks," in *2014 IEEE International Conference on Communications (ICC)*, pp. 2648–2653, IEEE, 2014.

[105] J. Li, Y. Chen, Z. Lin, W. Chen, B. Vucetic, and L. Hanzo, "Distributed caching for data dissemination in the downlink of heterogeneous networks," *IEEE Transactions on communications*, vol. 63, no. 10, pp. 3553–3568, 2015.

[106] X. Xu, M. Tao, and C. Shen, "Collaborative multi-agent multi-armed bandit learning for small-cell caching," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2570–2585, 2020.

[107] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep multi-agent reinforcement learning based cooperative edge caching in wireless networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2019.

[108] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning-based edge caching in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 48–61, 2020.

[109] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 247–257, 2020.

[110] D. Li, Y. Han, C. Wang, G. Shi, X. Wang, X. Li, and V. C. M. Leung, "Deep reinforcement learning for cooperative edge caching in future mobile networks," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2019.

[111] A. Sadeghi, G. Wang, and G. B. Giannakis, "Deep reinforcement learning for adaptive caching in hierarchical content delivery networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1024–1033, 2019.

[112] X. Wu, X. Li, J. Li, P. C. Ching, V. C. M. Leung, and H. V. Poor, "Caching transient content for iot sensing: Multi-agent soft

actor-critic," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 5886–5901, 2021.

[113] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[114] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[115] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[116] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[117] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[118] A. Gupta, Y. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2016.

[119] B. Da, Y.-S. Ong, L. Feng, A. K. Qin, A. Gupta, Z. Zhu, C.-K. Ting, K. Tang, and X. Yao, "Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results," 2017.

[120] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[121] A. D. Martinez, E. Osaba, J. D. Ser, and F. Herrera, "Simultaneously evolving deep reinforcement learning models using multifactorial optimization," 2020.

[122] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen, *et al.*, "A tutorial on thompson sampling," *Foundations and Trends® in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.

[123] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[124] Y. Sun, Y. Cui, and H. Liu, "Joint pushing and caching for bandwidth utilization maximization in wireless networks," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 391–404, 2019.

[125] "Amazon S3 pricing." `https://aws.amazon.com/s3/pricing/?nc=sn&loc=4`. Accessed: 2023-11-20.

[126] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[127] G. Shani and A. Gunawardana, "Evaluating recommendation systems," *Recommender systems handbook*, pp. 257–297, 2011.