SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

July 2024, Vol. 67, Iss. 7, 170303:1–170303:26
https://doi.org/10.1007/s11432-023-3987-y

Special Topic: XAI-Enabled Network Automation for 5G and Beyond: Move Closer to Zero-Touch

XRL-SHAP-Cache: an explainable reinforcement learning approach for intelligent edge service caching in content delivery networks

Xiaolong XU[1], Fan WU[1], Muhammad BILAL[2], Xiaoyu XIA[3], Wanchun DOU[4*], Lina YAO[5,6] & Weiyi ZHONG[7]

[1]*School of Software, Nanjing University of Information Science and Technology, Nanjing* 210044*, China;*
[2]*School of Computing and Communications, Lancaster University, Lancaster* LA1 4WA*, UK;*
[3]*School of Computing Technologies, Royal Melbourne Institute of Technology, Melbourne* VIC 3001*, Australia;*
[4]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210023*, China;*
[5]*School of Computer Science and Engineering, University of New South Wales, Sydney* NSW 2052*, Australia;*
[6]*Data 61, Commonwealth Scientific and Industrial Research Organization, Sydney* VIC 3169*, Australia;*
[7]*School of Computer Science, Qufu Normal University, Qufu* 273165*, China*

Received 20 July 2023/Revised 19 January 2024/Accepted 20 March 2024/Published online 27 June 2024

Abstract Content delivery networks (CDNs) play a pivotal role in the modern internet infrastructure by enabling efficient content delivery across diverse geographical regions. As an essential component of CDNs, the edge caching scheme directly influences the user experience by determining the caching and eviction of content on edge servers. With the emergence of 5G technology, traditional caching schemes have faced challenges in adapting to increasingly complex and dynamic network environments. Consequently, deep reinforcement learning (DRL) offers a promising solution for intelligent zero-touch network governance. However, the black-box nature of DRL models poses challenges in understanding and making trusting decisions. In this paper, we propose an explainable reinforcement learning (XRL)-based intelligent edge service caching approach, namely XRL-SHAP-Cache, which combines DRL with an explainable artificial intelligence (XAI) technique for cache management in CDNs. Instead of focusing solely on achieving performance gains, this study introduces a novel paradigm for providing interpretable caching strategies, thereby establishing a foundation for future transparent and trustworthy edge caching solutions. Specifically, a multi-level cache scheduling framework for CDNs was formulated theoretically, with the D3QN-based caching scheme serving as the targeted interpretable model. Subsequently, by integrating Deep-SHAP into our framework, the contribution of each state input feature to the agent's Q-value output was calculated, thereby providing valuable insights into the decision-making process. The proposed XRL-SHAP-Cache approach was evaluated through extensive experiments to demonstrate the behavior of the scheduling agent in the face of different environmental inputs. The results demonstrate its strong explainability under various real-life scenarios while maintaining superior performance compared to traditional caching schemes in terms of cache hit ratio, quality of service (QoS), and space utilization.

Keywords deep reinforcement learning (DRL), explainable artificial intelligence (XAI), multi-level cache, content delivery network (CDN), D3QN algorithm, Deep-SHAP

1 Introduction

Content delivery networks (CDNs) have emerged as critical infrastructures for the Internet, enabling providers to deliver content efficiently to users by caching frequently accessed content on nearby edge servers (ESs). According to a survey conducted by Akamai Technologies, the global CDN market is projected to reach $252.17 billion by 2029, with a compound annual growth rate of 10% [1]. This growth is driven by the escalating demand for high-quality streaming (e.g., Netflix, Hulu and YouTube), online gaming (e.g., Steam), and e-commerce services (e.g., Amazon and eBay) that require efficient content delivery across diverse geographical regions. In addition, a survey conducted by ABC Consultancy

* Corresponding author (email: douwc@nju.edu.cn)

revealed that 84% of enterprises have incorporated CDNs into their infrastructure, resulting in improved website performance, reduced latency, and enhanced user experience [2]. The proliferation of digital enterprises and growing demand for agility and faster content delivery have underscored the increasing significance of intelligent cache management. Concurrently, the adoption of software-defined networking (SDN) provides a programmable approach to network management, enabling the dynamic control and configuration of network resources [3].

To fully harness the potential of CDNs, the development of optimal cache-management mechanisms for autonomous network operations is imperative. The ultimate objective is to realize a zero-touch system that empowers network operations (NetOps) team to maintain control while minimizing manual interventions. However, in today's digital landscape, with 40% of website visitors abandoning the page, if it takes more than three seconds to load [4], efficient data caching at the edge is highly sophisticated to achieve low latency and reduce network traffic. Traditional caching algorithms rely on simple heuristics or statistical models that may not accurately capture complex patterns in user requests and content popularity. To overcome this limitation, deep reinforcement learning (DRL) has emerged as a powerful technique for network automation. DRL combines the strengths of deep learning (DL) and reinforcement learning (RL), allowing agents to learn optimal actions through environmental interactions. In the context of network caching, DRL agents can dynamically adapt their caching decisions based on real-time feedback such as user behavior and network conditions. DRL-driven network automation offers a new paradigm for intelligent and adaptive decision-making in diverse network resource scheduling scenarios, such as content caching, network slicing, and traffic routing [5]. By leveraging the real-time feedback on user behavior and network states, DRL agents can dynamically make decisions that account for content popularity [6], user access patterns, and network constraints, thereby improving cache hit rates, latency, and quality of service (QoS).

The application of DRL in network automation introduces challenges pertaining to the black-box nature [7] of DL models (i.e., not readily understandable by humans or easily explainable using intuitive reasoning). Traditional DRL agents lack interpretability, making it difficult for network administrators to understand the decision-making process and trust the agents' actions. This limitation hinders effective collaboration between humans and automated systems, potentially impeding the adoption of DRL in critical network operations. To address these challenges, explainable artificial intelligence (XAI) techniques must be incorporated into network automation. XAI creates a suite of AI techniques that enables human users to understand, appropriately trust, and effectively manage the emerging generation of artificial intelligent partners [8]. There is invariably a tradeoff between model explainability and performance. In other words, simple or shallow models (e.g., naive Bayes, logistic regression, decision trees (DTs)) that tend to be more explanatory may not achieve the same level of performance as more complex and deeper models based on neural networks (NNs).

Most current XAI approaches can be categorized as either intrinsic or post-hoc methods. Post-hoc explanations are derived from pre-trained models and can be model-agnostic, whereas intrinsic explanations are inherent to the model and are usually model-specific. Techniques such as visualization methods, knowledge extraction, contribution-based methods, and example-based explanations, fall under the category of post-hoc explanations. Visualization methods provide gradient propagation or activation masks to facilitate understanding of the model output. Knowledge extraction techniques (e.g., knowledge distillation [9]) focus on extracting explainable knowledge from complex models. The use of knowledge graphs [10] as a means of structured representation has also gained attention in recent studies. Contribution-based methods are commonly employed in model-agnostic approaches to estimate the importance or relevance of features. They perturb inputs or internal components and assess the impact on model performance [11]. Example-based explanations [12] clarify based on specific examples or instances. Specifically, the post-hoc algorithms commonly used in XAI include local interpretable model-agnostic explanations (LIME), Shapley additive explanations (SHAP), class activation map (CAM), and layer-wise relevance propagation (LRP).

However, the effective infusion of explainability into DRL-based caching strategies poses significant challenges. DRL agents map the states to actions based on a complex internal logic with opacity surrounding the weighting of the factors. Model-specific explanation techniques [10] necessitate pervasive changes to existing DRL algorithm implementations to inject explainability. This tight integration between target models and explanation methods hampers flexible adoption across various applications. Meanwhile, self-explainable methods that directly encode explainability into the model structure (e.g., policy networks) can potentially disrupt the emergence of optimal decisions.

In this study, the transparency of the autonomous network management system is enhanced by incorporating Deep-SHAP, which is an improvement over traditional SHAP methods. This enables NetOps teams to gain insights into the decision-making process, interpret system behavior, and ensure alignment with organizational goals and policies [13], which ultimately leads to a more reliable and accountable network automation solution.

The main contributions of this study are as follows:

• Based on the modeling of the multi-level CDN network architecture, a multi-objective optimization problem targeting an optimal cache policy is formulated to maximize the overall hit rate, QoS, space utilization, and load balancing of the entire system.

• To enhance the performance of CDNs in dynamic and complex network environments, a multi-level cache scheduling framework with two edge-side deep double dueling Q-network (D3QN) agents is proposed to handle caching and maintenance options. The scheduling agents can adjust their strategies based on real-time environmental feedback, thus making them capable of adapting to rapidly changing access patterns.

• To address the black-box nature of the D3QN decision-making process, explainability is integrated into the framework to propose an explainable reinforcement learning (XRL)-based intelligent edge service caching approach, namely XRL-SHAP-Cache, which combines DRL with the XAI technique for cache management in CDNs.

• The proposed XRL-SHAP-Cache approach was evaluated through extensive experiments in a simulator environment, with the results demonstrating its superior performance compared to baseline methods in terms of cache hit ratio, QoS, and resource utilization. In addition, case studies on real-world scenarios were conducted to showcase its outstanding interpretability for transparent and trustworthy decision-making.

2 Related work

2.1 DRL based service caching

Previous studies have mainly used static rules and heuristic methods, such as first in first out (FIFO), least frequently used (LFU), and CLOCK to address the cache management problem, struggling to adapt to dynamic and complex network environments. To address this issue, Yan et al. [14] proposed a cache space slicing technique and then leveraged the distributed distributional deep deterministic policy gradient (D4PG) to optimize the service caching strategy with the highest service coverage rate and low processing latency. Similarly, Kong et al. [15] designed a joint computing and caching framework by integrating the deep deterministic policy gradient (DDPG) algorithm.

To alleviate the unnecessary overhead on the backbone network caused by the rapidly growing internet video traffic and users' increasing quality of experience (QoE) demands, Wang et al. [16] proposed MacoCache, an intelligent edge-caching framework that is carefully designed to support a massively diversified and distributed caching environment, aiming to minimize both content access latency and traffic costs. Fang et al. [17] proposed a framework to improve content distribution in a layered fog radio access network (FRAN). For cross-layer cooperative caching and routing decisions, they included a new DRL policy design based on historical information and available network resources. Nikbakht et al. [18] further developed an RL-based caching technique that can adapt to time-location-dependent popularity patterns for on-demand video content, and Lim et al. [19] proposed a DRL-based offloading scheduler (DRL-OS) that considers the energy balance when selecting the method for task execution, namely local computing, offloading, or cache dropping.

Unlike most of the aforementioned works, which assume uniform cached content sizes, Zhou et al. [20] presented a novel size-adaptive content caching (SACC) algorithm using an actor-critic architecture. The SACC models the requests with random sizes and updates the cache after a batch of requests to satisfy the real-world requirements.

2.2 Explainability in DRL

DRL agents operate within a Markov decision process (MDP) framework to select actions based on the observed states and optimize the rewards. However, traditional DRL methods are often regarded as black boxes that lack transparency in decision-making processes. To address this limitation, recent

studies have focused on integrating explainability into DRL models, which is also known as XRL. Wells et al. [21] conducted a comprehensive review of the current approaches and trends in XAI and DRL. Their study emphasizes the significance of incorporating explainability into DRL models. Vouros [22] further conducted an extensive analysis of the state-of-the-art methods and challenges in XRL, identifying key areas for future research.

Several researchers have proposed approaches for enhancing the interpretability of their domain-specific DRL models. Zhang et al. [23] proposed a backpropagation deep explainer based on SHAP, as an interpretable approach for DRL-based power system emergency control applications. Dassanayake et al. [24] addressed the interpretability issue of deep neural networks (DNNs), explaining the reactions to predefined constraints and capturing the associated conditions that influence the DNN in a time series. Zhu et al. [25] focused on developing interpretable techniques for DRL in traffic signal control by utilizing modified DTs to extract models with simpler hierarchical structures from DRL policies, thereby enabling human-understandable decision processes. Leveraging knowledge graphs to empower DRL agents, Wu et al. [10] proposed a novel framework that enables transparent and interpretable explainable AI for communication network automation. Through a path selection case study, they demonstrated the feasibility of their proposed architecture by providing human-understandable explanations for network control actions. Several studies have focused on generating fuzzy rule-based systems to provide explainability. To address the stability challenges arising from independently controlled yet interdependent motion commands in autonomous vehicles, Chen et al. [26] presented a conditional deep Q-network (DQN) integrated with an explainable defuzzification scheme based on fuzzy logic control, to guide directional planning and improve predictive stability. To maintain the fidelity and accuracy of the original networks while lowering complexity, Aghaeipoor et al. [27] proposed fuzzy-rule-based explainer systems that learn compact yet accurate fuzzy rule sets based on the importance of features distilled from trained networks. Among practical industrial applications, there have also been attempts to extract IF-THEN rules [28], M-of-N rules [29], and DT rules [30, 31] to explain NNs.

In summary, few studies have focused on the explainability of service caching operations, and there has been little work on adopting DRL for multi-level caching. Therefore, a multi-level cache scheme incorporating XRL is proposed in this study.

3 Model formulation and problem definition

3.1 Network architecture

As illustrated in Figure 1, the CDN architecture discussed in this paper can be divided into three primary layers: cloud, edge, and end layers.

The cloud layer encompasses a centralized data center and K services. The datacenter, denoted by O, serves as the origin server. In this study, data center O is assumed to be always scalable, capable of accommodating all services within the system, and highly reliable, ensuring uninterrupted operation without any downtimes. The services, represented as $S = \{s_1, s_2, \ldots, s_K\}$, refer to the various applications and data hosted in data center O and accessed by end users.

The edge layer consists of N ESs responsible for caching and delivering content to end users, denoted as $E = \{e_1, e_2, \ldots, e_N\}$. ESs are strategically placed in optimal locations to ensure that they are geographically closer to as many end users as possible within their maximum coverage range r_i. To optimize energy consumption and boost performance, all ESs are equipped with a multi-layer cache architecture. Individual cache layers vary in capacity, I/O speed, and power efficiency, allowing intelligent data placement to maximize QoS. The three cache levels, denoted as L1, L2, and L3, are strategically tiered to align with distinct purposes within the system. Typically, L1 cache, which is implemented using random access memory (RAM), is the fastest and has the lowest capacity. In contrast, the L3 cache, which is often implemented using hard disk drives (HDDs) or tape drives, is the largest although slower in terms of I/O and consumes significantly more energy.

The end layer comprises M end users denoted by $U = \{u_1, u_2, \ldots, u_M\}$. These end users can be located anywhere around ESs and can access services using a wide range of devices, including desktop personal computers (PCs), laptops, smartphones, personal digital assistants (PDAs), and tablets. The connection method of users to ESs can be wired Ethernet, Wi-Fi, or cellular data, that is, global system for mobile (GSM) communication, long-term evolution (LTE), 4G, and 5G. which exhibit variations in bandwidth,

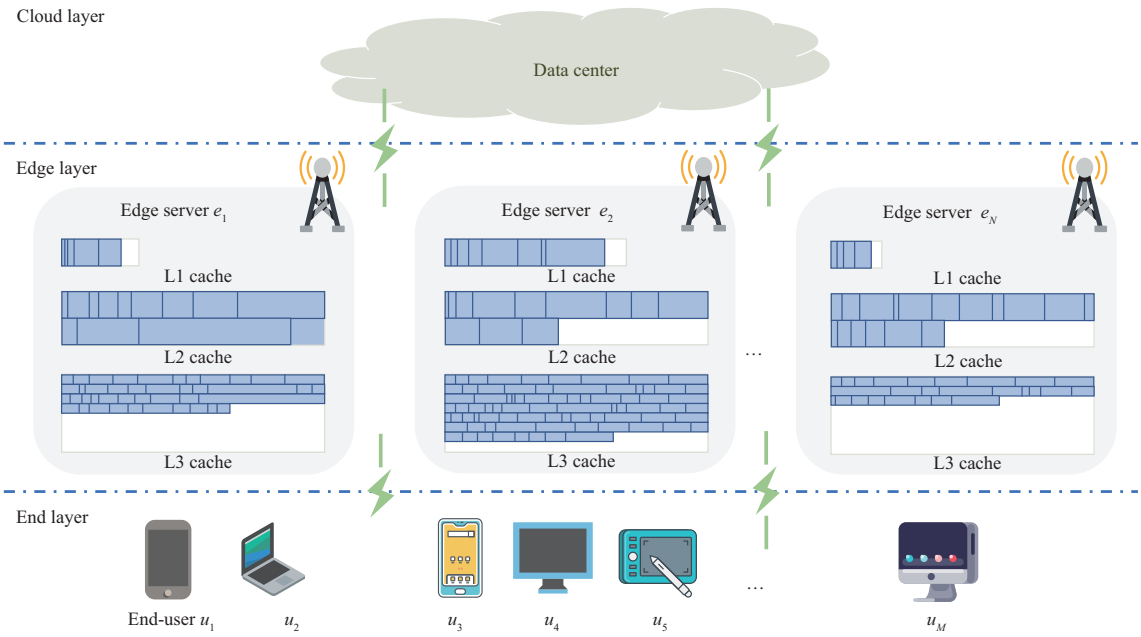

Figure 1 (Color online) Architecture of a multi-level cache CDN empowered by edge caching.

latency, and jitter.

End users iterate through all available ESs within the designated range, denoted by $\{e_j|\text{dist}(e_j, u_i) < r_i\}$, following the order of geographical proximity from closest to farthest. Upon encountering an ES with a cached service, the user halts further inspections and retrieves the data from that particular ES. In a scenario in which cache misses occur across all ESs within range, the user automatically resorts to fetching the required service from the nearest ES. Upon a cache miss, the selected ES retrieves the requested content directly from datacenter O and then uses its caching policy to determine whether to store a copy of the content in the cache for future requests.

3.2 Transmission rate model

Communication between users and ESs, as well as between ESs and data center O, can be regarded as individual channels. According to Shannon's theorem, the maximum data transmission rate $R_{\max}$ of a channel with bandwidth W is determined by the signal-to-noise ratio (SNR), which can be described as

$$R_{\max} = W \times \log_2(1 + \text{SNR}). \tag{1}$$

The SNR is a crucial metric used to measure signal quality. In particular, variations in the transmission distance can significantly affect the SNR. As distance increases, signal attenuation increases, which is referred to as propagation loss. Propagation loss l is commonly modeled as

$$l = (k \times d)^n, \tag{2}$$

where k is the coefficient, n is the path-loss exponent, and d is the transmission distance. The specific value of n is environment-dependent and can be obtained from experiments or theoretical calculations [32]. For instance, n is slightly greater than 1 for an ideal open environment, whereas in urban streets and indoors, n may range from 2 to 6.

In addition to user mobility, environmental noise is another factor that can dynamically affect SNR. Noise can arise from natural phenomena (such as thunder or rain), human activities (including traffic and machinery), and electronic devices. Its waveform and amplitude exhibit random variations in time and space, making it challenging to predict its impact on the signal. To incorporate this randomness, we introduce a random error term e, which follows a standard Gaussian distribution $e \sim N(0, 1)$ to simulate these fluctuations.

The total undesired noise is the sum of propagation loss l and environmental noise error term e. Accordingly, the SNR can be expressed as

$$\mathrm{SNR} = \frac{\rho}{l + |e|}, \tag{3}$$

where the fixed propagation loss l represents a conservative estimate of the signal attenuation, ensuring that the transmission rate does not overestimate the actual achievable rate even under ideal conditions. Simultaneously, the environmental noise term e introduces a level of uncertainty into the output. With the combination of fixed l and variable e, the proposed model can provide robust transmission rate estimates with conservativeness taken into account.

In this study, we adopt $k = 0.1$ and $n = 3$ in the propagation loss formula. The signal power ρ is determined based on the transmission power and gain at the sender, which is simplified as a constant $C = 10$. The channel bandwidth is determined considering the bottleneck with the minimum throughput in the link. Based on these assumptions, the transmission rate between nodes a and b can be modeled as

$$\mathrm{tr}(a,b) = \min\{bw(a), bw(b)\} \times \log_2 \left\{ 1 \ + \ \frac{C}{\left[0.1 \times \mathrm{dist}(a,b)\right]^3 + |e|} \right\}. \tag{4}$$

3.3 Response time model

We denote the request from user u_i for service s_k as $\mathrm{req}^n_{u_i,s_k}$, where n is the cumulative index of the requests. The lifecycle of $\mathrm{req}^n_{u_i,s_k}$ in a CDN system starts from when u_i sends a request to its target ES e_{target} and continues until the user receives the requested service s_k, which can be divided into the following three phases.

(1) User-to-edge delay. This refers to the time taken for user u_i's request to reach e_{target}. Given the small size of most request data packages, the transmission delay $T^{\mathrm{tran}}_{\mathrm{C2E}}$ on the user side is usually negligible. In contrast, the propagation delay $T^{\mathrm{prop}}_{\mathrm{C2E}}$ dominates this phase, which can be calculated as

$$T_{\mathrm{C2E}} \approx T^{\mathrm{prop}}_{\mathrm{C2E}} = \frac{\mathrm{dist}(u_i, e_{\mathrm{target}})}{v_{\mathrm{tran}} \times \eta}, \tag{5}$$

where v_{tran} is the speed of electromagnetic signal propagation, which equals the speed of light in a vacuum (approximately 299792 km/s), and η represents the reduction in signal propagation speed due to signal attenuation in the transmission medium, with a typical value between 0.7 to 0.8.

(2) Datacenter-to-edge delay (optional). For non-cached services, ESs are required to retrieve the original content from the remote. This involves transferring s_k from data center O to e_{target} via a backhaul link. Given the high quality of the dedicated backhaul link, the propagation delay is minimal during data transmission. Therefore, the primary source of time consumption is the transmission delay $T^{\mathrm{tran}}_{\mathrm{D2E}}$ between O and e_{target}, which can be modeled as

$$T^{\mathrm{tran}}_{\mathrm{D2E}} = \frac{s_k.\mathrm{size}}{\mathrm{tr}(O, e_{\mathrm{target}})}. \tag{6}$$

Subsequently, s_k is written to C^j_{target} utilizing different storage media that have varying write speeds. The time it takes to complete the write operation, denoted as $T^{\mathrm{write}}_{\mathrm{D2E}}$, can have a substantial impact on the overall duration of this phase and can be calculated using

$$T^{\mathrm{write}}_{\mathrm{D2E}} = \frac{s_k.\mathrm{size}}{\mathrm{io\_write}(C^j_{\mathrm{target}})}, \tag{7}$$

where $\mathrm{io\_write}(\cdot)$ is the write speed of the corresponding cache medium.

Thus, the total edge-to-datacenter time consumption T_{D2E} can be described as

$$T_{\mathrm{D2E}} = T^{\mathrm{tran}}_{\mathrm{D2E}} + T^{\mathrm{write}}_{\mathrm{D2E}}. \tag{8}$$

(3) Edge-to-user delay. This is identified as the most significant time expense in the entire request lifecycle and comprises three distinct components: queuing time $T^{\mathrm{queue}}_{\mathrm{E2U}}$, I/O retrieval time $T^{\mathrm{retrieval}}_{\mathrm{E2U}}$, and user downloading time $T^{\mathrm{download}}_{\mathrm{E2U}}$.

Due to I/O limitations, each level of cache C^j_{target} can only handle a finite number ω^j of concurrent reads. To prevent the system from becoming overwhelmed, any excessive requests are added to queue $Q = \{\text{task}_1, \text{task}_2, \ldots\}$. The queueing time $T^{\text{queue}}_{\text{E2U}}$ can be expressed as

$$T^{\text{queue}}_{\text{E2U}} = \begin{cases} 0, & \text{curr\_reads} < \omega^j, \\ \sum_{n=1}^{\text{len}(Q)} \text{duration}(\text{task}_n), & \text{curr\_reads} \geqslant \omega^j, \end{cases} \tag{9}$$

where curr_reads denotes the current number of concurrent reads, and $\text{len}(Q)$ represents the length of the queue Q containing excessive requests waiting to be processed.

The read speeds of different cache levels can have a significant impact on the retrieval time of services. For example, services stored in RAM tend to have shorter retrieval durations than services stored in HDD. The I/O retrieval time $T^{\text{retrieval}}_{\text{E2U}}$ can be calculated using

$$T^{\text{retrieval}}_{\text{D2E}} = \frac{s_k.\text{size}}{\text{io\_read}(C^j_{\text{target}})}. \tag{10}$$

Finally, the requested s_i is transmitted back to u_i. The time taken for u_i to download s_k can be calculated using

$$T^{\text{download}}_{\text{E2U}} = \frac{s_k.\text{size}}{\text{tr}(e_{\text{target}}, u_i)}. \tag{11}$$

Thus, the total edge-to-user time consumption T_{E2U} can be expressed as

$$T_{\text{E2U}} = T^{\text{queue}}_{\text{E2U}} + T^{\text{retrieval}}_{\text{E2U}} + T^{\text{download}}_{\text{E2U}}. \tag{12}$$

It is worth noting that our model focuses on the I/O-intensive nature of CDN and does not consider other time costs, such as processing time and system call time. These computation-related factors may also contribute to the overall response time and cannot be ignored in certain applications [14, 33, 34]. However, I/O operations are typically the primary bottleneck in CDNs, necessitating detailed modeling to alleviate their impact on system performance. Based on the factors mentioned above, the total response time T_{total} can be modeled as

$$T_{\text{total}}(\text{req}) = T_{\text{C2E}} + (1 - \mathbb{I}_{e_{\text{target}}.\text{has\_cache}(s_i)}) \times T_{\text{D2E}} + T_{\text{E2U}}, \tag{13}$$

where $\mathbb{I}_{(\cdot)}$ is the conditional function, which equals 1 when the condition $(\cdot)$ is true, and 0 otherwise.

3.4 Energy consumption model

According to [35], the energy consumption of ESs primarily originates from cache media and network interface controllers (NICs). This study focuses on service caching rather than offloading, and therefore computation-related factors are beyond the scope of our discussion.

The multi-level cache architecture exhibits inconsistent power consumption across cache levels due to differences in underlying storage technologies. The L1 cache typically adopts dynamic random-access memory (DRAM). Despite requiring occasional refreshes to avoid data loss, DRAM can provide rapid read/write speeds with reasonably low energy consumption. Besides, the small capacity of L1 cache keeps this power overhead manageable. The L2 cache, which typically employs flash storage, has moderate energy consumption with the number of charged storage units adjusted according to the represented data. The energy consumption of the L3 cache is relatively high due to the constant rotation of disks and the movement of heads for data retrieval. In addition to the variability in energy efficiency across the underlying cache media, the energy consumption also correlates directly with the amount of I/O operations. Let the energy coefficient τ_j denote the expenditure per unit of I/O throughput at the j-th level of cache. The energy consumption of cache C^j_i can then be calculated as

$$P_{\text{cache}}(C^j_i) = \tau_j \times \left(\sum \text{I/O history} \right).\text{size}. \tag{14}$$

In terms of NICs, the volume of data transmitted can significantly influence the energy consumption. Transmitting larger amounts of data requires sustained electrical power for signal propagation, amplification, and other overhead involved during the transmission process. Moreover, extended durations

of active requests can lead to prolonged operational states of the NIC, resulting in continuous energy consumption for maintaining connectivity, data processing, and communication facilitation. Thus, the energy consumption of NIC is intricately linked to both the volume of data transmitted and the duration of the requests, which can be expressed as

$$P_{\mathrm{NIC}}(e_i) = \sum_{\mathrm{req}\in\{\mathrm{reqs}\to e_i\}} \left[p_{\mathrm{keep\_alive}} \times T_{\mathrm{total}}(\mathrm{req}) + p_{\mathrm{trans}} \times \mathrm{req.size}\right], \tag{15}$$

where $p_{\mathrm{keep\_alive}}$ and p_{trans} are NIC-related energy consumption factors. Specifically, $p_{\mathrm{keep\_alive}}$ is the power required to maintain the connections, measured in watts per second, and p_{trans} is the power associated with data transmission, measured in watts per gigabyte.

Consequently, the total energy consumption of the ES e_i can be modeled as

$$P_{\mathrm{total}}(e_i) = \sum_{j\in\{L1,L2,L3\}} P_{\mathrm{cache}}(C_i^j) + P_{\mathrm{NIC}}(e_i). \tag{16}$$

3.5 Statistical quantities

Based on previous modeling, the following metrics can be obtained through statistical analysis and calculations in measuring the overall network performance.

(1) Cache hit rate. This metric indicates the proportion of requests that can be served directly from the cache. The cache hit rate for each e_i is defined by the ratio of cache hits to the total number of requests sent to e_i, which can be expressed as

$$\mathrm{hit\_rate}(e_i) = \frac{\sum_{\mathrm{req}^n_{u_i,s_k}\in\{\mathrm{reqs}\to e_i\}} \mathbb{1}_{e_i.\mathrm{has\_cache}(s_i)}}{\mathrm{count}\left(\mathrm{req}^n_{u_i,s_k} \in \{\mathrm{reqs}\to e_i\}\right)}. \tag{17}$$

(2) QoS. This metric quantifies the overall performance of the caching scheme in terms of user experience and energy consumption. $\mathrm{QoS}(e_i)$ was calculated using the following formula:

$$\mathrm{QoS}(e_i) = \frac{\ln\left(1+\mathrm{tr}_{\mathrm{effective}}(e_i)\right)}{P_{\mathrm{total}}(e_i)^2} = \frac{1}{P_{\mathrm{total}}(e_i)^2}\ln\left(1+\frac{\sum_{\mathrm{req}\in\{\mathrm{reqs}\to e_i\}}\mathrm{req.size}}{\sum_{\mathrm{req}\in\{\mathrm{reqs}\to e_i\}}T_{\mathrm{total}}(\mathrm{req})}\right). \tag{18}$$

where $\mathrm{QoS}(e_i)$ is proportional to effective transmission rate $\mathrm{tr}_{\mathrm{effective}}(e_i)$ and inversely proportional to the square of $P_{\mathrm{total}}(e_i)$, necessitating an optimal cache policy that can achieve lower energy consumption and higher $\mathrm{tr}_{\mathrm{effective}}(e_i)$.

(3) Space utilization. This metric quantifies the effectiveness of utilizing the cache space. A higher space utilization ratio indicates that more data are stored in the cache, which in turn can potentially lead to higher hit rates and faster response times. The space utilization ratio of e_i is given by

$$\mathrm{space\_utilize}(e_i) = \frac{\sum_{j\in\{L1,L2,L3\}} C_i^j.\mathrm{used}}{\sum_{j\in\{L1,L2,L3\}} C_i^j.\mathrm{size}}. \tag{19}$$

(4) Load balancing. This metric measures the ability of the policy to evenly distribute the workload among multiple ESs in the system. It is the reciprocal of the standard deviation of each ES's workload, calculated as

$$\mathcal{L} = \frac{1}{\sqrt{\frac{\sum_{i=1}^{N}\left[w(e_i)-\overline{w(e)}\right]^2}{N}}}, \tag{20}$$

where $w(e_i)$ indicates the workload of ES e_i, and $\overline{w(e)}$ denotes the mean workload of all ESs, calculated as $\frac{\sum_{i=1}^{N} w(e_i)}{N}$.

3.6 Problem definition

The objective is to obtain an optimal cache policy that maximizes the overall hit rate, QoS, space utilization, and load balancing of the entire CDN system. However, achieving high performance in all these metrics simultaneously involves tradeoffs between different objectives. To address this issue, let α_n

be the weighted indicators, with $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$ and $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in [0, 1]$. The multi-objective optimization problem can be formulated as

$$\begin{aligned}
&\text{maximize Obj} = \sum_{i=1}^{N} [\alpha_1 \times \text{hit_rate}(e_i) + \alpha_2 \times \text{QoS}(e_i) + \alpha_3 \times \text{space_utilize}(e_i)] + \alpha_4 \times \mathcal{L} \\
&\text{Subject to} \\
&C_1: \ 0 \leqslant s_k.\text{size} \leqslant C_i^j.\text{free_space}, \\
&C_2: \ \text{dist}(u_i, e_j) \leqslant r_i, \\
&C_3: \ 0 \leqslant w(e_i) \leqslant 1, \\
&C_4: \ T_{\text{total}}(\text{req}) \leqslant \text{threshold}_{\text{TIMEOUT}},
\end{aligned} \tag{21}$$

where the capacity constraint C_1 ensures that the size of content s_k about to be cached in C_i^j does not exceed the available free space. The distance constraint C_2 guarantees that user u_i is within the coverage range of ES e_j. Constraint C_3 limits the maximum workload for each ES to between 0 and 1. Constraint C_4 specifies the timeout for each request.

The set of constraints defined by C_1 to C_4 are assumed to be feasible and non-conflicting. In addition, we assume the implicit convexity of multi-objective space, which facilitates the identification of Pareto-optimal solutions. Intuitively, the formulated problem is a mixed-integer nonlinear programming (MINLP) problem, which is generally NP-hard and requires exponential computational time. Therefore, we will propose a DRL approach in the following section to address this problem in a more efficient manner.

4 D3QN-based multi-level cache scheme for CDNs

In real-world scenarios, caching resources are typically extensive in scale and exhibit dynamically changing access patterns. User requests also differ in temporal and spatial locality, resulting in a high degree of uncertainty and variability. Traditional cache algorithms [36, 37] that rely on manually designed rules may struggle to perform well in complex environments. In addition, other NN models [38, 39] that learn policies offline may become outdated due to dynamic changes in the environment and may no longer be suitable for real-time scenarios. Therefore, DRL provides a new paradigm for addressing the dynamic network environment by learning and updating cache policies in real time.

The proposed ES cache policy framework is presented in Figure 2, which is designed to be deployed on all ESs within the network. The agents operating on each ES continuously observe and interact with the dynamic environment, which enables them to make intelligent decisions regarding the cache content. Specifically, the framework consists of two distinct agents, each serving a specific role.

- Cache agent $\mathcal{C}$. This agent determines whether to cache the newly requested service s_new and the level of placement C_i^j after retrieving it from the origin O.
- Maintenance agent $\mathcal{M}$. This agent is responsible for traversing the cache and deciding whether to remove a particular service, which occurs during regular maintenance cycles or when the cache is about to reach its maximum capacity.

In this section, we initiate our discussion with an overview of the proposed decision-making process, which leverages D3QN to learn an optimal online caching policy. Subsequently, the feature and action space are defined to represent the environmental observations and feasible agent actions, respectively. Finally, reward functions are designed to evaluate the performance of both agents.

4.1 Decision-making process on the edge side

In RL, the Q-value function (i.e., action-value function) is primarily used to evaluate the expected reward or value of performing a specific action in a given state. The Q-value reflects the expected reward obtained by performing a certain action in the current state. Specifically, upon performing action a in state t, the agent transits to a new state t' and receives reward r. The Q-value function then estimates the expected cumulative reward that can be obtained by following the optimal policy, denoted as

$$Q(t, a) = E\left[r + \gamma \times \text{argmax}_{a'} Q(t', a') | t, \ a\right]. \tag{22}$$

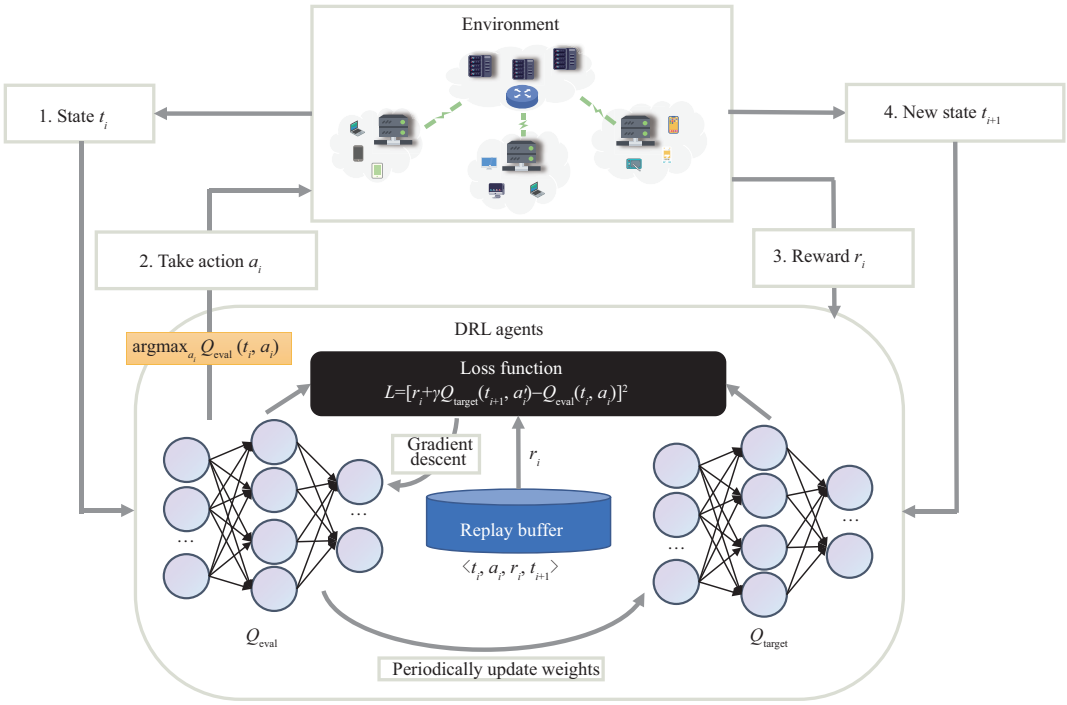

Figure 2 (Color online) D3QN-based cache policy framework.

In the early days, Q-learning was a classic algorithm that utilized the Q-value function by maintaining a Q-table that recorded all feasible $Q(t, a)$ for each state-action pair. However, storing and updating the Q-table is challenging when dealing with high-dimensional and continuous state spaces, resulting in low efficiency and convergence issues. DQN was then proposed to overcome this problem. DQN employs a DNN, also known as the Q-network, to approximate the Q-value function. Besides, the experience replay mechanism enables the DQN to store and learn from previous experiences, thereby increasing the diversity of samples and enhancing the generalization ability and stability of the network. Our agents are built based upon the D3QN algorithm, which is a notable upgrade to the DQN algorithm in terms of its double network mechanism [40]. This mechanism involves two NNs with the same structure, namely the eval network Q_{eval} and the target network Q_{target}. The eval network is used to estimate $Q(t, a)$ and select the action with the highest Q-value output, whereas the target network is used during training to calculate the target Q-values. This approach effectively alleviates the overestimation problem in the traditional DQN and requires fewer parameters.

Algorithm 1 outlines the decision process of a D3QN-based agent, which consists of the following five stages: (1) observing the current state; (2) selecting an action based on the current policy π; (3) executing the selected action in the environment; (4) observing the new state and receiving a reward; and (5) updating the weights of Q_{eval} and Q_{target}. By iterating through these steps, the agent learns the optimal policy π_{opt} through trial-and-error interactions within the environment.

4.2 Environment analysis and observation set

The two agents responsible for caching and maintaining service s_i in ES e_{target} rely on different sets of observations in making decisions.

In terms of the cache agent $\mathcal{C}$, its input features include the workload of e_{target}, available storage at each cache level (L1, L2, and L3), and the Boolean value indicating whether s_i can fit in. Other observations include the service size, charm, popularity, and request frequency of s_i, as well as the number of nearby cached servers and the request frequency for e_{target}. By accepting these input features, $\mathcal{C}$ can determine which services to cache and the optimal layer for caching, thereby maximizing QoS.

The maintenance agent $\mathcal{M}$'s observation set consists of free space and service size ratios, service charm and popularity, request frequency of s_i and e_{target}, cache miss rate, and the LFU index. In addition, the agent can identify whether s_i is marked as urgent. By assessing these features, $\mathcal{M}$ learns which

Algorithm 1 D3QN-based agent decision process

Input: Eval network Q_{eval}, target network Q_{target}, replay buffer buff, exploration fraction ε, action set A, learning rate α;
Output: Optimal actions A_{opt};

Initialize Q_{eval} and Q_{target} weights with random values;
Initialize replay buffer buff with the maximum capacity;
episode $\Leftarrow 0$;
while not converged **do**
  Observe the environment to get the current state t_i;
  for each a_i **in** A **do**
    Calculate the expected $Q(t_i, a_i)$ of action a_i given the current state t_i;
  end for
  Use the epsilon-greedy policy to select a random action with probability ε or action $\text{argmax}_{a_i} Q_{\text{eval}}(t_i, a_i)$ with probability $1-\varepsilon$;
  Agent takes action $a_i(a_{\text{opt}})$, obtains the next state t_{i+1} and reward r_i;
  Store the new experience $\langle t_i, a_i, r_i, t_{i+1}\rangle$ in replay buffer buff;
  if enough experiences in buff **then**
    Sample a random minibatch of transitions from buff;
    Calculate the target Q-values using $y_i = r_i + \gamma Q_{\text{target}}\left(t_{i+1}, a_i'\right)$;
    Calculate loss value using $L = [y_i - Q(t_i, a_i)]^2$;
    Update Q_{eval} by gradient descent $Q_{\text{eval}} \Leftarrow Q_{\text{eval}} - \alpha \times \frac{\partial L}{Q_{\text{eval}}}$;
    Update Q_{target} by copying the weights of Q_{eval}, expressed as $Q_{\text{target}} \Leftarrow Q_{\text{eval}}$;
  end if
  episode $\Leftarrow$ episode $+ 1$;
end while

services can be safely evicted from the cache, thereby liberating space for more valuable content while simultaneously maintaining satisfactory performance.

4.3 Design of the reward functions

The primary objective of D3QN is to maximize the accumulated reward through iterative parameter updates. The design of the reward function directly affects the quality of model convergence. Therefore, it is crucial to formulate a scientific and reasonable reward function for both agents $\mathcal{C}$ and $\mathcal{M}$ operating on e_i.

Inspired by [41, 42], as both agents operate on the same ES and share common goals, their reward functions contain a common part. To simplify the discussion, we first present the common part and then adapt it to the specific characteristics of each agent. The common part incentivizes both agents to collaborate and maximize their joint performance in terms of hit rate and space utilization, denoted as

$$R_{e_i}^{\text{base}} = \text{hit\_rate}(e_i) \times \text{space\_utilize}(e_i). \tag{23}$$

In addition to $R_{e_i}^{\text{base}}$, cache agent $\mathcal{C}$'s reward function incorporates factors that encourage a higher operation success rate σ_{success} and system's load-balancing factor $\mathcal{L}$, expressed as

$$R_{e_i}^{\mathcal{C}} = R_{e_i}^{\text{base}} \times \text{QoS}(e_i) \times \sigma_{\text{success}} + \mathcal{L}, \tag{24}$$

where σ_{success} represents the proportion of successful operations during a given period, which encourages $\mathcal{C}$ to take actions that lead to successful cache operations, thereby increasing the efficiency of the system. The load-balancing factor $\mathcal{L}$ motivates $\mathcal{C}$ to distribute the workload more evenly across servers, which can help improve the overall performance of the system.

For maintenance agent $\mathcal{M}$, the reward function $R_{e_i}^{M}$ incorporates a mechanism that adjusts to the available storage space and can be defined as

$$R_{e_i}^{\mathcal{M}} = R_{e_i}^{\text{base}} * \left[\text{QoS}(e_i)^2 \times \frac{C_i^j.\text{free}}{C_i^j.\text{size}} + \left(1 - \frac{\sigma_{\text{cache\_full}}}{\sigma_{\text{failed}}}\right) \times \frac{C_i^j.\text{used}}{C_i^j.\text{size}}\right]. \tag{25}$$

Specifically, $\mathcal{M}$ is more concerned with QoS at the start, as it is not yet affected by the limited available space. However, as more space is occupied, $\mathcal{M}$ gradually shifts its focus to optimizing the cache policy and preventing overflows.

4.4 Design of the action space

The cache agent $\mathcal{C}$'s action set A_C consists of four possible actions: $\{a_{\text{idle}}, a_{\text{L1}}, a_{\text{L2}}, a_{\text{L3}}\}$; $\mathcal{C}$ determines whether and where to cache service s_i: cache s_i to the L1 cache (a_{L1}) offers the fastest access but limited

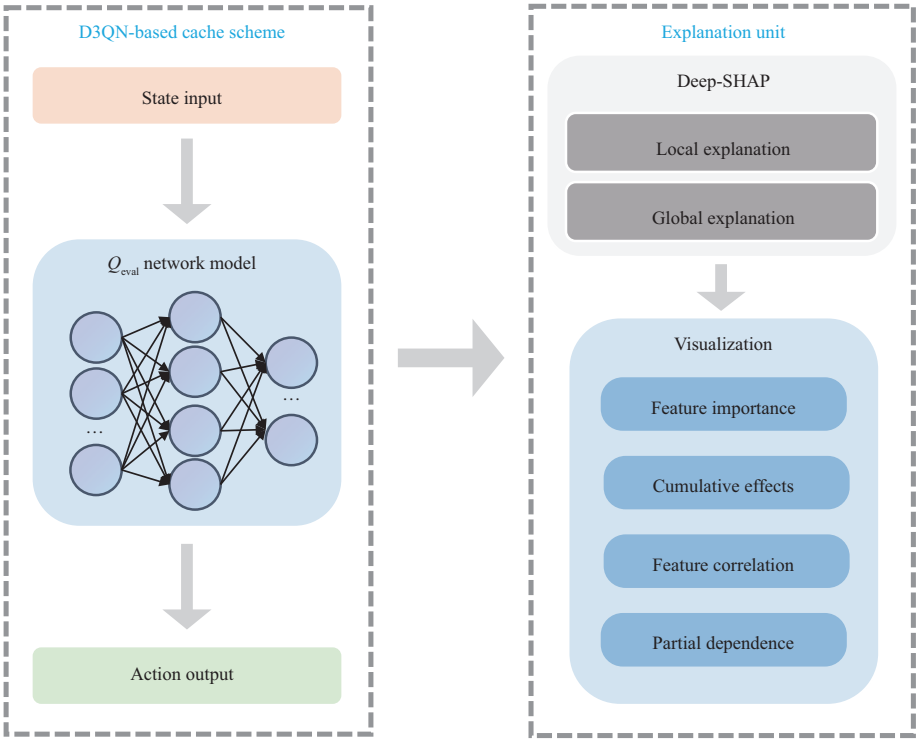

Figure 3 (Color online) D3QN decision-making framework integrated with Deep-SHAP.

capacity, cache s_i to the L2 cache (a_{L2}) has a higher latency but larger capacity, cache s_i to the L3 cache (a_{L3}) has the slowest I/O but the largest capacity, or take no action (a_{idle}). On the other hand, the action space for $\mathcal{M}$ includes two possible actions: $A_M = \{a_{\mathrm{preserve}}, a_{\mathrm{delete}}\}$, where a_{preserve} indicates that s_i should be left untouched, and a_{delete} evicts s_i to free up space.

5 Deep-SHAP integration for decision-making explainability

Modern AI-driven CDN systems in 5G and beyond impose new demands in areas such as regulatory audits, trustworthy decision-making, performance optimization, and human-machine collaboration [43]. However, the use of D3QN models in CDN cache policies often lacks transparency, hindering their explainability in the decision-making process. To address this issue, existing approaches include LIME, deep learning important features (DeepLIFT), LRP, and SHAP.

Deep-SHAP, a modern implementation of the traditional SHAP algorithm, was adopted to enable the interpretability of the D3QN agents. Deep-SHAP offers model-agnostic, global, and local explainability while maintaining robustness to correlated features. Figure 3 illustrates the integration of Deep-SHAP into the D3QN framework, which aligns the agents' decision-making with the needs and goals of network administrators, verifying whether the model robustly solves the problem. In this section, the computation of SHAP value, which serves as the foundation for Deep-SHAP, is demonstrated, followed by a discussion on integrating Deep-SHAP into our framework.

5.1 Computation of SHAP values

The SHAP values, initially proposed in [44], stem from the Shapley value concept of game theory. The Shapley value assigns a contribution value to each player in a cooperative game based on their marginal contributions to the overall payout. Similarly, in the context of SHAP values, this concept is applied to the contribution of individual input features to the output of a model. By viewing the prediction task as a "game" and the input features as the "players", SHAP values can provide explanations for complex and nonlinear models such as DRL models.

Given a dataset $\mathbb{D}$ consisting of k input samples $X = [x_1, x_2, \ldots, x_n]$, where each x_j represents a feature observed by the agent. Through the black-box D3QN eval network $Q_{\mathrm{eval}}(\cdot)$, state inputs are

mapped to Q-value outputs, denoted as $y_j = Q_{\text{eval}}(x_j)$. The SHAP value ϕ_j of a particular feature x_j can be calculated as follows.

First, let $\mathbb{S}$ be a coalition of features excluding x_j and fill the empty positions with $\frac{\sum_{i=1}^{n} x_i}{n}$. The marginal contribution $F_{\text{diff}}(x_j)$ of the model output with and without x_j is then calculated as

$$F_{\text{diff}}\left(x_j\right) = Q_{\text{eval}}\left(\mathbb{S} \cup \{x_j\}\right) - Q_{\text{eval}}\left(\mathbb{S}\right). \tag{26}$$

Next, the weight ω assigned to each coalition $\mathbb{S}$ is calculated as

$$\omega\left(\mathbb{S}\right) = \frac{|\mathbb{S}|!\left(k - |\mathbb{S}| - 1\right)!}{k!}, \tag{27}$$

where $|\mathbb{S}|$ is the size of coalition $|\mathbb{S}|$, and ! is the factorial operator that calculates the product of all positive integers up to a given number.

Finally, the Shapley value formula shap$(\cdot)$ is applied to each coalition:

$$\phi_j = \text{shap}\left(x_j\right) = \sum_{\mathbb{S} \subseteq \mathbb{D} \backslash \{x_j\}} \omega\left(\mathbb{S}\right) * F_{\text{diff}}\left(x_j\right) \ = \sum_{\mathbb{S} \subseteq \mathbb{D} \backslash \{x_j\}} \left[\frac{|\mathbb{S}|!\left(k - |\mathbb{S}| - 1\right)!}{k!} * F_{\text{diff}}\left(x_j\right)\right]. \tag{28}$$

By iterating the aforementioned steps, the SHAP values for each feature are calculated. $\phi_j > 0$ indicates a positive contribution of feature x_j to the model output, suggesting that its presence increases the predicted value, and vice versa.

Although SHAP provides insightful feature attributions for black-box models, its time complexity is $O(2^n)$ which increases exponentially with the number of features. This renders the calculation of SHAP values computationally infeasible for larger datasets or multi-dimensional feature spaces. Thus, approximation techniques and lower-bounding algorithms, such as kernel-SHAP, gradient-SHAP, and Deep-SHAP [45], have been proposed to mitigate the computational burden.

5.2 Local and global explainability with Deep-SHAP

The Deep-SHAP implementation in our framework was inspired by techniques used in DeepLIFT [46], which utilizes an LRP algorithm to backpropagate attribution scores from the model's output through its internal layers. Building upon this concept, Deep-SHAP combines the traditional SHAP values computed for smaller components of the network to obtain values for the entire network through the backpropagation of DeepLIFT's multipliers. Formally, the DeepLIFT multiplier is defined as

$$m_{\Delta x_j \Delta Y} = \frac{C_{\Delta x_j \Delta Y}}{\Delta x_j}, \tag{29}$$

where $\Delta x_j = x_j - r_j, \Delta Y = Q_{\text{eval}}(X) - Q_{\text{eval}}(R)$. r_j is the reference input for x_j and is calculated as $r_j = E(x_j)$. The set of individual reference inputs $\{r_1, r_2, \ldots, r_n\}$ for each x_j forms the total reference set for the state sample denoted by R.

According to Lundberg and Lee [45], DNNs comprise numerous simple components. When analytical solutions are available for these components, DeepLIFT's style of backpropagation enables fast approximations for the full model. Let $\phi_j = C_{\Delta x_j \Delta Y}$. For the network components depicted in Figure 4, the DeepLIFT multipliers can be computed as

$$m_{x_k f_3} = \frac{\phi_j\left(f_3, x\right)}{x_k - \mathbb{E}\left(x_k\right)}, \tag{30}$$

$$\forall_{k \in 1,2}\ m_{y_j f_k} = \frac{\phi_j\left(f_k, y\right)}{y_j - \mathbb{E}\left(x_j\right)}. \tag{31}$$

Using the chain rule,

$$m_{y_j f_3} = \sum_{k \in \{1,2\}} m_{y_j f_k} m_{x_k f_3}. \tag{32}$$

Ultimately, through linear approximations, the equations above can be simplified and expressed as

$$\phi_j\left(f_3, x\right) \approx m_{y_j f_3}\left(y_j - \mathbb{E}\left(x_j\right)\right) = \sum_{k \in \{1,2\}} \frac{\phi_j\left(f_k, y\right)}{y_j - \mathbb{E}\left(x_j\right)} * \frac{\phi_j\left(f_3, x\right)}{x_j - \mathbb{E}\left(x_k\right)}\left(y_j - \mathbb{E}\left(x_j\right)\right). \tag{33}$$

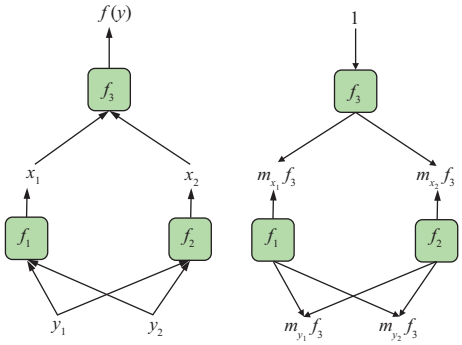

Figure 4 (Color online) Typical composition of a DNN that takes y_1, y_2 as input and outputs $f(y)$. The left side shows the forward-propagation inference process, whereas the right side illustrates the computation of SHAP value using Deep-SHAP.

Algorithm 2 Deep-SHAP integration for D3QN explainability

Input: Observation dataset $\mathbb{D}$, D3QN eval network Q_{eval};
Output: Local explanation ϕ, global explanation $\overline{\overline{\phi}}$;
Initialize empty local SHAP value matrix $\Phi \Leftarrow [\phi_1, \phi_2, \ldots, \phi_k]$;
Initialize empty global SHAP value vector $\overline{\overline{\phi}}$;
for each observation X_i **in** $\mathbb{D}$ **do**
  for each feature x_j **in** X **do**
    Calculate shap$(x_{i,j})$ using (33);
    $\phi \Leftarrow \phi \cup \text{shap}(x_{i,j})$;
  end for
end for
for $j = 1$ **to** n **do**
  Calculate $\overline{\overline{\phi_j}}$ using (34);
  $\overline{\overline{\phi}} \Leftarrow \overline{\overline{\phi}} \cup \overline{\overline{\phi_j}}$;
end for
return ϕ, $\overline{\overline{\phi}}$.

In the context of Deep-SHAP, the scope can be global or local. Local explainability pertains to understanding individual predictions, whereas global explainability aims to provide a more comprehensive understanding on how the model behaves across the entire dataset. Considering a specific scenario represented by dataset $\mathbb{D}$, which is composed of observations X originating from a particular case, the SHAP values ϕ_j obtained can be considered as local explanations that help understand the importance of each input feature for a given prediction. In addition to local explainability through scenario-based reasoning, global explainability can provide a more comprehensive overview on how agents make decisions globally and the overall importance of features. The global SHAP value $\overline{\overline{\phi_j}}$ for each feature is calculated by averaging the SHAP values across all observations, to reflect the overall contribution of each feature to the model prediction. It can be formulated as

$$\overline{\overline{\phi_j}} = \frac{1}{k}\sum_{i=1}^{k} \text{shap}\left(x_{i,j}\right), \tag{34}$$

where $\overline{\overline{\phi_j}}$ denotes the global SHAP value for the j-th feature, k represents the number of samples in the dataset, and $x_{i,j}$ represents the j-th feature of the i-th sample.

Algorithm 2 summarizes the main steps of the Deep-SHAP integration in our framework. Note that the computational complexity of Deep-SHAP is $O(n)$. By employing this approach, the scalability of Deep-SHAP was significantly improved for high-dimensional inputs, thus generating both local and global interpretations without compromising the accuracy or processing time.

Table 1 Parameter configuration of ESs

Index	Bandwidth (mBps)	Cache capacity (GB)			Service range (km)	Num of users within range	Geographic location	
		L1	L2	L3			X	Y
0	331	16	256	1000	22	31	27	33
1	973	64	512	4000	45	87	101	18
2	648	32	512	2000	41	78	67	42
3	70	4	64	320	24	22	12	15
4	255	8	128	500	21	26	103	34
5	670	16	128	1000	48	47	141	14
6	752	8	256	2000	34	66	58	34

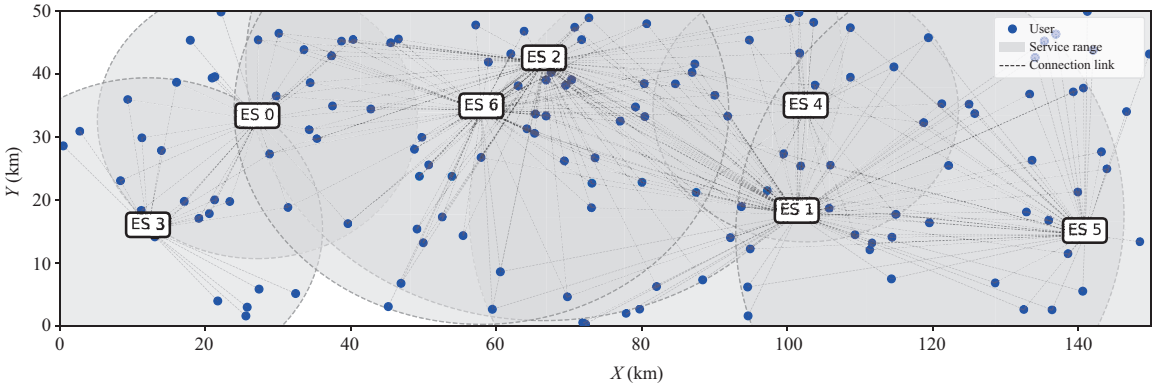

Figure 5 (Color online) Geographic distribution of ESs and users.

Table 2 Running environment details

Component name	Specifications
CPU	Intel Core i7-12700KF
Memory (RAM)	64 GB
GPU	Nvidia GTX 4090
Programing language	Python 3.11.3
IDE	VS Code 1.80.0
DL framework	PyTorch 1.12.1
Operating system	Ubuntu 22.04.2

6 Experiments

6.1 Simulation setup and system environment

Experiments were conducted using our self-built simulator (code: GitHub Link[1)]) to emulate the operation of CDNs with the goal of better visualizing the results and evaluating performance. The experimental environment consisted of 10000 services stored in a centralized data center, 150 users, and 7 ESs (Table 1) distributed over a 150 km × 50 km area (Figure 5). The size of each service followed a Zipf distribution ($\alpha = 2$) between 50 MB to 10 GB, and the charm of each service followed a normal distribution ($\mu = 1$, $\sigma^2 = 5$). Users have limited downstream bandwidth, and all edge nodes have limited maximum concurrent requests, coverage range, bandwidth, and storage capacity. In addition, the energy consumptions of cache levels L1, L2, and L3 were defined as 0.2, 0.5, and 1 watt per gigabyte (W/GB), with the corresponding I/O throughputs set to 20 GB/s, 1 GB/s, and 150 MB/s, respectively.

Furthermore, the simulator integrates features such as user inactivity, new service uploads, changes in service attractiveness values, user preferences over time, man-made trends, and popularity rankings. It can replicate various complex application scenarios such as cold starts, high concurrency, server offline, and stress testing, to simulate realistic network conditions. The experiments were performed using the system configuration outlined in Table 2.

1) https://github.com/zzxjl1/CDNCache_XRL/tree/main/simulator.

Table 3 Hyper parameters during training

Parameter	Value
Learning rate	1e−4
Discounted reward factor γ	0.9
Replay buffer size	1e6
Target network update frequency	5e−3
Exploration fraction ε	Start: 0.95, end: 0.05
Exploration decay factor	1e−4
Batch size	128

6.2 Model convergence analysis

For all agents, the target network Q_{target} and evaluation network Q_{eval} utilized the same four-part structure. The first part served as the input layer and received the state feature vector as input, with the number of neurons equal to the number of input features. The second part consisted of four convolutional layers with 64, 128, 256, and 256 filters, each with a stride of 1. The first three convolutional layers had a kernel size of 3×3, whereas the size of the last layer was 1×1 to merge different channels. The output of each convolutional layer was passed through a rectified linear unit (ReLU) activation function to extract latent features. The third part was a fully connected hidden layer that received the output of the final convolutional layer as a one-dimensional vector. This layer plays a critical role in linking the convolutional layers' output with the final Q-value estimation, thereby enabling the model to capture higher-level abstractions and complex interactions. The last layer provided the Q-value output, and the number of neurons in this layer corresponded to the possible actions of the given agent.

In training the network, the mean squared error (MSE) loss function was used to measure the discrepancy between the predicted and target Q-values obtained from the Bellman equation. The Adam optimizer was used to update the network parameters. The hyperparameters used during the training are listed in Table 3.

The choice of the learning rate is a critical determinant of the training dynamics of the proposed DRL model. An excessively high learning rate can induce instability and hinder convergence, whereas an excessively low learning rate can lead to slow convergence. To strike a balance between these considerations, we set the learning rate to 1e−4 through grid search iterations, to enable stable policy updates without the risk of divergence. The discounted reward factor γ was set to 0.9, emphasizing the importance of immediate rewards while acknowledging the potential influence of future rewards. This balanced weighting ensures that the model effectively captures both short-term gains and long-term considerations. The replay buffer, with its size set to 1e6, served as a reservoir for past experiences, allowing the agent to learn from a diverse range of situations. Although a larger buffer can accommodate more varied experiences for robust learning, it requires more memory resources. The chosen size accommodates this tradeoff, fostering effective learning without imposing excessive memory demands. The exploration fraction ε represents the proportion of exploratory actions performed by the agent. As the system starts without any knowledge (or bias) of the task at hand, using the best-known action at each step, the system may get trapped in a local optimum. To overcome this, ε was initially set to 0.95, implying that the system explores 95% of the time at the beginning of the training phase. Then, ε was linearly reduced to 0.05 over the first 9000 steps. This ensures that the agent explores the environment extensively at the beginning but gradually relies more on learned knowledge as training progresses. After 9000 steps, ε remained constant at 0.05, allowing continued exploration to improve performance. Finally, a batch size of 128 was chosen to balance the computational efficiency and effective learning from experience.

The learning process for both agents can be divided into three stages, as shown in Figure 6. In the starting stage (first 22000 episodes), both agents gradually learn to manipulate cache based on environmental observations. Initially, the average reward is low as the agents begin to explore and learn. However, the average reward increases over time as the agents improve their decision-making abilities. Subsequently, moving on to the transition stage (episodes 22000 to 38000), the agents face the challenge of handling cache overflow. The average reward fluctuates as the agents experiment with different strategies. A slight dip in the average reward is observed as the agents tackle this challenge, followed by a gradual increase as they find better approaches. Finally, in the fine-tune stage (after 38000 episodes), the average reward stabilizes. This indicates that the agents converge to an optimal strategy and a point of consistent and predictable performance. Random variations in the environment or slight adjustments made by the

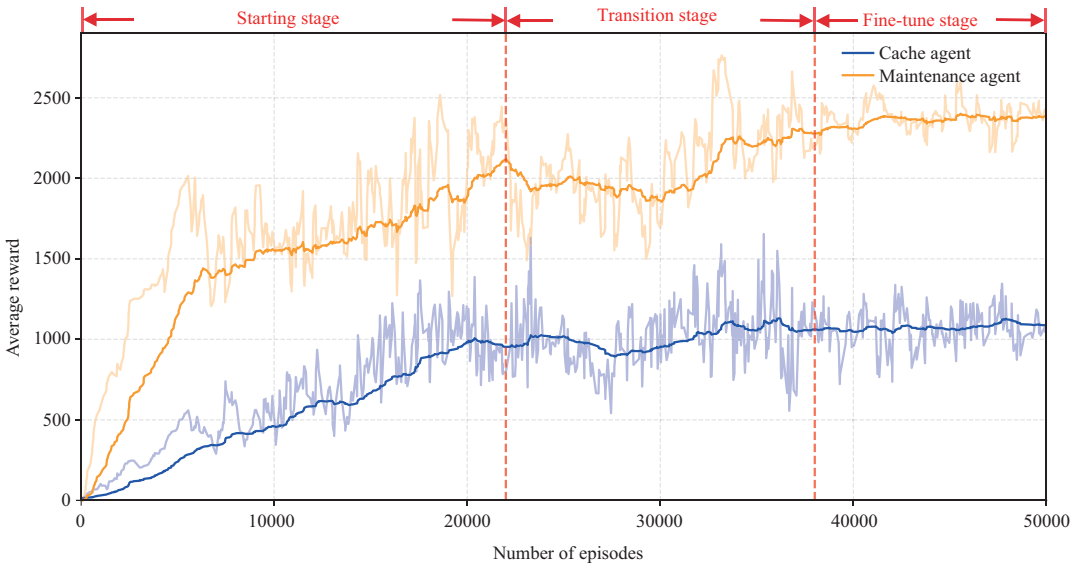

Figure 6 (Color online) Average reward over episodes.

agents to fine-tune their strategies can still generate minor fluctuations during this stage.

6.3 Performance comparison with established baselines

To evaluate the feasibility and performance of the proposed multi-level cache scheme based on D3QN, we compared RL-SHAP-Cache with five well-established algorithms. The baselines used in our evaluation experiment are as follows:

• Random selection. Randomly selects a cache level that can accommodate the requested service as the destination. When the cache is full, services are randomly selected for eviction.

• FIFO. FIFO caches services in the order of their arrival, starting from the L1 cache and progressing to the L3 cache. When the cache is full, the oldest cached service is removed first.

• LFU. LFU considers the number of accesses within a specific time frame to determine cache actions. When the cache is full, it prioritizes the removal of the least accessed data.

• Q-learning. As a model-free reinforcement learning algorithm that aims to maximize a Q-function, it estimates the expected cumulative reward for taking a specific action in a given state.

• DQN. DQN extends the Q-learning algorithm by leveraging an NN to approximate the Q-function. DQN incorporates techniques like experience replay and target networks to enhance stability and efficiency during learning, enabling better adaptability to complex scenarios.

During the experiments, we set up anchor points within the simulation environment to collect performance data at predefined time intervals. These anchor points were triggered when the timestamp reached a specified value, thereby enabling the collection of performance metrics for each baseline algorithm. The results of the performance comparison are presented in Table 4.

During the initial 2-minute cold start period, the proposed XRL-SHAP-Cache cached as many services as possible in the L1 cache, achieving the highest L1 cache utilization of 92.72%. This resulted in the best QoS and average response time among all the baselines. At the 2-hour mark, the performances of all baseline algorithms were generally at their peak as the caches had just been filled, ranking their highest hit rates during the entire lifecycle. At this point, the proposed XRL-SHAP-Cache outperformed other baselines in all evaluation metrics. At the 1-day mark, the previously cached services were no longer relevant due to shifts in user interests. Hard-coded baselines struggled to adapt to the new environment at this point, with the random selection hit rate decreasing by 63.54%. In contrast, XRL-SHAP-Cache continued to perform well by continuously learning from environmental observations, achieving an 87.91% hit rate, 43.24 s average response time and a load balancing factor of 92.71. Note that the utilization of L2 and L3 cache in XRL-SHAP-Cache is slightly lower than that of the other baselines. This is because completely filling the cache leads to frequent evictions, increasing energy consumption and adversely affecting QoS. Therefore, XRL-SHAP-Cache agents prefer to retain cache space to improve overall performance. Among all caching schemes, RL-based algorithms achieve superior performance owing to their ability to

Table 4 Performance comparison of different caching schemes over time[a)]

Time	Algorithm	Hit rate (%)	Avg response time (s)	QoS	Space utilization (%)			Load balancing
					L1	L2	L3	
2 min	Random	89.71	45.49	1.32	62.58	1.95	0.36	50.59
	FIFO	89.42	43.92	1.56	87.41	**6.32**	0	52.41
	LFU	**91.54**	37.15	1.71	88.36	6.27	0	55.87
	Q-learning	88.67	35.40	1.84	88.03	5.12	0	60.25
	DQN	90.33	33.92	1.93	90.24	5.44	**0.47**	59.04
	Ours	90.64	**32.56**	**2.04**	**92.72**	5.52	0.02	**61.88**
5 hour	Random	90.81	47.65	0.90	92.75	96.77	99.52	27.11
	FIFO	91.59	48.36	0.85	89.24	97.16	99.39	34.92
	LFU	91.02	45.19	0.78	91.39	98.64	99.28	48.30
	Q-learning	92.13	43.13	1.22	93.21	97.20	99.87	79.03
	DQN	92.94	42.62	1.41	93.69	98.52	99.41	88.20
	Ours	**93.46**	**41.86**	**1.53**	**93.77**	**98.81**	**99.89**	**90.52**
1 day	Random	34.27	68.35	0.30	87.87	**98.63**	**99.38**	14.46
	FIFO	63.58	60.13	0.44	91.02	95.15	99.27	34.32
	LFU	74.11	52.84	0.59	90.76	97.81	99.14	45.25
	Q-learning	77.19	49.10	0.94	91.18	94.97	94.84	77.84
	DQN	83.01	47.06	1.21	90.14	98.09	97.46	87.92
	Ours	**87.91**	**43.24**	**1.47**	**91.40**	94.13	95.49	**92.71**

a) The best performance in each evaluation metric is in bold.

adapt dynamically to changing environments. In particular, DQN surpassed Q-learning by 5.82% in hit rate and 3.96 s in response time. Our proposed D3QN-powered XRL-SHAP-Cache further outperformed DQN by 4.9% in hit rate and 5.2 s reduction in response time.

These results validate the feasibility and effectiveness of XRL-SHAP-Cache in CDNs. The proposed approach consistently outperformed the other algorithms in terms of QoS and load-balancing metrics. It adapted well to varying workloads and conditions over time, demonstrating its robustness and suitability in real-life environments.

6.4 Case study with local explainability

In this subsection, we present a series of case studies demonstrating the local explainability of the XRL-SHAP-Cache model in various real-world scenarios. These case studies utilized samples collected during the training process of the XRL-SHAP-Cache model. Each case study focuses on unique conditions, providing insight into the decision-making process of the model.

6.4.1 *Cold start*

CDNs may encounter the "cold start" scenario when deployed in new environments, with little to no cached data or optimization for the upcoming workload. In this scenario, CDNs struggle to adaptively optimize their performance due to the lack of historical usage records, whereby an issue known as "cache breakdown" may arise when all requests fail to hit the cache, causing a surge of requests being forwarded to the origin server. Traditionally, to address this problem, in prewarming [47] the cache is filled with randomly selected content. However, the proposed approach relies on D3QN agents to manage the cache dynamically during the request process, thereby allowing for a more adaptive and elastic caching strategy.

Figure 7 illustrates the impact of different input features on the Q-value output of each action for the cache agent when the cache is almost empty. The Q-value for action a_{idle} is significantly reduced by the high free-storage size ratio of the cache. Conversely, the Q-values for $a_{\text{L1}}, a_{\text{L2}}, a_{\text{L3}}$ are positively affected. Consequently, the cache agent is confident in not choosing a_{idle}. These behaviors are reasonable because the cache agent must prioritize caching as much content as possible to improve the cache hit rate.

Figure 8 further analyzes the impact of input features on the expected Q-value output of maintenance agent's actions a_{preserve} and a_{delete} for 100 newly deployed ES request samples. The analysis shows that the majority of input features contribute positively to the expected Q-value of the maintenance agent's output a_{preserve}. Specifically, the high free-space ratio and high ES cache miss rate during cold start have the most significant impact, leading to Q-values of approximately 600. However, the high free-space ratio

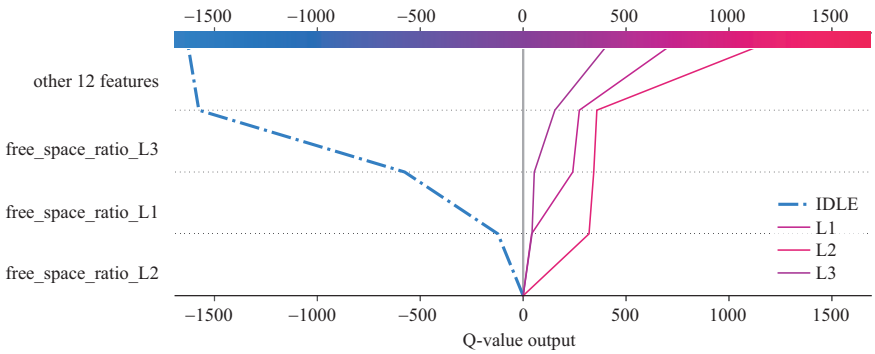

Figure 7 (Color online) Multi-output decision plot for cache agent actions captured on a newly deployed ES, where from bottom to top of the plot, SHAP values for each feature are added, showing how features contribute to the overall Q-value output.

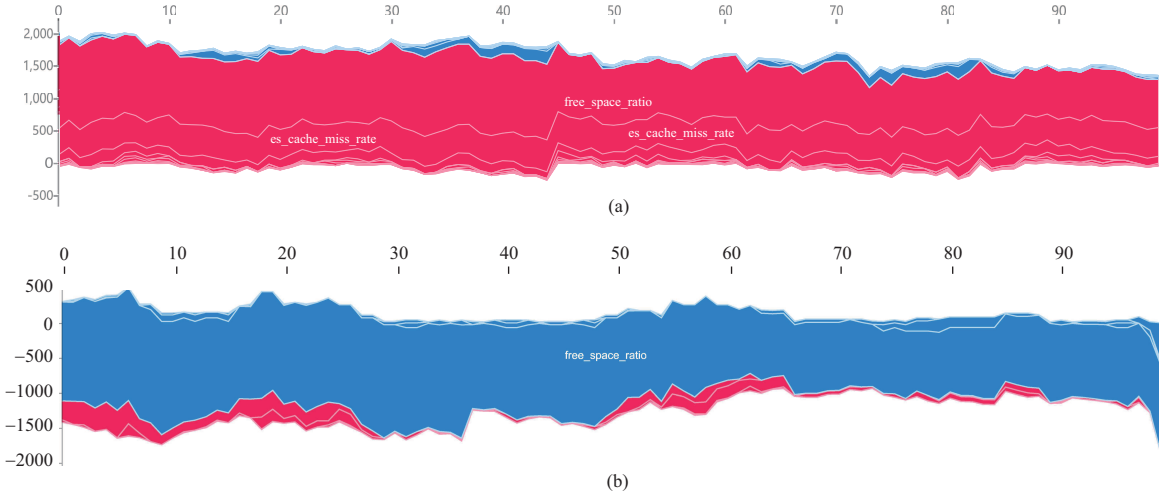

Figure 8 (Color online) Maintenance agent actions on 100 newly deployed ES request samples, where red (blue) indicates a positive (negative) impact on the Q-value output. (a) PRESERVE; (b) DELETE.

negatively affects the expected output of a_{delete}, resulting in Q-values below zero. This suggests that the maintenance agent takes a conservative approach, prioritizing a_{preserve} over a_{delete} to allow for a higher tolerance for cache hits and fewer evictions. Such decision-making tendencies help prevent the premature eviction of potentially popular content and excessive energy consumption, as the agent lacks sufficient historical data to accurately identify which content is truly unpopular.

6.4.2 *Cache management under resource-constrained conditions*

Due to the complexity of the proposed multi-level cache architecture, selecting the optimal cache level is critical for ensuring QoS. In addition, the cache maintenance process requires a delicate balance between preserving and evicting data. Aggressive cache eviction policies can lead to Cache Thrashing, causing frequent cache emptying and refilling, which results in significant performance degradation. Conversely, excessive data preservation can increase cache hit rate but also increase the risk of cache overflow.

Figure 9 provides insights into the influence of free storage space at different cache levels on cache level selection by the agent. The key findings are as follows:

(a) Adaptive cache level selection. When the free-space ratio is high in the L1 cache, the agent tends to favor caching the data at the L1 level. As the free-space ratio in the L1 cache decreases, the agent gradually shifts its preference to L2 and eventually to L3 cache. This demonstrates that the agent is adapting its decisions based on the available cache resources to maintain an optimal cache hit rate.

(b) Cache level preference. The agent's sensitivity to the free storage space ratio varies across different cache levels. Specifically, the agent shows higher sensitivity (indicated by higher SHAP values) to the free-space ratio in the L1 cache compared to the L2 and L3 caches. This preference aligns with the characteristics of the L1 cache, which has the lowest latency and highest I/O speed, making it crucial for ensuring QoS.

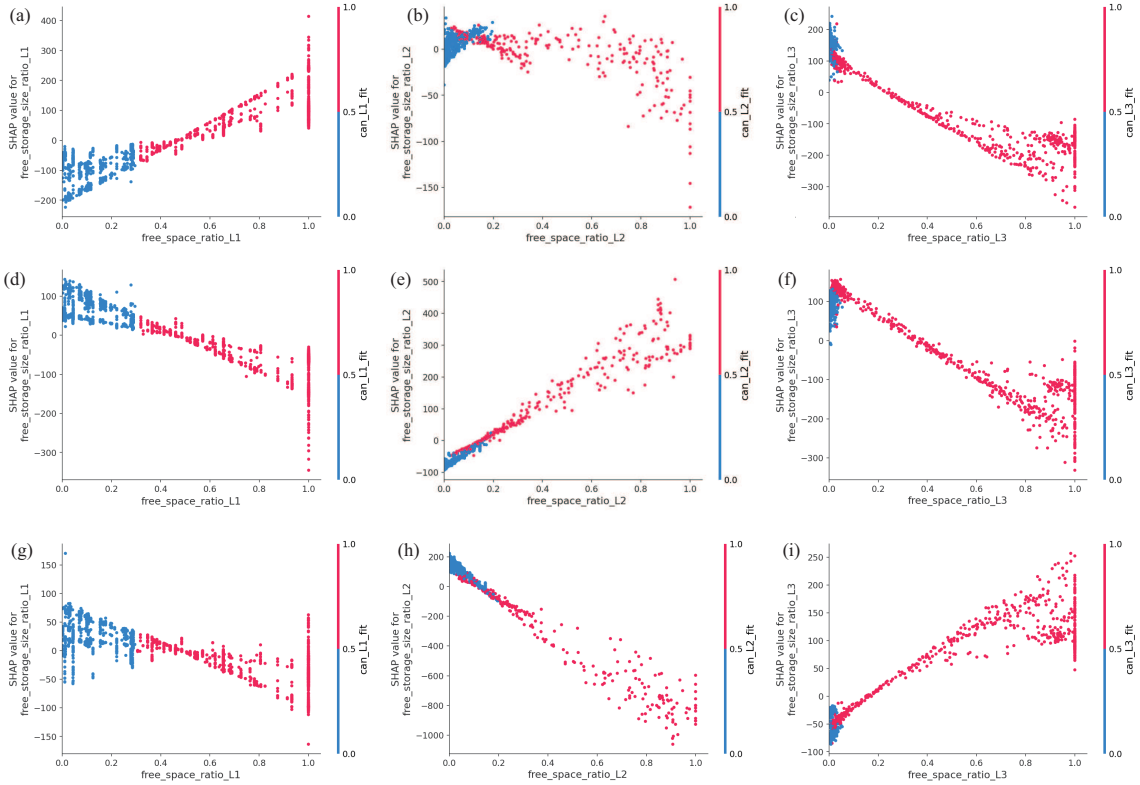

Figure 9 (Color online) Dependence plot for the impact of free storage space on cache level selection. Consecutive rows (cache to L1: (a)–(c), cache to L2: (d)–(f), cache to L3: (g)–(i)) represent actions of caching at different levels. The three graphs display the impact of the free-space ratio on the decision-making process for each of the three cache levels respectively.

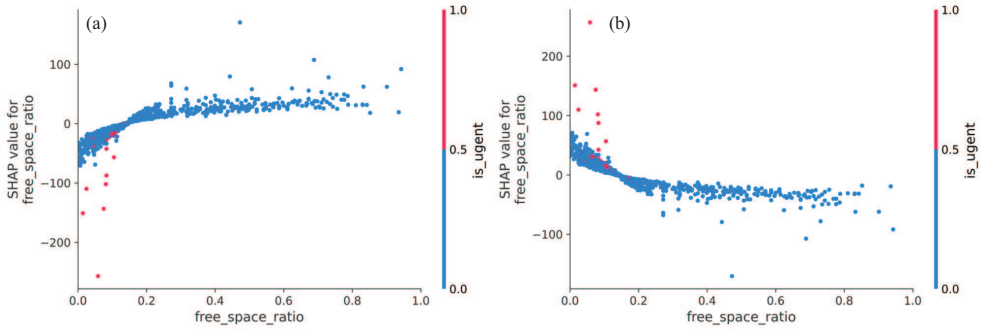

Figure 10 (Color online) Dependence plot displaying the impact of free space on maintenance agent actions. (a) PRESERVE; (b) DELETE.

(c) Improved action success rate. When the cache has a high free-space ratio, the can_fit feature has a greater impact on the decision-making process. This indicates that the agent is more likely to select a cache level capable of accommodating the data, thereby increasing the probability of a successful cache action.

In Figure 10, the maintenance agent exhibits the following patterns:

(a) Progressive cache eviction. The RL agent avoids aggressive cache eviction policies, particularly for the L1 cache. It only starts to consider evicting data from cache (SHAP values above 0) when the free space ratio is below a certain threshold possibly because the agent wants to maintain a certain level of cached data to prevent cache thrashing and maximize overall performance.

(b) Priority control. When the is_urgent flag is set to true, the agent becomes more aggressive in cache eviction to free up space for high-priority data. This strategy helps avoid exceeding cache limits, thereby preventing performance degradation.

6.4.3 *Services of intensive or moderate demand*

Modern CDNs implement strategies that prioritize services based on resource usage, which is crucial for an efficient allocation of resources as services are not accessed equally by users, and their resource requirements can vary significantly. Efficient allocation involves storing services with high resource demands for longer durations in the cache to avoid quick eviction. Conversely, services with a lower demand should be evicted more readily.

Highly attractive (e.g., charming) services tend to attract more traffic, resulting in increased bandwidth and CPU requirements. Figure 11 illustrates how the model dynamically adjusts its caching strategy based on the popularity of requested services. By analyzing the SHAP value patterns in the input features charm, req_freq and is_popular, the following conclusions can be drawn.

(a) Favoring high-demand services. The agent prioritizes popular services over less popular ones such that services with higher appeal and request frequency are more likely to be cached earlier and preserved for longer periods.

(b) Tolerance variation. Even for less popular services, larger cache capacities demonstrate higher tolerance. With abundant space, they can afford caching some lower-demand services in case their popularity increases over time.

(c) Decisive is_popular feature. The is_popular feature has substantial weight in agent decisions. If a service is designated as a popular trend, it can induce agents to boost their caching priority and lengthen their retention period. This is likely because of the possibility of sudden traffic surges to hot-listed services, which encourages agents to provide them with preferential treatment.

Service size is another significant factor that determines the demand for resources. Figure 12 provides a detailed analysis of the agent actions when dealing with services of varying sizes. The behavior of the cache agent in Figure 12(a) is consistent with the findings discussed in Subsection 6.4.2, and highlights the following points. The L1 cache is well-suited for lightweight services. However, it may struggle to accommodate bulky services because of limited capacity, resulting in significant adverse effects. By contrast, the L2 cache proved to be more versatile with moderate size and satisfactory I/O throughput, allowing it to effectively handle services of various sizes. For bulky services, the L3 cache is the most suitable with its large capacity. However, the relatively low speed makes it less favorable for minuscule services that require quick access times.

The heatmap in Figure 12(b) is somewhat noisy in that as the service scale increases, its impact on the Q-value of a_{delete} does not exhibit the expected linear upward trend. Instead, it reaches its maximum positive impact at the 4-th sample and then gradually decreases to approximately 0. This phenomenon may indicate that in real-world applications, an increase in service volume does not always directly lead to a linear positive impact on a_{delete} but can be influenced by various complex factors. Specifically, significantly large services that are not subject to eviction often possess greater flexibility in terms of popularity, causing the agent to shift its focus to state inputs other than service_size. Nevertheless, the maintenance agent may be perceived as attempting to strike a balance between two competing objectives: avoiding cache overflow (thereby evicting bulky services) and minimizing the need for a time-consuming reloading process for these services.

6.4.4 *Collaboration across geo-distributed nodes*

Recent advancements [48] in multi-agent reinforcement learning (MARL) have enabled collaboration across geo-distributed nodes. When selecting a DRL algorithm for interpretability, we opted for D3QN as the interpretable model. While multi-agent systems are known to excel in scheduling tasks in distributed systems, the inherent complexity introduced by the coordination among multiple agents can impede the interpretability goals. For a more straightforward demonstration and analysis of interpretable results, we focused on a single-agent approach.

Figure 13 demonstrates that even without employing the MARL algorithms, XRL-SHAP-Cache still allows for a certain level of collaboration between ESs. This collaboration is facilitated by the utilization of the nearby_cached_server_count features. As the number of nearby nodes that are caching the requested service increases, cache agents tend to either remain idle or cache the data to a high-capacity storage level, such as HDD. This behavior aims to reduce the repetition overhead and improve overall performance.

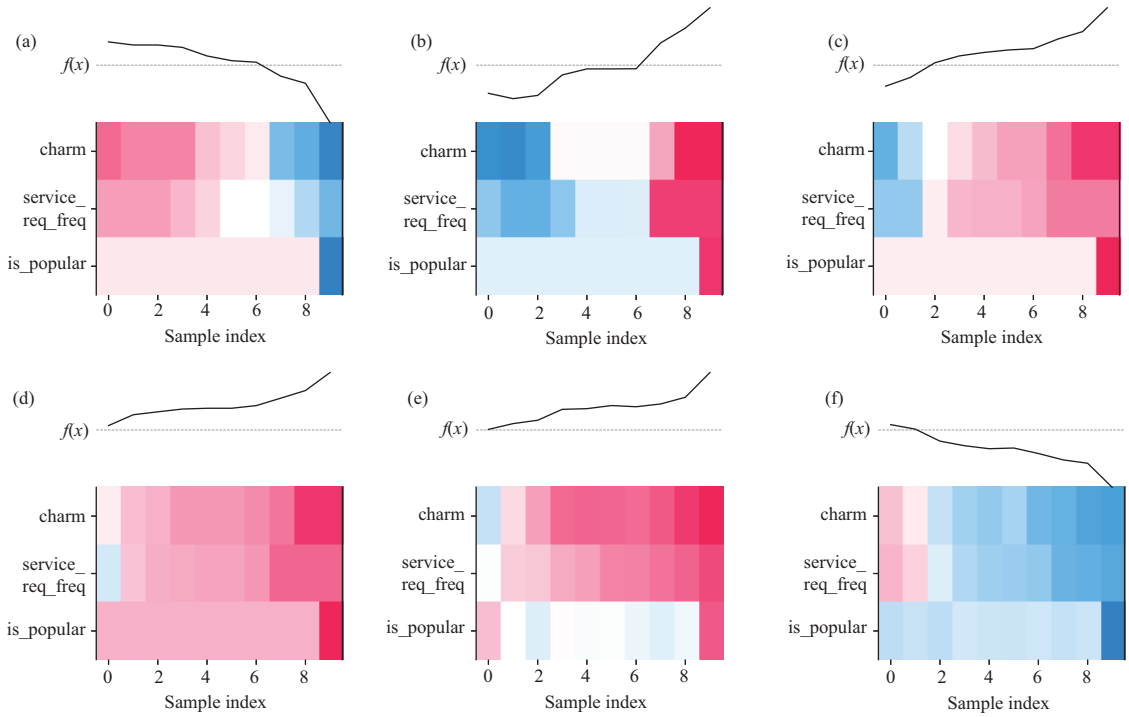

Figure 11 (Color online) Heatmap for the correlation between service popularity and agent actions (cache agents: (a)–(d) and maintenance agents: (e) and (f)) based on 10 samples with their charm increasing over time, where $f(x)$ counts the cumulative effect of all SHAP values shown in the figure and the grey horizontal line represents the threshold $f(x) = 0$. (a) IDLE; (b) cache to L1; (c) cache to L2; (d) cache to L3; (e) PRESERVE; (f) DELETE.

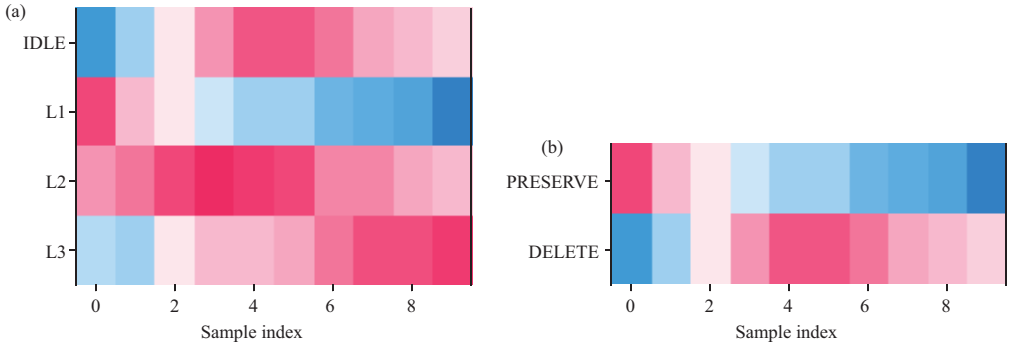

Figure 12 (Color online) Heatmap for the correlation between service size and agent actions based on 10 samples sorted by gradually increasing size. Each color block represents the proportion of input feature service_size SHAP values among all features, calculated as $\phi_{\text{service\_size}}/\sum_{j=1}^{N}|\phi_j|$. (a) Cache agent; (b) maintenance agent.

6.5 Visualizing feature importance for agent actions with global explainability

The contribution magnitude of each state input to the agent's decision is depicted in Figure 14. By examining these global explainability visualization graphs, deeper insights can be acquired into the overarching propensity underlying the strategies learned by agents across all state inputs.

For the cache agent, the storage-space-related features (i.e., can_X_fit, free_storage_size_ratio_X and service_size) exhibit the largest impact on the Q-values across all actions, as cache capacity limitation is the most fundamental feasibility constraint in its environment. This aligns with our objective of maximizing the success rate of caching actions, as the agent consistently prioritizes services that can fit within the available storage capacity. Once the fundamental capacity constraint is addressed, the next objective is to maximize cache hits by retaining the most reusable services. Thus, service popularity-related features (i.e., is_popular and charm) were observed to have the second-most substantial impact.

For the maintenance agent, the is_urgent flag stands out as the most dominant impact, demonstrating that the agent's top priority in promptly addressing the critical space release, is to ensure system avail-

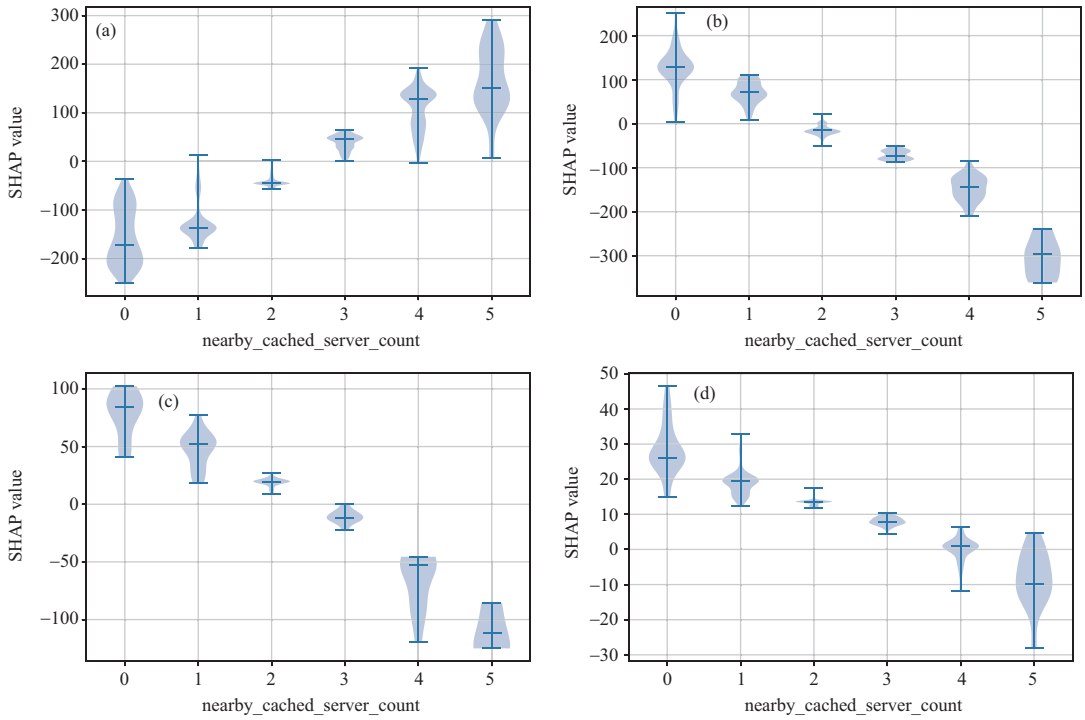

Figure 13 (Color online) Violin plot for the influence of nearby_cached_server_coun features on cache agent actions. (a) IDLE; (b) cache to L1; (c) cache to L2; (d) cache to L3.

ability. This robust behavior aligns seamlessly with the role of is_urgent as a final warning flag when the storage resources on an ES reach dangerously scarce levels. The free_space_ratio feature emerges as having the next most impact on decisions based on the heightened risks of cache failures when capacity becomes highly constrained. This learned prioritization of avoiding capacity bottlenecks aligns with reward function design and real-world system reliability objectives. Features reflecting request frequency (i.e., least_freq_index and service_req_freq) exert moderate impacts, emphasizing in-demand services for maintaining performance despite pressing capacity needs. By contrast, feature es_req_freq has the lowest importance, as the workload and network burden on ESs are less directly relevant to cache eviction decisions versus end user access patterns.

The global proportionality of feature impacts reveals structured prioritization cascades that are intrinsic to the agents' specialized contexts. Cache agents exhibit storage-feasibility filtering, prioritizing capacity constraints and only thereafter maximize service reusability. Maintenance agents demonstrate decisive capacity safeguarding during critical shortages, retaining frequently accessed services for their continued availability. Ultimately, the explanation hierarchies validate how agents intrinsically develop precise domain-tuned logics spanning from foundational constraints to goal optimization in a graded manner.

6.6 Feasibility analysis of model inference speed

The proposed XRL-SHAP-Cache incorporates D3QN to create a more adaptive and intelligent caching strategy, capable of handling high-dimensional state inputs and dynamic environments. However, the computational burden associated with the NN backbone of D3QN raises concerns regarding its practical application in real-world scenarios. To assess the feasibility of XRL-SHAP-Cache under high-concurrency conditions, we conducted a benchmark of its inference speed across various hardware configurations. The stress-test results are listed in Table 5.

Table 5 reveals that the cache agent $\mathcal{C}$ and maintenance agent $\mathcal{M}$ exhibit comparable transactions per second (TPS) results across different hardware configurations. This similarity arises from the adoption of identical network structures, including the same number of layers and weights for both agents. Both agents achieved an inference speed exceeding 3500 TPS even on a low-power laptop processor (Core i5-8250U). When utilizing a more potent processor, such as Core i7-12700KF, the system attained an even higher inference speed, exceeding 8000 TPS. CDN servers are typically I/O bound, and ESs experiencing higher workloads are likely to be equipped with more advanced hardware. Therefore, the demonstrated

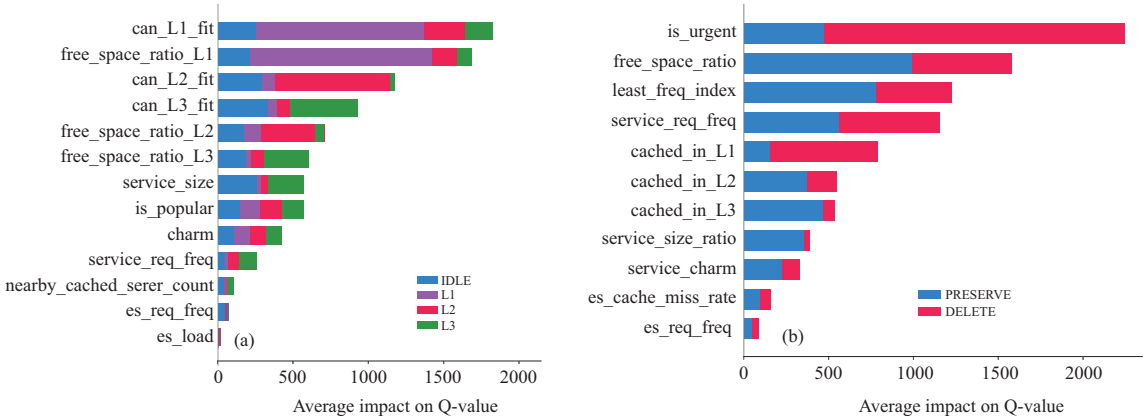

Figure 14 (Color online) Summary plot for global feature importance ranking of both agents, where each color stands for a specific action. (a) Cache agent; (b) maintenance agent.

Table 5 Inference speed comparison across different hardware specifications

Device type	Model name	Inference speed (TPS)	
		Cache agent	Maintenance agent
CPU	Core i5-8250U	3614.21	3568.38
	E3-1230 v3	5507.35	5421.96
	Core i7-12700KF	8495.42	8204.35
GPU	Tesla T4	14926.73	14893.99
	RTX 4090	26283.98	26106.93
TPU	Google TPU v2	7531.60	7317.13

performance is deemed sufficient for addressing the demands of most real-world scenarios. Additionally, notable increases in TPS have been observed by leveraging graphical processing units (GPUs) and application-specific integrated circuits (ASICs) for hardware acceleration. Transitioning to the Tesla T4 GPU resulted in a remarkable 176% inference performance increase compared with the Core i7-12700KF, reaching over 14000 TPS. Similarly, employing RTX 4090 yielded a substantial 309% TPS increase for both agents, escalating from 8200+ TPS on Core i7-12700KF to 26000+ TPS on RTX 4090. Notably, the GPU performance gain over the CPU was not as prominent as that observed in other DL applications. This is attributed to the design of the XRL-SHAP-Cache ensuring real-time decision making, such that multiple requests are not merged, thus leading to a constant batch size of 1. Consequently, GPUs cannot fully utilize their parallel processing capabilities. Future efforts will include improving GPU efficiency by batching input samples based on time slices, whereby requests within a designated time interval can be aggregated and batched, thereby allowing for better parallelism during the inference process. Even in extreme cases where the inference speed is still insufficient to meet the required standard, model compression techniques (e.g., quantization and pruning) can be implemented to further optimize the performance. Moreover, given the inherently distributed nature of CDNs, where both clouds and ESs are equipped with substantial computational resources, a distributed inference approach [33] can be adopted to allocate resources efficiently within the network for collaborative decision-making model inference.

In conclusion, the robust performance of XRL-SHAP-Cache, coupled with its capacity for continuous refinement through diverse optimization techniques, underscores its promise as a feasible method for deployment in real-world CDN environments.

7 Conclusion and future work

In this study, a D3QN-based multi-level cache scheme combined with Deep-SHAP technology was proposed to enhance the transparency of decision-making in network automation. The feasibility of the proposed solution was demonstrated through various scenarios, presenting the explainability results of the DRL agents through comprehensive case studies. Furthermore, the XRL-SHAP-Cache approach was compared with five well-established baselines to validate its superiority. The experimental results confirm its outstanding performance and effectiveness, showcasing significant improvements in key performance

metrics.

Several promising directions exist for future research in this field. One direction involves incorporating node interactions, such as service replication and service movement, into our network model to better address real-world challenges. Additionally, applications that can be built upon XAI techniques for error analysis, robustness testing, refinement, and human-machine interactions deserve further investigation. Finally, the assessment of the accuracy of such XAI techniques should continue to be a paramount focus for future advancements in this field.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant No. 92267104) and Natural Science Foundation of Jiangsu Province of China (Grant No. BK20211284).

References

1 Phillips N A. Content Delivery Network (CDN) Market: Global Industry Trends, Share, Size, Growth, Opportunity and Forecast 2023–2028. IMARC Market Research Report. 2022
2 Zolfaghari B, Srivastava G, Roy S, et al. Content delivery networks. ACM Comput Surv, 2020, 53: 1–34
3 Nisar K, Jimson E R, Hijazi M H A, et al. A survey on the architecture, application, and security of software defined networking: challenges and open issues. Internet Things, 2020, 12: 100289
4 Nygren E, Sitaraman R K, Sun J. The Akamai network. SIGOPS Oper Syst Rev, 2010, 44: 2–19
5 Zhu G X, Lyu Z H, Jiao X, et al. Pushing AI to wireless network edge: an overview on integrated sensing, communication, and computation towards 6G. Sci China Inf Sci, 2023, 66: 130301
6 Hu Z, Fang C, Wang Z, et al. Many-objective optimization-based content popularity prediction for cache-assisted cloud-edge-end collaborative IoT networks. IEEE Int Things J, 2024, 11: 1190–1200
7 He C, Ma M, Wang P. Extract interpretability-accuracy balanced rules from artificial neural networks: a review. Neurocomputing, 2020, 387: 346–358
8 Rachha A, Seyam M. Explainable AI In education: current trends, challenges, and opportunities. SoutheastCon, 2023, 2023: 232–239
9 Kaadoud I C, Bennetot A, Mawhin B, et al. Explaining Aha! moments in artificial agents through IKE-XAI: implicit knowledge extraction for explainable AI. Neural Netw, 2022, 155: 95–118
10 Wu Y, Lin G, Ge J. Knowledge-powered explainable artificial intelligence for network automation toward 6G. IEEE Netw, 2022, 36: 16–23
11 Bacciu D, Numeroso D. Explaining deep graph networks via input perturbation. IEEE Trans Neural Netw Learn Syst, 2023, 34: 10334–10345
12 Du Y, Antoniadi A M, McNestry C, et al. The role of XAI in advice-taking from a clinical decision support system: a comparative user study of feature contribution-based and example-based explanations. Appl Sci, 2022, 12: 10323
13 Padovan P H, Martins C M, Reed C. Black is the new orange: how to determine AI liability. Artif Intell Law, 2023, 31: 133–167
14 Yan H, Xu X, Dai F, et al. Service caching for meteorological emergency decision-making in cloud-edge computing. In: Proceedings of IEEE International Conference on Web Services (ICWS), 2022. 120–128
15 Kong X, Duan G, Hou M, et al. Deep reinforcement learning-based energy-efficient edge computing for Internet of Vehicles. IEEE Trans Ind Inf, 2022, 18: 6308–6316
16 Wang F, Wang F, Liu J, et al. Intelligent video caching at network edge: a multi-agent deep reinforcement learning approach. In: Proceedings of IEEE Conference on Computer Communications, 2020. 2499–2508
17 Fang C, Xu H, Yang Y, et al. Deep-reinforcement-learning-based resource allocation for content distribution in fog radio access networks. IEEE Int Things J, 2022, 9: 16874–16883
18 Nikbakht R, Kahvazadeh S, Mangues-Bafalluy J. Video on demand streaming using RL-based edge caching in 5G networks. In: Proceedings of IEEE Conference on Standards for Communications and Networking (CSCN), 2022. 208
19 Lim D, Lee W, Kim W T, et al. DRL-OS: a deep reinforcement learning-based offloading scheduler in mobile edge computing. Sensors, 2022, 22: 9212
20 Zhou X, Liu Z, Guo M, et al. SACC: a size adaptive content caching algorithm in fog/edge computing using deep reinforcement learning. IEEE Trans Emerg Top Comput, 2022, 10: 1810–1820
21 Wells L, Bednarz T. Explainable AI and reinforcement learning — a systematic review of current approaches and trends. Front Artif Intell, 2021, 4: 550030
22 Vouros G A. Explainable deep reinforcement learning: state of the art and challenges. ACM Comput Surv, 2023, 55: 1–39
23 Zhang K, Zhang J, Xu P D, et al. Explainable AI in deep reinforcement learning models for power system emergency control. IEEE Trans Comput Soc Syst, 2021, 9: 419–427
24 Dassanayake P M, Anjum A, Bashir A K, et al. A deep learning based explainable control system for reconfigurable networks of edge devices. IEEE Trans Netw Sci Eng, 2021, 9: 7–19
25 Zhu Y, Yin X, Chen C. Extracting decision tree from trained deep reinforcement learning in traffic signal control. IEEE Trans Comput Soc Syst, 2023, 10: 1997–2007
26 Chen L, Hu X, Tang B, et al. Conditional DQN-based motion planning with fuzzy logic for autonomous driving. IEEE Trans Intell Transp Syst, 2022, 23: 2966–2977
27 Aghaeipoor F, Sabokrou M, Fernández A. Fuzzy rule-based explainer systems for deep neural networks: from local explainability to global understanding. IEEE Trans Fuzzy Syst, 2023, 31: 3069–3080
28 Soares E, Angelov P P, Costa B, et al. Explaining deep learning models through rule-based approximation and visualization. IEEE Trans Fuzzy Syst, 2020, 29: 2399–2407
29 Mereani F, Howe J M. Exact and approximate rule extraction from neural networks with Boolean features. In: Proceedings of the 11th International Joint Conference on Computational Intelligence, Vienna Austria, 2019. 424–433
30 Dhebar Y, Deb K, Nageshrao S, et al. Toward interpretable-AI policies using evolutionary nonlinear decision trees for discrete-action systems. IEEE Trans Cybern, 2024, 54: 50–62
31 Qiao L, Wang W, Lin B. Learning accurate and interpretable decision rule sets from neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, New York, 2021. 4303–4311

32 Singh I, Smith P J, Dmochowski P A. Optimal SNR analysis for single-user RIS systems in Ricean and Rayleigh environments. IEEE Trans Wireless Commun, 2022, 21: 9834–9849
33 Xu X, Tian H, Zhang X, et al. DisCOV: distributed COVID-19 detection on X-ray images with edge-cloud collaboration. IEEE Trans Serv Comput, 2022, 15: 1206–1219
34 Gao Z H, Chen X M, Shao X D. Robust federated learning for edge-intelligent networks. Sci China Inf Sci, 2022, 65: 132306
35 Yang C, Xu X, Zhou X, et al. Deep Q network-driven task offloading for efficient multimedia data analysis in edge computing-assisted IoV. ACM Trans Multimedia Comput Commun Appl, 2022, 18: 1–24
36 Zhao H, Wang Q, Wang J, et al. Popularity-based and version-aware caching scheme at edge servers for multi-version VoD systems. IEEE Trans Circ Syst Video Technol, 2020, 31: 1234–1248
37 Einziger G, Eytan O, Friedman R, et al. Lightweight robust size aware cache management. ACM Trans Storage, 2022, 18: 1–23
38 Cho M, Kang D. ML-CLOCK: efficient page cache algorithm based on perceptron-based neural network. Electronics, 2021, 10: 2503
39 Araf S, Saha A S, Kazi S H, et al. UAV assisted cooperative caching on network edge using multi-agent actor-critic reinforcement learning. IEEE Trans Veh Technol, 2022, 72: 2322–2337
40 Hu H, Wu D, Zhou F, et al. Intelligent resource allocation for edge-cloud collaborative networks: a hybrid DDPG-D3QN approach. IEEE Trans Veh Technol, 2023, 72: 10696–10709
41 Yang Y, Lou K, Wang E, et al. Multi-agent reinforcement learning based file caching strategy in mobile edge computing. IEEE ACM Trans Netw, 2023, 31: 3159–3174
42 Zhang D, Wang W, Zhang J, et al. Novel edge caching approach based on multi-agent deep reinforcement learning for Internet of Vehicles. IEEE Trans Intell Transp Syst, 2023, 24: 8324–8338
43 Paleja R, Ghuy M, Arachchige N R, et al. The utility of explainable AI in ad hoc human-machine teaming. Adv Neural Inform Process Syst, 2021, 34: 610–623
44 Datta A, Sen S, Zick Y. Algorithmic transparency via quantitative input influence: theory and experiments with learning systems. In: Proceedings of IEEE Symposium on Security and Privacy (SP), 2016. 598–617
45 Lundberg S M, Lee S-I. A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17), Red Hook, 2017. 4768–4777
46 Shrikumar A, Greenside P, Kundaje A. Learning important features through propagating activation differences. In: Proceedings of International Conference on Machine Learning, 2017. 3145–3153
47 Romero F, Chaudhry G I, Goiri í, et al. Faa$T: a transparent auto-scaling cache for serverless applications. In: Proceedings of the ACM Symposium on Cloud Computing, Seattle, 2021. 122–137
48 Yan H, Bilal M, Xu X, et al. Edge server deployment for health monitoring with reinforcement learning in Internet of Medical Things. IEEE Trans Comput Soc Syst, 2024. doi: 10.1109/TCSS.2022.3161996