Adaptive Layer-wise Personalized Federated Deep Reinforcement Learning for Heterogeneous Edge Caching

Tan Li, *Member, IEEE*, Zhen Li, Hai Liu, *Member, IEEE*, Chao Yang, Tse-Tin Chan, *Member, IEEE*, and Jun Cai, *Senior Member, IEEE*

***Abstract*—Proactive caching is essential for minimizing latency and improving Quality of Experience (QoE) in heterogeneous edge networks. While Federated Deep Reinforcement Learning (FDRL) shows promise for developing cache policies, it faces challenges such as an expanding action space and difficulty in balancing global knowledge sharing with local environmental adaptation. In this paper, we propose a Layer-wise Relevance Propagation-aided Personalized Federated (LRP-PFed) Deep Reinforcement Learning framework for edge caching to maximize system utility while satisfying caching constraints. To handle the expanding action space, we design a Multi-Head Double Deep Q-Network (MH-DDQN) that reshapes the action output layers into a multi-head structure, where each head generates a sub-dimensional action. Furthermore, we introduce an LRP-based adaptive personalization mechanism that dynamically determines the optimal number of personalized layers for each edge server during training. This approach enables automatic adaptation to heterogeneous environments while leveraging global information to accelerate learning convergence. Extensive experiments validate the effectiveness of our approach, showing that MH-DDQN achieves superior cache hit rates and reduced computational complexity compared to traditional DRL methods, while our LRP-guided personalization strategy achieves superior performance, scalability, and adaptivity compared to existing FDRL methods.**

***Index Terms*—Proactive caching, deep reinforcement learning, layer-wise relevance propagation, personalized federated learning.**

I. Introduction

THE explosive growth of mobile and connected devices is reshaping digital content consumption, with networked devices expected to reach 29.3 billion by 2023 and over 70% of the global population having mobile connectivity [2]. Traditional cloud-centric content delivery architectures struggle to meet the latency requirements of emerging applications such as augmented reality, autonomous driving, and real-time gaming, as users must fetch content from distant cloud servers through backhaul networks. This centralized approach leads to significant network congestion, increased latency, and degraded Quality of Experience (QoE).

To address these limitations, Mobile Edge Computing (MEC) has emerged as a paradigm that brings computation and storage capabilities closer to end users [3]–[5]. Within this paradigm, proactive caching enhances edge intelligence by strategically pre-positioning popular content at edge servers before user requests arrive [6]–[8]. By proactively caching frequently requested content at the network edge, service providers can minimize the frequent back-and-forth communication with remote cloud servers, enabling faster content access and reducing backhaul traffic. However, one of the core challenges in proactive caching lies in the fact that content popularity is both unknown and dynamic, varying across different geographical locations, time periods, and user demographics [9].

In recent years, deep learning-based methods have been increasingly utilized to solve the proactive caching problem. A typical approach usually involves a two-stage pipeline: first predicting content popularity, followed by designing a caching policy [10]. While various deep neural networks [11], [12] have been applied to predicting popularity, the caching performance is bottlenecked by prediction accuracy. Moreover, the models need to be re-trained when user request patterns change, which can be computationally expensive.

To overcome these challenges, deep reinforcement learning (DRL) offers an end-to-end solution by learning caching policies through direct interactions with the environment, eliminating the need for explicit popularity predictions. Various DRL models [13]–[17] have shown success in optimizing caching strategies, but their performance relies on access to substantial observation data (e.g., user feedback). In scenarios where a single edge node has insufficient observations, DRL models may suffer from slow convergence or even fail to converge.

To mitigate this issue, federated deep reinforcement learning (FDRL) [18], [19] enables multiple edge nodes to collaboratively train a global DRL model by sharing only model parameters, rather than raw user feedback. This approach not only preserves user privacy but also facilitates knowledge sharing across distributed edge nodes, making it particularly well-suited for the edge caching problem. By aggregating diverse

This work was supported in part by Concordia University PERFORM Research Chair Program; in part by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant; in part by Guangdong Basic and Applied Basic Research Foundation under Grants 2024A1515012745; in part by the Research Grants Council of Hong Kong, SAR, under Grant UGC/FDS14/E04/25; in part by the Big Data Intelligent Center of HSUHK; and in part by the Research Matching Grant Scheme from the Research Grants Council of Hong Kong. (*Tan Li and Zhen Li are co-first authors.*) (*Corresponding author: Zhen Li.*)

Tan Li and Hai Liu are with the Department of Computer Science, The Hang Seng University of Hong Kong, Hong Kong SAR. (*email: tanli@hsu.edu.hk;hliu@hsu.edu.hk*)

Zhen Li and Jun Cai are with the Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada. (*email: zhen_li@ieee.org;jun.cai@concordia.ca*)

Chao Yang is with the School of Automation, Guangdong University of Technology, Guangzhou, China. (*email: chyang513@gdut.edu.cn*)

Tse-Tin Chan is with the Department of Mathematics and Information Technology, The Education University of Hong Kong, Hong Kong, China. (*email: tsetinchan@eduhk.hk*)

Part of this work was previously published in *WiOpt 2024* [1].

knowledge from different nodes, FDRL can accelerate model training and improve caching decisions, even for nodes with insufficient local observations [20], [21]. However, three critical challenges remain unaddressed:

1) *The curse of dimensionality in the caching action space*: The exponentially growing action space with increasing content numbers poses significant challenges for DRL methods. For each content item, edge servers must decide whether to cache, update, or remove it, leading to a combinatorial explosion where the action space grows exponentially with the number of content items. Value-based methods [13], [14] become computationally impractical as they need to evaluate Q-values for all possible action combinations, while actor-critic methods [15]–[17] require discretization of continuous policies for caching decisions, introducing sampling instability and suboptimal performance.

2) *Heterogeneity across edge server environments*: Conventional FDRL implementations typically employ uniform model aggregation, leading to action homogenization [22] across servers. However, edge servers in different geographical locations and time zones serve diverse user populations with distinct content popularity patterns. For example, business districts favor work-related content during daytime while residential areas prioritize entertainment content in evenings. This spatial-temporal heterogeneity makes one-size-fits-all approaches ineffective, as globally averaged models fail to capture local user preferences and environmental characteristics.

3) *Content staleness in dynamic edge environments*: Existing edge caching solutions primarily focus on optimizing cache hit rates and reducing retrieval costs, overlooking the temporal validity of cached content. In dynamic scenarios where content frequently updates (e.g., news feeds, social media, live streaming), serving outdated cached versions can significantly degrade user experience. This creates a fundamental trade-off between maintaining cache efficiency and ensuring content freshness, as frequent updates increase operational costs while infrequent updates compromise information timeliness.

To address the challenges of dimensionality and heterogeneity, we propose a Layer-wise Relevance Propagation Personalized Federated (LRP-PFed) DRL framework. The objective of this framework is to maximize system utility by jointly considering user satisfaction, cache payment cost, and the Age of Information (AoI) for cached content. In our framework, each edge server trains a DRL model locally using user feedback. These local models are then partially uploaded to a central server after the layer-wise selection, enabling each edge server to incorporate global knowledge while retaining the ability to adapt to its unique local environment. The key contributions of this work are summarized as follows.

• We design a new multi-head DRL network, called Multi-Head Double Deep Q-Network (MH-DDQN), which serves as a local cached model for each edge server in the proposed framework. Specifically, each head in the output layer of the MH-DDQN is responsible for generating a one-dimensional sub-action. This reduces the discrete cache action space from exponential to linear complexity regarding content size.

• To enable collaborative training among edge servers while adapting to local heterogeneity, we propose the LRP-PFed framework where each MH-DDQN-aided edge server acts as a participant. Our framework employs Layer-wise Relevance Propagation (LRP), a neural network interpretability tool, to quantitatively analyze the contribution of different network layers to caching decisions. Based on LRP scores and local-global distribution divergence, we adaptively determine the optimal split between base layers (shared globally) and personalized layers (retained locally), enabling automatic adaptation to varying degrees of environmental heterogeneity.

• Through extensive experiments, we first show that the proposed MH-DDQN outperforms other DRL algorithms in terms of convergence speed and system utility. Moreover, our LRP-PFed achieves superior performance compared to fully-federated, non-federated, fixed-federated algorithms, and other baselines under various system settings.

II. Related Work

Recent years have witnessed increasing interest in applying DRL to edge caching problems. Early attempts employed value-based methods such as DQN [13] and DDQN [14] to learn caching policies directly from user interactions. The Q-network in these models takes the current state as input and outputs Q-values for all possible caching actions. However, these methods face a combinatorial explosion in the action space, which grows exponentially with the number of content items, rendering them computationally infeasible for large-scale caching scenarios. To circumvent this challenge, one line of research has focused on actor-critic methods, such as DDPG [15], PPO [16], and SAC [17]. These approaches handle the high-dimensional action space by relaxing the discrete caching decisions into a continuous probability space. While this avoids the combinatorial explosion, it introduces a discrepancy between the learned continuous policy and the required discrete actions, often requiring heuristic rounding that can lead to suboptimal performance. An alternative approach is to directly address the discrete combinatorial action space by decomposing the joint action-value function. For instance, action branching architectures [23] reduce the output complexity from exponential to linear by using parallel network heads for each action dimension. This principle is further advanced in [24], [25], which factorized a global Q-value into individual components. Despite their success in other domains, these principles of action space decomposition and value function factorization have not yet been explored for the proactive edge caching problem.

Beyond the action space challenge, the efficacy of DRL-based caching is often bottlenecked by the availability of training data at individual edge servers. To mitigate this issue, FL has emerged as a promising paradigm that enables collaborative model training across distributed servers while preserving local data privacy. Early applications focused on collaborative content popularity prediction [26], where knowledge aggregation across servers improved prediction accuracy. More recently, this paradigm was extended to FDRL [20], enabling edge servers to collaboratively train a shared policy network by exchanging model parameters. Subsequent studies [19], [21] refined the FDRL framework by optimizing for system metrics like latency and energy efficiency. However, a critical limitation persists in these works: they typically rely on uniform model aggregation, forcing a single, "*one-size-fits-all*" global model upon all participants. This approach inherently overlooks the environmental heterogeneity across servers, where local user preferences can diverge substantially from the global average, leading to suboptimal local performance [22].

Personalization is an emerging topic in FL to address local heterogeneity. Existing personalized FL methods can be broadly categorized into three main paradigms: meta-learning-based methods, adaptive model fusion, and layer-wise personalization.

Meta-learning-based methods [27], [28] train a global meta-model that can be quickly adapted to local data by a few gradient updates. This balances collaborative learning with individual adaptation and is effective when clients share similar base representations but require customized refinement. However, repeated local fine-tuning imposes high computational overhead, which can be impractical for resource-limited edge nodes. Adaptive model fusion methods (e.g., FedFomo [29], FedAMP [30]) pursue lightweight personalization by adaptively adjusting the fusion ratio between local and global models at each client. Such methods efficiently mitigate client drift through parameter-level adaptation but operate with a uniform fusion ratio across the network, providing only coarse-grained control and limited insight into which model components should be globally shared or locally specialized.

In contrast, layer-wise personalization [31] delivers a more structure-aware and fine-grained form of adaptation by explicitly partitioning the model into shared "base" layers and locally trained "personal" layers. The underlying principle, supported by extensive research [32]–[35], is that shallow layers learn general features suitable for global aggregation, while deeper layers capture task-specific patterns that should be trained privately. According to a recent empirical analysis in [36], layer-wise approaches such as FedPer [31] and FedBABU [34] achieve not only competitive accuracy but also the lowest memory overhead among major personalized FL methods.

Despite its promise, the application of layer-wise personalization to FDRL for edge caching is still in its infancy. Initial attempts [37], [38] have explored this direction but often rely on heuristic layer splitting without a systematic methodology. To the best of our knowledge, our work is the first to employ interpretability techniques to guide the layer partitioning process in FDRL for edge caching.

III. System Model

A. Network Model

We consider an edge caching system consisting of a central content server (CCS) and a network of MEC servers, denoted by $\mathcal{M} = \{1, 2, \ldots, M\}$. Each MEC server is connected to a corresponding base station and provides content caching service to users.

The system manages a total of C content items, represented as $\mathcal{C} = \{1, 2, \ldots, C\}$, which can be requested by users. Each content $c \in \mathcal{C}$ is characterized by a tuple (η_c, ϕ_c, χ_c), where $\eta_c \in \mathbb{R}^+$ represents the data size of content c. The parameters $\phi_c \in \mathbb{R}^+$ and $\chi_c \in \mathbb{R}^+$ denote the downloading cost and updating cost, respectively. Specifically, ϕ_c represents the cost of downloading content c for the first time from the CCS to an MEC server. In contrast, χ_c represents the cost of updating an existing cached version of content c, which typically only requires transferring the modified portions of the content. Therefore, we have $\chi_c < \phi_c$ for each content c.

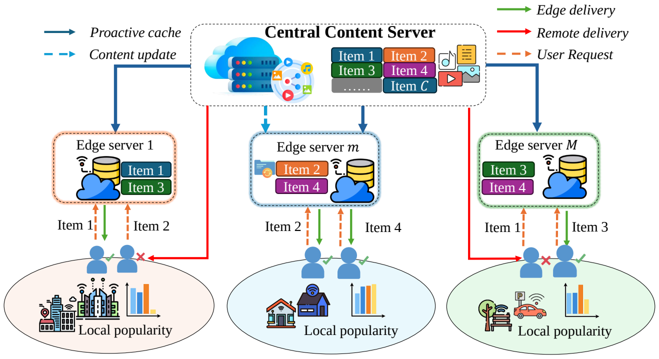


Fig. 1: System model.

The set of users is defined as $\mathcal{U} = \{1, 2, \ldots, U\}$, with each subset $\mathcal{U}_m \subseteq \mathcal{U}$ representing the users within the coverage area of MEC server m. To simplify the model, we assume that the coverage areas of different MEC servers do not overlap, i.e., $\mathcal{U}_i \cap \mathcal{U}_j = \emptyset$ for any $i, j \in \mathcal{M}, i \neq j$. The system model is depicted in Fig. 1.

B. Proactive Caching Model

The system operates over finite time slot set $\mathcal{T} = \{1, 2, \ldots, T\}$ and each slot contains the following stages:

- *Content Placement and Update.* At the beginning of time slot t, the content is deployed at MEC server m according to the caching policy $\alpha_m^t = \{\alpha_{m,1}^t, \alpha_{m,2}^t, \ldots, \alpha_{m,C}^t\}$, where $\alpha_{m,c}^t = 1$ if the MEC server m caches content c at time slot t, otherwise $\alpha_{m,c}^t = 0$. Given the physical storage limitation N_m of each MEC server m, the caching policy α_m^t must adhere to the following storage constraint:

$$\sum\nolimits_{c=1}^{C} \alpha_{m,c}^t \eta_c \leq N_m, \ \forall m \in \mathcal{M}, \ \forall t \in \mathcal{T}. \tag{1}$$

However, a policy $\alpha_{m,c}^t$ alone is insufficient because it does not capture the dynamic nature of content freshness. To address this, β_m^t is introduced to govern the active decision of when to update a cached item by fetching a new version. The update policy is denoted by $\beta_m^t = \{\beta_{m,1}^t, \beta_{m,2}^t, ..., \beta_{m,C}^t\}$, where $\beta_{m,c}^t = 1$ indicates that MEC server m updates content c at time slot t, and $\beta_{m,c}^t = 0$ otherwise. Note that content updates can only be performed on currently cached content, which leads to the following constraint:

$$\beta_{m,c}^t \leq \alpha_{m,c}^t, \ \forall m \in \mathcal{M}, \ \forall c \in \mathcal{C}, \ \forall t \in \mathcal{T}. \tag{2}$$

This separation cache and update policy is essential because these two decisions manage different aspects of system performance: one controls cache occupancy, while the other directly mediates the trade-off between the costs of performing an update and the benefits of serving fresh content. The interplay between these policies will be formally captured in the system utility model presented later.

• *User Request.* Following the content placement phase, each user $u \in \mathcal{U}_m$ generates a request vector $\mathbf{d}_u^t = \{d_{u,1}^t, d_{u,2}^t, \ldots, d_{u,C}^t\}$ and sends it to their associated MEC server m. Here, $d_{u,c}^t = 1$ indicates that user u requests content c at time slot t, and $d_{u,c}^t = 0$ otherwise. For each MEC server m, we aggregate all user requests into a received request vector $\mathbf{d}_m^t = \{d_{m,1}^t, d_{m,2}^t, \ldots, d_{m,C}^t\}$, where $d_{m,c}^t = \sum_{u \in U_m} d_{u,c}^t$ represents the total number of requests for content c. This aggregated demand directly reflects the content popularity $P_{c,m}$ in the service area of MEC server m.

In our proactive caching model, the content popularity $P_{c,m}$ is assumed to have an unknown distribution, and its statistical characteristics need to be learned through observations over time. Besides, due to varying user demographics and behavior patterns across different locations, the popularity of the same content exhibits **heterogeneity** across different MEC servers, i.e., $P_{c,m} \neq P_{c,n}$ for $m \neq n$. To model this heterogeneous popularity distribution, many works adopt the Mandelbrot-Zipf (MZipf) distribution [39]. Under this model, the popularity of content c at server m can be expressed as:

$$P_{c,m} = \frac{(\kappa_c + q_m)^{-k_m}}{\sum_{c' \in \mathcal{C}} (\kappa_{c'} + q_m)^{-k_m}}, \tag{3}$$

where κ_c represents the popularity rank of content c (with $\kappa_1 = 1$ for the most popular content). The plateau factor q_m determines how flat the popularity distribution is at the head, while the Zipf factor k_m controls how quickly the popularity decays for less popular content. Different combinations of these parameters can capture various user preference patterns in different regions.

• *Cache Service.* At the end of each time slot, the system processes all user requests through one of the two ways:

Edge delivery (cache hit): If the requested content has been cached at the local MEC server, i.e., $\alpha_{m,c}^t = 1$, the request is served directly from the edge cache, resulting in lower latency and reduced backhaul network load.

Remote delivery (cache miss): If the requested content is not available in the local cache, i.e., $\alpha_{m,c}^t = 0$, the content must be retrieved from the CCS through backhaul links, incurring additional delay and network costs.

C. System Utility Model

In this work, the system utility is composed of three parts:

• *Cache Payment Cost.* From the perspective of MEC servers, retrieving content from the CCS incurs significant costs that should be minimized. These costs primarily arise from two operations: (1) downloading new content that was not previously cached and (2) updating existing cached content to maintain freshness. The total payment cost of MEC server m at time slot t can be expressed as:

$$E_m^t = \sum\nolimits_{c=1}^{C} \underbrace{\mathbb{I}(\alpha_{m,c}^t, \alpha_{m,c}^{t-1})\phi_c}_{\text{downloading cost}} + \underbrace{I(\alpha_{m,c}^t, \alpha_{m,c}^{t-1})\beta_{m,c}^t \chi_c}_{\text{updating cost}}, \tag{4}$$

where $\mathbb{I}(i,j) = 1$ if and only if $i = 1$ and $j = 0$; $I(i,j) = 1$ if and only if $i = 1$ and $j = 1$. The first term captures the downloading cost when content c transitions from uncached ($\alpha_{m,c}^{t-1} = 0$) to cached ($\alpha_{m,c}^t = 1$) status. This corresponds to fetching the entire content from the CCS. The second term represents the updating cost for content that remains cached ($\alpha_{m,c}^{t-1} = \alpha_{m,c}^t = 1$) and is selected to update ($\beta_{m,c}^t = 1$) in the current time slot. In this case, only incremental updates of the content are required.

• *Cache Hit Ratio.* Regarding the users, a cache hit allows immediate edge delivery via cellular links, while a cache miss requires fetching the content from remote CCS via backhaul links, leading to larger delays. Therefore, the Cache Hit Ratio (CHR), as the metric used to evaluate the user QoE, is another factor of our system utility. Specifically, the instantaneous CHR can be calculated as:

$$H_m^t = \frac{\sum_{u \in \mathcal{U}_m} \sum_{c=1}^{C} \alpha_{m,c}^t d_{u,c}^t}{\sum_{u \in \mathcal{U}_m} \sum_{c=1}^{C} d_{u,c}^t}, \tag{5}$$

where the numerator represents the number of requests successfully served by the edge cache, and the denominator represents the total number of requests received by MEC server m. A higher H_m^t value indicates better caching efficiency and improved user QoE.

• *AoI cost.* To quantify the freshness of content delivered to users, we employ the AoI metric. For a content request, AoI measures the elapsed time since the most recent update of the delivered content items, directly reflecting the staleness of information received by users. Let $\delta_{m,c}^t$ denote the AoI of content c delivered by MEC server m at time slot t. Its evolution follows [40]:

$$\delta_{m,c}^t = \begin{cases} 1, & \text{if } \alpha_{m,c}^t = 0 \text{ or } \mathbb{I}(\alpha_{m,c}^t, \alpha_{m,c}^{t-1}) = 1 \text{ or} \\ & I(\alpha_{m,c}^t, \alpha_{m,c}^{t-1}) = 1, \beta_{m,c}^t = 1, \\ \delta_{m,c}^{t-1} + 1, & \text{if } I(\alpha_{m,c}^t, \alpha_{m,c}^{t-1}) = 1, \beta_{m,c}^t = 0. \end{cases} \tag{6}$$

The first case $\delta_{m,c}^t = 1$ represents the situation where users receive the latest version of the content, which occurs in three situations: (1) when the content is not cached ($\alpha_{m,c}^t = 0$) and retrieved directly from the remote server, (2) when the content is newly cached ($\mathbb{I}(\alpha_{m,c}^t, \alpha_{m,c}^{t-1}) = 1$), or (3) when the cached content is updated ($I(\alpha_{m,c}^t, \alpha_{m,c}^{t-1}) = 1, \beta_{m,c}^t = 1$). The second case reflects situations where users receive cached but non-updated content, resulting in linearly increasing AoI.

The average AoI for all requests served by MEC server m at time slot t is defined as:

$$\Delta_m^t = \frac{\sum_{u \in \mathcal{U}m} \sum_{c=1}^{C} \delta_{m,c}^t d_{u,c}^t}{\sum_{u \in \mathcal{U}m} \sum_{c=1}^{C} d_{u,c}^t}. \tag{7}$$

This metric only considers the AoI of requested content, reflecting our design that optimizes the user-centric trade-off between cache efficiency, operational costs, and information freshness. By focusing on actually requested content rather than the entire content library, our utility model ensures that optimization efforts are directed toward content that directly impacts user experience, making the subsequent reward function design both practical and meaningful.

D. Problem Formulation

Based on the three-component system utility model established above, we formulate the edge caching optimization as a multi-objective problem that balances user experience, operational efficiency, and content freshness. The inherent trade-offs among these objectives, such as the conflict between maximizing cache hit rates and minimizing payment costs when popular content requires frequent updates. In this paper, we aim to optimize the averaged system utility over a time horizon of T and all MEC servers M by jointly maximizing the CHR, minimizing the payment cost, and reducing the AoI. This optimization also considers the limited MEC server storage capacity and ensures that the AoI of any cached content does not exceed a predefined threshold. The optimization problem can be formulated as:

$$\max_{\{\mathbf{A},\mathbf{B}\}} \frac{\sum_{t=1}^{T}\sum_{m=1}^{M}\omega_1 H_m^t - \omega_2 E_m^t - \omega_3 \Delta_m^t}{TM}, \tag{8}$$

$$\text{s.t.} \begin{cases} \text{C1: } \sum_{c=1}^{C}\alpha_{m,c}^t \eta_c \le N_m,\ \forall m \in \mathcal{M},\ \forall t \in \mathcal{T}, \\ \text{C2: } \beta_{m,c}^t \le \alpha_{m,c}^t,\ \forall m \in \mathcal{M},\ \forall c \in \mathcal{C},\ \forall t \in \mathcal{T}, \\ \text{C3: } \delta_{m,c}^t \le \Delta_{\max}, \forall m \in \mathcal{M},\ \forall c \in \mathcal{C},\ \forall t \in \mathcal{T}, \\ \text{C4: } \alpha_{m,c}^t, \beta_{m,c}^t \in \{0,1\},\ \forall m \in \mathcal{M},\ \forall c \in \mathcal{C},\ \forall t \in \mathcal{T}, \end{cases}$$

where $\mathbf{A} = \{\alpha_1^t, \alpha_2^t, \dots, \alpha_M^t,\ \forall t \in \mathcal{T}\}$ and $\mathbf{B} = \{\beta_1^t, \beta_2^t, \dots, \beta_M^t,\ \forall t \in \mathcal{T}\}$ represent the caching and updating decision variables of MEC servers, respectively. The weighted parameters ω_1, ω_2, and ω_3 control the relative importance of the three utility components, enabling system operators to adapt the optimization focus to specific deployment scenarios. For instance, news delivery services may emphasize content freshness with higher ω_3, while video streaming platforms may prioritize cache efficiency with higher ω_1. The constraint C1 ensures that the sum of the content size will not exceed the storage capacity of each MEC server. The constraint C2 indicates that content can only be updated if it is already cached in the MEC server. The constraint C3 enforces an upper bound on the AoI to maintain content freshness. The constraint C4 ensures that our caching and updating actions are discrete.

The above optimization problem is an integer linear programming (ILP) problem, which is known to be NP-hard. The complexity comes from the fact that the combinatorial action spaces make it computationally intractable to find the optimal solution through exhaustive search. Due to the temporal dependencies in system state evolution, previous works have naturally modeled such problems as Markov Decision Processes (MDPs) and solved them using DRL approaches. However, as discussed in the introduction, (Federated) DRL approaches face several limitations when solving our problem. Therefore, we propose a novel approach in the next section that addresses these limitations through two key innovations: (1) redesigning the DRL model structure to reduce the action space from exponential to linear complexity, and (2) introducing a layer-wise personalized aggregation mechanism to handle heterogeneous content popularity while maintaining effective knowledge-sharing.

IV. Algorithm Design

In this section, we first propose a multi-head structure to improve the existing DRL algorithm, enabling a single MEC server to handle the large discrete action space. Subsequently, we construct a layer-wise personalized federated training architecture that allows multiple MEC servers to enhance collaborative learning performance while simultaneously accommodating local heterogeneity.

A. Multi-head Double Deep Q-Network

1) Markov Decision Process: To present our new DRL method, we first formulate the proactive caching process as an MDP, which can be defined by a 4-tuple $(\mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R})$, where $\mathbb{S}$ represents the set of states, $\mathbb{A}$ represents the set of actions, $\mathbb{P}_a(s, s')$ is the set of probability that action a in state s will lead to state s', $\mathbb{R}_a(s, s')$ consists of reward values after transitioning from state s to state s', due to action a. For each MEC server m, the detailed state space, action space, and reward function are summarized as follows.

- *State Space*. The state is the system information observed by the MEC server m at the beginning of each time slot, described as $\boldsymbol{s}_m^t = [\bar{\boldsymbol{d}}_m^t, \boldsymbol{\alpha}_m^{t-1}, \boldsymbol{\delta}_m^{t-1}] \in \mathbb{S}$. Here, $\boldsymbol{\alpha}_m^{t-1} \in \{0,1\}^C$ represents the current cache status after the previous time slot, and $\boldsymbol{\delta}_m^{t-1} \in \mathbb{R}^C$ denotes the current AoI of cached content. $\bar{\boldsymbol{d}}_m^t \in \mathbb{R}^C$ captures the historical content request patterns, defined as:

$$\bar{\boldsymbol{d}}_m^t = \frac{\sum_{k=1}^{w-1}\rho^k \cdot \boldsymbol{d}_m^{(t-k)}}{\sum_{k=1}^{w-1}\rho^k}, \tag{9}$$

where $\boldsymbol{d}_m^{(t-k)}$ represents the content request vector at time $(t-k)$. We adopt an exponentially weighted moving average (EWMA) approach to model content popularity, with window size w and decay factor $0 < \rho < 1$. This approach assigns exponentially decreasing weights to past observations, allowing the model to adapt more rapidly to temporal variations and capture the latest popularity trends more effectively.

- *Action Space*. The MEC server m determines which content should be cached and updated for the next time slot through two action vectors: $\boldsymbol{\alpha}_m^t = [\alpha_{m,1}^t, \alpha_{m,2}^t, \dots, \alpha_{m,C}^t]$ for caching decisions and $\boldsymbol{\beta}_m^t = [\beta_{m,1}^t, \beta_{m,2}^t, \dots, \beta_{m,C}^t]$ for updating decisions. For each content c, the possible combinations of $(\alpha_{m,c}^t, \beta_{m,c}^t)$ are constrained by C2, resulting in three valid action pairs: $(0,0)$ when content is not cached, $(1,0)$ when content is cached but not updated, and $(1,1)$ when content is both cached and updated. Consequently, for C content items, the total action space size is 3^C, which grows exponentially with the number of contents.

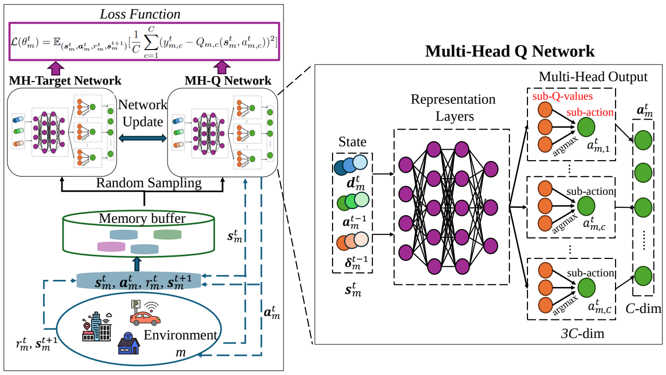


Fig. 2: Overview of MH-DDQN.

- *Reward Function*. Given the state-action pair, the reward obtained from the environment is defined as:

$$r_m^t = \omega_1 H_m^t - \omega_2 E_m^t - \omega_3 \Delta_m^t - \varrho, \tag{10}$$

where the first three terms directly correspond to our optimization objective in Eq. (8). The penalty term ϱ is designed to enforce hard constraints through reward shaping, defined as:

$$\varrho = \begin{cases} \lambda_1, & \text{if } \sum_{c=1}^{C} \alpha_{m,c}^t \eta_c > N_m \text{ (violates C1)}, \\ \lambda_2 \cdot V_m^t, & \text{if } V_m^t > 0 \text{ (violates C3)}, \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

where $V_m^t = \sum_{c=1}^{C} \alpha_{m,c}^t \cdot \mathbb{I}(\delta_{m,c}^t > \Delta_{\max})$ represents the number of cached content items that violate the freshness constraint at MEC server m during time slot t. Here, $\mathbb{I}(\cdot)$ is the indicator function that equals 1 when the condition is true and 0 otherwise. λ_1 and λ_2 are positive penalty values. For constraint C3, the penalty is proportional to the number of cached content items that exceed the freshness threshold, encouraging the agent to minimize both the occurrence and extent of constraint violations. This graduated penalty design provides more nuanced feedback to the learning agent, enabling faster convergence to feasible and high-quality solutions.

To handle the unknown transition probabilities and high-dimensional state-action space, we employ Double Deep Q-Network (DDQN) [14], which reduces overestimation bias compared to DQN [13]. However, traditional DDQN still faces challenges with exponentially growing discrete action spaces in our caching scenario.

2) Multi-head Q Network: Traditional value-based DRL methods, such as DQN and DDQN, learn a Q-function that estimates the expected cumulative reward of taking specific actions in a given state. During action selection, these methods evaluate Q-values for all possible actions and choose the one with the highest value. However, in our caching problem, each MEC server m needs to make joint decisions for C content items through two action vectors: $\boldsymbol{\alpha}_m^t$ for caching decisions and $\boldsymbol{\beta}_m^t$ for updating decisions. Due to constraint C2, each content c has three valid action combinations: $(\alpha_{m,c}^t, \beta_{m,c}^t) \in \{(0,0),(1,0),(1,1)\}$. This results in a joint action space of size 3^C, which grows exponentially with the number of contents and makes traditional DQN-based approaches computationally prohibitive for evaluating all possible action combinations.

To address this exponential action space challenge, we propose a Multi-Head Double Deep Q-Network (MH-DDQN), shown in Fig. 2. Our key insight is to decompose the joint decision-making process into C independent sub-decisions, transforming the exponential complexity into linear complexity. Instead of learning a single Q-function for the entire joint action space, our MH-DDQN decomposes it into C independent sub-Q-functions. As shown in the right part of Fig. 2, the MH-DDQN network uses C parallel output layers (referred to as *heads*), each responsible for estimating Q-values for the three possible action combinations of a single content. For notation convenience, we define the sub-action for content c as:

$$a_{m,c}^t = \begin{cases} 0, & \text{if } (\alpha_{m,c}^t, \beta_{m,c}^t) = (0,0), \\ 1, & \text{if } (\alpha_{m,c}^t, \beta_{m,c}^t) = (1,0), \\ 2, & \text{if } (\alpha_{m,c}^t, \beta_{m,c}^t) = (1,1). \end{cases} \tag{12}$$

Each head c learns an independent sub-Q-function, enabling the MEC server to make independent optimization decisions for each content dimension given the current state $\boldsymbol{s}_m^t$. This decomposition is well-suited for our caching problem because: (1) the reward function (Eq. 10) can be naturally decomposed into content-specific contributions, and (2) while content popularity may exhibit correlations, the caching decision for each content can be made independently given sufficient state information. Consequently, the computational complexity for action selection reduces from $O(3^C)$ to $O(3C)$, achieving a fundamental reduction from exponential to linear complexity.

The whole MH-DDQN consists of two neural networks with identical multi-head structures: the MH-Q network, which interacts with the environment to estimate Q-values, and the MH-target network, which computes target Q-values for training stability. When an MEC server m observes a state $\boldsymbol{s}_m^t$, the MH-Q network outputs sub-Q-values for each dimension $c \in \{1,\ldots,C\}$. In particular, the sub-Q-value of sub-action $a_{m,c,j} \in \{0,1,2\}$ in the c-th action dimension is defined as $Q_{m,c,j}(\boldsymbol{s}_m^t, a_{m,c,j})$. The sub-action with the highest Q-value is selected for each dimension:

$$a_{m,c}^t = \arg\max\nolimits_{a_{m,c,j} \in \{0,1,2\}} Q_{m,c,j}(\boldsymbol{s}_m^t, a_{m,c,j}). \tag{13}$$

The final joint action $\boldsymbol{a}_m^t$ is generated by combining each dimension action, i.e., $\boldsymbol{a}_m^t = \{a_{m,c}^t | c = 1,\ldots,C\}$.

The MH-target network $Q_{m,c}^-(\cdot)$ has the same C parallel heads with the MH-Q network, which is used to compute the target Q-values for updating the MH-Q network, defined as:

$$y_{m,c}^t = r_m^t + \gamma Q_{m,c}^-(\boldsymbol{s}_m^{t+1}, \underset{a_{m,c,j} \in \{0,1,2\}}{\arg\max} Q_{m,c,j}(\boldsymbol{s}_m^{t+1}, a_{m,c,j})), \tag{14}$$

where γ is the discounted factor.

The interaction between the MEC server and the environment generates transitions $(\boldsymbol{s}_m^t, \boldsymbol{a}_m^t, r_m^t, \boldsymbol{s}_m^{t+1})$, which are stored in a memory buffer. In each training step, a mini-batch of transitions is randomly sampled to compute the loss and update the MH-Q

Algorithm 1 MH-DDQN Algorithm for MEC Server m

```
Input: Reward function weights $\omega_1, \omega_2, \omega_3, \lambda_1, \lambda_2$, learning
    rate $\xi \in [0, 1]$, update coefficient $\tau \in [0, 1]$, number of
    episodes $E$ and training steps $T$.
Output: MH-DDQN model $\theta_m^t$ of MEC server $m$.
Initialize memory buffer $\mathfrak{B}_m$;
Initialize MH-Q network with random parameters $\theta_m$ and
    MH-target network with parameters $\theta_m^- = \theta_m$;
for episode $e = 1, \ldots, E$ do
    Initialize state $\boldsymbol{s}_m^0$ randomly;
    for $t = 0, 1, 2, \ldots, T$ do
        With probability $\epsilon$ select $\boldsymbol{a}_m^t$ randomly; Otherwise
            select $\boldsymbol{a}_m^t = \{a_{m,c}^t | c = 1, \ldots, C\}$ by Eq. (13);
        Cache content based on $\boldsymbol{a}_m^t$, observe next state $\boldsymbol{s}_m^{t+1}$
            and reward $r_m^t$ according to Eq. (10);
        Store transition $(\boldsymbol{s}_m^t, \boldsymbol{a}_m^t, r_m^t, \boldsymbol{s}_m^{t+1})$ into $\mathfrak{B}_m$;
        Sample a mini-batch of transitions randomly from
            $\mathfrak{B}_m$ and calculate loss function in Eq. (15);
        Update MH-Q Network by:
            $\theta_m^{t+1} = \theta_m^t - \xi \nabla_{\theta_m^t}(\mathcal{L}(\theta_m^t))$;
        Update MH-target network by:
            $\theta_m^{\overline{t+1}} = \tau\theta_m^{t+1} + (1-\tau)\theta_m^{\overline{t}}$;
    end for
end for
```

network parameters. The loss function also takes into account all C dimensions, defined as:

$$\mathcal{L}(\theta_m^t) = \mathbb{E}_{(\boldsymbol{s}_m^t, \boldsymbol{a}_m^t, r_m^t, \boldsymbol{s}_m^{t+1})}[\frac{1}{C}\sum_{c=1}^{C}(y_{m,c}^t - Q_{m,c}(\boldsymbol{s}_m^t, a_{m,c}^t))^2]. \tag{15}$$

The parameters θ_m of the MH-Q network are updated using gradient descent based on the loss calculated above, while the parameters θ_m^- of the target network are softly updated to slowly align with the MH-Q network, ensuring stability in the learning process. The primary steps of the MH-DDQN algorithm are summarized in Algorithm 1.

B. Layer-wise Personalized FDRL for Caching

While the proposed MH-DDQN successfully addresses the exponential action space challenge, DRL methods still face the fundamental requirement of sufficient training data to achieve satisfactory decision accuracy. In scenarios with limited local request observations, individual MEC servers may struggle to learn good caching policies. Therefore, we consider a federated learning approach that enables collaborative learning among multiple MEC servers. At each time slot, the MEC servers upload their locally trained MH-DDQN model parameters to the CCS for aggregation. This process ensures that raw user request data never leaves the local MEC server, providing a fundamental layer of privacy.

However, the federated approach introduces a critical question: *how much local knowledge should be shared?* Traditional FDRL that shares entire models would lead to a "compromise" strategy that performs suboptimally for heterogeneous servers. Consider two MEC servers: one in a business district where news content dominates, and another in a residential area where entertainment videos are popular. A promising solution is personalized FDRL where each edge server autonomously decides how to utilize global and local knowledge.

Specifically, we adopt a layer-wise approach where servers selectively replace certain layers with global parameters while retaining local parameters for others. This design is motivated by the hierarchical nature of neural networks: early layers learn general feature representations that are consistent across regions, while later layers learn content-specific decision policies highly dependent on local popularity patterns. The key challenge becomes: *how to determine which layers to share?* Existing layer-wise methods rely on empirical trial-and-error with fixed splitting points, facing prohibitive computational overhead and ignoring server-specific characteristics. To address this, we propose an LRP-based strategy that automatically determines server-specific optimal splitting points.

1) LRP-based Layer Selection: LRP is a widely-used explainability technique that traces the contribution of each neuron to the final output by backpropagating a "relevance" score through the network [41]. While the original LRP was designed for standard classifier networks, our MH-DDQN architecture, with its multi-head structure, requires a specific adaptation. The following process describes the computation of the layer-wise relevance score, $\text{LRP}_m^t(l)$, for a layer $l \in \{1, \ldots, L\}$ at a given MEC server m and time slot t. For notational simplicity, we omit the time index t from all variables throughout this derivation. The process is defined in three steps:

Step 1: Relevance Initialization. The LRP process begins at the output layer (denoted by layer L). For our MH-DDQN, this layer consists of C parallel heads, with each head containing 3 neurons that output the Q-values for the sub-actions defined in Eq. (12). We initialize the relevance of each output neuron as its Q-value, as this value is the quantity we aim to explain.

$$R_m^i(L) = Q_{m,c,j}(s_m, a_{m,c,j}), \tag{16}$$

Here, the neuron index i has a direct mapping to a specific content-action pair (c, j), for example, via $i = 3\times(c-1)+j+1$, where $c \in \{1, \ldots, C\}$ is the content index and $j \in \{0, 1, 2\}$ is the sub-action index. This ensures that the relevance attributed to each output neuron is precisely its contribution to the overall policy evaluation.

This initialization reflects the contribution of each sub-action decision to the overall caching policy. The output layer contains

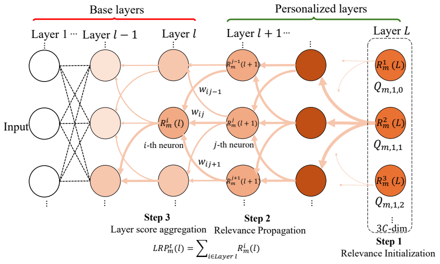


Fig. 3: Calculation of LRP values.

$3C$ neurons in total, organized into C heads, each responsible for evaluating three possible actions for one content item.

Step 2: Relevance Propagation. Following the layer-wise relevance propagation principle in [41], we propagate relevance from layer $l+1$ backwards to layer l by computing:

$$R^i_m(l) = \sum\nolimits_{k\in\text{layer } l+1} \frac{a^i_m(l) w^{ik}_m}{z^k_m + \epsilon \cdot \text{sign}(z^k_m)} \cdot R^k_m(l+1) \quad (17)$$

In this equation, $R^i_m(l)$ is the relevance of neuron i in layer l. The sum is over all neurons k in layer $l+1$ that are connected to neuron i. $a^i_m(l)$ is the activation of neuron i, and w^m_{ik} is the weight of the connection between neuron i and neuron k. The term $z^k_m = \sum_{j\in\text{layer } l} a^j_m(l) \cdot w^{jk}_m + b^k_m$ is the pre-activation of neuron k. A small stabilization constant ϵ (e.g., 0.01) is used to prevent division by zero. This rule redistributes the relevance of each neuron in layer $l+1$ to its contributors in layer l proportionally to their weighted activations, thereby conserving the total relevance across the network.

Step 3: Aggregation to a Layer-wise Relevance Score. After propagating relevance from the output layer all the way down to a specific layer l, we obtain the relevance $R^{m,(l)}_i$ for every neuron i in that layer. To get a single, comprehensive score representing the overall importance of layer l to the final decision, we sum the relevance of all its neurons:

$$\text{LRP}_m(l) = \sum\nolimits_{i\in\text{layer } l} R^i_m(l) \quad (18)$$

The whole process is shown in Fig. 3. The layer-wise relevance scores $\text{LRP}^t_m(l)$ computed through this process quantify the importance of different network layers for the caching decisions. Layers with higher relevance scores are deemed more critical for decision-making and thus more suitable for personalization, while layers with lower scores encode more general features that can be safely shared across MEC servers.

Given a set of layer-wise relevance scores $\{LRP^t_m(l)\}_{l\in\mathcal{L}}$, the next critical step is determining the optimal splitting point between base layers (shared globally) and personalized layers (retained locally). A straightforward approach would apply a uniform split threshold across all MEC servers. However, this ignores the fundamental heterogeneity in local environments.

In this work, we propose an adaptive splitting rule where the personalization threshold adapts to each server's local characteristics. The key insight is that servers with more distinctive local patterns should retain more personalized layers to preserve their unique decision-making capabilities, while servers with patterns similar to the global average can benefit more from knowledge sharing. To quantify this distinction, we use the Kullback-Leibler divergence (KLD) between local and global content request distributions as a measure of environmental heterogeneity:

$$KL(\mathbf{P}^t_m||\mathbf{P}^t_G) = \sum\nolimits_{c=1}^{C} P^t_m(c) \log \frac{P^t_m(c)}{P^t_G(c)}, \quad (19)$$

where $\mathbf{P}^t_m$ and $\mathbf{P}^t_G$ are C-dimensional probability vectors representing the local and global content request distributions, respectively:

$$P^t_m(c) = \frac{\bar{d}^t_{m,c}}{\sum_{c'\in\mathcal{C}} \bar{d}^t_{m,c'}}, P^t_G(c) = \frac{\sum_{m\in\mathcal{M}} \bar{d}^t_{m,c}}{\sum_{m\in\mathcal{M}} \sum_{c'\in\mathcal{C}} \bar{d}^t_{m,c'}}. \quad (20)$$

Based on this measure of local-global divergence, we design an adaptive personalization threshold for each MEC server that automatically adjusts to their environmental heterogeneity:

$$Sp^t_m = \min\{1, base_{sp} \cdot (1 + \lambda \cdot KL(\mathbf{P}^t_m||\mathbf{P}^t_G))\}, \quad (21)$$

where λ is a scaling factor and $base_{sp}$ is a basic sharing ratio. This formulation offers several key advantages: (1) servers with higher KL divergence (indicating more distinctive local patterns) are assigned higher thresholds, leading to more personalized layers; (2) the use of $\min$ naturally bounds the threshold within the [0, 1) range, ensuring its mathematical stability when camparing the LRP values; and (3) the scaling factor λ controls the sensitivity of the threshold to environmental heterogeneity, allowing the system to tune the balance between personalization and knowledge sharing.

With the adaptive threshold determined, we identify the splitting point using a cumulative relevance analysis. Since higher-relevance layers are more critical for decision-making and thus more suitable for personalization, we calculate the cumulative relevance ratio from the output layer backwards:

$$CLRP^t_m(l) = \frac{\sum_{k=l}^{L} LRP^t_m(k)}{\sum_{k=1}^{L} LRP^t_m(k)}. \quad (22)$$

With this backward definition, $CLRP^t_m(l)$ is non-increasing in l and satisfies $CLRP^t_m(1) = 1$. The rationale is to prioritize personalizing the most decision-critical deeper layers that exhibit higher LRP scores. The splitting point $l^*_m(t)$, i.e., the first personalized layer (closest to the input among the personalized ones), is chosen as the deepest layer whose backward cumulative relevance meets the personalization threshold:

$$l^*_m(t) = \max\{l|CLRP^t_m(l) \geq Sp^t_m\}. \quad (23)$$

The splitting point $l^*_m(t)$ identifies the deepest layer whose backward-cumulative relevance exceeds the personalization threshold Sp^t_m. Layers $\{l^*_m(t), l^*_m(t)+1, \ldots, L\}$ from this point to the output are personalized to preserve local decision-making capacity, while shallower layers $\{1, 2, \ldots, l^*_m(t)-1\}$ are shared as base layers to leverage global knowledge. Larger Sp^t_m (due to higher KL) pushes $l^*_m(t)$ toward the input, resulting in fewer shared base layers and more personalized layers.

Illustrative Example: Consider two MEC servers with different heterogeneity levels: Server A with $KL(\mathbf{P}_A||\mathbf{P}_G) = 0.8$ and Server B with $KL(\mathbf{P}_B||\mathbf{P}_G) = 0.2$. With a scaling factor $\lambda = 1.0$ and $base_{sp} = 0.5$, the adaptive thresholds become $Sp_A = 0.9$ and $Sp_B = 0.6$. Given a model with five layers and a backward cumulative relevance ratio array of $CLRP = [1.0, 0.95, 0.85, 0.70, 0.40]$, the splitting points are determined as follows. For Server A, we find $l^*_A = \max\{l|CLRP(l) \geq 0.9\} = 2$. Thus, Server A personalizes layers $\{2, 3, 4, 5\}$. For Server B, we find $l^*_B = \max\{l|CLRP(l) \geq 0.6\} = 4$. Thus, Server B only personalizes the layers, $\{4, 5\}$. This demonstrates

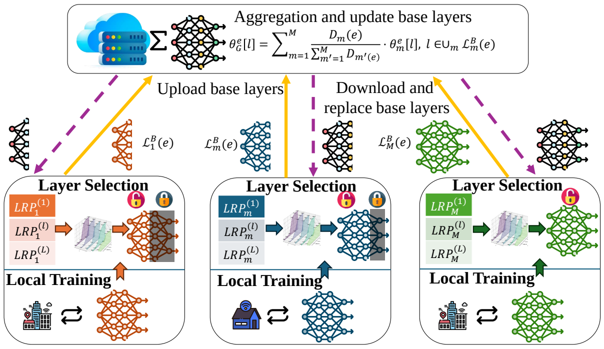


Fig. 4: Overview of proposed LRP-PFed DRL.

how our adaptive mechanism automatically assigns more personalized layers to servers with more distinctive local patterns.

2) LRP-PFed DRL Algorithm: After determining the optimal splitting points using LRP analysis, we integrate this layer selection mechanism into the FL process. The complete algorithm, referred to as *LRP-based Personalized Federated (LRP-PFed) DRL*, is illustrated in Algorithm 2 and Fig. 4.

As shown in Fig. 4, the training process operates in episodes. Each episode begins with parallel local training at all MEC servers, where both the MH-Q network and MH-target network are updated following Algorithm 1. At the end of each episode (after $e \times T$ time steps), the federated aggregation process is triggered with the following steps:

Step 1: Request Distribution Exchange. Each MEC server uploads its local historical request vector $\{\bar{d}_{m,c}^e\}_{c\in\mathcal{C}}$ to the CCS. The CCS computes the global request distribution $\mathbf{P}_G^e$ and broadcasts it back to all servers.

Step 2: Dynamic Layer Selection. Using the local distribution $\mathbf{P}_m^e$ and global distribution $\mathbf{P}_G^e$, each server independently determines its splitting point $l_m^*(e)$ through LRP analysis and adaptive thresholding. This divides the network into base layers $\mathcal{L}_m^B(e) = \{1, 2, \dots, l_m^*(e) - 1\}$ and personalized layers $\mathcal{L}_m^P(e) = \{l_m^*(e), \dots, L\}$.

Step 3: Selective Parameter Sharing. Each MEC server uploads only its base layer parameters $\{\theta_m^e[l] : l \in \mathcal{L}_m^B(e)\}$ to the CCS. The CCS performs weighted aggregation over the union of all uploaded base layers $\bigcup_m \mathcal{L}_m^B(e)$:

$$\theta_G^e[l] = \frac{\sum_{m:l\in\mathcal{L}_m^B(e)} D_m(e) \cdot \theta_m^e[l]}{\sum_{m:l\in\mathcal{L}_m^B(e)} D_m(e)}, \quad \forall l \in \bigcup_m \mathcal{L}_m^B(e),$$

where $D_m(e)$ represents the request volume at server m during episode e. This weighting scheme follows the same principle as sample-size-based aggregation in supervised FL: servers with higher request volumes have observed more state-action pairs (equivalent to training samples in supervised learning) and thus possess more reliable parameter estimates. By weighting according to request volumes, we ensure that the global model benefits more from servers with richer observations while reducing the influence of servers with limited training data.

Step 4: Model Update. The aggregated base layers are broadcast back to all servers, which update their corresponding base layers while keeping personalized layers unchanged.

Algorithm 2 LRP-PFed DRL for Edge Caching

```
Input: Number of episodes $E$, steps per episode $T$, and scaling factor $\lambda$, $base_{sp}$.
Output: Personalized DRL models for all MEC servers.
Initialize MH-DDQN models $\{\theta_m^0, \bar{\theta}_m^0\}$ for $m \in \mathcal{M}$;
for episode $e = 1, \dots, E$ do
    for each MEC server $m \in \mathcal{M}$ parallel do
        /* Local training at MEC Servers
        Train models $\theta_m^e$, $\bar{\theta}_m^e$ for $T$ steps using Alg. 1;
        Upload historical request vector $\bar{\mathbf{d}}_m^e$ to CCS;
        Calculate local and global distribution $\mathbf{P}_m^e$, $\mathbf{P}_G^e$;
        Calculate threshold $Sp_m^e$ based on Eq. (21);
        Randomly sample an action, calculate LRP scores $\{\mathrm{LRP}_m^e(l)\}_{l\in\mathcal{L}}$ and cumulative ratios $\{CLRP_m^e(l)\}_{l\in\mathcal{L}}$ based on Eq. (18) and (22);
        Determine split point $l_m^*(e)$ based on Eq. (23);
        Divide layers into set $\mathcal{L}_m^B(e)$ and $\mathcal{L}_m^P(e)$;
        Upload $\{\theta_m^e[l] : l \in \mathcal{L}_m^B(e)\}$ to CCS;
    end for
     /* Global Aggregation at CCS
    for each base layer $l \in \bigcup_m \mathcal{L}_m^B(e)$ do
        $\theta_G^e[l] = \sum_{m=1}^{M} w_m^e \theta_m^e[l]$, where aggregation weight $w_m^e = \frac{\sum_{c\in\mathcal{C}} \bar{d}_{m,c}^e}{\sum_{m'\in\mathcal{M}} \sum_{c\in\mathcal{C}} \bar{d}_{m',c}^e}$;
    end for
     /* Model Update at MEC Servers
    for each MEC server $m \in \mathcal{M}$ parallel do
        for each layer $l \in \mathcal{L}_m^B(e)$ do
            $\theta_m^e[l] = \bar{\theta}_m^e[l] = \theta_G^e[l]$;
        end for
        Keep layers $l \in \mathcal{L}_m^P(e)$ unchanged;
    end for
end for
```

Importantly, the LRP computation occurs only once per episode before each communication round, introducing minimal computational and communication overhead.

C. Algorithm Analysis and Discussion

Our LRP-PFed algorithm introduces minimal computational overhead compared to conventional federated DRL approaches. The LRP analysis requires a single backward pass through the network with $O(L \cdot \Theta)$ operations, where Θ represents the average number of parameters per layer. Since this computation occurs once per episode (every T time steps), the amortized per-step overhead is $\frac{O(L\cdot\Theta)}{T}$, which becomes negligible as $T \gg 1$ in typical DRL settings. The local training complexity remains unchanged at $O(L \cdot \Theta)$ per gradient update, while aggregation complexity reduces from $O(M \cdot L \cdot \Theta)$ to $O(M \cdot \bar{l}^* \cdot \Theta)$ where $\bar{l}^*$ is the average splitting point across servers. The total computational overhead per episode is $O(L\cdot\Theta)+O(T\cdot L\cdot\Theta)+O(M\cdot\bar{l}^*\cdot\Theta)$, dominated by the training cost $O(T \cdot L \cdot \Theta)$, making LRP overhead practically negligible. Additional memory requirements are minimal at $O(L+C)$ per server for storing relevance scores and request distributions, compared to model parameters $O(L \cdot \Theta)$.

Beyond computational efficiency, our LRP-PFed algorithm also offers privacy advantages. While standard FL provides a baseline of privacy by keeping raw data local, sharing model updates can still be vulnerable to inference attacks. Our partial sharing mechanism inherently mitigates this risk. By transmitting only a subset of generic layers and retaining personalized layers locally, we reduce the information exposed to potential adversaries, which makes it more difficult to infer sensitive information from the shared parameters. Although our method enhances privacy, robust security often requires dedicated protocols that are orthogonal to the learning algorithm. For future work, our framework could be integrated with advanced mechanisms, such as the blockchain-based reputation management system [42] or differential privacy mechanism [43], [44], to further enhance security without changing our algorithmic core.

V. Performance Evaluation

In this section, we conduct extensive experiments to evaluate the performance of our proposed approaches.

We consider a system consisting of five MEC servers ($M = 5$), each handling heterogeneous content requests from users within its coverage area. To simulate the heterogeneous local environment, we adopt the MZipf distribution with different plateau factors q_m and Zipf factors k_m across MEC servers. Additionally, we configure different user densities within the coverage areas of the MEC servers to represent another aspect of heterogeneity. The content sizes range from 1 to 8 GB, with payment cost varying between 0.03 and 0.55 HKD per content. The MH-DDQN architecture in our implementation consists of 6 hidden layers with 128 neurons per layer. The input and output dimensions are both $3C$, where C represents the number of content in the system. All DRL algorithms are implemented using PyTorch 1.8 framework. The detailed simulation parameters are summarized in Table I [1].

A. Performance Evaluation of MH-DDQN Algorithm

1) Compare with Other DRL Methods: In this subsection, we set the number of content to 50 and evaluate the effectiveness of the proposed MH-DDQN algorithm for a single MEC server. Since traditional DQN [13] cannot handle the large action space, we compare our MH-DDQN with two baseline methods: 1) **MH-DQN** [1], which uses a similar multi-head structure but based on DQN, and 2) **PPO** [16], an actor-critic based DRL method that generates continuous action policy and requires an additional sampling process.

TABLE I: Simulation Parameters

Parameter	Value
Plateau factors of MZipf q_m	[100, 200, 90, 40, 80]
Zipf factors of MZipf k_m	[0.60, 0.60, 0.75, 0.90, 0.90]
Downloading cost of contents ϕ_c	0.05∼0.55 HKD
Updating cost of contents χ_c	0.03∼0.45 HKD
Learning rate ξ	0.003
Update coefficient τ	0.005
Discounted factor γ	0.99
Scaling factor λ	0.5

TABLE II: Effect of reward function weights.

Weight Parameter	CHR	Payment Cost	AoI
ω_1 only	0.963	5.395	2.952
ω_2 only	0.244	1.317	1.488
ω_3 only	0.181	1.588	1.221
Multi-objective	0.518	2.037	1.663

Fig. 5 presents the learning curves of these three methods regarding overall system utility and its four components. The results demonstrate that the multi-head structure-based approaches significantly outperform the PPO method, with MH-DDQN achieving the best performance. Specifically, MH-DDQN converges to a higher system utility of approximately 0.6, compared to 0.5 for MH-DQN and −0.1 for PPO.

In terms of individual metrics, MH-DDQN achieves the highest CHR of 0.52, outperforming MH-DQN at 0.48. PPO shows unstable performance with CHR dropping to 0.25, mainly due to its action discretization process when sampling from continuous policy. For payment cost, MH-DDQN and MH-DQN both stabilize around 2.2, while PPO incurs higher costs at 3.1. Similarly, MH-DDQN maintains the lowest AoI at 1.8, comparable to MH-DQN, while PPO fluctuates between 2.5 and 3.0. All methods effectively minimize constraint violation penalties to below 0.1. However, MH-DQN and PPO show higher policy entropy fluctuations, indicating unstable learning dynamics.

2) Effect of Weights in Reward Function: To investigate how different weight combinations in the reward function affect the system performance and guide policy learning, we conduct experiments with four different weight settings, including three single-objective scenarios and one multi-objective scenario. Table. II presents the results in terms of three sub-objectives.

When focusing on optimizing CHR (ω_1 only), the algorithm achieves the highest CHR of 0.963, but at the cost of excessive payment cost of 5.395 and a high AoI of 2.952. This is because the policy prioritizes maintaining a large number of content to maximize CHR, leading to frequent cache operations and updates that incur high operational costs. Similarly, when only considering payment cost minimization (ω_2 only), the system maintains a low caching cost at 1.317, but sacrifices cache efficiency with a CHR of only 0.244. The AoI-oriented optimization (ω_3 only) yields the lowest content staleness of 1.221 but results in suboptimal performance in both CHR of 0.181 and cost of 1.588. In contrast, our proposed multi-objective weight setting achieves a balanced performance across all metrics. Although it may not outperform single-objective scenarios in their respective targeted metrics, it maintains reasonable performance levels with a CHR of 0.518, payment cost of 2.037, and AoI of 1.663.

B. Performance Evaluation of LRP-PFed DRL

Next, we extend MH-DDQN to a multi-server environment to explore the overall system performance. We evaluate the

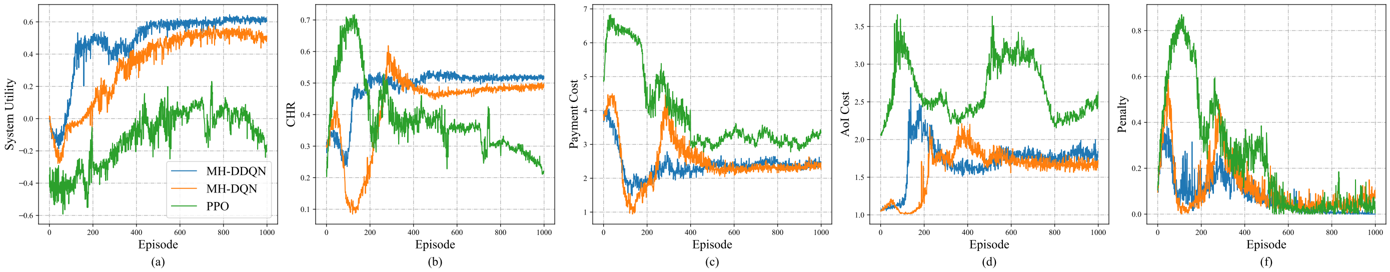


Fig. 5: Performance comparison of MH-DDQN, MH-DQN, and PPO algorithm.

performance of the proposed LRP-PFed DRL by comparing it with three learning-based algorithms and three traditional non-learning caching approaches, as described below.

Learning-based methods: i) **Fully-Fed**: A conventional FDRL framework without the personalized layers. All model parameters will be uploaded for aggregation; ii) **Non-Fed**: A conventional DRL framework without the FL paradigm. Each MEC server runs its own MH-DDQN model without communication; iii) **Fix-PFed** [1]: A personalized FDRL framework in which all MEC servers upload fixed, predetermined base layers for aggregation throughout all communication rounds.

Non-learning-based methods: iv) **LRU** [45]: A heuristic approach where the least recently used contents are replaced whenever the cache is full; v) **LFU** [45]: A heuristic approach where the least frequently used contents are replaced based on access frequency; vi) **Random**: A baseline strategy in which contents are cached randomly.

1) Convergence Performance: Fig. 6 illustrates the convergence behavior of our proposed method in comparison to Fully-Fed and Non-Fed approaches. Overall, our method demonstrates superior and stable convergence, reaching the highest system utility of approximately 0.7 and maintaining consistent performance after convergence. The Fully-Fed achieves better performance than the Non-Fed method but falls short of our solution, converging to around 0.5. This limitation can be attributed to the homogenization of actions across all agents due to complete parameter sharing, which fails to accommodate local heterogeneity in user feedback and content preferences. The Non-Fed exhibits significant performance instability and converges to the lowest system utility, with large fluctuations around 0.3. This inferior performance arises from insufficient local user feedback for training.

2) Effect of LRP: In this subsection, we analyze how the LRP can help select layers for our LRP-PFed. The LRP values are calculated by the Python library [46]. Fig. 7 presents the LRP values distribution across network layers for each MEC server. The LRP analysis across five MEC servers reveals consistent patterns in how different network layers contribute to the final caching decisions. A clear pattern is that the input layer and first hidden layer show relatively low LRP values, indicating that the initial processing of state vectors contributes less significantly to the final decisions. This suggests these layers primarily encode and transform the raw state information. Moving deeper into the network, we observe a consistent increase in LRP values. The middle hidden layers (#2 and #3) show moderate relevance, suggesting their role in feature extraction and preliminary decision-making. The final hidden layer and output layer demonstrate the highest LRP values, indicating their crucial role in synthesizing previous layer information for final action selection. The increasing values of LRP correspond to the idea that deeper layers in a neural network play a more crucial role in the decision-making process, while the earlier layers concentrate on feature extraction and learning representations. This analysis of relevance by layer offers valuable insights for improving model interpretability and optimization in the future.

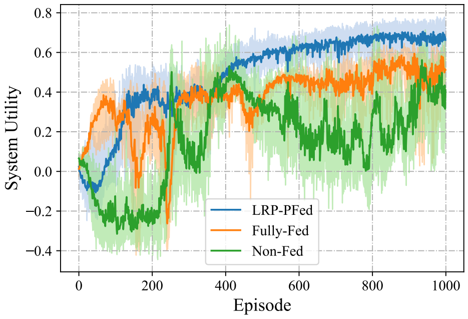


Fig. 6: Convergence comparison of learning-based methods.

Next, we compare the system utility achieved by our LRP-PFed and Fix-PFed on different number of personalized layers. The x-axis in Fig. 8 shows different numbers of personalized layers (0-6) for fixed strategies, while the last column repre-

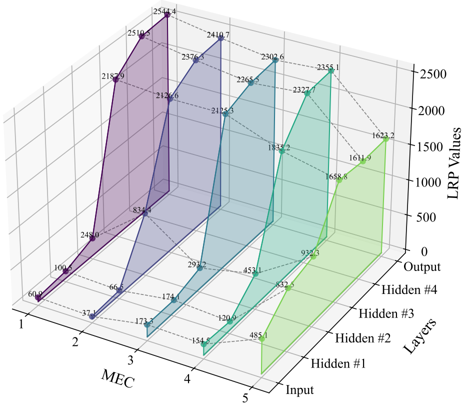


Fig. 7: LRP values for each MEC.

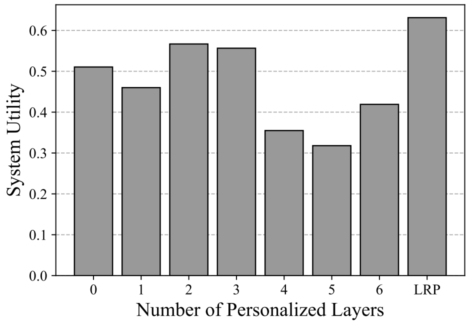


Fig. 8: Performance comparison with Fix-PFed on different numbers of personalized layers.

sents our proposed LRP-based approach. Among all fixed-layer strategies, personalizing 2 layers achieves the best performance with a system utility of approximately 0.57, followed by 3 layers at around 0.55. This indicates that neither too few nor too many personalized layers lead to optimal performance. Our proposed LRP-based approach achieves superior performance at around 0.62 compared to all fixed-layer strategies. This demonstrates the effectiveness of dynamically selecting personalized layers based on LRP analysis during training.

Another significant advantage of our approach is its computational efficiency in finding optimal personalized layers. While fixed-layer approaches require multiple training runs (up to 6 times) to identify the optimal number of layers to personalize, our method automatically adjusts the number of base layers for each MEC server during a single training process. The adaptive nature of our approach allows different MEC servers to upload different numbers of base layers in each communication round, based on their individual LRP patterns. This flexibility enables the system to better adapt to local characteristics and dynamics, improving overall performance.

These results validate that LRP-based layer selection provides better performance and training efficiency than fixed personalization strategies while offering the additional benefit of automaticity in finding optimal personalization configurations.

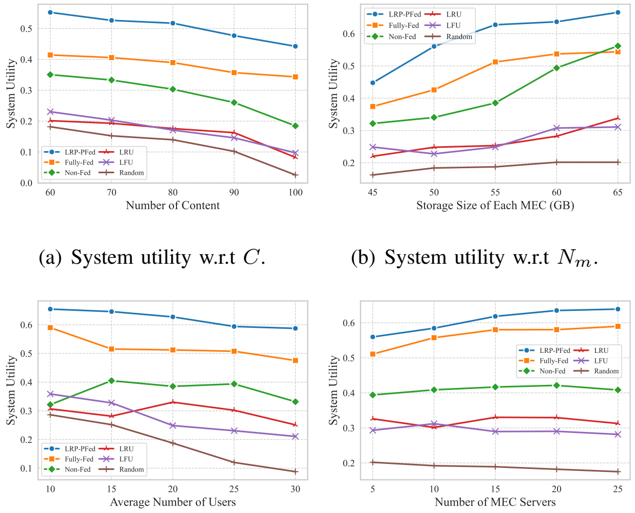
C. Performance under Different System Settings

We compare the performance of our proposed algorithm with other baseline methods under different system settings, including the number of content, the storage size of the MEC servers, and the average number of users. These experiments demonstrate the scalability and adaptability of our method across various scenarios.

1) Scalability analysis: Fig. 9(a) illustrates the system utility as the number of content items C is varied from 60 to 100. A general downward trend in utility is observable for all methods, which reflects the increased difficulty of learning a caching policy within a larger and more complex action space.

Our proposed LRP-PFed framework consistently yields the highest system utility, starting at 0.54 and decreasing to 0.44 at $C = 100$, demonstrating its superior performance and robustness as the problem scale increases. The Fully-Fed approach ranks second, with its utility declining from 0.41 to 0.34. The performance gap between LRP-PFed and Fully-Fed highlights the advantages of our personalized layer-sharing mechanism.

A more significant degradation occurs to the Non-Fed approach, whose utility decreases from 0.35 to 0.19. This decline is particularly notable as C exceeds 80. The underlying reason is the growing sparsity of user requests in a large content library, which hampers the ability of isolated local models to learn reliable patterns. Consequently, the utility of LRP-PFed is over 130% higher than that of Non-Fed when $C = 100$, indicating the critical need for collaboration. The non-learning baselines (LRU, LFU, Random) consistently exhibit the lowest performance, with their utilities all converging to values around or below 0.10, confirming their inefficiency in this setting. Fig. 9(b) presents a performance comparison of six algorithms as the cache size of each MEC server varies from 45 GB to 65 GB. From the results, we can observe that the proposed LRP-PFed algorithm consistently outperforms all baseline methods across different storage sizes, achieving the highest system utility that increases from 0.45 to around 0.65. The Fully-Fed approach ranks second, showing a similar increasing trend but with lower system utility values, reaching approximately 0.55 at 65 GB storage. The Non-Fed method demonstrates moderate performance, with system utility improving from 0.3 to 0.56. Notably, when the storage size reaches 65 GB, it can be seen that the Non-Fed approach surpasses the Fully-Fed. This is because the larger storage capacity enables each MEC server to interact with more cache feedback in each round, thereby better learning the local model. In contrast, the Fully-Fed approach suffers from performance degradation due to its neglect of heterogeneity among MEC servers, resulting in homogenized actions that fail to adapt to local characteristics. Non-learning based methods like LRU, LFU, and Random show relatively poor performance, with utility remaining below 0.35 regardless



(a) System utility w.r.t C.

(b) System utility w.r.t N_m.

(c) System utility w.r.t $|\mathcal{U}_m|$.

(d) System utility w.r.t M.

Fig. 9: Performance comparison under different system settings.

of storage size increases.

Fig. 9(c) presents the system utility performance of the caching algorithms as the average number of users per MEC server is varied from 10 to 30. The results show that the utility of most algorithms declines as the number of users increases. Our proposed LRP-PFed algorithm consistently outperforms all baseline methods, with a slight decrease in system utility from 0.62 to 0.59 as the user population grows. The learning-based methods generally exhibit higher utility than the non-learning-based methods. An interesting observation is that the system utility of the Non-Fed baseline initially increases and then decreases as the average number of users grows. This is because, with a larger user base, the Non-Fed algorithm can use more requests and feedback to enhance its caching strategy. However, the overall performance of Non-Fed remains lower than that of LRP-PFed and Fully-Fed, indicating the substantial benefits provided by the server communication, especially when the number of users in the server region is relatively small.

Fig. 9(d) demonstrates the system utility as the number of MEC servers is varied from 5 to 25. The performance of the non-collaborative methods, including Non-Fed and the non-learning baselines (LRU, LFU, Random), remains largely stable across the range of servers, exhibiting only minor random fluctuations. For instance, Non-Fed's utility consistently hovers around 0.4. This is an expected outcome, as these methods operate in isolation. Since the local request workload is constant, increasing the number of servers in the system provides no additional information for these isolated agents.

In contrast, the two collaborative frameworks, LRP-PFed and Fully-Fed, both demonstrate significant performance improvements as more servers participate in the learning process. This is because a larger number of servers contributes to a larger knowledge pool for aggregation, allowing the models to learn more generalized and effective caching policies. Specifically, the utility of our proposed LRP-PFed increases from approximately 0.56 (5 servers) to 0.63 (25 servers). Similarly, Fully-Fed's utility grows from around 0.51 to 0.59.

Throughout this scaling experiment, our LRP-PFed method consistently outperforms the standard Fully-Fed approach, further validating the effectiveness of our personalized aggregation mechanism. Additionally, it is noteworthy that the performance gains for both collaborative methods begin to plateau as the number of servers surpasses 15. This suggests that once enough models are participating in the aggregation, the marginal benefit of adding more servers' knowledge becomes less significant.

2) Adaptability analysis: To evaluate the adaptability of our proposed LRP-PFed algorithm under varying environmental conditions, we construct different system heterogeneity levels by simulating diverse content popularity distributions across edge servers. The heterogeneity level is quantified as the average KL divergence between each edge server's local content request distribution and the global distribution, calculated using Eq. (17). We categorize three heterogeneity levels: Low, Medium and High, representing increasing degrees of content popularity divergence across edge servers. Fig. 10 presents the system utility performance of four learning-based methods under different heterogeneity levels.

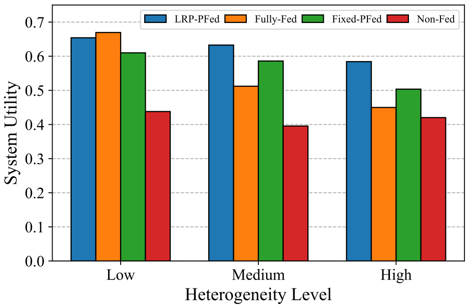


Fig. 10: Performance under different heterogeneity levels.

First, the Non-Fed approach demonstrates relatively stable performance across all heterogeneity levels, achieving system utilities of approximately 0.44, 0.40, and 0.42 for low, medium, and high heterogeneity level, respectively. This consistency stems from its reliance solely on local observations without any inter-server communication, making it insensitive to system-wide heterogeneity variations but consistently achieving the lowest performance among all federated approaches.

Besides, under low heterogeneity conditions where content request patterns are relatively homogeneous across edge servers, our LRP-PFed algorithm achieves comparable performance to Fully-Fed. This near-optimal performance occurs because our adaptive thresholding mechanism in Eq. (19) automatically encourages edge servers to share more layers when local-global distribution divergence is minimal, effectively leveraging the benefits of knowledge sharing. The Fixed-PFed approach achieves slightly lower performance at 0.61, representing an 8.7% decrease compared to Fully-Fed, due to its inability to dynamically adjust the amount of shared knowledge.

As system heterogeneity increases, the performance gap between our method and alternatives becomes more pronounced. In the medium heterogeneity level, LRP-PFed maintains superior performance at 0.64, outperforming Fixed-PFed by 7.8% and Fully-Fed by 25.5%. Under high heterogeneity conditions, this advantage further increases, with LRP-PFed achieving 0.60 compared to Fixed-PFed at 0.51 and Fully-Fed at 0.46, achieving 17.6% and 30.4% improvement, respectively. The significant performance degradation of Fully-Fed demonstrates the limitations of uniform model aggregation when local environments exhibit distinct characteristics.

All FL-based algorithms experience performance decline as heterogeneity increases, but with notably different degradation rates. LRP-PFed shows the most graceful degradation, declining only 10.4% from low to high heterogeneity scenarios, compared to Fixed-PFed's 16.4% decline and Fully-Fed's 33.3% decline. This demonstrates our algorithm's superior adaptability to environmental heterogeneity through its dynamic layer personalization mechanism.

VI. Conclusion

In this paper, we proposed the LRP-PFed DRL approach for multi-server proactive caching, effectively tackling three key challenges: immense action space, unknown content popularity, and heterogeneous content requests. We utilized a multi-head structure to reshape the DDQN's output layer, making the action output space scale linearly with the number of contents. We further proposed a layer-wise personalized federated training architecture. Based on the LRP-guided layer-splitting rules, each MEC server uploads only the base layers to the CCS for aggregation, while retaining the personalized layers locally. This balance leverages the benefits of collaborative knowledge sharing and allows individual MEC servers to adapt to their local content popularity.

References

[1] Z. Li, T. Li, H. Liu, and T.-T. Chan, "Personalized federated deep reinforcement learning for heterogeneous edge content caching networks," in *Proc. 22nd Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, Dec. 2024, pp. 313–320.

[2] U. Cisco, "Cisco annual internet report (2018–2023) white paper," *Cisco: San Jose, CA, USA*, vol. 10, no. 1, pp. 1–35, 2020.

[3] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, 2017.

[4] S. Yoo, S. Jeong, J. Kim, and J. Kang, "Cache-assisted mobile-edge computing over space–air–ground integrated networks for extended reality applications," *IEEE Internet Things J.*, vol. 11, no. 10, pp. 18 306–18 319, 2024.

[5] Y. Shi *et al.*, "Service migration or task rerouting: A two-timescale online resource optimization for MEC," *IEEE Trans. Wireless Commun.*, vol. 23, no. 2, pp. 1503–1519, 2024.

[6] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 82–89, Aug. 2014.

[7] D. Huang, X. Tao, C. Jiang, S. Cui, and J. Lu, "Trace-driven QoE-aware proactive caching for mobile video streaming in metropolis," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 62–76, Jan. 2020.

[8] C. Yi, S. Huang, and J. Cai, "An incentive mechanism integrating joint power, channel and link management for social-aware d2d content sharing and proactive caching," *IEEE Trans. Mobile Comput.*, vol. 17, no. 4, pp. 789–802, 2018.

[9] X. Xu, C. Feng, S. Shan, T. Zhang, and J. Loo, "Proactive edge caching in content-centric networks with massive dynamic content requests," *IEEE Access*, vol. 8, pp. 59 906–59 921, 2020.

[10] T. Li and L. Song, "Federated online learning aided multi-objective proactive caching in heterogeneous edge networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 9, no. 4, pp. 1080–1095, Aug. 2023.

[11] L. Ale, N. Zhang, H. Wu, D. Chen, and T. Han, "Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5520–5530, Jun. 2019.

[12] Y. Zhang, Y. Li, R. Wang, J. Lu, X. Ma, and M. Qiu, "PSAC: Proactive sequence-aware content caching via deep learning at the network edge," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2145–2154, Oct.-Dec. 2020.

[13] M. Ma and V. W. Wong, "A deep reinforcement learning approach for dynamic contents caching in HetNets," in *Proc. IEEE Int. Conf. Commun. (ICC)*. IEEE, 2020, pp. 1–6.

[14] J. Yan, M. Zhang, Y. Jiang, F.-C. Zheng, Q. Chang, K. M. Abualnaja, S. Mumtaz, and X. You, "Double deep Q-network based joint edge caching and content recommendation with inconsistent file sizes in fog-rans," *IEEE Trans. Veh. Technol.*, vol. 73, no. 3, pp. 4264–4276, 2023.

[15] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning-based edge caching in wireless networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 1, pp. 48–61, 2020.

[16] Y. Wang, S. Fu, C. Yao, H. Zhang, and F. R. Yu, "Caching placement optimization in UAV-assisted cellular networks: A deep reinforcement learning-based framework," *IEEE Wirel. Commun. Lett.*, vol. 12, no. 8, pp. 1359–1363, 2023.

[17] X. Gao, Y. Sun, H. Chen, X. Xu, and S. Cui, "Joint computing, pushing, and caching optimization for mobile-edge computing networks via soft actor-critic learning," *IEEE Internet of Things Journal*, vol. 11, no. 6, pp. 9269–9281, 2023.

[18] A. Tian *et al.*, "Efficient federated DRL-based cooperative caching for mobile edge networks," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 1, pp. 246–260, Mar. 2023.

[19] D. Qiao, S. Guo, D. Liu, S. Long, P. Zhou, and Z. Li, "Adaptive federated deep reinforcement learning for proactive content caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4767–4782, Dec. 2022.

[20] X. Wang, C. Wang, X. Li, V. C. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.

[21] H. Wu, B. Wang, H. Ma, X. Zhang, and L. Xing, "Multiagent federated deep-reinforcement-learning-based collaborative caching strategy for vehicular edge networks," *IEEE Internet Things J.*, vol. 11, no. 14, pp. 25 198–25 212, Jul. 2024.

[22] H. Jin, Y. Peng, W. Yang, S. Wang, and Z. Zhang, "Federated reinforcement learning with environment heterogeneity," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2022, pp. 18–37.

[23] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018.

[24] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *J. Mach. Learn. Res.*, vol. 21, no. 178, pp. 1–51, 2020.

[25] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. Int. Conf. Auton. Agents Multi-Agent Syst.*, 2018, pp. 2085–2087.

[26] K. Qi and C. Yang, "Popularity prediction with federated learning for proactive caching at wireless edge," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–6.

[27] Z. Zhan and X. Zhang, "Computation-effective personalized federated learning: A meta learning approach," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2023, pp. 957–958.

[28] Y. Chen, C. Wu, Z. Du, Y. Lin, S. Djahel, and L. Zhong, "Hier-fedmeta: A hierarchical federated meta-learning framework for personalized and efficient iov systems," in *Proc. IEEE Veh. Technol. Conf. (VTC2024-Spring)*, 2024, pp. 1–5.

[29] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, "Personalized cross-silo federated learning on non-iid data," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 9, 2021, pp. 7865–7873.

[30] M. Zhang, K. Sapra, S. Fidler, S. Yeung, and J. M. Alvarez, "Personalized federated learning with first order model optimization," *arXiv preprint arXiv:2012.08565*, 2020.

[31] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *arXiv preprint arXiv:1912.00818*, 2019.

[32] S. Lee, T. Zhang, and A. S. Avestimehr, "Layer-wise adaptive model aggregation for scalable federated learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 37, no. 7, 2023, pp. 8491–8499.

[33] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Exploiting shared representations for personalized federated learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 2089–2099.

[34] J. Oh, S. Kim, and S.-Y. Yun, "FedBABU: Towards enhanced representation for federated image classification," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[35] P. P. Liang, T. Liu, L. Ziyin, N. B. Allen, R. P. Auerbach, D. Brent, R. Salakhutdinov, and L.-P. Morency, "Think locally, act globally: Federated learning with local and global representations," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019.

[36] A. A. Khan, A. F. Khan, H. Ali, and A. Anwar, "Personalized federated learning techniques: Empirical analysis," in *Proc. IEEE Int. Conf. Big Data (BigData)*. IEEE, 2024, pp. 1333–1339.

[37] J. Gao, W. Wang, F. Nikseresht, V. Govinda Rajan, and B. Campbell, "PF-DRL: Personalized federated deep reinforcement learning for residential energy management," in *Proc. ICPP*, 2023, pp. 402–411.

[38] C. F. Mendoza, M. Kaneko, M. Rupp, and S. Schwarz, "Enhancing the uplink of cell-free massive mimo through prioritized sampling and

personalized federated deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, 2025.

[39] M.-C. Lee, M. Ji, A. F. Molisch, and N. Sastry, "Throughput–outage analysis and evaluation of cache-aided D2D networks with measured popularity distributions," *IEEE Trans. Wireless Commun.*, vol. 18, no. 11, pp. 5316–5332, Nov. 2019.

[40] X. Wang, M. Yi, J. Liu, Y. Zhang, M. Wang, and B. Bai, "Cooperative data collection with multiple UAVs for information freshness in the Internet of Things," *IEEE Trans. Commun.*, vol. 71, no. 5, pp. 2740–2755, May 2023.

[41] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.

[42] W. Zhu, L. Shi, J. Li, B. Cao, K. Wei, Z. Wang, and T. Huang, "Trustworthy blockchain-assisted federated learning: Decentralized reputation management and performance optimization," *IEEE Internet Things J.*, vol. 12, no. 3, pp. 2890–2905, 2025.

[43] T. Li, L. Song, and C. Fragouli, "Federated recommendation system via differential privacy," in *2020 IEEE international symposium on information theory (ISIT)*. IEEE, 2020, pp. 2592–2597.

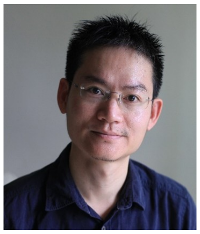
[44] S. D. Okegbile, J. Cai, H. Zheng, J. Chen, and C. Yi, "Differentially private federated multi-task learning framework for enhancing human-to-virtual connectivity in human digital twin," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 11, pp. 3533–3547, 2023.

[45] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of Facebook photo caching," in *Proc. ACM SOSP*, Nov. 2013, p. 167–181.

[46] N. Kokhlikyan *et al.*, "Captum: A unified and generic model interpretability library for pytorch," 2020. [Online]. Available: https://arxiv.org/abs/2009.07896

Tan Li received the Ph.D. degree in Computer Science from City University of Hong Kong (CityU) in 2023. Prior to that, she received the M.Eng. degree in Control Engineering from University of Science and Technology of China (USTC) in 2019, and the B.Eng. degree in Automation from Central South University (CSU) in 2016. She is currently an Assistant Professor in the Department of Computer Science at the Hang Seng University of Hong Kong. Her research interests lie in federated learning and machine learning for wireless communication.

Zhen Li received his B.Eng. degree from the School of Automation, Guangdong University of Technology, Guangzhou, China, in 2023. From 2023 to 2024, he was a Research Assistant in the Department of Mathematics and Information Technology, The Education University of Hong Kong, HKSAR, China. He is currently pursuing the M.A.Sc. degree with the Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada. His research interests include network optimization, Internet of Things, and machine learning.



Hai Liu is Professor with Department of Computer Science, The Hang Seng University of Hong Kong (HSUHK). Before joining HSUHK, he held several academic posts at University of Ottawa and Hong Kong Baptist University. Prof Liu received PhD in Computer Science at City University of Hong Kong, and received MSc and BSc in Applied Mathematics at South China University of Technology. His research interests include wireless networking, cloud computing, artificial intelligence and algorithm design and analysis.

Chao Yang received the Ph.D. degree in Signal and Information Processing from South China University of Technology, Guangzhou, China, 2013. He currently work in the School of Automation, Guangdong University of Technology. From July, 2014 to July, 2016, he was a Research Associate in the Department of Computing, The Hong Kong Polytechnic University. His research interest focuses on VANETs, edge computing, smart grid.

Tse-Tin Chan (Member, IEEE) received the B.Eng. and Ph.D. degrees in information engineering from The Chinese University of Hong Kong (CUHK), Hong Kong, China, in 2014 and 2020, respectively. From 2020 to 2022, he was an Assistant Professor with the Department of Computer Science, The Hang Seng University of Hong Kong (HSUHK), Hong Kong. He is currently an Assistant Professor with the Department of Mathematics and Information Technology, The Education University of Hong Kong (EdUHK), Hong Kong. His research interests include wireless communications and networking, age of information (AoI), and AI-native wireless systems. He has served as a Guest Editor for the IEEE Open Journal of the Communications Society, a TPC Chair for an IEEE ICC 2026 Workshop, and a TPC member for various flagship conferences, such as IEEE ICC and IEEE GLOBECOM.

Jun Cai received the Ph.D. degree from the University of Waterloo, Canada, in 2004. From June 2004 to April 2006, he was with McMaster University, Canada, as a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellow. From July 2006 to December 2018, he has been with the Department of Electrical and Computer Engineering, University of Manitoba, Canada, where he was a full Professor and the NSERC Industrial Research Chair. Since January 2019, he has joined the Department of Electrical and Computer Engineering, Concordia University, Canada, as a full Professor and the PERFORM Research Chair. His current research interests include digital twin, distributed learning, edge/fog computing, radio resource management, and performance analysis. Dr. Cai served as the General Chair of IEEE BSC 2023; Technical Program Committee (TPC) Co-Chair for IEEE GreenCom 2018; Track/Symposium TPC Co-Chair for the IEEE VTC-Fall 2019, IEEE CCECE 2017, IEEE VTC-Fall 2012, IEEE Globecom 2010, and IWCMC 2008. He also served on the editorial board of IEEE Transactions on Vehicular Technology, IEEE Internet of Things Journal, IEEE Wireless Communications Magazine, and IEEE Access. He was appointed as the IEEE Vehicular Technology Society Distinguished Lecturer in 2024.