Multi-Level Deep Reinforcement Learning-based Edge Caching Strategies in Vehicular Networks

Taban Soleymani
Electrical and Computer Engineering Department
University of Tehran
Tehran, Iran
taban.soleymani@ut.ac.ir

Nasser Yazdani
Electrical and Computer Engineering Department
University of Tehran
Tehran, Iran
yazdani@ut.ac.ir

Seyed Pooya Shariatpanahi
Electrical and Computer Engineering Department
University of Tehran
Tehran, Iran
p.shariatpanahi@ut.ac.ir

***Abstract*— Recent advances in internet of vehicles (IoV) have highlighted the importance of vehicular networks in intelligent transportation systems (ITS). The rapid growth in the number of connected sensors and users requires a better networking scheme. However, the dynamic nature of actual networks poses significant challenges. In addition, content delivery should occur within a specific time frame, known as the delivery deadline, and conventional quality of service (QoS) metrics must be met. In this paper, we implemented cooperative edge caching strategies aimed at reducing the weighted average of energy consumption and content access delay in a vehicular networks while considering the data delivery deadline. By utilizing an abstract layer, along with federated deep reinforcement learning (DRL) and hierarchical DRL, we enhanced the efficiency of the caching scheme. This approach mitigates the challenges posed by the non-stationary nature of file popularity distribution and user behaviors. Our results show that the use of DRL agents in federated and hierarchical structures allows us to develop an effective caching strategy without requiring prior knowledge, such as file popularity distribution or user request patterns while improving cache hit rates, reducing data transmission power and access delay in IoV networks.**

Keywords*— *IoV, federated deep reinforcement learning, hierarchical deep reinforcement learning, vehicular edge computing, content caching, data delivery deadline, multi-agent.

I. INTRODUCTION

Recent advancements in cloud computing and smart city technologies have heightened interest in the internet of vehicles (IoV). Meanwhile, the rapid expansion of IoV networks presents challenges for traditional IP-based networks, particularly regarding addressing, mobility and transmission delay, leading to traffic congestion which reduces quality of service (QoS). According to Ericsson, the volume of data generated by approximately 230 million producer nodes is expected to reach 288 exabytes per month by 2027 [1].

Unlike traditional host-centric services, IoV applications are content-centric, prioritizing the content of data packets over their storage location [2]. Information-centric networks (ICN) technologies, with their in-network caching capabilities, offer a promising solution by caching popular content in road side units (RSUs) or cache-enabled vehicles (CEVs) [3]. This approach allows frequent data requests to be served locally, reducing the need for long data retrieval paths.

Centralized machine learning-based caching approaches [3]-[4], while common, raise privacy concerns due to data aggregation on central servers. To address this, recent research works have focused on distributed approaches using multiple servers to train models. For instance, the authors of [5] proposed a system combining auto-encoders and federated learning to predict content popularity in dynamic vehicle clusters, improving resource efficiency while protecting user privacy.

Despite differences from video services, IoV systems still face storage constraints and deadline-driven content replacement policies. Other studies have explored various approaches, such as leveraging vehicle movement patterns for opportunistic caching [6], using hierarchical edge node architectures to minimize content access costs [7], and applying machine learning techniques like recurrent neural networks (RNNs) and deep Q-networks (DQN) to optimize caching strategies [8]. Moreover, Sun et al. in [9] have integrated edge caching concept with recommender systems to enhance user experience by considering factors like content similarity and data access latency.

In this paper, we introduce a novel approach to content caching in IoV networks by incorporating an abstract layer designed with federated and hierarchical structures. Our approaches consider a weighted sum of transmission power and delay as the primary objective, optimizing network performance across these critical metrics. By leveraging the strengths of federated and hierarchical architectures, our method not only enhances cache efficiency but also significantly increases cache hit rates and the number of fulfilled data requests. This dual focus on reducing energy consumption and minimizing delays, while maximizing content availability, represents a significant advancement in the design of efficient and scalable caching strategies for IoV networks.

The rest of the article is organized as follows. In Section II, we review the related works in recent years. In Section III, firstly we present system model including network model, vehicles' mobility pattern, and communication model. Then, we discuss the state space, action space and reward function. In Section IV, the designs of federated and hierarchical DRL are given. The simulation results are analyzed in Section V. Finally, Section VI concludes this article.

II. RELATED WORKS

The following research work is the first study addressing freshness of data constraint in content replacement decisions. Authors of [2] presented the LFF (Least Fresh First) policy resorting data freshness as a function of future sensor movements through a multihop retrieval model; Thus, data that is expected to expire before the next refresh can be removed. LFF depicted an enhanced replacement desirable according to data freshness than FIFO, RR, LRU and LFU.

Ling et al. in [10] solved the problem of how to predict service popularity in terms of vehicle properties. They proposed a cooperative caching scheme named TCCS (Transportation Correlations-based Caching Strategy), which looks at the traces of vehicles to know what services should be stored on RSUs. Their approach also computed the service

sharing probability among RSUs, which enhanced network delay and hit rate of service acquisition. Their proposed platform used a content replacement policy called DCRS (Dynamic Cache Replacement Strategy), which adjusts to user habits and storage characteristics by replacing services that are requested less often.

The authors of [11] indicated that LSTM (Long Short Term Memory) can predict the popularity of IoT data as a time series problem. In [11], a two-layer LSTM has been used which significantly improved the learning accuracy. While the proposed algorithm works quite well in predicting long-term and short- term popularity, it struggles to initialize when there are not enough historical data available.

Challenges, such as vehicle-to-network communication breakages and the costly price of downloads from a base station over vehicle-to-infrastructure (V2I); are debated in [12]. To overcome these challenges, authors introduced a content caching architecture with edge computing termed as VeSoNet. VeSoNet stores social content in the vehicle that is popular at a given time and delivers context-relevant data based on a pruning search algorithm. The method further optimizes the content distribution by using DRL, and introduces a refinement recommendation called vehicle2vec that uses low-dimensional vectors to represent vehicles as well as previous used contents.

Wu et al. in [13], propose a multi-agent soft actor-critic (SAC) framework for age-of-information-aware item caching, memory cost updating, network traffic reduction, and sensor energy consumption minimization. Unlike many RL methods, this one works with discrete multi-agent systems in two modes: decentralized mode, in which each edge node decides independently of any other node, and centralized mode, where the cloud server coordinates decisions across nodes using a central soft Q function.

In [7], the authors proposed a hierarchical edge node scheme for storing content in IoV networks to reduce long-term access costs. They presented the distributed multi-agent reinforcement learning-based edge caching (DMRE) model and casted content swapping at RSUs as a Markov decision process (MDP). The MDP is then extended to multiple agents and a combinatorial multi-armed bandit algorithm was employed for resolution. For model refinement, they add DQN and DMRE to produce an actually trained taste using the Nash Equilibria estimate neural network.

Authors of [5] investigated an update problem about content popularities and user privacy in IoV networks. In this paper a content popularity prediction approach utilizing autoencoders integrated with federated learning has been suggested for decreasing communication overhead. A complementary storage framework learns a theoretical optimal strategy using DRL.

The problem of inefficient use of cache storage on edge servers and different mobility dynamics in the movement of vehicles is handled in [8]. Authors of [8] introduced a federated deep reinforcement learning-based cooperative storage scheme, with the integration of vehicle-to-vehicle (V2V) and V2I. The method is based on a RNN to predict the connectivity and interactions, and decides which vehicles are right for storage according to their connectivity as well as density, followed by a multi-dimensional content popularity prediction framework. It does so by leveraging features such as popularity history, social interactions, and geographical-based traits. The storage strategy is formulated as MDP, which is trained independently on Q-Networks for each vehicle to gain long-term rewards within a DQN framework and content replacement algorithm loss via federated learning.

The article [6] addressed the problems of changing content popularity, post position for nodes in IoV, and user privacy laws. The proposed approach uses consistent hashing to enable efficient management of cached content for dynamic vehicle clusters. They designed content popularity prediction algorithm based on autoencoders and federated learning for estimating the time-varying content popularity, such that communication overhead is minimized. Also, they designed a cooperative caching system via DRL to learn deployment policies that minimizes the average recommendation delay.

Authors in [9] improved the storage efficiency through edge-storage and recommendation system of cloud computation. Their suggested algorithm supports both soft and direct hits. They evaluated user experience concerning a system cost (e.g., content similarity, access delay, and retrieval costs). Then, they formulated content replacement problem as a multi-agent MDP aiming at minimizing long-term costs. For heterogeneous requests, the applied federated discrete SAC algorithm with a federated learning-based critic to address request diversity.

III. System Model And Problem Formulation

This section is associated with introducing the network model, the timeline of vehicle movements during the caching process, the movement patterns of vehicles, the dataset used for generating requests, and communication model employed to define cost function and rewards.

A. Network Model

To implement the environment for the proposed strategies, similar to [3], we consider three types of storage nodes: MBS, RSUs, and CEVs, each with different processing and storage capacities, denoted as G_i. Each node is identified by an index ($i \in \mathcal{I} = \{0,1, \dots, K, K+1, \dots, K+M\}$). The index (0) represents MBS, while RSUs are identified by the set of components ($\mathcal{K} = \{1, \dots, K\}$). CEVs, which are responsible for V2V caching, are represented by the set of ($\mathcal{M} = \{K+1, \dots, K+M\}$). In addition, the set of vehicles that send content access requests, referred to as requesting vehicles (RVs), is defined by ($\mathcal{N} = \{1, \dots, N\}$).
The set ($\mathcal{F} = \{1, \dots, F\}$) identifies each content, while ($\mathcal{D}_f = \{1, \dots, D_f\}$) defines the set of data items associated with content (f). We assume that MBS provides all data items for each content in ($\mathcal{F}$). By considering the limited storage capacity of RSUs and CEVs, the set of data items stored in their memory is a subset of ($\mathcal{F}$). Accordingly, the MBS node is placed at the center of the map, and RSUs are randomly located using the *geopy* tool in Python.

Since the timescale for changes in content or its popularity is typically larger than the timescale for data delivery (which is often influenced by vehicle movement and channel conditions), our proposed model operates on a dual-timescale framework. A time step is defined to track the vehicles' movement and current locations, and each time step consists

of several time slots denoted as Δt. During each time slot, an RSU or CEV can respond to a maximum of one request, and each vehicle is permitted to send only one request at the beginning of a time step. If an RV has an additional request that remains unfulfilled by the end of a time step, it must defer to the next time step. Furthermore, requests sent during a time step are buffered at each MBS, RSU, and CEV. Since the processing capabilities of these devices vary, their buffer length also differs, and if a buffer becomes full, overflow occurs. Then, the proposed strategy is expected to effectively reduce the number of overflows.

Each data item corresponding to content is defined by three elements $\{\rho_f, \sigma_{d_f}, \tau_{d_f}\}, f \in \mathcal{F}, d \in \mathcal{D}_f$, where ρ_f represents the popularity of content f, σ_{d_f} denotes the size of data d_f, and τ_{d_f} indicates the maximum allowable delay called deadline for accessing data d_f. Additionally, two storage identifiers, $\alpha_{f,i}^t$ and $\beta_{d_f,i}^t$, are used to indicate the status of memory, each taking either the value 0 or 1. Specifically, $\alpha_{f,i}^t = 1$ indicates that content f is stored in memory i during time step t, while $\beta_{d_f,i}^t = 1$ signifies that the requested data d_f is stored in memory i during time step t. Since the stored content in cache memory must be unique, if $\alpha_{f,i}^t = 1$ and $\beta_{d_f,i}^t = 0$, the requested data d_f will replace the currently stored data in cache memory i.

In contrast to existing works like [4], where their proposed caching schemes are designed based on the assumption of content popularity distribution following a Zipf distribution, the content popularity distribution in the utilized dataset is derived from a subset of the *MovieLens 25M* dataset [14]. For each piece of content, there are randomly selected up to 5 data items. This approach of defining data items is aimed to enable the system to support both direct and soft access discussed in [9] and storing data fragments based on user request patterns described in [15].

The rate of new requests sent to access data is modeled as a Poisson distribution with parameter λ. It is also important to note that the regeneration of incomplete requests remains independent of the Poisson process. To provide a realistic model of content popularity, each agent tracks the popularity of the content up to the given time interval, based on its geographical location. This value for specific content is determined by the ratio of the number of requests received for accessing content (f), $\mu_{f,i}^t$, to the total number of requests received by the agent, M_i, i.e. $\rho_f := \frac{\mu_{f,i}^t}{M_i}$

B. Vehicle Movement Pattern

The paths for individual RSUs and CEVs are determined using the method outlined in [16]. The points along each path are stored in a KML file, with the distance between any two consecutive points on the map being 40 meters. Given that each time step is assumed to be 1 second, the speed of each vehicle is randomly selected from the set $\{0, 40, 80\}$. This means that within a single time step, a CEV or RV has three possible actions: it can either remain at the previous point, move to the next point, or advance by two points.

C. Communications Model

We enable the M-PSK modulation scheme for any kind of communication, either among agents or between agents and users. Consequently, the transmission power for each agent node is calculated by (1) [17], in which, N_0 represents the noise spectral density, and β is defined as $\beta = \log_2(M)$. The symbol duration T_s is formulated as $T_s = \frac{\beta \Delta T}{L}$, where h denotes the channel attenuation coefficient and L is the size of individual data packets. For both agent-to-MBS and MBS-to-agent communications, a larger value of h is assumed to encourage content caching by the agents. Moreover, in (1), BEP_{target} refers to the target bit error probability, and Q^{-1} is the inverse of the Q-function known as the error function.

$$P_{TX}(h, BEP_{target}) = \frac{N_0}{2h\beta T_s\left(\sin\left(\frac{\pi}{M}\right)\right)^2}\left[Q^{-1}\left(\frac{\beta}{2}BEP_{target}\right)\right]^2 \quad (1)$$

Using (1) for calculating the consumed power for data transmission, the signal power level can be derived from (2):

$$S = \frac{P_{TX_i}^t \times g_{a_i}}{\xi(\delta_{i,n}(t), w_i^t) \times \delta_{i,n}^{\kappa}(t)} \quad (2)$$

where (g_{a_i}) refers to the gain of each type antenna. In addition, (w_i^t) denotes the allocated bandwidth to storage node (i). The term $(\delta_{i,n})$ represents the distance between the user n and the agent i, while (κ) indicates the channel path loss exponent. To obtain the channel path loss, the free space path loss model is used which is $\xi(\delta, w) = \left(\frac{4\pi\delta w}{c}\right)^2$.

Therefore, to calculate the interference in the communication channel, we proceed as shown in (3):

$$I = \Sigma_{j \neq i} \frac{P_{TX_j}^t \times g_{a_j}}{\xi\left(\delta_{j,n}(t), w_j^t\right) \times \delta_{j,n}^{\kappa}(t)} \quad (3)$$

Now, by using (2) and (3), the signal-to-noise-plus-interference ratio (SNIR) is calculated as $SNIR_{i,n}^t = \frac{S}{I+\sigma^2}$, where σ^2 represents the power of the additive white Gaussian noise (AWGN). Also, the data transmission rate for fetching a fraction of data item can be derived based on Shannon's theory $r_{i,n}^t = w_i^t \log_2(1 + SNIR_{i,n}^t)$.

D. State Space

At the beginning of each time slot, the intelligent agent acquires information about the requested data item. The state of the system is defined by 8 components as follows:

$$s_t = \{[f^t], [d_f^t], [\rho_f^t], \left[SLA_{d_f}^t\right], \left[o_{d_f}^t\right], [l^t], [\alpha_{f,i}^t], \left[\beta_{d_f,i}^t\right]\} \quad (4)$$

Where $SLA_{d_f}^t$ is a service level fulfillment of the requested data item by time interval t, and $o_{d_f}^t$ represents the data delivery deadline calculated by adding τ_{d_f} and data generated time step. Also, l^t indicates the index of the current region where RV is driving.

E. Action Space

After receiving content access requests, the agent calculates the content's popularity and decides whether to store it. $a_t = 0$ means that the requested data item will not be stored, while if the agent selects $a_t = 1$, the requested data item will be stored. In the case of $a_t = 2$, the requested data

item will be updated, and $a_t = 3$ means the data item has been already cached.

F. Reward Function

This function takes into account the following cases:

1) For each of the storage nodes, the transmission power is calculated according to (1).

2) Delay of data access can be derived from (5). Generally, during each time slot, a portion of data with length L is sent from the agent to RV.

$$t_{i,n}^{d} = \frac{L}{r_{i,n}^{t}} \tag{5}$$

3) If the popular content is available in memory, the storage agent receives a reward for caching the requested content, denoted as $reward_{\text{cache hit}} > 0$, which is defined as a fixed positive value.

4) If the request is fully fulfilled, the storage agent will receive the fulfillment reward denoted as $reward_{\text{fulfillment}} > 0$.

So, the reward function can be obtained by (6), where $\lambda_1, \lambda_2 < 0$:

$$r_{total}^{t} = \lambda_1 P_{TX_i}^{t} + \lambda_2 t_{i,n}^{d} + reward_{\text{cache hit}}\mathbb{I}\{\text{cache hit}\} + reward_{\text{fulfillment}}\mathbb{I}\{\text{fulfillment}\} \tag{6}$$

IV. Deep Reinforcement Learning-based Solution for Cooperative Content Caching

The goal of the proposed cooperative caching strategy is to achieve an optimal balance between energy consumption and file access delay. The policy $\pi: S \rightarrow A$ is a stationary policy that maps the current state of the system to a set of actions, such that $a = \pi(s)$. Additionally, Π denotes the set of all stationary policies. Here, the optimal policies at each time step are denoted by π^*, and the state value function is defined as follows in (7) which represents a different form of the Bellman equation:

$$V^*(s) = \max_{a \in A} \{r(s,a) + \gamma Pr(s'|s,a)V^*(s')\} \tag{7}$$

Due to the continuous and high-dimensional nature of the state space, calculations using conventional approaches become very complex and practically infeasible. Consequently, it will not be possible to estimate the state-action value or the state value explicitly. In the proposed algorithm of this paper, model-free actor-critic learning is utilized to provide learning rules based on a large number of past experiences.

As value-based algorithms using deep learning estimators like DQN have been unsuccessful in very simple problems and perform poorly in terms of learning efficiency, to solve the optimization problem, we use the deep deterministic policy gradient (DDPG) algorithm. In fact, DDPG is one of the algorithms within the gradient-based policy methods. Mathematical explanations of the algorithm are discussed further below.

A. Deep Deterministic Policy Gradient

In the deep deterministic policy gradient (DDPG) algorithm, the critic network is responsible for evaluating policies based on the value function of state-action pairs. The values obtained must satisfy the Bellman equation $Q(s,a) = r(s,a) + \gamma \sum_{s' \in S} Pr(s'|s,a)V^*(s')$.

We use a deep neural network-based estimator to approximate the state-action value function and update the parameters using past experiences stored in the replay buffer. Specifically, the state-action function can be approximated using the parameter matrix θ^Q as $Q(s,a|\theta^Q) \approx Q(s,a)$. The mapping function of each two consecutive layers of the critic network is defined as $\theta^{Q^T}\Phi(s,a) = \sum_{h=1}^{H}\sum_{h'=1}^{H'}\left(\theta_{hh'}^{Q}\phi_{mh}(s,a) + \theta_{0h'}^{Q}\right)$. For the m-th training sample, the activation function Φ is defined as $\Phi = \left(\phi_{m1}(s,a), \dots, \phi_{mH}(s,a)\right)$, and θ^Q is the parameter matrix of the neural network. Here, $\left[\theta_{01}^{Q}, \dots, \theta_{0H'}^{Q}\right]$ represents the bias vector, and $\left[\theta_{h1}^{Q}, \dots, \theta_{hH'}^{Q}\right]$ represents the weight vector, with H and H' denoting the number of hidden units in two consecutive layers of the critic network.

Furthermore, we create a replay buffer to store sequences of past experiences $[s,a,r,s']$, which helps to avoid correlations between samples during the learning process. To improve the efficiency of learning algorithm, a random batch of network parameters is selected from the replay buffer during the parameter update process. The size of the replay buffer is set to B, and the temporal difference is determined as $\delta_{TD} = r(s,a) + \gamma Q'\left(s', a'\middle|\theta^{Q'}\right) - Q(s,a|\theta^Q)$, where $Q'\left(.\middle|\theta^{Q'}\right)$ is the target state-action value function derived from the target critic network, and γ is the learning rate, with a defined range of $\gamma \in [0,1]$. After this stage, the parameter θ^Q in the primary critic network is updated using gradient descent on the selected batch as shown in (8):

$$\theta^Q \coloneqq \theta^Q - \eta_{\theta^Q}\delta_{TD}\nabla_{\theta^Q}Q(s,a|\theta^Q) \tag{8}$$

Instead of learning the distribution of the stochastic policy $\pi(a|s)$, the actor network can learn the deterministic policy function $a = \mu(s|\theta^\mu)$ with the parameter vector $\theta^\mu = \left(\theta_0^\mu, \theta_1^\mu, \dots, \theta_n^\mu\right)$. Similar to before, a deep neural network estimator is used to approximate the deterministic policy function, where the mapping function between any two consecutive layers in the actor network is expressed as $\theta^{\mu^T}\Psi(s,a) = \sum_{h=1}^{H}\sum_{h'=1}^{H'}\left(\theta_{hh'}^{\mu}\psi_{mh}(s,a) + \theta_{0h'}^{\mu}\right)$, where Ψ is the activation function, and the gradient of the objective function is determined as $\nabla_{\theta^\mu}Q(s,a|\theta^Q,\theta^\mu) = \frac{\partial\mu(s|\theta^\mu)}{\partial\theta^\mu}\frac{\partial^Q(s,a|\theta^Q)}{\partial a}\Big|_{a=\mu(s|\theta^\mu)}$.

Thus, the local/global values of the state-action value function are obtained using the gradient descent of the selected policy group, and for updating the policy gradient parameters, we have:

$$\theta^\mu \coloneqq \theta^\mu - \eta_{\theta^\mu}\nabla_{\theta^\mu}Q_{\left(s,a|\theta^Q,\theta^\mu\right)} \tag{9}$$

where η_{θ^μ} represents a redefined learning rate.

Instead of using traditional greedy search or sampling from a Gaussian distribution, we utilize the Ornstein-Uhlenbeck noise mechanism for exploration-exploitation balance. Target networks in DDPG algorithm can be considered approximate copies of the primary networks. They should have the same number of layers or hidden units as the primary networks. We employ the exponentially weighted moving average (EWMA) approach to update the parameters $\theta^{Q'}$ and $\theta^{\mu'}$ instead of directly copying

θ^{Q} and θ^{μ}. The update process for the parameters $\theta^{Q'}$ and $\theta^{\mu'}$ is as follows:

$$\theta^{Q'} \leftarrow \tau_{\theta^{Q'}}\theta^{Q'} + \left(1 - \tau_{\theta^{Q'}}\right)\theta^{Q} \quad (10)$$

and

$$\theta^{\mu'} \leftarrow \tau_{\theta^{\mu'}}\theta^{\mu'} + \left(1 - \tau_{\theta^{\mu'}}\right)\theta^{\mu} \quad (11)$$

B. Federated Deep Reinforcement Learning

To implement the Federated DRL algorithm, agents are initially clustered based on geographical location and the pattern of user requests for content access, using the K-means clustering algorithm. Within each cluster, the presence of a RSU as a cluster head is required due to its higher processing capability compared to CEV. In the first step, the local model of DDPG is trained as previously described. After a fixed number of time steps, (n_{round}), the global model of the cluster is updated as a weighted sum of average received rewards by agents. And the weights are calculated based on the confidence interval of average agent rewards in the previous round:

$$\theta_{Q_G} = \frac{\sum_{i=1}^{N} c_i \theta_{Q_i}}{\sum_{i=1}^{N} c_i} \quad (12)$$

As seen in (12), the global model is obtained solely based on the weights of the critic network. This is because if local models are updated according to (13), the policies of the agents will become overly similar to each other. Consequently, the overlap of stored contents in the system increases progressively, which is not desirable. Therefore, the parameters of the primary and target critic networks in each agent are updated according to (12). The parameter (α) is an adjustable parameter that plays a role in maintaining the stability of the designed system. For more details, see Algorithm (Fig. 1).

$$\theta_{Q_i} = (1 - \alpha)\theta_{Q_i} + \alpha\theta_{Q_G} \quad (13)$$

Algorithm 1 Cooperative Edge Caching based on FMADDPG

Initialize global model G.
Initialize local critic network.
Initialize local actor network with random weights θ^{Q_i} and θ^{μ_i} respectively.
Initialize target networks with weights $\theta^{Q_i'} \leftarrow \theta^{Q_i}$ and $\theta^{\mu_i'} \leftarrow \theta^{\mu_i}$.
Initialize local replay buffers R_i for agent i
for episode = $1, \ldots, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_0
  for t = $1, \ldots, T$ **do**
    for each agent i **do**
      Observe state $s_{i,t}$ for agent i
      Select local action $a_{i,t} = \mu_{i,t}(s_{i,t}|\theta^{\mu_i}) + \mathcal{N}_{i,t}$.
      Execute action $a_{i,t}$ and observe reward $r_{i,t}$ and next state $s_{i,t+1}$
      Store transition $(s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1})$ in replay buffer R_i
      Sample a random mini-batch of N transitions $(s_{ij}, a_{ij}, r_{ij}, s_{ij+1})$
      Update local critic by (8).
      Update local actor using the sampled policy gradient (9).
      Update local target based on (10) and (11).
    end for
    if $T \bmod n_{rounds} = 0$ **then**
      Obtain global model by (12)
      Update each local critic based on (13)
    end if
  end for
end for

Figure 1- Cooperative Edge Caching based on FMADDPG

C. Hierarchical Deep Reinforcement Learning

The MBS can be considered as a fully observer agent that manages several partially observer agents and is aware of the overflows and the contents stored in low-level agents.

The state space of the upper layer includes the union of the state spaces of the lower layer and the number of overflows for each agent at the current time slot that has cached the requested content, i.e. $s_t^H = \{[f^t], [d_f^t], [\rho_f^t], [SLA_{d_f}^t], [o_{d_f}^t], [l^t], [\alpha_{f,i}^t], [\beta_{d_f,i}^t], [\{\mathcal{O}_i^t\}]\}$

The action taken by the upper layer determines from which cache storage the data item should be fetched, and this action is combined with the state space of the agent as the goal state: $s_{i,t}^L = \{[f^t], [d_f^t], [\rho_f^t], [SLA_{d_f}^t], [o_{d_f}^t], [l^t], [\alpha_{f,i}^t], [\beta_{d_f,i}^t], [a_t^H]\}$

The reward for the upper-layer agent is determined based on the weighted sum of the delay duration in the destination node and the number of overflows in the destination buffer, $\mathcal{O}_i^t$. The steps of the proposed approach can be observed in Algorithm (Fig. 2).

Algorithm 2 Cooperative Edge Caching based on HMADDPG

Randomly initialize high-level critic and actor networks.
Initialize high-level target networks with weights $\theta^{Q'^H} \leftarrow \theta^{Q^H}$ and $\theta^{\mu'^H} \leftarrow \theta^{\mu^H}$.
Randomly initialize low-level critic and actor networks.
Initialize low-level target networks with weights $\theta^{Q_i'^L} \leftarrow \theta^{Q_i^L}$ and $\theta^{\mu_i'^L} \leftarrow \theta^{\mu_i^L}$.
Initialize high-level replay buffer R^H and low-level replay buffers R_i^L for agent i.
for episode = $1, \ldots, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_0^H
  for t = $1, \ldots, T$ **do**
    Select high-level action $a_t^H = \mu_t^H(s_t^H|\theta^{\mu^H}) + \mathcal{N}_t^H$.
    for each agent i **do**
      Observe state s_{it}^L for agent i
      Select action $a_{i,t}^L = \mu_{i,t}^L(s_{i,t}^L|\theta^{\mu_i^L}) + \mathcal{N}_{i,t}^L$
      Execute action $a_{i,t}^L$ and observe reward $r_{i,t}^L$ and next state $s_{i,t+1}^L$
      Store transition $(s_{i,t}^L, a_{i,t}^L, r_{i,t}^L, s_{i,t+1}^L)$ in replay buffer R_i^L
      Sample a random mini-batch of N transitions $(s_{ij}^L, a_{ij}^L, r_{ij}^L, s_{ij+1}^L)$
      Update low-level critic by (8).
      Update low-level actor using the sampled policy gradient (9).
      Update low-level target based on (10) and (11).
    end for
    Receive reward r_t^H and next state s_{t+1}^H
    Store transition $(s_t^H, a_t^H, r_t^H, s_{t+1}^H)$ in replay buffer R^H
    Sample a random mini-batch of N transitions $(s_j^H, a_j^H, r_j^H, s_{j+1}^H)$
    Update high-level critic by (8).
    Update high-level actor using the sampled policy gradient (9).
    Update high-level target based on (10) and (11).
  end for
end for

Figure 2 - Cooperative Edge Caching based on HMADDPG

V. Evaluation

The proposed approaches are evaluated by comparing them with conventional caching methods (LRU, LFU, LFRU[18], SLRU[19]) and both non-cooperative and cooperative distributed approaches, using various performance metrics.

Table 1 provides the system parameter values and the hyper-parameters for the neural networks used in the RL-based algorithms. The proposed approaches were evaluated based on their effectiveness in terms of average received rewards. Figure 3 shows that the DDPG algorithm generally outperforms proximal policy optimization (PPO) across different architectures, except in non-cooperative settings. The hierarchical structure achieved the highest final convergence value, followed by the distributed and federated structures. The initial peak in the hierarchical structure is due to a high number of overflows early on, which the reward function penalizes. As the system stabilizes, the algorithm converges. In the federated structure, rewards initially drop sharply due to significant variations in request patterns, but these patterns become more realistic over time as vehicles traverse different regions.

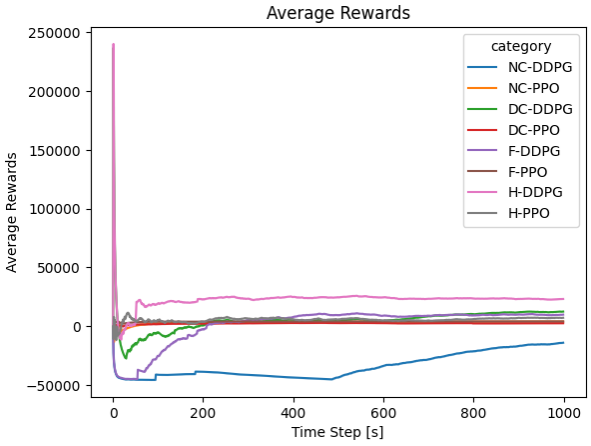
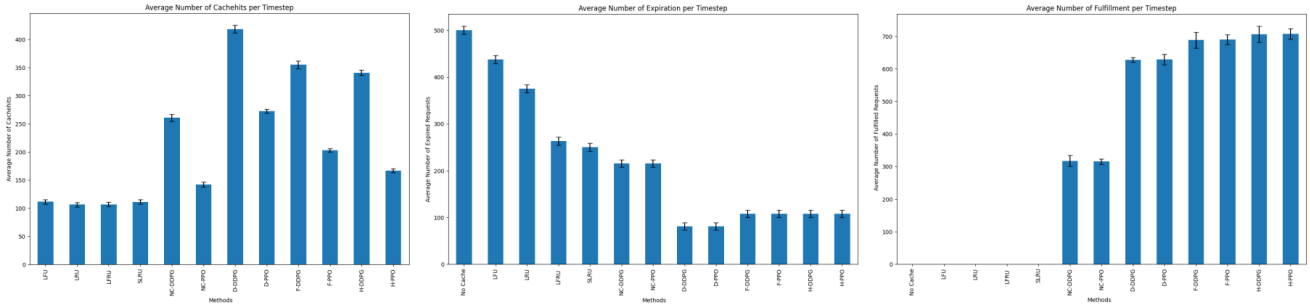


Figure 3 - Average received rewards in DRL-based Approaches

Figure 4(a) shows that the cache hit rates are relatively consistent across conventional content replacement algorithms, with slightly higher rates observed in the federated structure due to node clustering based on request patterns. The distributed approach allows each node to retrieve content from the nearest cache, although full knowledge of content across nodes may pose security concerns. The federated and hierarchical structures have similar cache hit rates, with the federated structure performing slightly better. The distributed algorithms exhibit lower request expiration rates, while the hierarchical structure is more effective in controlling buffer overflows and fulfilling requests, as shown in Fig. 4(b) and Fig. 4(c).

Table 1- Simulation Setup

Hyperparameters	
Parameter	***Value***
N_0	$-174dBm$
$M\ in\ M$ −PSK Modulation	64
BEP_{target}	$9.89e-5$
L	1024 Bytes
ω_i	$\in [5e6,20e6]$ Hz
h	$\in \{-13,-8.4,-5.4,3.2,-1.6,-0.08,1.4,3.1\}$
g_{a_i}	$\in \{5,10,15\}\ dBi$
κ	1
ΔT	$5e-3$
n_{RSU}	18
n_{CEV}	20
n_{RV}	100
$n_{content}$	200
$n_{cluster}$	10
n_{rounds}	25
τ_{d_f}	$\in$ [150,170] s
σ_{d_f}	$\in$ [32,64] MB
p_{ROC}	0.7
b_{RSU}	8
b_{CEV}	4
α (DDPG)	$25e-6$
β (DDPG)	$25e-5$
tau (DDPG)	$1e-3$



(a) cache hits (b) expired requests (c) fulfilled requests

Figure 4 - Cache hits, fulfilled requests, and expired requests for different approaches over the evaluated time intervals

Figure 5 and Figure 6 depict the average energy consumption and delay per time step for different approaches. DRL-based methods generally outperform traditional ones in reducing both metrics. The federated and hierarchical structures yield similar results, both effectively managing request response delays better than other algorithms. The federated structure reduces response times by retrieving content from nearby nodes, while the hierarchical structure minimizes request duplication and delays by incorporating queue delay and buffer overflows into the reward function.

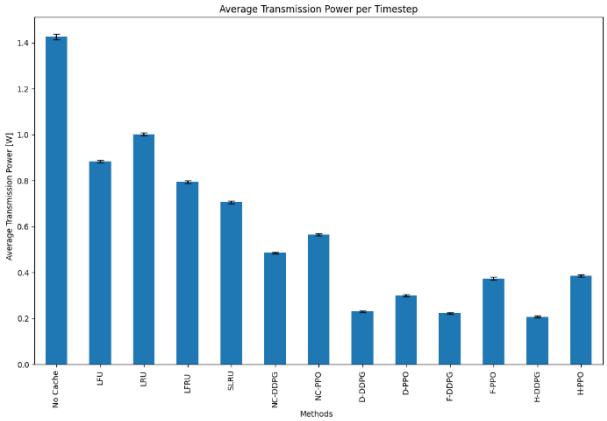


Figure 5 - Analysis of the average transmission power

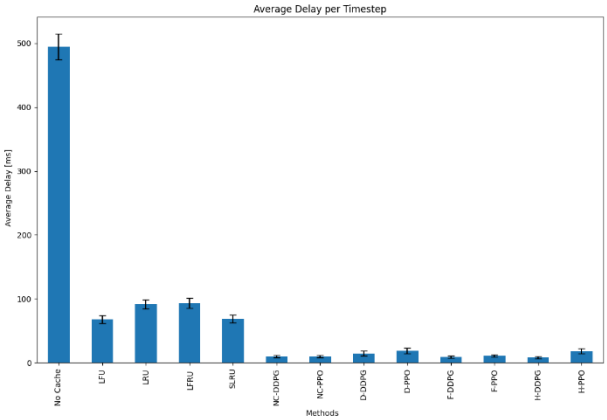


Figure 6 - Analysis of the average content access delay

VI. CONCLUSION

In this paper, we introduced multi-layered federated and hierarchical approaches for edge content caching in IoV networks aimed at enhancing network performance. The proposed methods dynamically manage cache placement and replacement, adapting to frequent changes in user requests by adjusting cache storage for locally and neighboring popular data items, which minimizes redundant storage duplication. Simulation results show that these approaches outperform traditional caching schemes by reducing transmission delay, energy consumption, and improving request fulfillment rates. In recent literature, one of the significant research directions proposed for improving cache storage and data access while protecting users' sensitive information from unauthorized access. by considering the high importance of privacy in vehicular networks, integrating these privacy-preserving approaches with the proposed strategies could help reduce concerns and increase the acceptance of technology among users.

REFERENCES

[1] Ericsson mobility report. [online] available: https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/november-2021

[2] S. Vural, N. Wang, P. Navaratnam, and R. Tafazolli, "Caching transient data in internet content router," *IEEE/ACM Transactions on Networking*, 25(2), pp.1048--1061, October 2017.

[3] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet of Things Journal*, 7(1), pp.247-257, October 2019.

[4] H. Wu, A. Nasehzadeh, and P. Wang, "A deep reinforcement learning-based caching strategy for IoT networks with transient data," *IEEE Transactions on Vehicular Technology*, 71(12), pp.13310-13319, 2022.

[5] H. Wu, J. Jin, H. Ma, and L. Xing, "Federation-based deep reinforcement learning cooperative cache in vehicular edge networks," *IEEE Internet of Things Journal*, 2023.

[6] H. Wu, Y. Fan, J. Jin, H. Ma, and L. Xing, "Social-Aware Decentralized Cooperative Caching for Internet of Vehicles," *IEEE Internet of Things Journal*, 10(16), pp.14834-14845, January 2023.

[7] H. Zhou, K. Jiang, S. He, G. Min, and J. Wu, "Distributed deep multi-agent reinforcement learning for cooperative edge caching in internet-of-vehicles," *IEEE Transactions on Wireless Communications*, 22(12), pp.9595-9609, 2023.

[8] H. Wu, B. Wang, H. Ma, X. Zhang, and L. Xing, "Multi-Agent Federated Deep Reinforcement Learning Based Collaborative Caching Strategy for Vehicular Edge Networks," *IEEE Internet of Things Journal*, April 2024.

[9] C. Sun, X. Li, J. Wen, X. Wang, Z. Han, and V. Leung, "Federated deep reinforcement learning for recommendation-enabled edge caching in mobile edge-cloud computing networks," *IEEE Journal on Selected Areas in Communications*, 41(3), pp.690-705, January 2023.

[10] C. Ling, W. Zhang, Q. Fan, Z. Feng, J. Wang, R. Yadav, and D. Wang, "Cooperative Service Caching in Vehicular Edge Computing Networks Based on Transportation Correlation Analysis," *IEEE Internet of Things Journal*, 2024.

[11] B. Chen, L. Liu, M. Sun, and H. Ma, "IoTCache: Toward data-driven network caching for Internet of Things," *IEEE Internet of Things journal*, 6(6), pp.10064--10076, 2019

[12] N. Aung, S. Dhelim, L. Chen, A. Lakas, W. Zhang, H. Ning, S. Chaib, and M. T. Kechadi, "VeSoNet: Traffic-aware content caching for vehicular social networks using deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 24(8), pp.8638-8649, 2023.

[13] X. Wu, X. Li, C. Jun, L. PC, CM. Victor, P. Victor, and V. H, "Caching transient content for IoT sensing: Multi-agent soft actor-critic," *IEEE Transactions on Communications*, 69(9), pp.5886-5901, 2021.

[14] F.M. Harper, and J.A. Konstan, "The movielens datasets: History and context," *ACM transactions on interactive intelligent systems (tiis)*, 5(4), pp.1-19, December 2015.

[15] Y. Liu, T. Zhi, H. Xi, X. Duan, and H. Zhang, "A novel content popularity prediction algorithm based on auto regressive model in information-centric IoT," *IEEE Access*, 7, pp.27555-27564, 2019.

[16] I.A. Alablani, and M.A. Arafah, "A new vehicle dataset in the city of Los Angeles for V2X and machine learning applications," *Applied Sciences*, 2(8), p.3751, April 2022.

[17] N. Sharma, N. Mastronarde, and J. Chakareski, "Accelerated structure-aware reinforcement learning for delay-sensitive energy harvesting wireless sensors," *IEEE Transactions on Signal Processing*, 68, pp.1409-1424, February 2020.

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

[19] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Transactions on Knowledge and data Engineering*, 11(1), pp.94-107, 1999.