A Reinforcement Learning and Recommendation-Based Approach to Content Caching Optimization in Vehicular Networks

by

Zhenhuan Cui

A thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
Master of Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Edge caching for the Internet of Vehicles (IoVs) has emerged as one of the most active research areas in computer networks, aiming to alleviate network congestion and reduce latency by deploying content caching and computational capabilities at Roadside Units (RSU). Given the high mobility of vehicles and the diversity of content requests, RSUs require caching strategies that ensure both high accuracy and content diversity. RSUs naturally act as in-network caches consistent with the Information-Centric Networking (ICN) paradigm.

To address these challenges, we propose a proactive caching strategy named (**GCN**) -based **R**ecommender System and **D**eep **R**einforcement **L**earning **C**aching (GCNR-DRL). This approach integrates a GCN-based recommender system with deep reinforcement learning to predict vehicles' content demands and dynamically optimize caching decisions. The reinforcement learning agent captures the complex spatial-temporal dynamics of vehicular networks and the intricate relationships between vehicles and content items, enabling precise cache replacement actions that significantly reduce transmission delays and mitigate network congestion during peak traffic hours.

Furthermore, we extend our work by introducing an innovative method that hierarchically learns recommendation and caching strategies using an Actor-Critic reinforcement learning framework. This method couples a cache controller with a per-vehicle recommender. By jointly optimizing recommendation and caching decisions, the HiCaRe-RL approach further improves cache hit ratios, reduces latency, and lowers link load compared to existing strategies.

By combining predictive power with adaptive caching decisions, our methods effectively enhance content dissemination efficiency in highly dynamic vehicular environments. The ability of GNN-based models to learn latent representations of vehicular interactions, coupled with reinforcement learning's adaptability, makes these approaches highly suitable for real-world IoV scenarios.

Comprehensive experimental results demonstrate that both GCN-based Recommender System and Deep Reinforcement Learning Caching (GCNRDRL) and the HiCaRe-RL method outperform traditional caching strategies. Our investigations underscore the potential of leveraging advanced GNN-based models and reinforcement learning techniques to drive efficient content placement and network resource management in dynamic IoV environments.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Amiya Nayak, for his invaluable guidance, advice, and support throughout my master's study. I am also grateful to my senior colleague, Dr. Jiacheng Hou and Haolong Zhang, for their insightful advice and support during my thesis work. I would like to thank my family as well, who continuously support me both physically and mentally. Finally, I wish to honor the memory of my late grandmother, Heying Cao, whose loving care and encouragement from my earliest days have shaped who I am today.

Contents

List of Tables

List of Figures

Acronyms

AC Actor-Critic.

ARIMA Auto Regressive Integrated Moving Average.

CNN Convolutional Neural Network.

DCGRU Diffusion Convolutional Gated Recurrent.

DL Deep Learning.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

DSAC Discrete Soft Actor-Critic.

DT Decision Tree.

DTPRO Deep Q-Network and Traffic Prediction based Routing Optimization.

FIFO First-In-First-Out.

GAT Graph Attentional Network.

GCN Graph Convolutional Network.

GCNRDRL GCN-based Recommender System and Deep Reinforcement Learning Caching.

GGAE Gated Graph Auto Encoder Network.

GNN Graph Neural Network.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

HA Historical Average.

ICN Information-Centric Networking.

IoT Internet of Things.

IoV Internet of Vehicles.

KDN Knowledge-Defined Networking.

KP Knowledge Plane.

KPI Key Performance Indicators.

LightGCN Light Graph Convolutional Network.

LR linear regression.

LRU Least Recently Used.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

MDP Markov Decision Process.

MEC Mobile Edge Computing.

ML Machine Learning.

MLP Multi-layered Perceptron.

MPNN Message Passing Neural Networks.

MRE Mean Relative Error.

MRU Most Recently Used.

MSE Mean Squared Error.

NB Naive Bayes.

NC Nearest Centroid.

NDN Named Data Networking.

NN Neural Network.

NOS Networking Operating System.

OD Origin-Destination.

OVS open virtual switch.

QoE Quality of Experience.

QoS Quality of Service.

RL Reinforcement Learning.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

RS Recommender System.

RSU Roadside Units.

SAC Soft Actor-Critic.

SAE Stacked Auto-Encoder.

SDN Software-Defined Networking.

SFS Sequential Feature Selection.

SHAP Shapley additive explanation.

SPF shortest path first.

SVM Support Vector Machine.

TnC Traffic and Capacity.

TnCwD Traffic and Capacity with Delays.

TO Traffic Only.

TPU Tensor Processing Unit.

V2I Vehicle to infrastructure.

WMAPE Weighted Mean Absolute Percentage Error.

XGBoost eXtreme Gradient Boosting.

Chapter 1

Introduction

1.1 Motivation

In IoV, robust communication between vehicles and Roadside Units (RSU) is critical. RSUs act as highly configurable edge nodes that dynamically manage caching and recommendation services based on real-time data from vehicles, such as location, speed, and user demand. This dynamic interaction is essential for adapting to varying traffic conditions and optimizing resource allocation.

The rapid development of the IoV has led to an exponential increase in data traffic, placing immense pressure on network infrastructure. IoV enables vehicles to exchange information with RSU and cloud servers, supporting applications such as autonomous driving, intelligent traffic management, and infotainment systems. However, the high mobility of vehicles and the massive volume of data being transmitted in real-time pose significant challenges in ensuring low-latency, high-reliability communication.

IoV applications demand ultra-low latency for real-time decision-making, particularly in safety-critical scenarios such as collision avoidance, emergency braking, and cooperative driving. Even in infotainment systems, long buffering times and high response delays degrade user experience. However, conventional cloud-based content retrieval often suffers from long backhaul delays, especially during peak hours. This

necessitates intelligent caching strategies at the edge to ensure fast content retrieval and seamless connectivity for vehicles.

Modeling the complex spatial and temporal relationships in IoV is challenging. By leveraging GNNs, the system can represent the network as a graph with RSU as central node and vehicles as peripheral nodes, capturing both individual node features and inter-node dependencies. This approach provides a more detailed state representation than simple averaging. This thesis applies GNN to solving caching difficulties and focuses on the following two main aspects:

1. Applying GNNs to model the architecture of the vehicular network to capture spatial dependencies among neighboring nodes for cooperative caching decisions considering different node positions and characteristics.

2. Utilizing GNNs to model a rating matrix between users and content, such as movies, to predict accurate ratings from users to unviewed content.

One promising solution to mitigate these challenges is edge caching, which involves storing frequently requested content at the network edge, such as in RSU, to reduce redundant transmissions and improve access speed. By bringing content closer to vehicles, edge caching can significantly reduce network congestion and transmission delays. However, designing an optimal caching strategy is non-trivial due to the dynamic nature of vehicle mobility and the diversity of content requests. Traditional caching methods, which rely on simple heuristics like Least Recently Used (LRU), fail to adapt to changing request patterns, leading to suboptimal caching performance.

To achieve adaptive caching strategies, DRL is employed to learn optimal actions through sequential decision-making. The DRL agent interacts with the environment to identify which caching actions yield the highest long-term Quality of Experience (QoE), adjusting its policy based on real-time reward feedback. This continuous learning enables the system to respond effectively to dynamic conditions.

By preloading and storing popular content at RSU, edge caching minimizes the reliance on distant cloud servers, reducing transmission delays and bandwidth consumption. This proactive caching mechanism is particularly beneficial in vehicular networks, where intermittent connectivity and dynamic topology make it difficult for vehicles to continuously access centralized servers. However, the fundamental challenge lies in predicting which content should be cached to maximize the cache hit ratio while adapting to dynamic vehicular environments.

Meanwhile, a Recommender System (RS) can significantly enhance caching efficiency by predicting which content will likely be requested next. Unlike traditional caching techniques that rely solely on past frequency statistics, RS-based caching leverages user preferences, contextual information, and social relationships to make informed caching decisions. This personalized caching strategy ensures that highly relevant content is proactively stored at RSUs, improving cache hit rates and reducing content retrieval time.

GNN-based recommendation models further enhance this approach by capturing complex user-item interactions and higher-order relationships among vehicles, content, and contextual factors. Integrating GNNs into edge caching allows for a more adaptive and personalized content placement strategy, ensuring that the cached content aligns with evolving user preferences.

While RS improve caching by predicting user preferences, they do not inherently account for dynamic network conditions, cache capacity constraints, or vehicle mobility patterns. RL, particularly DRL, is a natural fit for edge caching due to its ability to optimize long-term rewards through continuous interaction with the environment.

RL-based caching models learn adaptive policies that dynamically update cache contents based on real-time observations, such as content popularity trends, vehicle movement patterns, and network congestion levels. Unlike static caching approaches, RL continuously refines caching decisions to maximize cache hit rates and minimize transmission delays.

Given the challenges associated with vehicular mobility, dynamic content demands, and low-latency requirements, this research aims to develop an adaptive and intelligent caching framework based on recommender systems and DRL By integrating GNN-powered recommendation with RL-driven cache optimization, this research seeks to enhance cache hit rates, minimize transmission delays, and improve overall network efficiency in IoV scenarios.

In addition to the traditional IP-based V2I communication model, vehicular networks are conceptually aligned with the principles of ICN. ICN shifts the network paradigm from host-centric communication to content-centric access, allowing named data objects to be cached and retrieved directly within the network. In ICN architectures such as Named Data Networking (NDN), intermediate nodes maintain in-network caches that can serve popular content without contacting the original publisher, thereby reducing end-to-end latency and backhaul load. Roadside Units naturally inherit this role in the IoV ecosystem, functioning as distributed edge caches positioned along vehicular routes.

However, while ICN provides the architectural foundation for in-network caching, it does not specify how limited cache space should be allocated under highly dynamic vehicular mobility, shifting content popularity, and heterogeneous user preferences. This motivates the need for intelligent, learning-driven caching strategies capable of predicting future content demand and proactively managing RSU caches. In this thesis, we leverage recommender systems and reinforcement learning to design adaptive caching policies that align with the ICN paradigm while addressing the unique challenges of the IoV environment.

1.2 Objective and Contribution

Motivated by the ICN paradigm where RSUs act as in-network caches, our objective is to design learning-driven caching strategies that proactively place content at RSUs

based on predicted demand.

In IoV, vehicles frequently request diverse content while moving through highly dynamic environments. To improve caching performance, recent research has explored RS and DRL as potential solutions. However, existing approaches either fail to capture the complex relationships between content and users or lack adaptability to real-time changes in vehicle mobility and network conditions. A more integrated and learning-driven approach is necessary to address these limitations.

To tackle these challenges, this thesis proposes two key contributions: first, a Graph Convolutional Network (GCN)-based RL framework that enhances caching decisions by leveraging structured content-user relationships; second, a hierarchical recommender and caching policy, optimized using actor-critic structure, to enable hierarchical learning of content prediction and cache management.

Traditional caching methods rely on heuristic approaches that fail to generalize in dynamic environments. Meanwhile, recent advancements in GNN-based RS have demonstrated their ability to capture complex relationships between users and content. In IoV, where vehicle mobility and content preferences exhibit strong spatial-temporal correlations, a GNN-based RS can effectively predict content demand. However, static recommendation alone is insufficient for efficient caching, as cache management requires dynamic adaptation to changing network conditions.

Integrating hierarchical structure into the caching framework enables simultaneous learning of RS and caching strategies in a unified manner. Actor and critic structure improves over traditional RL methods by enabling discrete action spaces and more stable learning, ensuring that caching decisions remain aligned with the evolving RS model. By hierarchically optimizing both components, the RS continuously refines content predictions, while the RL agent learns caching strategies that maximize long-term cache hit rates and minimize latency.

The first approach consists of two key components. First, a pretrained GCN-based RS predicts vehicles' future content demands based on historical access patterns, vehi-

cle mobility, and spatial-temporal correlations. The RS is trained on past interaction data to generate content preference scores for each vehicle. Second, a GCN-based RL caching agent optimizes cache management by learning an adaptive caching policy. The RL agent considers key factors when making caching decisions, including RS scores, vehicle positions, mobility patterns, available bandwidth, and the current RSU cache state. By continuously updating its policy based on real-time content requests, cache hit/miss statistics, and observed network conditions, the agent formulates caching as a Markov Decision Process (MDP), where actions correspond to cache replacement decisions, rewards reflect cache hit improvements, and states capture network dynamics.

By leveraging the synergy between GNN-based RS and RL-driven caching, this approach represents a significant advancement in intelligent edge caching for IoV networks. The combination of predictive modeling with adaptive caching decisions ensures efficient content dissemination in highly dynamic IoV environments. Additionally, these methods have the potential to extend beyond vehicular caching, offering applications in Mobile Edge Computing (MEC), and future intelligent transport systems.

To further enhance caching performance, we introduce a hierarchical-based method that unifies RS and caching optimization in a level-based RL framework. The RL agent dynamically adjusts cache contents by selecting the optimal content mix based on real-time traffic and RS predictions. By operating in a discrete action space, hierarchical allows finer control over caching policies. The reward function is designed to optimize hit ratio.

In this thesis, we use the GCN-based framework to optimize caching decisions in IoV environments by integrating recommender systems and reinforcement learning. We evaluate our approach in dynamic vehicular networks and compare its performance against traditional caching strategies. The contributions of this thesis are as follows:

- We propose a GNN-based caching strategy that integrates a pretrained recommender system with a reinforcement learning agent to dynamically manage cache content at RSU.
- We introduce a hierarchical-based framework to learn both recommendation and caching strategies, ensuring consistency between predicted content demands and cache placement decisions.
- We evaluate our proposed method on real-world mobility datasets and compare its performance against traditional caching policies, including Least Recently Used (LRU) and First-In-First-Out (FIFO), Most Recently Used (MRU), Random, and 2Q.
- We incorporate a learning-based DSAC baseline[5] for comparison and analyze its limitations under highly dynamic IoV conditions, highlighting the need for hierarchical adaptation.

Our results demonstrate that the GNN-based approach significantly outperforms traditional caching strategies, achieving up to **93%** improvement in cache hit ratio and **10%** reduction in latency. The hierarchical approach outperforms traditional caching strategies by achieving average **96%** improvement in cache hit ratio and **19%** reduction in latency. After testing multiple seeds with different network conditions and vehicular mobility patterns, our findings highlight the necessity of incorporating spatial-temporal correlations in caching strategies and validate the effectiveness of leveraging GCN for intelligent edge caching in IoV environments.

1.3 Thesis Organization

The remainder of this thesis is organized as follows.

- Chapter 2 provides an overview of fundamental concepts relevant to this research, including IoV, Recommender Systems, Reinforcement Learning, GNN, and traditional caching strategies such as LRU.

- Chapter 3 reviews related work on edge caching in IoV, highlighting existing machine learning and reinforcement learning-based approaches. This includes prior applications of GNNs, Deep Q-Networks (DQN), Actor-Critic methods, and discrete action space optimization techniques for caching.

- Chapter 4 introduces the proposed GNN-based Reinforcement Learning caching strategy (GCNRDRL). This chapter details the system architecture, the integration of a pretrained recommender system, and the reinforcement learning agent responsible for cache decision-making. A paper from this chapter (Z. Cui, J. Hou, A. Nayak, *"A GCN and Recommender Based Approach for Optimizing Edge Caching Performance in IoV"*) has been accepted for publication at 2025 IEEE Globecom Conference.

- Chapter 5 presents the hierarchical-based caching strategy, which optimizes caching policies using Actor-Critic MLP structure. This chapter discusses the technical implementation, including state representation, discrete action space formulation, and reward function design for caching in dynamic vehicular environments.

- Chapter 6 concludes this thesis with a summary of key findings and discusses potential future directions for improving intelligent caching in IoVs.

Chapter 2

Background

2.1 IoV

IoV relies on the strong communication between vehicles and RSU, where RSU acts as highly configurable edge nodes. These RSU dynamically manages data services such as caching and content recommendation based on real-time vehicle information.

In this architecture, RSU gathers data—like vehicle location, speed, and user demand—from connected vehicles and adjust their configurations accordingly. This flexibility enables RSU to optimize resource allocation and service delivery, ensuring that network policies meet the varying traffic conditions and user requirements.

The robust link between vehicles and RSU supports rapid updates to caching strategies and network settings. By leveraging RSU configurability, the system can adapt to changing environments and maintain high QoE for end users.

2.2 Neural Network Fundamentals (for MLP and GNN)

Neural Network is a machine learning technique that uses artificial neurons to simulate the way how biological neurons in the human brain work.

Neural networks are composed of differentiable layers of linear transformations followed by nonlinear activation functions. Each layer maps an input vector to a new representation using learnable weights and bias terms, and the parameters are optimized using gradient-based training.

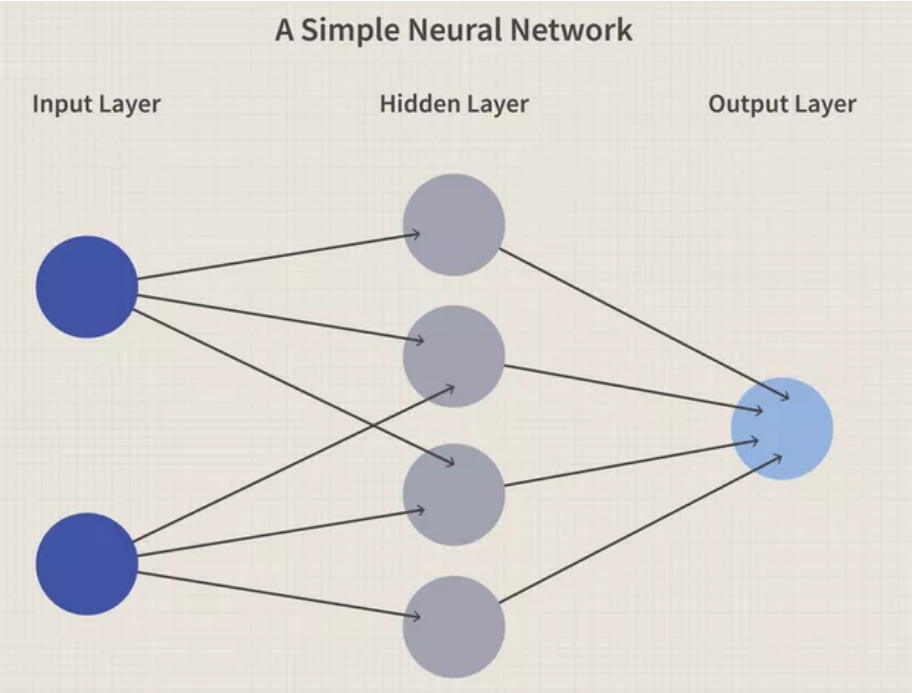


Figure 2.1: A Simple Neural Network with Layers[27]

In Figure 2.1, it shows a sample of a simple neural network. In this MLP, perceptrons are arranged in interconnected layers. The input layer is responsible for collecting input patterns. The output layer contains classifications or output signals that may be mapped to by input patterns. In the middle, there is a hidden layer. Hidden layers adjust the input weightings until the neural network's margin of error

is as little as possible by experiments.

Neural networks form the foundation of the deep learning models used in this thesis. Although standalone fully-connected networks are not used as the primary prediction models, their principles directly support two key components:

- the MLP architectures employed by the actor and critic networks in the reinforcement learning framework, and
- the learnable message-update and aggregation functions in GNN

Thus, the core ideas introduced in this section provide essential background for understanding the deep learning architectures developed in later chapters.

2.3 Graph Neural Network

GNNs[41, 53, 62] have become a powerful tool for analyzing and learning structured graph data. In contrast to other popular neural network architectures such as Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), or Transformers, which operate on grid-like or sequential data, GNNs excel in handling complex relational data represented as graphs.

The most fundamental part of GNN is a Graph. In computer science, a graph is a data structure consisting of two components: nodes (vertices) and edges. A graph G can be defined as G = (V, E), where V is the set of nodes, and E are the edges between them.

To represent graphs, we can use adjacency matrix:

$$W_{i,j} = \begin{cases} 1, \ i \ and \ j \ share \ a \ link, \\ 0, \ otherwise \end{cases} \tag{2.1}$$

where $W_{i,j}$ is the edge value from node i to j. If a graph has n nodes, adjacency matrix has a dimension of (n $\times$ n).

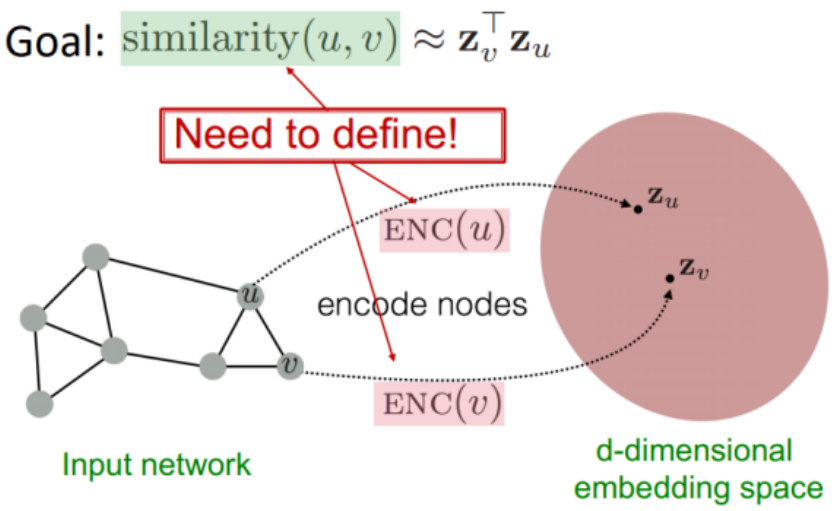


Figure 2.2: Basics of Deep Learning for graphs [31]

Recent advances in GNN research have further expanded their capabilities by introducing novel architectures, including convolutional-based GNNs [28], attention-based GNNs [47], transformer-based GNNs [58], and federated-based GNNs [33]. These advances have improved the applicability of GNNs in various applications, such as molecular classification of cancer [32], intelligent transportation systems [39], etc. In GNN, our goal is to map nodes, so that similarity in the embedding space approximates similarity in the network. Let's define u and v as two nodes in a graph; x_u and x_v are two feature vectors. Now we'll define the encoder function $Enc(u)$ and $Enc(v)$, which convert the feature vectors to z_u and z_v. The whole process is shown in Figure 2.2 from [31].

To find the encoder function, we need locality (local network neighborhoods) information, aggregate information, and stacking multiple layers (computation) information. Locality information can be achieved by using a computational graph. After that, we start aggregating by using neural networks.

In Figure 2.3 from [31], Neural Networks are presented in light and dark grey boxes. The required aggregations should be order-invariant, like sum, average, and maximum, as the result of permutation-invariant functions. Aggregations can be done

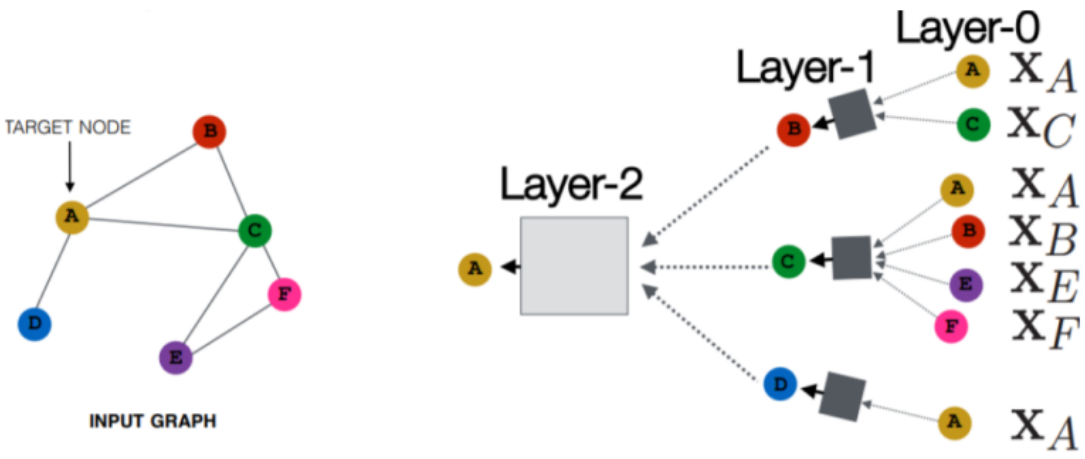


Figure 2.3: Neural Networks with layers [31]

prior to this attribute.

Based on the forward propagation rule, the information is transferred from layer to layer. For example, in Figure 2.3, every node has a feature vector. In Layer-0, the inputs are three groups of feature vectors, and the box will take the three groups of feature vectors, aggregate them, and then pass them on to the next layer.

After performing forward propagation in the computational graph layer by layer, we can train the model by defining a loss function on the embeddingS. There are two types of training:

- Unsupervised Training:

  Use simply the graph structure; nodes with comparable embeddings have similar embeddings. An unsupervised loss function can be a loss based on graph node proximity or random walks.

- Supervised Training:

  Train model for a supervised task such as node classification, normal or anomalous node.

The loss function measures the difference between the predicted output and the outcome generated by the machine learning model. We can obtain the gradients that

are utilized to update the weights and parameters from the loss function to get better training performance.

2.4 Reinforcement Learning

RL[45] is a branch of machine learning where agents learn to make sequential decisions by interacting with an environment. At each step, the agent receives observations and chooses actions in order to maximize cumulative rewards over time. The core idea is learning through trial and error, guided by feedback in the form of rewards or penalties. There are several key approaches to RL, including value-based methods such as Q-learning[50], online bandit[14], and some DRL approaches will be introduced in the following section.

In RL, an agent interacts continuously with an environment, aiming to learn optimal decisions through trial-and-error interactions. The environment is the context or system within which the agent operates and provides feedback based on the agent's decisions. At each step, the environment is described by a state, which represents the current situation or configuration perceived by the agent. The agent chooses an action, influencing the subsequent state and receiving feedback in the form of a reward, a scalar signal that guides the agent toward desirable behaviors. The decision-making process of the agent is guided by a policy, which defines the strategy or rule for choosing actions given states. To evaluate states, the agent estimates a state value, indicating the expected cumulative reward when starting from that particular state and following a given policy. Similarly, the Q-value (or action-value) represents the expected cumulative reward of performing a specific action in a particular state and then continuing according to the policy.

Besides the fundamental concepts discussed above, reinforcement learning requires additional mechanisms to ensure correct learning and maximize cumulative rewards. Like many recursive problem-solving methods, reinforcement learning relies on the

Markov Decision Process (MDP), meaning the next state depends only on the current state and the action chosen by the agent, independent of previous states. Based on the structure of MDPs, researchers[3] have derived the Bellman equation:

$$V(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V(s') \right] \tag{2.2}$$

$V(s)$ stands for the value of state s, representing the expected cumulative reward when starting from state s and following the optimal policy thereafter. a is an action chosen by the agent from state s. $R(s,a)$: The immediate reward received after executing action a in state s. γ is the discount factor ($0 \leq \gamma \leq 1$), determining how much future rewards are valued compared to immediate rewards. $P(s' \mid s,a)$ accounts for the transition probability, representing the probability of transitioning to the next state s' given current state s and action a.

An important concept not yet introduced is the discount factor γ, a crucial component in reinforcement learning. The discount factor balances immediate rewards and long-term rewards, specifying whether the agent should prioritize short-term benefits or aim for strategies that yield higher cumulative rewards over time. Formally, γ is a floating-point number between 0 and 1; higher values of γ mean that future rewards are considered more significant, guiding the agent towards long-term strategies, and vice versa.

Besides the choice of γ, obtaining rewards remains one of the core objectives in reinforcement learning. To understand which behaviors lead to the highest possible rewards, the agent must sufficiently explore different actions to grasp the environment's dynamics fully. Conversely, to maximize rewards, the agent should exploit actions known to yield the best outcomes. This fundamental tension between exploration and exploitation is commonly known as the *exploration-exploitation dilemma*. Researchers have developed several approaches to address this dilemma, such as:

- Setting an initial exploration rate ϵ[7] (epsilon-greedy approach), where the

agent randomly selects actions with probability ϵ to maintain exploration and selects the optimal known actions otherwise.

- Computing probabilities for all actions and selecting them probabilistically using a softmax function[2], which transforms action-values into a probability distribution favoring actions with higher expected rewards.

- Employing an entropy coefficient[12], where entropy regularization is introduced to encourage a more diverse selection of actions, helping the agent avoid prematurely converging to suboptimal policies.

Many reinforcement learning methods have already demonstrated effectiveness in various scenarios, such as the previously mentioned Q-learning and online bandit approaches. Researchers have further extended these methods, leading to techniques like Double Q-learning, Monte Carlo methods, and Temporal Difference (TD) learning.

2.4.1 Online Bandit

Online bandit methods address decision-making problems where an agent repeatedly selects actions (often called "arms") to maximize cumulative expected rewards. Unlike general reinforcement learning setups, online bandit problems assume no state transitions—each action immediately results in a reward drawn from an unknown probability distribution. The primary goal of bandit algorithms is to maximize cumulative rewards, or equivalently, minimize cumulative regret, defined as the difference between obtained rewards and the optimal possible rewards.

Formally, bandit algorithms aim to maximize the expected cumulative reward:

$$\max \mathbb{E}\left[\sum_{t=1}^{T} r_{a_t}\right] \tag{2.3}$$

where r_{a_t} is the reward obtained by choosing action a_t at time t, and T is the total number of steps.

Since bandit problems do not involve state transitions, the classical Bellman equation does not directly apply. Instead, the value estimate (expected reward) for each action a is updated iteratively using a simpler rule, typically:

$$Q(a) \leftarrow Q(a) + \alpha\left[r - Q(a)\right] \tag{2.4}$$

Online bandit algorithms are most suitable for scenarios with no explicit state dynamics, such as recommendation systems, A/B testing, or online advertising, where the environment is relatively static or stationary. Their primary limitation lies in their inability to handle state transitions or delayed rewards effectively. Consequently, online bandit methods struggle to manage problems involving long-term consequences or sequential decision-making complexities, restricting their use to simpler contexts.

2.4.2 Q-learning

Q-learning is one of the most widely-used model-free reinforcement learning algorithms designed to handle decision-making problems involving sequential state transitions. Unlike online bandit methods, Q-learning explicitly deals with state-action pairs and aims to find the optimal policy by maximizing cumulative discounted rewards over time. Specifically, the goal of Q-learning is to learn an optimal action-value function, often referred to as the Q-value, which indicates the expected cumulative reward starting from a particular state-action pair and following the optimal policy thereafter.

Formally, Q-learning aims to maximize the expected cumulative discounted reward:

$$\max \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{2.5}$$

where r_t is the reward obtained at time step t, and $\gamma \in [0, 1]$ is the discount factor determining the importance of future rewards.

The Q-learning algorithm updates the Q-value estimates iteratively based on the **Bellman equation**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.6}$$

where α is the learning rate, controlling the magnitude of the Q-value update. s' stands for the resulting state after performing action a from state s. $\max_{a'} Q(s',a')$ isthe maximum Q-value achievable from the next state s', reflecting the best-known future action.

Q-learning performs effectively in environments characterized by discrete states and actions, especially when environment dynamics (transitions and rewards) are unknown or difficult to model explicitly. However, the main limitation of classical Q-learning lies in handling environments with large or continuous state-action spaces, which can lead to the "curse of dimensionality," necessitating advanced variants such as Deep Q-Networks (DQN) or function approximation methods which will be introduced in next section.

2.4.3 Temporal Difference (TD) Learning

TD learning is a fundamental approach in reinforcement learning, combining elements from both Monte Carlo methods and dynamic programming. Unlike Monte Carlo methods, which require complete episodes to update value estimates, TD learning updates predictions incrementally at each time step using current estimates, enabling it to learn more efficiently in ongoing tasks. The primary objective of TD learning is to estimate the value function by iteratively adjusting predictions based on differences between temporally successive value estimates, known as the *temporal difference error*.

Formally, TD methods aim to approximate the state-value function $V(s)$, representing the expected cumulative discounted reward from state s, defined as:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right] \tag{2.7}$$

The basic update rule for TD learning, commonly known as the **TD(0)** algorithm, is derived from the Bellman equation and defined by:

$$V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right] \tag{2.8}$$

where The term $r + \gamma V(s') - V(s)$ is known as the *temporal difference error*, indicating how different the current prediction is from the updated estimate based on the observed reward and the next state's value.

TD methods are especially suitable for environments with continuous interactions, where learning must happen incrementally without waiting for complete episodes. They are efficient and capable of adapting rapidly to changes in the environment. However, TD learning has limitations, such as sensitivity to parameter choices like the learning rate α and the discount factor γ. Additionally, TD(0) may suffer from bias, as updates rely on estimates that might initially be inaccurate, requiring careful tuning or combining with other methods, like eligibility traces (e.g., TD(λ)), to enhance stability and convergence.

Deep reinforcement learning provides powerful methods allowing agents to learn optimal strategies through interactions with their environment. Fundamental techniques, such as Q-learning, TD learning, and Online Bandit, each offer unique strengths tailored to different problem scenarios and assumptions. While Q-learning effectively addresses sequential decision-making tasks through state-action value estimation, TD learning balances computational efficiency and continuous value updates. Online Bandit methods provide simplicity and efficiency in scenarios lacking explicit state transitions. Together, these approaches and their extensions offer flexible and robust tools for solving diverse sequential decision-making problems

2.5 Deep Reinforcement Learning

DRL[1] has emerged as a powerful paradigm within artificial intelligence, combining deep learning and reinforcement learning techniques. Specifically, DRL is a type of online learning, where instead of collecting data in advance and training a neural network, then deploying the trained neural network to the environment, DRL employs an online agent that interacts with the environment continuously. The agent is trained using the data collected through interaction with the environment. Additionally, DRL masters sequential decision-making tasks. At each step, the DRL agent receives a state from the environment, takes an action based on the received state, and receives a reward from the environment indicating the quality of the action taken. Afterward, the environment transitions to a new state, and the process repeats. The ultimate objective of DRL is to maximize the agent's long-term reward.

Given its ability to quickly capture environmental dynamics and excel in making sequential decisions to maximize long-term rewards, DRL has drawn increasing attention in recent years. Various DRL frameworks have emerged, including Deep Recurrent Q- Network [43], proximal policy optimization algorithm [42], etc. Furthermore, DRL has been applied in diverse applications such as task offloading in wireless networks [60], routing optimization [15], etc. Since caching decision-making is inherently a sequential task, two chapters in this thesis leverage DRL to make caching decisions. Through iterative trial-and-error learning, the DRL-based caching agent learns from interactions with the environment, discovering which actions lead to optimal outcomes, and updating its policies accordingly.

2.5.1 Deep Actor-Critic Methods

Deep Actor-Critic methods[29] represent an advanced family of reinforcement learning techniques combining aspects of policy-based and value-based methods through neural network approximations. The key idea behind actor-critic methods is to maintain

two distinct structures simultaneously: an *actor*, responsible for selecting actions according to a learned policy, and a *critic*, which evaluates these actions by estimating the corresponding value functions.

In the deep learning variant, neural networks are employed to approximate both the policy (actor) and the value function (critic), enabling these methods to handle complex environments with large or continuous state-action spaces efficiently. The actor aims to directly optimize the policy parameters by maximizing the expected cumulative reward, while the critic guides this optimization process by providing an estimate of how valuable the selected actions are.

Formally, the actor-critic method updates its policy parameters (θ) based on gradients derived from the critic's evaluation, typically through policy gradients such as:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|s) A^\pi(s,a)\right] \tag{2.9}$$

where $\pi_\theta(a|s)$ is the policy defined by the actor network, parameterized by θ. $A^\pi(s,a)$ stands for the advantage function computed by the critic, indicating how much better an action is compared to the average action available at state s.

Deep actor-critic methods, such as Advantage Actor-Critic (A2C), Asynchronous Advantage Actor-Critic (A3C), and Proximal Policy Optimization (PPO), have been highly successful in various challenging reinforcement learning tasks, including robotics, video games, and control problems.

2.5.2 Proximal Policy Optimization (PPO)

PPO is a state-of-the-art deep actor-critic algorithm designed to efficiently optimize neural network-based policies in reinforcement learning tasks. PPO addresses the instability often encountered in policy gradient methods by introducing a novel objective function that constrains policy updates, ensuring more stable and robust training.

The core idea behind PPO is the clipped surrogate objective, defined as follows:

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{2.10}$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the probability ratio between the new and old policy.
- $\hat{A}_t$ is the advantage estimate at timestep t.
- ϵ is a small hyperparameter that limits how much the policy can change in each training iteration, thus preventing excessively large updates.

To further enhance exploration and prevent premature convergence, PPO incorporates an `entropy regularization` term into its objective function. The entropy term encourages the policy to maintain a certain degree of randomness, defined as:

$$L^{ENT}(\theta) = \mathbb{E}\left[H\left(\pi_\theta(a|s)\right)\right] \tag{2.11}$$

where $H(\pi_\theta(a|s))$ represents the entropy of the policy distribution. By maximizing entropy, PPO ensures diverse action selection, enabling the policy to explore a broader range of actions and states effectively.

PPO achieves stable and efficient policy optimization by combining the clipped surrogate objective with entropy regularization, making it particularly suitable for complex and high-dimensional reinforcement learning tasks.

2.6 Traditional Caching Policies

Traditional caching policies play a fundamental role in determining which content should be stored or evicted when cache space is limited. These policies follow heuristic-based strategies, including LRU, FIFO, MRU, Random Replacement, and 2Q Caching.

LRU evicts the least recently accessed content, assuming that recently used items are more likely to be needed again soon. FIFO removes the oldest cached content in the order it was added, regardless of access frequency, which can lead to suboptimal evictions. MRU, in contrast to LRU, discards the most recently used content, which may be beneficial in workloads where older data is more relevant. Random Replacement selects an item at random for eviction, making it computationally simple but often inefficient in optimizing cache hit rates. 2Q Caching extends LRU by maintaining two queues—one for newly accessed content and another for frequently accessed items—helping to balance recency and frequency in caching decisions.

While these traditional caching policies offer computational efficiency, they lack adaptability in dynamic environments like IoV, where request patterns continuously change. This limitation motivates the use of learning-based approaches, such as RL and RS, to develop more intelligent and adaptive caching strategies.

Although these classical caching strategies are widely used, they are reactive and lack the ability to adapt to dynamic mobility and evolving content demand. Moreover, they do not leverage the architectural advantages offered by emerging networking paradigms such as ICN, where caching is a native network function. We therefore introduce ICN as an important conceptual foundation for edge caching in the IoV.

2.6.1 ICN and Edge Caching

Traditional IP networks follow a host-centric communication model, where data is retrieved from a specific server identified by an IP address. In contrast, ICN shifts the focus from host locations to named data objects. In ICN architectures such as NDN, intermediate routers maintain in-network caches, enabling content to be delivered from the nearest node storing the requested data rather than the original producer.

ICN naturally supports edge caching: any node along the delivery path may

store content in a local Content Store, reducing end-to-end latency and alleviating core network congestion. This architectural property aligns closely with the needs of the IoV, where RSU serves as distributed edge nodes along vehicular routes. RSUs function analogously to ICN forwarders, providing fast access to popular content for passing vehicles.

However, while ICN specifies a framework for named-data delivery and in-network caching, it does not define how limited cache space should be allocated under rapidly changing vehicular mobility and content preferences. Classical ICN caching strategies such as Leave-Copy-Everywhere (LCE) or probabilistic caching often perform suboptimally in dynamic IoV environments.

This motivates the use of intelligent, learning-driven caching mechanisms that can predict future demand and proactively manage cache contents at RSU. In this thesis, we build upon the ICN principle of in-network caching while introducing reinforcement learning and recommender-based decision-making to determine which content objects should remain cached to maximize hit ratio and reduce latency in highly dynamic vehicular scenarios.

Recent studies have applied learning-driven approaches to enhance caching decisions in ICN and Software-Defined Networking (SDN)-ICN architectures. Hou et al. have proposed a series of GNN-based and DRL-based caching schemes that exploit request patterns, network topology, and feature extraction to improve cache-hit performance. Their work spans statistical preference modeling for SDN-ICN [23], GNN-assisted caching for NDN routers [19, 21], and proactive or predictive caching strategies for NDN networks [20]. Follow-up efforts integrate optimized GNN-based decision-making within SDN-controlled ICN infrastructures [22] and explore broader edge-intelligent caching frameworks for Internet of Things (IoT) services [24]. These studies collectively demonstrate that combining ICN's in-network caching with representation learning and reinforcement-driven adaptation can significantly improve hit ratio and reduce delivery latency in dynamic environments, further motivating the

learning-oriented caching design adopted in this thesis.

2.7 Recommender System and Graph Neural Network

While ICN provides architectural support for in-network caching, it does not directly model user preferences or predict future content demand. Recommender systems complement ICN-based caching by estimating which content is likely to be requested, enabling more informed caching decisions at RSU.

RS have become an essential component of modern digital platforms, providing personalized suggestions to users based on their preferences and past interactions. These systems play a crucial role in various applications, including e-commerce, online streaming, and social media, where vast amounts of content must be efficiently matched to user interests. Traditional recommendation approaches primarily fall into two categories: collaborative filtering[18] and content-based filtering[46]. Collaborative filtering leverages user-item interaction patterns to infer preferences, while content-based filtering analyzes item attributes to make recommendations. More advanced hybrid models combine these techniques to enhance recommendation accuracy and mitigate challenges such as data sparsity and the cold-start problem.

In recent years, GNN-based recommender systems[52] have emerged as a powerful paradigm, leveraging the expressive power of graphs to model complex user-item relationships. Unlike traditional methods that rely on matrix factorization or deep learning architectures operating on tabular data, GNN-based models explicitly represent users and items as nodes in a graph, with edges encoding various interactions, such as purchase history, implicit feedback, and social connections. By iteratively aggregating and propagating information across graph structures, GNNs enable more effective representation learning for both users and items.

Specifically, GNN-based recommendation can be viewed as a message-passing process. Each node (user or item) updates its embedding by aggregating information from its neighboring nodes, capturing both direct and high-order interactions. Through multiple layers of aggregation, the model learns expressive representations that preserve structural and semantic relationships within the graph. Compared to conventional models, GNN-based recommenders excel in addressing sparsity issues, improving generalization, and handling evolving user preferences dynamically.

Given their ability to model complex dependencies and enhance recommendation quality, GNN-based recommenders have gained increasing attention. Various architectures, including GCN [17], GAT [48], and Transformer-based recommendation[57] have been developed to optimize information propagation and weighting mechanisms. Furthermore, GNN-based approaches have been successfully applied in domains such as personalized content recommendation, social network analysis, and online advertising, demonstrating superior performance compared to conventional collaborative filtering techniques.

2.8 Summary

This chapter introduced the core background required for the remainder of this thesis, including IoV system concepts, neural networks, graph neural networks, reinforcement learning, deep reinforcement learning, and classical caching policies. These foundations support the methods developed in the next chapters: Chapter 4 applies GNN-based recommendation and reinforcement learning for cache optimization, while Chapter 5 introduces a hierarchical actor–critic caching strategy suitable for dynamic IoV environments.

Chapter 3

Related Works

Traditional caching methods, such as Greedy algorithms and MRU policies, often fail to account for the complex relationships between vehicles and content demands, leading to suboptimal cache utilization and increased latency. These approaches typically lack the ability to predict future content requests accurately or adjust caching policies in real-time based on vehicular mobility patterns. As a result, they may not effectively handle the dynamic and heterogeneous nature of IoV environments, where content popularity and network topology can change rapidly.

3.1 Intelligent Caching in IoV

Several studies have explored advanced machine learning techniques to enhance edge caching strategies in vehicular networks. Chen *et al.* [49] proposes a proactive caching framework for high-mobility IoV networks that leverages reinforcement learning, specifically, both non-contextual and contextual MAB techniques—to predict the next RSU a vehicle will enter and dynamically cache fixed-size content fragments at the network edge. Simulations in diverse urban layouts demonstrate enhanced QoE by tailoring caching decisions to real-time mobility and network conditions, while the system uniquely incorporates uncertainty modeling through an extended Subjective

Logic framework. Key advantages include accurate prediction of RSU coverage and proactive caching that minimizes service interruptions; however, a critical limitation is that RSU may face unbounded caching demands when many vehicles simultaneously request remaining content fragments.

Similarly, the authors in [36] compared multi-agent deep reinforcement learning strategies, including non-cooperative, federated, and personalized learning, finding that personalized learning achieves the highest cache hit ratio and lowest delivery delay. However, this approach assumes each vehicle requests only one content per time slot and that content popularity is known in advance, which may not reflect real-world IoV conditions where vehicles may have multiple content requests and content popularity is highly dynamic.

The authors in [59] proposed an edge caching method for the IoV using multi-agent deep reinforcement learning. It treats each moving vehicle as an adaptive agent that makes caching and access decisions to minimize content distribution delays in a dynamic environment. Experiments show that this approach improves content hit rate and success rate while reducing delays compared to other methods. Nonetheless, it also assumes single content requests per time slot and does not consider vehicular mobility, limiting its effectiveness in dynamic environments. Moreover, relying on vehicles for caching may not be reliable due to their transient connections and limited storage capacities.

The paper[61] presents a novel approach to edge caching within the IoV utilizing Distributed Multi-Agent Reinforcement Learning (MARL). The authors extend the MDP to a multi-agent context and address the challenge through a stochastic game framework. They introduce Distributed MARL-based Edge caching (DMRE) and an enhanced version, DeepDMRE, which incorporates Deep-Q Networks for better performance. The system optimizes caching strategies for each agent (e.g., RSU), considering their mutual influence within the IoV environment. However, the simplified RL state using only cache hit ratio omits key dynamics like vehicle count,

positions, and speeds. This hinders the RL agent's ability to learn effective caching strategies under changing conditions. Richer metrics are needed to capture essential system dynamics.

3.2 Using GNN with DRL

DRL has gained popularity in addressing sequential decision-making problems. Traditional DRL approaches like Deep Q-learning [35], policy gradient methods [44], and Actor-Critic (AC) algorithms typically utilize MLP as the underlying Neural Network (NN) architecture. However, with the emergence of various advanced NN models, researchers are increasingly integrating DRL with other neural network architectures such as CNN [26, 40], LSTM[37, 54], Transformer[4, 8], and even GNN. Authors in [38] employed dynamic GCNs and DRL for long-term traffic flow prediction. They represented traffic flow as a graph, where each station is a node and directed weighted edges are used to indicate traffic flow occurrence. A Graph Convolutional Policy Network (GCPN) model generates dynamic graphs at each time step, and the RL agent receives a reward if the generated graph closely resembles the target graph. The paper further utilizes GCN and LSTM to extract spatial and temporal features from the generated dynamic graph sequences, enabling traffic flow prediction in future time steps. Another study [56] introduces the Inductive Heterogeneous Graph Multi-agent Actor–critic (IHG-MA) algorithm for traffic signal control. The traffic network is modelled as a heterogeneous graph, with each traffic signal controller considered as an agent.

These studies demonstrate the effectiveness of combining DRL and GNN in tackling a range of issues, including traffic prediction, traffic signal control, cooperative control of autonomous vehicles, and routing optimization in IoV scenarios. By leveraging the respective strengths of DRL and GNN, these approaches enable intelligent decision- making and enhance performance in intricate systems.

3.3 Popularity-based Caching Strategies

This section delves into popularity-based caching strategies, which calculate content popularities based on various network factors. Consequently, caching decisions rely on content popularities and node positions within the network. It is important to note that this section does not involve any DNN-based caching strategies, which typically predict content popularities using a neural network. Instead, this section focuses on strategies that utilize predefined rules to compute content popularities.

In the context of optimizing caching environments for the IoV, integrating RS with RL is gaining increasing attention among researchers. RS can precisely capture user preferences for specific items, thus effectively narrowing the selection space of cached files and improving cache accuracy. When appropriately combined with RL, RL agents can comprehensively perceive user characteristics within the RSU's coverage area, such as arrival rates, speeds, and relative positions to the RS. This integration enables the agent to devise optimal caching strategies or directly select the most suitable files for caching.

To address content popularity prediction and privacy concerns, Wang *et al.* [51] combined asynchronous federated learning with deep reinforcement learning, allowing RSUs to adaptively manage cache space based on predicted content popularity with autoencoder. Yet, the method assumes availability of global user ratings and personal information for similarity calculations, potentially conflicting with privacy preservation objectives and facing challenges in data availability and synchronization.

3.4 Discrete Soft Actor Critic and Joint Policies Learning

The Discrete Soft Actor-Critic (DSAC)[6] method is a variant of the Soft Actor-Critic algorithm adapted specifically for discrete action spaces. Soft Actor-Critic (SAC)

methods are known for maximizing both cumulative rewards and policy entropy, thus encouraging exploration while optimizing performance. Unlike traditional actor-critic methods, SAC introduces an additional entropy term to its objective, explicitly balancing exploration and exploitation through entropy regularization.

DSAC optimizes the policy (π_θ) by maximizing the following entropy-regularized objective function:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))\right)\right] \tag{3.1}$$

where:

- $r(s_t, a_t)$: the immediate reward for action a_t in state s_t.
- α: the entropy temperature parameter that determines the trade-off between maximizing rewards and encouraging exploration.
- $H(\pi(\cdot|s_t))$: the entropy of the policy distribution at state s_t, encouraging diverse action selection.

In DSAC, the policy update is achieved by explicitly computing action probabilities through a softmax function, defined as:

$$\pi_\theta(a|s) = \frac{\exp\left(Q_\phi(s, a)/\alpha\right)}{\sum_{a'} \exp\left(Q_\phi(s, a')/\alpha\right)} \tag{3.2}$$

where $Q_\phi(s, a)$ represents the soft Q-value for each discrete action, parameterized by neural networks with parameters ϕ.

The critic network updates its parameters using the soft Bellman backup, defined as:

$$Q_\phi(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{s' \sim p}\left[V_{\bar{\phi}}(s')\right] \tag{3.3}$$

where the soft value function $V_{\bar{\phi}}(s')$ is computed using the log-sum-exp function of the next state's Q-values:

$$V_{\bar{\phi}}(s') = \alpha \log \sum_{a'} \exp\left(\frac{Q_{\bar{\phi}}(s', a')}{\alpha}\right) \tag{3.4}$$

DSAC has proven highly effective in discrete-action environments, offering stable policy optimization, improved exploration, and strong empirical performance across diverse decision-making tasks.

3.4.1 Joint Policies Optimization

DSAC is inherently suited to learn joint policies, especially in multi-agent or multi-objective scenarios[9, 11, 34]. Rather than optimizing a single deterministic policy, DSAC employs an entropy-regularized objective that encourages exploration across the entire action space. By explicitly maximizing the entropy of the policy distribution, DSAC maintains a probabilistic representation over multiple potentially optimal policies simultaneously, effectively capturing a distribution over various decision-making strategies rather than converging prematurely to a single optimal action.

The key reason DSAC can concurrently learn multiple policies is its use of the *soft Q-values*, which encode both expected cumulative rewards and uncertainty through entropy. Formally, DSAC optimizes a soft policy defined by the softmax Equation (3.2) over Q-values.

The probabilistic representation inherently supports simultaneous learning of multiple policies because each action's probability reflects its relative value and uncertainty. Rather than converging directly to a deterministic action, DSAC continuously adjusts and updates these joint probabilities based on observed rewards and transitions, effectively enabling the agent to explore and learn multiple effective strategies concurrently.

Consequently, DSAC's entropy-maximizing approach naturally facilitates joint-

policy learning by maintaining a balanced exploration across actions, making it particularly advantageous in environments requiring diverse or adaptive policies.

3.4.2 Applying Discrete Soft Actor Critic (DSAC)

To address the challenges of autonomous decision-making in dynamic environments, we adopt the DSAC approach due to its effectiveness in handling discrete action spaces and encouraging exploration through entropy regularization. In this section, we outline how DSAC is specifically applied to construct an efficient and safe discrete decision-making strategy for variant environments.

The paper[10] proposes a novel discrete decision-making strategy for autonomous driving using a Discrete Soft Actor-Critic algorithm enhanced by a sample filter method (DSAC-SF). The DSAC-SF approach increases both the efficiency and stability of learning by selectively utilizing effective training samples. Additionally, the authors design an area observation method and a multi-objective reward function aimed at improving vehicle safety and efficiency in dynamic freeway scenarios. Experiments conducted in the SMARTS simulation environment demonstrate that the proposed DSAC-SF algorithm significantly improves decision-making success rates, driving speed, training efficiency, and stability compared to existing discrete reinforcement learning methods.

The paper[5] introduces a cooperative edge video caching framework designed to improve video transmission efficiency within the IoV. By integrating recommendation, caching, and transmission optimization using a Discrete Soft Actor–Critic (DSAC) based deep reinforcement learning algorithm, the proposed approach dynamically adapts to changing network conditions and diverse user preferences. Simulation results show significant improvements in service utility, delivery rate, and latency reduction compared to traditional baseline methods, demonstrating the framework's potential for delivering seamless, high-quality video content in vehicular networks.

The recommendation model proposed in this paper jointly considers intrinsic user preferences and video recommendations provided by RSU to determine the probability of video requests. Specifically, user preferences for video content are modeled using a Zipf-like distribution. Given a video file f and a vehicle i, the inherent request probability $p_{i,f}^t$ at time t can be calculated as follows:

$$p_{i,f}^t = \frac{f^{-\eta_i^{(t)}}}{\sum_{f=1}^{F} f^{-\eta_i^{(t)}}} \tag{3.5}$$

where $\eta_i^{(t)}$ determines the shape of the Zipf distribution, reflecting the concentration of content popularity. The RSU proactively recommends cached videos to vehicles, thereby increasing request probabilities for these recommended videos. The actual probability that vehicle i requests video f at time t, incorporating both inherent user preferences and recommendation influence, is computed by:

$$p_{i,f}^t = \begin{cases} \lambda_i p_{i,f}^{t,R} + (1-\lambda_i) p_{i,f}^t, & x_{i,f}^t = 1 \\ (1-\lambda_i) p_{i,f}^t, & x_{i,f}^t = 0 \end{cases} \tag{3.6}$$

where λ_i denotes the probability that vehicle i accepts the RSU's recommendations, $x_{i,f}^t$ indicates whether video f is recommended to vehicle i, and $p_{i,f}^{t,R}$ is the recommendation gain, assumed equal across recommended videos.

The caching model optimizes the placement and retrieval of video content within RSUs and vehicles in IoV scenarios. Video files can be simultaneously cached at multiple nodes to enhance delivery efficiency and alleviate bandwidth constraints. Caching decisions depend on multiple factors such as video popularity, resource availability, and request probabilities. The caching capacity at each RSU and vehicle is limited, necessitating strategic management to maximize cache hits and minimize delays. Formally, the caching decision $y_{f,j}^t$ for video file f at node j (either RSU or vehicle i) at time t is constrained by their storage capacities as follows:

$$\sum_{f=1}^{F} D_f y_{f,0}^t \leq C_{RSU}^t, \quad \sum_{f=1}^{F} D_f y_{f,i}^t \leq C_i^t, \quad \forall i \in \mathcal{M} \tag{3.7}$$

where D_f is the data size of video f, and C_{RSU}^t, C_i^t represent the caching capacities of the RSU and vehicle i, respectively. By jointly leveraging recommendation and caching strategies, the proposed approach significantly enhances the efficiency and user experience of video services in IoV environments.

3.5 GNN-based Recommender

With the advancement of recommender systems, GNN have gradually gained prominence due to their capability to effectively capture interactions between users and items, as well as their ability to model multi-hop associations. This enables GNN to be applicable to both collaborative filtering and content-based filtering tasks simultaneously. Moreover, in reinforcement learning settings, the user and item embeddings constructed by GNN-based recommenders during training can significantly enhance the perception capabilities of GNN-based reinforcement learning agents regarding the environment. In this section, we introduce state-of-the-art GNN-based recommenders and discuss their advantages and limitations.

The paper [16] introduces LightGCN, a simplified and effective GCN-based collaborative filtering model designed specifically for recommendation tasks. Through empirical analysis, the authors found that traditional GCN components such as feature transformation and nonlinear activation contribute minimally to recommendation performance. Therefore, LightGCN retains only the most critical GCN component—neighborhood aggregation—to linearly propagate user and item embeddings across the user-item interaction graph. The final embeddings are computed by aggregating embeddings from all propagation layers. This streamlined design significantly enhances model simplicity, ease of implementation, and training efficiency, achieving

around 16.0% average relative improvement over Neural Graph Collaborative Filtering (NGCF), a leading state-of-the-art method.

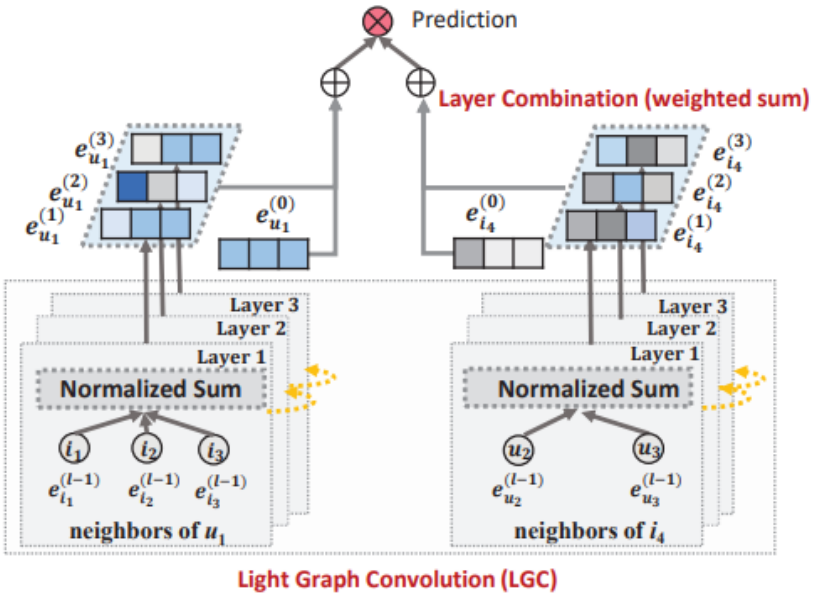


Figure 3.1: Illustration of the LightGCN model architecture. (from [16])

Figure 3.1 illustrates the architecture of LightGCN, highlighting its simplified GCN structure for collaborative filtering. The model consists of two main operations: Light Graph Convolution (LGC) and Layer Combination. In the LGC step, embeddings are propagated through the graph by aggregating normalized embeddings from neighboring nodes. Unlike traditional GCN models, LightGCN omits self-connections, feature transformations, and nonlinear activation functions, which simplifies the network significantly. In the Layer Combination step, embeddings from each layer are aggregated using a weighted sum to generate the final user and item representations for prediction.

DGRec[55] is a diversified graph neural network-based recommender system designed to enhance the diversity of recommendations by directly improving the embedding generation process. It employs three specialized modules: submodular neighbor selection to ensure diverse neighbor aggregation, layer attention for adaptive layer

importance, and loss reweighting to improve performance on long-tail items. The primary advantage of DGRec is its clear, distinct, and reusable user and item embeddings. However, emphasizing diversity prevents DGRec from significantly outperforming state-of-the-art methods in terms of recall. Additionally, when applied in IoV environments, the high mobility of vehicles restricts their capacity to request multiple types of content, resulting in redundant computational overhead.

MixGCF[25] addresses the fundamental challenge of negative sampling in graph neural network (GNN)-based collaborative filtering. Specifically designed as a flexible negative sampling module, MixGCF generates hard negatives by synthesizing embeddings through a novel *hop mixing* technique, which aggregates embeddings from different layers of raw negatives' neighborhoods. A theoretical optimization strategy guides the selection of layers and neighborhoods for constructing these negatives. Extensive experimental results demonstrate substantial improvements over existing state-of-the-art GNN models, including a 26% improvement for NGCF and 22% for LightGCN in terms of NDCG@2, highlighting MixGCF's effectiveness in enhancing recommendation performance.

3.6 Summary

This chapter reviewed advanced caching and recommendation strategies in IoV, highlighting the benefits of integrating graph neural networks and discrete reinforcement learning techniques. Approaches such as LightGCN, DGRec, and MixGCF demonstrate significant improvements in recommendation diversity, caching efficiency, and predictive performance, addressing key IoV challenges such as mobility and dynamic user demands.

Chapter 4

Reinforcement Learning-Enhanced and GNN Recommender Caching

4.1 Introduction

This chapter focuses on improving caching performance (i.e., maximizing the cache hit ratio) in IoV using a pretrained GNN recommender and an online RL agent. A higher cache hit ratio indicates that IoV nodes can cache contents that users are interested in. In this case, more user requests are satisfied by IoV nodes' cache stores rather than the content producer (i.e., the server). It improves network performance regarding caching space utilization and data delivery latency.

4.2 System Design

4.2.1 Environment Architecture

We consider a vehicular network as shown in Figure 4.1, where a set of vehicles $\mathcal{V} = \{v_1, v_2, \ldots, v_V\}$ communicate with a RSU deployed along a highway or urban road. Each vehicle is equipped with wireless communication capabilities and gener-

ates content requests from a catalog $\mathcal{F} = \{f_1, f_2, \ldots, f_F\}$ based on individual preferences and mobility patterns. The RSU has a limited cache capacity C and aims to proactively store popular content to serve vehicles with minimal latency. To achieve this, we leverage a GNN-based recommendation system that predicts future content demands by analyzing the associations between vehicles and content.

Vehicles connect to the RSU via Vehicle to infrastructure (V2I) communication channels when they are within its coverage area. The mobility of vehicles is modeled using realistic traffic patterns, where each vehicle's position changes over time according to its speed and direction. The arrival of vehicles within the RSU's coverage is modeled as a Poisson process, capturing the stochastic nature of vehicular traffic. The set of vehicles connected to the RSU at time t is dynamic, denoted as $\mathcal{V}_t \subseteq \mathcal{V}$.

Furthermore, the environment captures additional network metrics critical to content delivery performance. Specifically, each vehicle is associated with a communication delay d_i after their first request and updated in each time frame, an available bandwidth B_i, and an average speed v_i, which collectively influence the quality of service. These parameters are used to dynamically adjust caching strategies and provide a more comprehensive view of the network state.

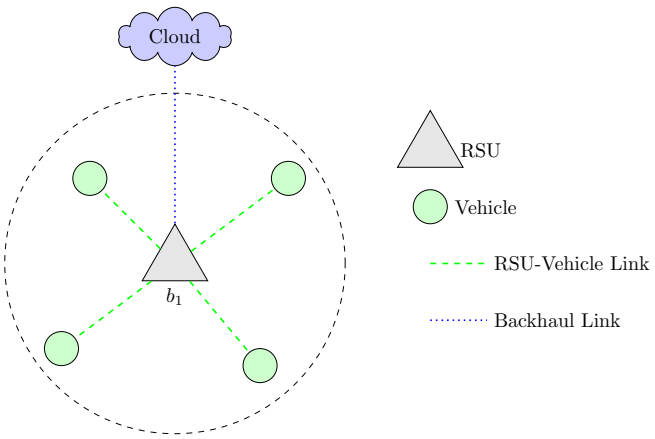


Figure 4.1: System Architecture

4.2.2 Request Pattern

4.2.2.1 Random Requesting GCNRDRL

In vehicular caching scenarios within IoV, various request strategies beyond Zipf-based models can also yield effective caching policies. For instance, prior research[49] demonstrates successful caching strategies by employing random vehicle generation and randomized file requests. Specifically, by predicting the next RSU position, the system pre-caches files to enhance the overall QoE across the network. Such random request patterns can also be integrated within GCNRDRL frameworks to simulate diverse vehicular behaviors.

Formally, given a vehicle v, it maintains a candidate set of requestable items denoted by $\mathcal{V}$. At each time step t, the vehicle randomly selects n items from $\mathcal{V}$, forming the request set $\mathcal{V}_t$ for that time step.

4.2.2.2 Zipf Request Distribution

In the proposed RL environment for vehicular file caching, each vehicle $v \in \mathcal{V}$ requests items according to user-specific preferences combined with a Zipf distribution. Specifically, the probability of a vehicle v requesting a particular file is influenced both by individual user-item ratings and a Zipf-based popularity bias.

Given a vehicle v with user identifier u, let $\mathcal{I}_u^+$ denote the set of positively rated items. Each item $i \in \mathcal{I}_u^+$ has an associated preference rating $r_{u,i}$. To reflect realistic variability and popularity bias, item request probabilities are weighted by the Zipf distribution. Formally, for a sorted sequence of items $i_1, i_2, \ldots, i_n$ ranked in decreasing order of the user's preference ratings, the probability of selecting the k-th ranked item is:

$$p(i_k) = \frac{\frac{1}{k^s}}{\sum_{j=1}^{n} \frac{1}{j^s}}, \quad k = 1, 2, \ldots, n \tag{4.1}$$

Here, the parameter s (denoted as `zipf_s`) controls the skewness of the distribution; larger values intensify the bias toward higher-ranked items.

When a vehicle v requests an item, the environment computes the communication delay based on the Euclidean distance between the vehicle position $\mathbf{x}_v$ and the RSU position $\mathbf{x}_{\text{RSU}}$. The normalized distance is computed as:

$$d_{\text{norm}} = \frac{|\mathbf{x}_v - \mathbf{x}_{\text{RSU}}|}{d_{\max}}, \tag{4.2}$$

where $d_{\max}$ is the maximum communication range.

If the requested file is available in the cache (cache hit), the vehicle experiences reduced communication delay:

$$\text{delay}_{\text{hit}} = \text{base\_delay} - \text{hit\_delay\_reduction} \cdot (1 - d_{\text{norm}}), \tag{4.3}$$

where *base_delay* represents the baseline service delay experienced by a vehicle under ideal conditions — for instance, when it is close to the RSU and the channel is stable. *hit_delay_reduction* quantifies how much the service delay is further reduced when the requested content is already cached. Notably, the vehicle's request frequency is increased to model user satisfaction and the tendency to request content more actively in response to a positive experience:

$$fr_v \leftarrow \min(fr_v + fr_{\text{inc}}, fr_{\max}), \tag{4.4}$$

where fr_{inc} denotes the incremental increase in frequency upon cache hits, and $fr_{\max}$ is the maximum allowed request frequency.

Conversely, if the request results in a cache miss, the vehicle incurs an additional delay penalty:

$$\text{delay}_{\text{miss}} = \text{base\_delay} + \text{miss\_delay\_penalty}, \tag{4.5}$$

and consequently, the vehicle's request frequency is decreased to reflect user dissatisfaction and reduced immediate engagement:

$$fr_v \leftarrow \max(fr_v - fr_{\text{dec}}, fr_{\text{base}}), \tag{4.6}$$

where fr_{dec} is the decrement applied upon cache misses, and fr_{base} represents the minimum request frequency. This dynamic adjustment of request frequencies effectively captures realistic user behavior, incentivizing efficient caching strategies within the RL environment.

4.3 Proposed Methodology

Before introducing the details of each component, we provide a high-level overview of the proposed GCNRDRL framework. The system combines an offline recommender module and an online reinforcement learning controller to make caching decisions at the RSU. First, a LightGCN-based recommender is trained on historical user–item interactions to identify globally popular or semantically related content. The resulting candidate item set reflects the long-term preference structure of the dataset and serves as the content pool from which the RSU may select cached files.

On top of this, a PPO-based reinforcement learning agent operates in real time, observing the dynamic IoV environment and deciding which items should be kept or replaced in the cache. The RL agent does not rely on item scores; instead, it reacts to network conditions and request dynamics to optimize hit ratio and service delay.

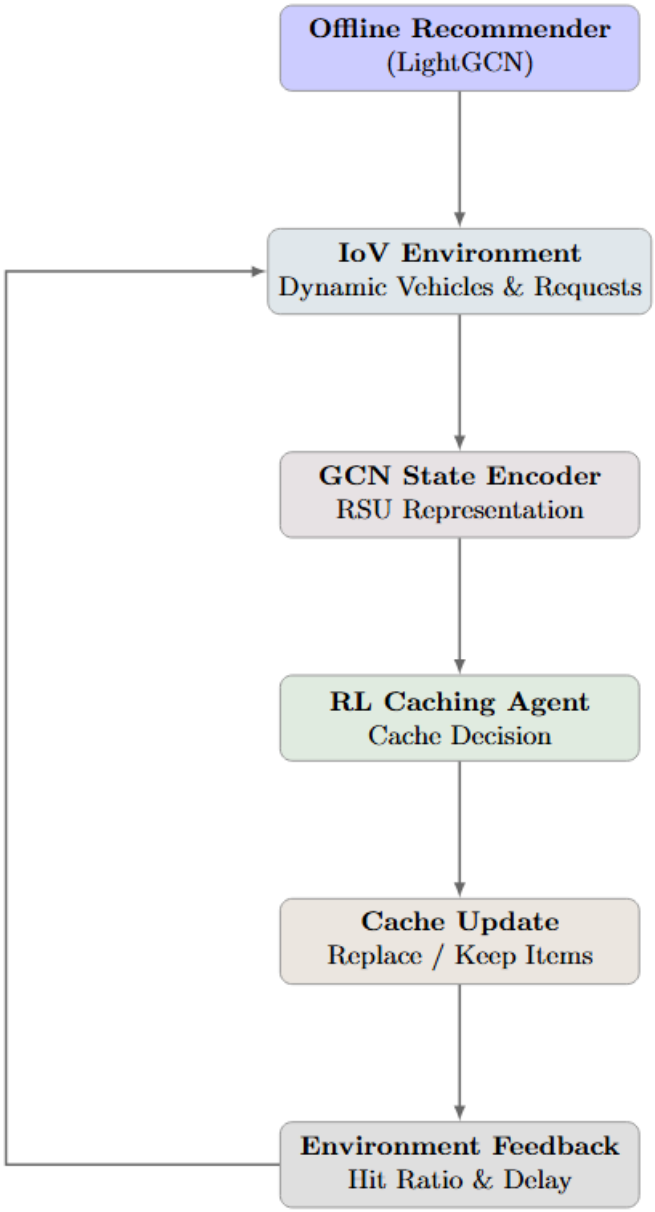


Figure 4.2: Overall GCNRDRL System framework.

The overall workflow of the proposed GCNRDRL framework is illustrated in Fig.4.2.

4.3.1 Recommender Building

To enhance caching decisions, we integrate pretrained embeddings generated by Light Graph Convolutional Network (LightGCN) [16]. After training, LightGCN yields

embedding matrices for all users and items, denoted respectively as $\mathbf{U} \in \mathbb{R}^{|\mathcal{U}| \times d}$ and $\mathbf{V} \in \mathbb{R}^{|\mathcal{I}| \times d}$, where d is the embedding dimension.

In our vehicular caching environment, these embeddings are employed dynamically to construct candidate sets of items to cache at each decision timestep. Specifically, the set of candidate items is updated as follows: given a set of active vehicles $\mathcal{V}_t$ currently within communication range of an RSU at timestep t, each vehicle $v \in \mathcal{V}_t$ is associated with a unique user identifier u_v. The embedding of each active user is retrieved from the user embedding matrix $\mathbf{U}$, and the mean user embedding vector for timestep t, given by $\mathbf{u}_t$, is computed as:

$$\mathbf{u}_t = \frac{1}{|\mathcal{V}_t|} \sum_{v \in \mathcal{V}_t} \mathbf{U}_{u_v} \tag{4.7}$$

Utilizing this aggregated representation $\mathbf{u}_t$, candidate items for caching are selected by calculating scores through inner products with all item embeddings $\mathbf{V}_i$:

$$s_{t,i} = \mathbf{u}_t^\top \mathbf{V}_i, \quad i \in \mathcal{I} \tag{4.8}$$

Subsequently, we select the top-K highest-scoring items to form the candidate set at timestep t, given by $\mathcal{C}_t$ generated by LightGCN:

$$\mathcal{C}_t = \{i \in \mathcal{I} \mid s_{t,i} \text{ ranks among top-}K\} \tag{4.9}$$

These candidate scores are directly used as inputs within the environment's state representation, guiding the reinforcement learning agent's decisions regarding caching strategies. The initial cache configuration at the beginning of the simulation defaults to the highest-ranked items within the candidate set, ensuring meaningful initialization and consistent evaluation throughout the environment's operation.

4.3.2 State Space

We define state representations for both the MLP and GNN-based methods to effectively capture essential network dynamics relevant to caching.

4.3.2.1 MLP-based State Space

The state s_t at time step t is given by:

$$s_t = [\mathcal{C}_t, d_t^{\text{avg}}, fr_t^{\max}] \tag{4.10}$$

Here, $\mathcal{C}_t \in \mathbb{R}^K$ represents the normalized candidate scores generated by LightGCN for the top-K candidate items. The scalar d_t^{avg} denotes the average communication delay across all vehicles at time t, normalized by the baseline delay. Finally, $fr_t^{\max}$ is the normalized user id with the highest current request frequency, defined as:

$$fr_t^{\max} = \frac{\max_{v \in \mathcal{V}}(fr_v)}{fr_{\max}}, \tag{4.11}$$

4.3.2.2 GNN-based State Space

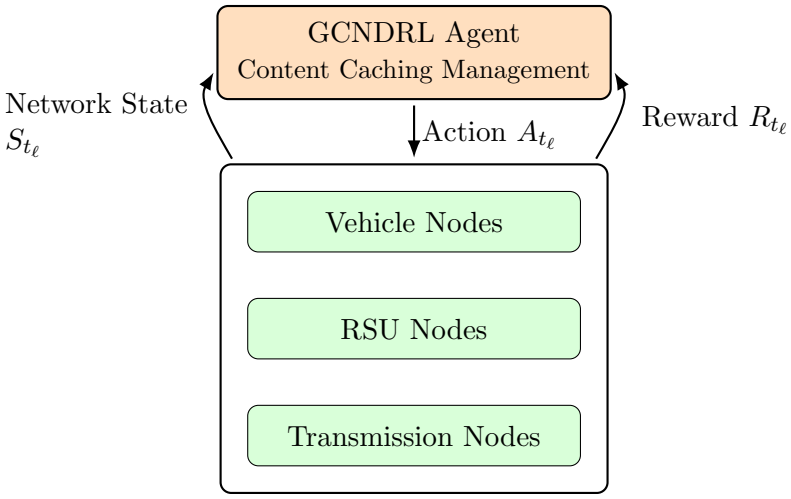


Figure 4.3: GCNDRL RL Architecture

For the GNN-based representation shown in Figure 4.3, the environment at time

t is modeled by a graph $G_t = (X_t, E_t)$. Each node (RSU or vehicle) in the graph has features succinctly captured as:

$$x_i = [d_i, f_i, 1] \tag{4.12}$$

where d_i is the normalized communication delay for node i, f_i is its normalized request frequency, and the last term is a constant indicator feature. The nodes are padded to a fixed size to form the node feature matrix $X_t \in \mathbb{R}^{(1+\text{gnn\_max\_vehicles})\times 3}$. Edges form a star topology around the RSU, and a node mask is applied to differentiate valid nodes from padding.

4.3.3 Action Space

The action at time t, consistent across both MLP and GNN methods, is defined as:

$$a_t = [a_{t,1}, a_{t,2}, \ldots, a_{t,K}] \in \mathbb{R}^K \tag{4.13}$$

This action vector assigns continuous caching scores to each candidate item. The final caching decision involves selecting the top-Z items:

$$C_t = \text{Top-}Z(a_t), \tag{4.14}$$

where Z is the cache capacity.

4.3.4 Reward Function

The reward function at timestep t incentivizes cache hits and low delays, expressed concisely as:

$$r_t = \frac{1}{|Q_t|} \sum_{q \in Q_t} [R(q) - w_d \cdot d_q] - c_u \cdot \mathbb{I}(C_t \neq C_{t-1}) \tag{4.15}$$

where Q_t is the set of requests at timestep t, d_q is the experienced delay for request q, w_d penalizes delay, and c_u penalizes cache updates. The reward $R(q)$ for each request is succinctly defined by the piecewise function:

$$R(q) = \begin{cases} R_{\text{hit}} \cdot \sigma(d_{\text{base}} - d_q), & q \in C_t \quad \text{(cache hit)}, \\ -R_{\text{miss}} \cdot [1 - \sigma(d_{\text{base}} - d_q)], & q \notin C_t \quad \text{(cache miss)} \end{cases} \tag{4.16}$$

where $\sigma(x) = \frac{1}{1+\exp(-kx)}$ is a sigmoid scaling function that smoothly rewards delay improvements, and R_{hit}, R_{miss} are constants representing rewards and penalties, respectively.

4.3.5 Modified GNN Architecture

This work employs a two-layer generic Graph Neural Network to update node features on a star-shaped topology. The central RSU node is connected to multiple vehicle nodes. At each decision step, every vehicle node carries its latest communication delay, recommender score and other features $\mathbf{x}_v \in \mathbb{R}^{d_{\text{in}}}$, whereas the RSU node stores an embedding of its previous cache state. Graph convolutions aggregate information from neighbouring nodes so that the RSU can jointly reason over the behaviour of all vehicles and decide which contents to cache next.

Network architecture: Let $\mathbf{X} \in \mathbb{R}^{N \times d_{\text{in}}}$ be the input feature matrix and $\mathcal{E}$ the edge set. The forward pass consists of two convolutional layers:

$$\mathbf{H}^{(1)} = \sigma\big(\text{Conv}_1(\mathbf{X}, \mathcal{E})\big), \qquad \mathbf{H}^{(2)} = \text{Conv}_2\big(\mathbf{H}^{(1)}, \mathcal{E}\big),$$

where $\sigma(\cdot)$ denotes the activation function (ReLU in our experiments). Conv_ℓ can be instantiated as GCNConv, GATConv, or TransformerConv, switchable through the `conv_type` hyper-parameter. The RSU row of $\mathbf{H}^{(2)}$ is treated as the

embedding for the next-state cache decision.

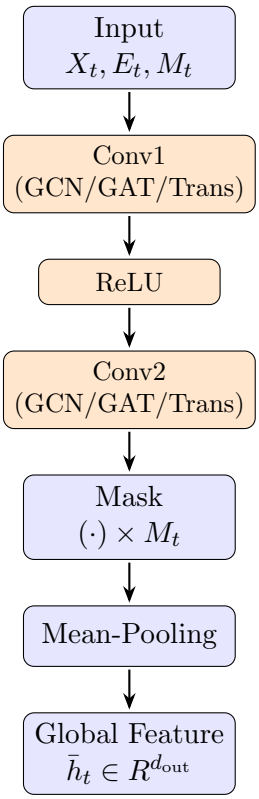


Figure 4.4: Pipeline of the `TorchGeoGNNFeatureExtractor`.

In a star graph, the maximum shortest-path length is 2, so two layers are sufficient for the RSU to receive information from every vehicle node. Deeper networks would not offer additional structural reach but would incur extra computation and potential over-fitting.

4.3.6 Hyper-parameter settings

Table 4.1: GNN hyper-parameters

Category	Hyper-parameter	Value
General	Number of layers L	2
	Hidden dimension d_{hidden}	64
	Output dimension d_{out}	32
	Activation function σ	ReLU
	Dropout p	0.4
GATConv only	Number of heads H	1

The listed values remain fixed across all experiments unless stated otherwise. For GATConv and TransformerConv, multi-head attention or larger hidden sizes can be enabled if greater capacity is required, but empirical tests show that a single head suffices for the star topology considered here.

4.3.7 GNN-based Feature Extractor

To interface the star-shaped graph observations with the stable baseline policy network, a dedicated feature extractor maps each raw observation to a single fixed-length vector. An observation is a dictionary containing

- `node_features`: the feature matrix $\mathbf{X} \in \mathbb{R}^{N \times d_{\text{in}}}$,
- `edge_index`: the edge list $\mathcal{E} \subseteq \{1, \ldots, N\} \times \{1, \ldots, N\}$,
- `node_mask`: a Boolean vector that flags valid nodes (needed because a fixed *maximum* number of nodes is pre-allocated for batching).

The extractor applies the two-layer GNN model to $(\mathbf{X}, \mathcal{E})$, producing node embeddings $\mathbf{H} \in \mathbb{R}^{N \times d_{\text{out}}}$. Embeddings whose mask bit is zero are discarded, and the remaining vectors are averaged to yield the global state representation

$$\bar{\mathbf{h}} = \frac{1}{|\mathcal{V}_{\text{valid}}|} \sum_{v \in \mathcal{V}_{\text{valid}}} \mathbf{H}_v,$$

which is passed to the downstream policy. This simple mean-pooling strategy preserves permutation invariance, keeps computational cost low and—by virtue of the node mask—remains unaffected by the zero-padded placeholders required for mini-batch processing.

4.4 NP-Completeness of the Rec-Cache-Hit Problem

This section formalises an *offline, single-shot* version of recommendation-aware caching (Rec-Cache-Hit), proves that it is NP-complete, and briefly discusses the impact of this hardness on RL approaches for cache optimisation.

4.4.1 Problem statement

Definition 1 (Rec-Cache-Hit)**.**

Input.

- A finite set of content items $I = \{i_1, \ldots, i_m\}$;
- A finite set of users (or predicted requests) $U = \{u_1, \ldots, u_n\}$;
- For each user $u_j \in U$ the recommender outputs a *single* item $r(u_j) \in I$ that u_j will request with certainty;
- A cache capacity $C \in \mathbb{N}$;

- A hit target $T \in \mathbb{N}$.

Question. Does there exist a cache set $S \subseteq I$ with $|S| \leq C$ such that at least T users find their predicted item in the cache, i.e.

$$\left|\{u_j \in U \mid r(u_j) \in S\}\right| \;\geq\; T\,?$$

The above already captures the core combinatorial difficulty; allowing multiple candidate items per user, probabilities, or time steps only *strengthens* the hardness result.

4.4.2 Membership in NP

Given a candidate cache S, counting the number of covered users takes $O(|U|)$ time, hence Rec-Cache-Hit $\in$ NP.

4.4.3 NP-hardness

We reduce from the classic Max-k-Cover problem, known to be NP-complete.

Definition 2 (Max-k-Cover (decision form))**.**

Input.

- A universe U of elements;
- Subsets $S_1, \ldots, S_m \subseteq U$;
- An integer budget k and a coverage target T.

Question. Do there exist at most k subsets whose union covers at least T elements of U?

Reduction. Given an instance of MAX-k-COVER, construct an instance of REC-CACHE-HIT as follows:

1. **Elements → users.** For every element $e \in U$ create a user u_e.
2. **Subsets → items.** For every subset S_i create a content item i_i.
3. **Recommender mapping.** For each element e pick *any* subset S_{i_e} containing e, and set $r(u_e) = i_e$.
4. **Parameters.** Set cache capacity $C := k$ and hit target T unchanged.

Correctness.

($\Rightarrow$) If the MAX-k-COVER instance admits $\leq k$ subsets covering $\geq T$ elements, caching the corresponding items covers the same $\geq T$ users.

($\Leftarrow$) Conversely, any cache S of $\leq k$ items that hits $\geq T$ users identifies $\leq k$ original subsets whose union covers those $\geq T$ elements.

The reduction runs in time polynomial in $|U| + |I|$, hence MAX-k-COVER $\leq_{\mathrm{p}}$ REC-CACHE-HIT. Combined with Definition 1, we obtain the main theorem.

Theorem 1. REC-CACHE-HIT is NP-complete.

Proof. Membership in NP was shown above. NP-hardness follows from the polynomial reduction described in the preceding paragraph. □

4.4.4 Implications for RL-based caching

Even computing an *optimal cache for a single time step* is NP-complete. Optimising over multiple time steps—i.e., the finite-horizon Markov decision process addressed by an RL agent—is at least as hard; in fact, infinite-horizon optimal control is NP-hard. Consequently, practical systems resort to heuristics or learning-based methods (e.g. reinforcement learning) to approximate near-optimal policies within reasonable time.

4.5 Algorithm

Algorithm 1: GCNRDRL

Input : Rating matrix $\mathcal{R}$, system parameters Θ

Output: Optimised caching policy $\pi^{\star}$

`/* Data pre-processing */`

Split $\mathcal{R}$ into $\mathcal{R}_{\text{train}}$ and $\mathcal{R}_{\text{test}}$;

Construct normalised adjacency $\mathbf{A} \leftarrow g(\mathcal{R}_{\text{train}})$.

`/* Recommender pre-training */`

Learn LightGCN embeddings $(\mathbf{P}, \mathbf{Q})$ by minimising $\mathcal{L}_{\text{BPR}}(\mathbf{P}, \mathbf{Q}; \mathbf{A})$.

`/* MDP formulation */`

Define $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$ where:

- *State* $s \in \mathcal{S}$: node features of vehicles and the RSU encoded by a two-layer GNN, mean-pooled to $\bar{\mathbf{h}}$;
- *Action* $a \in \mathcal{A}$: continuous scores assigned to a candidate set of k items;
- *Reward* $r(s, a)$: weighted sum of cache-hit indicator and negative delay.

`/* RL optimisation (PPO) */`

Initialise policy parameters θ_0;

for $t = 1$ **to** $T_{\max}$ **do**

- Collect trajectories τ_t under π_{θ_t};
- Estimate advantages $\hat{A}_t$ on τ_t;
- Update θ_{t+1} by maximising the clipped objective $J_{\text{PPO}}(\theta_t)$.

end for

`/* Baselines */`

foreach $\pi^{base} \in \{LRU,\ FIFO,\ MRU,\ Random,\ 2Q\}$ **do**

- Simulate N episodes in $\mathcal{M}$ and record hit ratios.

end foreach

4.6 Experiments

In this section, we present the experiment of our proposed GCNRDRL caching policy and provide a comprehensive illustration of the key components.

4.6.1 Setup

In the experiment, we used Python 3.9.21 and PyTorch 2.6.0 for real-world cross-road simulations and NN training and predictions. We simulated the algorithm on a Macbook Pro (M4 Pro and 24GB memory) running macOS Sequoia 15.3.1 operating system, and a computer (Intel Core Xeno, 32GB memory, and 2 GTX1080 GPUs) running Ubuntu 16.04 LTS operating system.

4.6.2 Dataset

We use the MovieLens-1M (ml-1m)[13] benchmark, which contains 1,000,209 explicit 1–5 ratings with timestamps from 6,040 users on 3,952 movies. Each user has at least 20 ratings. In our setting, users serve as proxies for vehicles: we derive per-vehicle baseline preference vectors from the user–item interactions and use only the interaction matrix (demographic fields are ignored). We apply standard preprocessing (e.g., minimum ratings per user/item and optional caps) to control scale and keep the state dimension consistent.

4.6.3 Simulation Platform

The proposed vehicular caching environment is implemented entirely in Python, designed specifically for this study without reliance on external simulation software. The reinforcement learning components are integrated using the Stable-Baselines3 (SB3) library, providing a reliable and modular framework for training and evaluating reinforcement learning agents. Neural network components, including the pretrained

LightGCN embeddings and Graph Neural Networks, are implemented using PyTorch.

4.6.4 Simulation Parameters

The simulation environment utilizes a series of configurable parameters designed to model realistic vehicular caching scenarios. At the start of each simulation episode, the environment is initialized with 20 vehicles. However, the actual number of vehicles within the RSU's communication range dynamically changes throughout the simulation, influenced by vehicle movement and the spawning mechanism. Specifically, vehicles move with a maximum speed of 1.0 units within a bounded positional area, and new vehicles continuously enter the environment at an average spawn rate of 1.2 vehicles per timestep. As a result, the RSU interacts with a realistic number of vehicles over time.

The RSU itself is configured with a cache capacity of 3 items. Each simulation episode has a maximum duration of 128 timesteps, with each timestep representing an interval of 5 units. Network delay and reward parameters include a baseline communication delay of 1.0 unit, a delay reduction of 0.9 units for cache hits, and an additional delay penalty of 0.5 units for cache misses. Rewards are computed using a base reward of 5 for cache hits and a penalty of -1 for misses, weighted according to experienced delays by a factor of 1. Additionally, an adjustable cache update cost is included, set by default to zero.

Table 4.2: Simulation Environment Configuration Parameters

Parameter	**Value**
Initial number of vehicles	20
Vehicle maximum speed	10m/s
Vehicle spawn rate	1.2/s
RSU cache capacity	3
Episode duration	128
Timestep interval	5s
Baseline communication delay	1.0
Delay reduction (cache hit)	0.9
Delay penalty (cache miss)	0.5
Base reward (cache hit)	5
Penalty (cache miss)	-1
Reward weighting factor	1

4.6.5 Evaluation Metrics

The following three metrics are adopted to evaluate various caching algorithms:

- CHR (Cache Hit Ratio): It defines the percentage of requests that can be satisfied by the cached data packets.
- ALT (Average Latency): It defines the average delay between the time the consumer sends an Interest packet and the time it receives a Data packet. The formula for ALT is given in Eqn. 4.3.
- Rewards: It defines the total reward that RL agent acquired from the training episodes.

4.7 Results

This section shows the experimental results of our GCNRDRL algorithm with MLP, GAT, GCN, Transformer, and caching results from traditional policies under the same configurations. For deep learning-based cache replacement strategies, the models are trained and tested using the same dataset. All deep learning models use the node features of the previous 1 time slots to predict the content caching probability for the next time slot. By default, a time slot is 5 seconds. Therefore, CHR, rewards and ALT are computed 640 seconds after the start of the experiment.

4.7.1 Impact of Zipf Exponent

We examined how the Zipf exponent s influences cache hit ratio, average delay, and the cumulative reward, and why the GCN-based agent outperforms its GAT and Transformer variants. Cache capacity is set to 3 in this experiment.

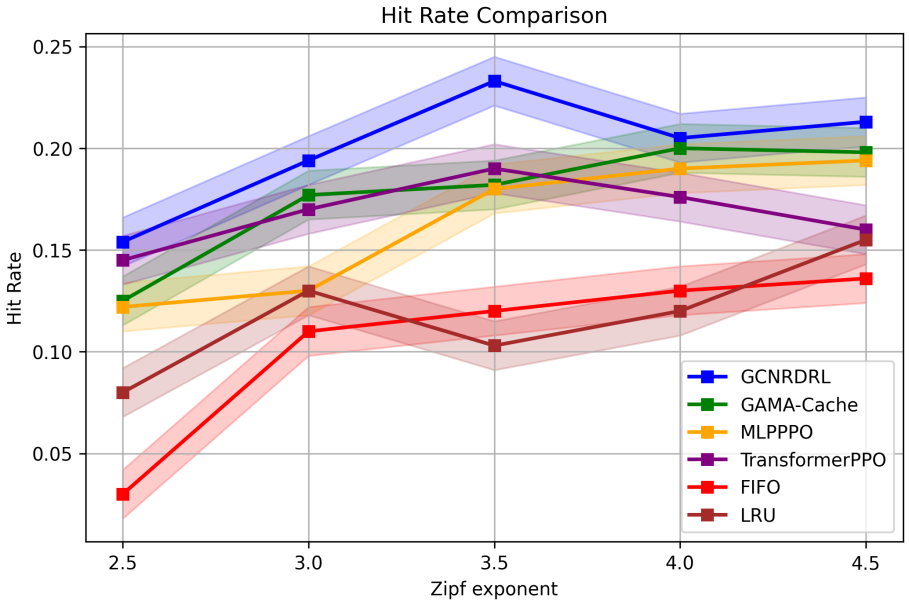


Figure 4.5: The hit rate distribution of GCNDRL, GAT, MLP, Transformer, LRU and FIFO vary with the value of *Zipf* Exponents requested by vehicles covered by each RSU. Results are a 95% confidence interval of across 10 **training** runs.

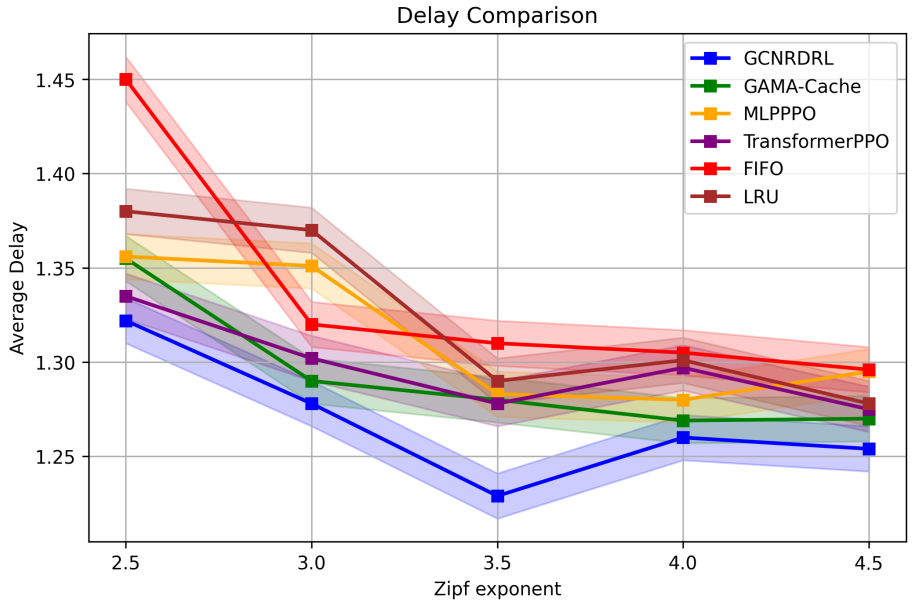


Figure 4.6: The delay distribution of GCNDRL, GAT, MLP, Transformer, LRU and FIFO vary with the value of *Zipf* Exponents requested by vehicles covered by each RSU. Results are a 95% confidence interval of across 10 **training** runs.

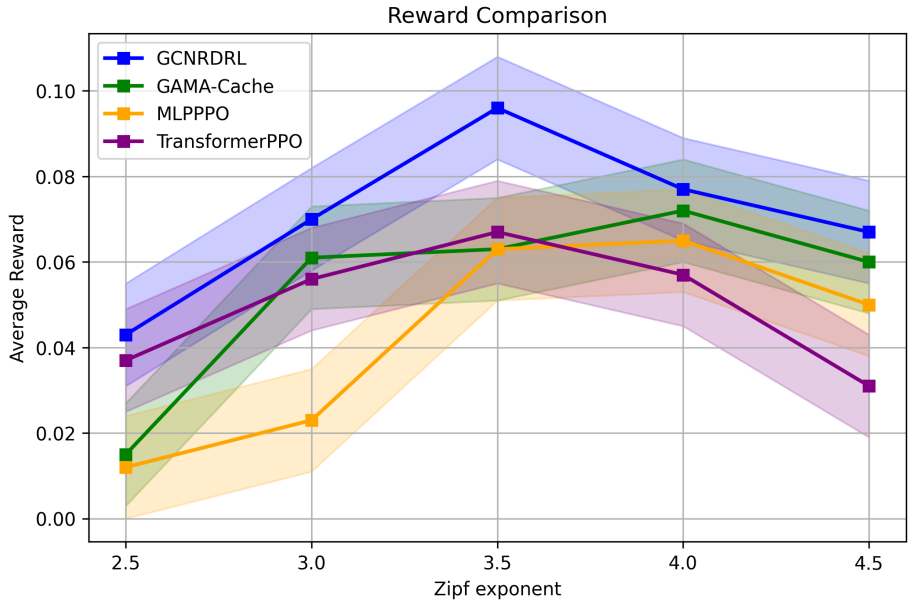


Figure 4.7: The reward distribution of GCNDRL, GAT, MLP, Transformer, LRU and FIFO vary with the value of *Zipf* Exponents requested by vehicles covered by each RSU. Results are a 95% confidence interval of across 10 **training** runs.

Figure 4.7 plots the hit rate as s increases from 2.5 to 4.0. All methods improve—requests concentrate on fewer items—but GCNDRL consistently leads. At $s = 4.0$, GCNRDRL achieves 27.2%, versus 19.4% for GAT and 20.8% for Transformer. The simple neighbor averaging of the GCN encoder produces stable embeddings that smoothly capture the local topology, making caching decisions robust to noisy or transient links. In contrast, GAT's edge-wise attention can overfocuson outliers, and Transformer's self-attention introduces greater variance in the natural featuress of urban and rural crossroads. Figure 4.6 shows the corresponding average delay. GCNDRL again achieves the lowest latency: 1.32 units at $s = 2.5$ down to 1.19 at $s = 4.0$ - while GAT and Transformer remain around 1.29–1.30. The GCN's locality bias helps minimize both cache misses and distance-dependent delays, whereas attention models occasionally misallocate cache space to spurious patterns. Finally, Figure 4.7 reports the episode reward under PPO. GCNDRL secures the highest reward curve across all s, reflecting its superior balance of hits and low delay. Because the GCN backbone produces less noisy state embeddings, the policy learns more consistent value estimates—leading to smoother, higher rewards—whereas GAT and Transformer variants exhibit larger reward fluctuations due to their more flexible but noisier attention weights.

Overall, these results confirm that for IoV caching with skewed request patterns, a well-regularized GCN backbone yields better hit rates, lower delays, and higher RL rewards than more complex attention-based encoders.

4.7.2 Impact of GNN

In this section, we analyze the performance of different caching models, including a baseline MLP and several GNN-based architectures, specifically GCN, GAT, and Transformer. The simulation environment is configured with an RSU cache capacity of 3 items, an initial coverage of 20 vehicles per RSU, and vehicle request frequencies

ranging from a base frequency of 1 to a maximum of 5.

Figure 4.8 depicts the cache hit rates of the evaluated methods. Clearly, all three GNN-based caching strategies significantly outperform the MLP baseline across the entire training duration, demonstrating the advantage of explicitly modeling graph-structured interactions among vehicles and the RSU. Among the GNN variants, GCN consistently achieves the highest cache hit rate, closely followed by GAT, while the Transformer-based GNN performs somewhat less effectively, though still notably better than the MLP. Specifically, the best-performing GCN strategy achieves approximately a 14% improvement in hit rate over the MLP baseline, reflecting its superior ability to aggregate neighboring node information to make informed caching decisions.

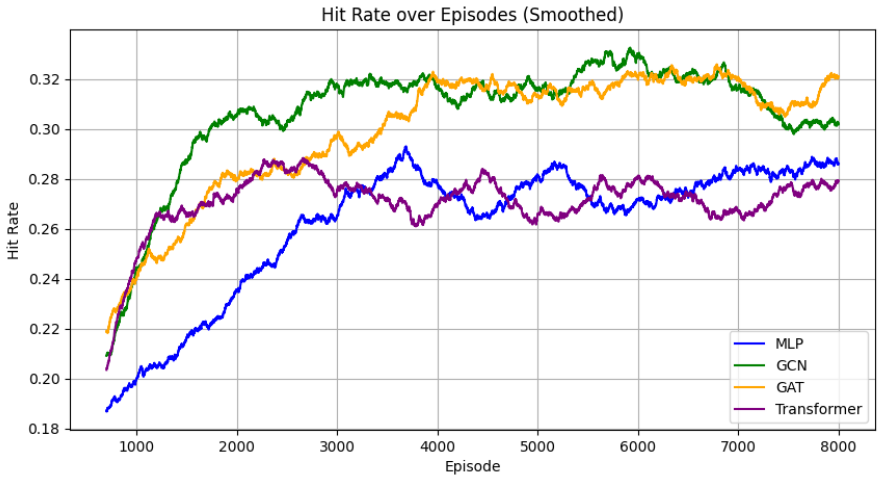


Figure 4.8: Cache hit rate comparison among different caching models.

Similarly, Figure 4.9 presents the average communication delay experienced by vehicles. Consistent with cache hit rate results, all GNN-based methods exhibit noticeably lower delays compared to MLP. The GCN approach achieves the lowest overall delay, highlighting its effectiveness in reducing communication latency through cooperative caching decisions influenced by vehicle interactions. GAT closely matches the delay performance of GCN, while the Transformer, although outperforming MLP, does not achieve as significant a delay reduction as GCN or GAT. Specifically, the

GCN approach reduces delay by approximately 12% compared to MLP.

Figure 4.10 summarizes the average reward performance over training episodes. Once again, the GNN-based approaches clearly surpass the MLP baseline, with GCN and GAT achieving substantially higher cumulative rewards, reflecting their enhanced capability to adaptively optimize caching strategies over episodes. GCN maintains the highest reward throughout training, evidencing stable and consistent improvements in caching efficiency. Overall, the results illustrate that the integration of structural vehicle relationships through GNN architectures substantially improves the effectiveness of caching decisions compared to models without GCN, such as MLP.

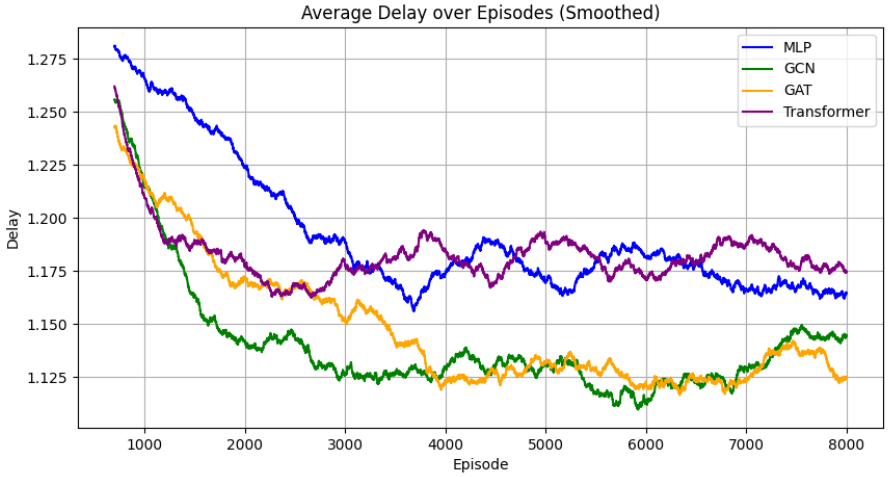


Figure 4.9: Average delay comparison among different caching models.

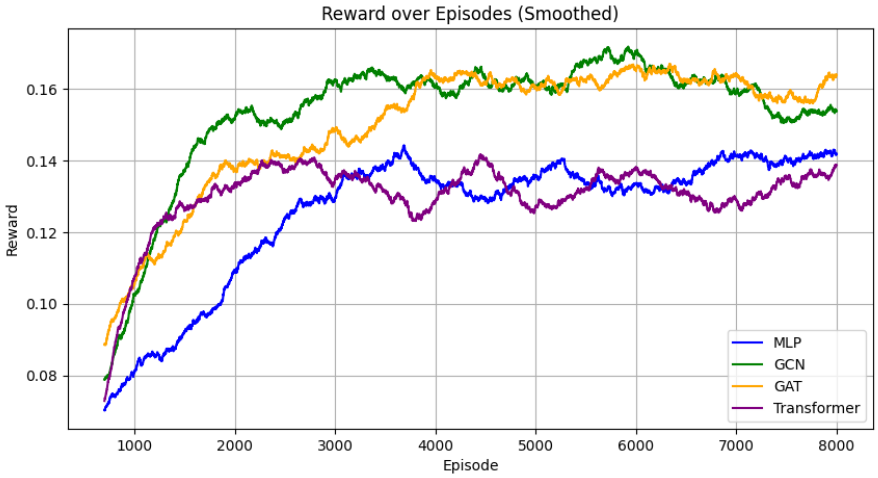


Figure 4.10: Reward performance comparison among different caching models.

These results collectively confirm the superiority of graph-based caching strategies, with GCN providing the best performance across all metrics. This further emphasizes the critical role of effectively leveraging graph-structured vehicle interactions in vehicular content caching scenarios.

Table 4.3: Performance Comparison of Trained RL Agent with Traditional Caching Policies

Caching Policy	**Average Cache Hit Ratio**
RL Agent (GNN-based)	**0.38**
FIFO	0.09
LRU	0.08
MRU	0.07
2Q	0.09
Random	0.10

This table succinctly demonstrates that the trained GCNRDRL algorithm with GCN significantly surpasses classical caching strategies such as FIFO, LRU, MRU, 2Q, and Random, under identical environmental configurations.

4.7.3 Impact of Cache Size

In this section, we explore how different cache sizes affect the performance of the same caching model, specifically using a GAT-based GCNRDRL approach. We conducted experiments over 3200 episodes, each lasting 128 timesteps, comparing two cache capacities: 5 and 8. A larger cache size increases the possible actions the RL agent must explore. Because our simulation environment involves significant randomness, a bigger action space makes it harder and slower for the agent to learn effective caching strategies within a limited training period.

Figure 4.11 compares the cache hit rates for cache sizes of 5 and 8. The results show that the smaller cache size (5 items) reaches a stable and high hit rate more quickly. Although the larger cache size (8 items) theoretically allows better performance, the results fluctuate more and improve slowly, indicating that the model needs more training to fully benefit from the larger action space.

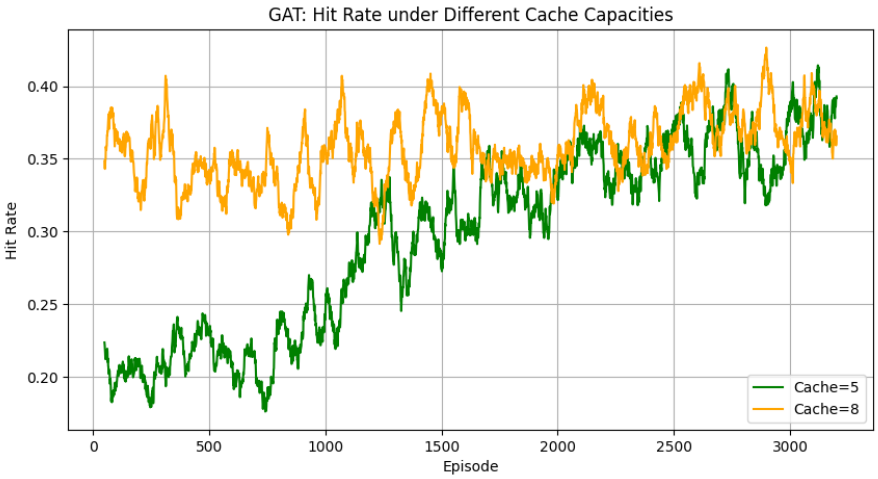


Figure 4.11: GAT Cache Hit Rate under Different Cache Capacities

Figure 4.12 shows the average delays. Here, the smaller cache size leads to faster and more stable delay improvements over timesteps. The larger cache size again performs slightly better overall but struggles with stability and slower improvements due to the difficulty in convergence.

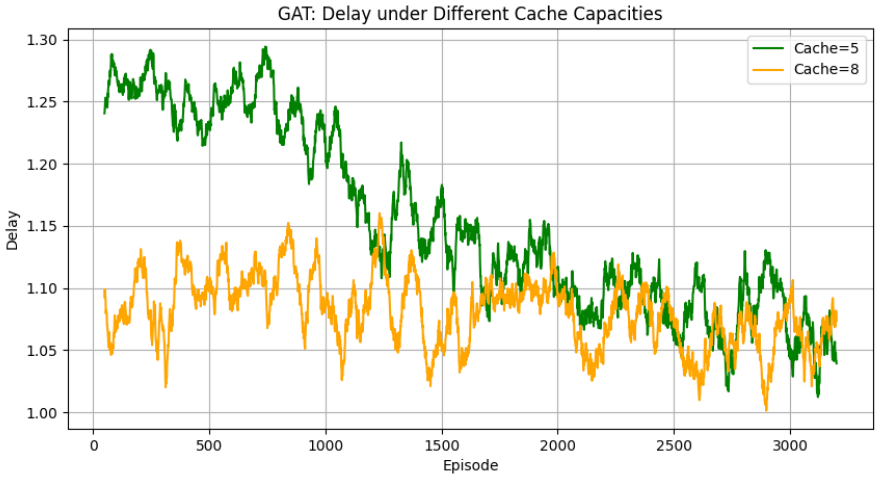


Figure 4.12: GAT Average Delay under Different Cache Capacities

In Figure 4.13, we observe the overall reward achieved by the model for the two cache sizes. Similar to the hit rate and delay, the cache size of 5 demonstrates more rapid and stable reward improvements, while the larger cache size (8 items) improves more slowly and experiences greater fluctuations.

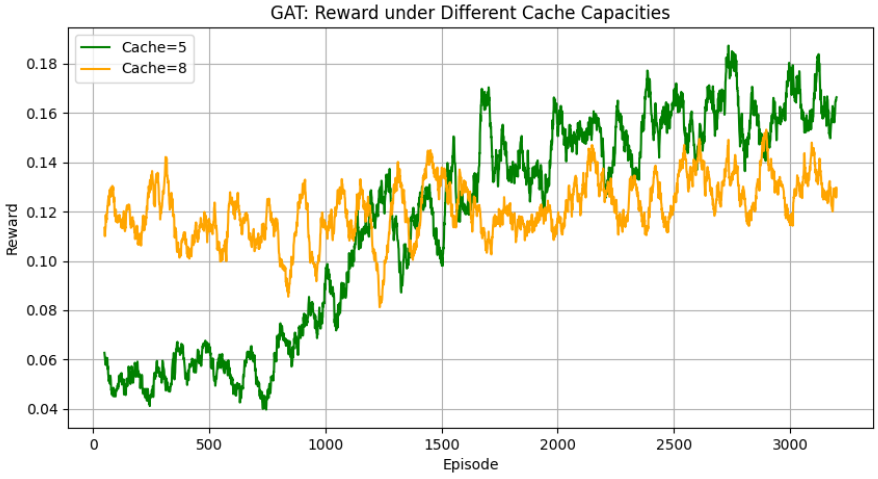


Figure 4.13: GAT Reward under Different Cache Capacities

These results suggest an important practical insight: although a larger cache can potentially improve performance, it also greatly increases the complexity of the learning process. As a result, the GCNRDRL-based caching model tends to perform

best and learn faster in situations where the RSU's caching capacity is relatively small.

4.7.4 Impact of Standard Deviation of GCNRDRL Model Training

In this section, we study the effect of varying the standard deviation parameter (`std`) in the GCNRDRL algorithm, under a fixed configuration using GAT, cache size 3, and a training horizon of 3200 episodes with 128 timesteps each. The standard deviation controls the degree of exploration in the policy's action sampling process—higher values encourage broader exploration, while lower values promote more conservative behavior.

Despite the noticeable difference in learning curves during early training, both settings (`std` = 1.0 and `std` = 1.8) ultimately converge to similar levels in terms of cache hit rate, as shown in Figure 4.14. However, the lower `std` variant exhibits more stable behavior in the later stages, with reduced fluctuation and smoother convergence.

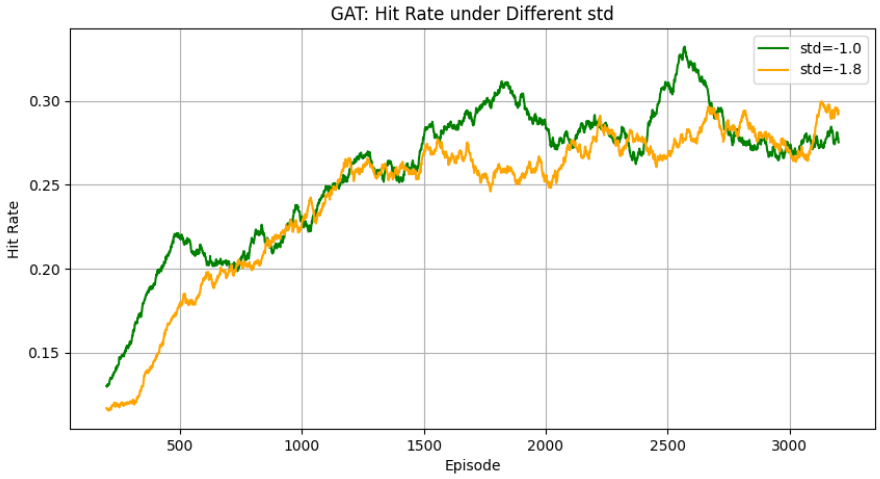


Figure 4.14: GAT: Cache Hit Rate under Different `std` Values

The delay curves in Figure 4.15 support this observation. While both configurations lead to meaningful delay reductions over time, the policy trained with `std` = 1.0

consistently reaches lower average delays and maintains greater stability throughout training. This suggests that once the model has sufficiently explored the environment, a lower exploration variance helps fine-tune caching decisions more precisely.

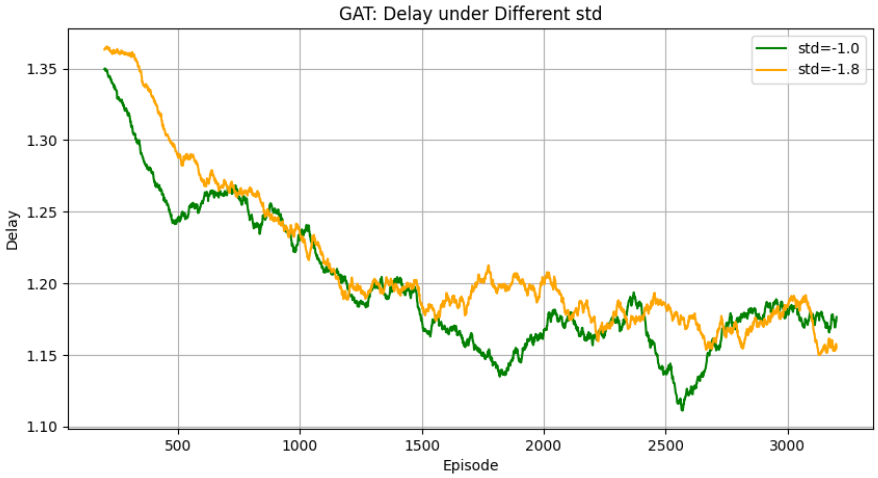


Figure 4.15: GAT: Average Delay under Different `std` Values

Similarly, as shown in Figure 4.16, the overall reward for the `std` = 1.0 configuration grows steadily and converges slightly faster, reinforcing the idea that controlled exploration is beneficial in later stages of training. These results demonstrate that al-

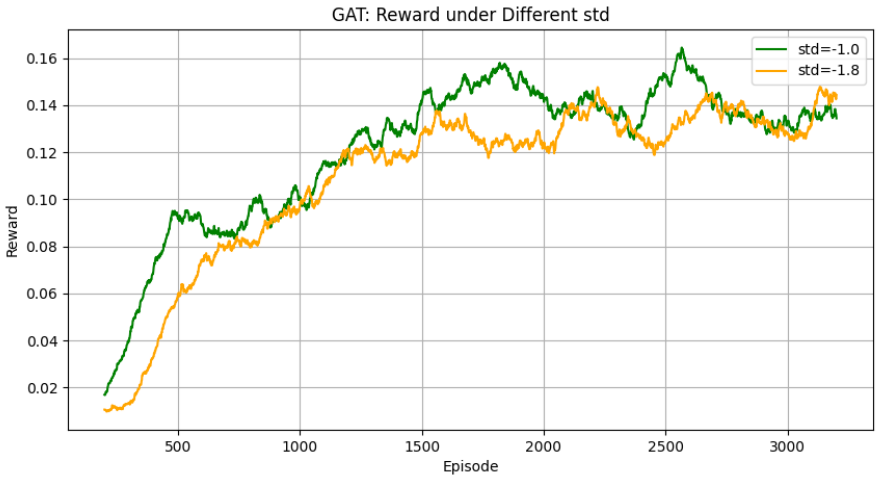


Figure 4.16: GAT: Reward Performance under Different `std` Values

though larger standard deviation values can facilitate early exploration, smaller values

can help stabilize policy behavior and improve convergence quality. This highlights the adaptability of GCNRDRL methods under different learning dynamics, and their ability to balance exploration and exploitation effectively depending on environmental needs.

4.8 Conclusion

In this chapter we presented a GCNRDRL caching framework for IoV roadside units. The approach builds a recommendation-driven candidate set from pretrained user–item embeddings and then employs graph-based policies—GCN, GAT, and Transformer decide which files should be stored. Across 8000 training episodes, the GCN policy raises the cache-hit rate by about 30% and lowers the average delay by roughly 10% compared with a multilayer perceptron baseline, while achieving nearly three times the hit rate of classical rules such as FIFO, LRU, and random. Experiments also reveal that a moderate cache capacity lets the agent converge faster and more smoothly than a larger capacity, and that reducing the exploration standard deviation helps the policy stabilise once the main exploration phase is complete. These findings show that topology-aware reinforcement learning offers an efficient and robust solution for roadside units that must make timely caching decisions with limited storage in dynamic vehicular networks.

Chapter 5

Hierarchical Decision-Making for Joint Caching and Recommendation (HiCaRe-RL)

5.1 Introduction

This chapter focuses on improving caching performance (i.e., maximizing cache hit ratio) in IoV via a hierarchical RL framework, **HiCaRe-RL**, that couples a cache controller with a per-vehicle recommender. The cache policy selects a set with cache capacity size at the RSU, and—conditioned on this set—the recommendation policy proposes cached items for each vehicle, aligning what is stored with what is suggested as shown in Figure 5.1. As a result, more requests are served directly from RSUs rather than the origin server, improving cache space utilization and reducing delivery latency/backhaul.

Unlike GCNRDRL in Chapter 4, which couples a *pretrained, centrally trained* GNN recommender with a PPO cache agent, **HiCaRe-RL** learns both caching and recommendation online and jointly, without requiring any offline recommender or centralized user data. HiCaRe-RL removes the embedding pretraining pipeline and

its privacy burden, and lets the system adapt immediately to non-stationary, time-of-day preferences using only current, local signals. Structurally, HiCaRe-RL enforces recommended items are in cached files, aligning what can be suggested with what is actually stored; this factorization shrinks the effective action space and improves sample efficiency. Practically, we use lightweight MLPs and a fixed-size state (personalized preference summaries, Bloom-encoded cache, presence mask), avoiding graph construction/inference overhead while retaining responsiveness on resource-limited RSU.

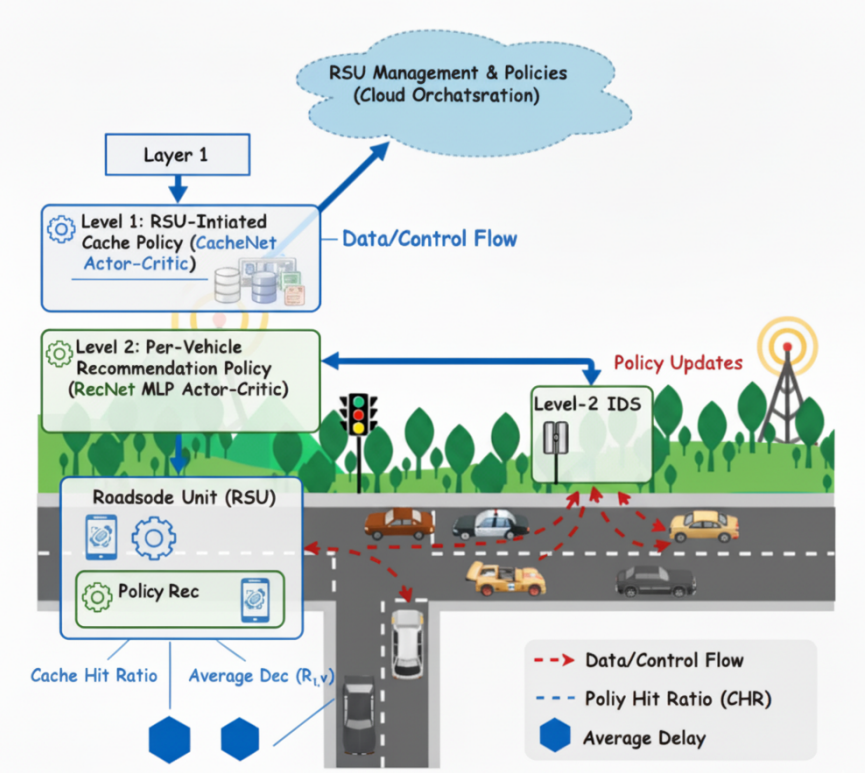


Figure 5.1: HiCaRe-RL Overview

5.2 System Design

We adopt the highway RSU system model described in Section 4: an RSU with cache size C serves a time-varying set of vehicles $\mathcal{V}_t$, whose arrivals follow a Poisson process and whose requests come from catalog $\mathcal{F}$. Each connected vehicle is characterised by

its current delay d_i, available bandwidth B_i, and speed v_i, all of which are updated every time frame and fed into the caching and recommendation algorithm. Moreover, we set a maximum number of vehicles in current RSU coverage area, denoted as V_{max}. To interface a time-varying $|\mathcal{V}_t|$ with fixed-size neural inputs, we maintain a bank of $V_{\max}$ vehicle slots: at each frame t, active vehicles are assigned to slots and the remaining slots are zero-padded. A binary *presence mask* $m_t \in \{0,1\}^{V_{\max}}$ marks which slots are occupied (1) vs. empty (0), and downstream modules ignore padded rows (see the *Presence mask* section in Sec. 5.3.3.3).

5.3 Proposed Methodology

5.3.1 Hierarchical Learning Structure

We adopt a shallow hierarchy with two cooperative RL policies, consisting of π_{cache} and π_{rec}. These two policies act at different granularities and time scales.

5.3.1.1 Notation

Let $\mathcal{F}$ be the available files ($|\mathcal{F}| = F$) and $\mathcal{V}$ the vehicle pool. At step t, the environment provides an observation s_t and a set of active vehicles $\mathcal{V}_t \subseteq \mathcal{V}$. The cache has budget C items and each vehicle may receive up to K recommendations.

5.3.1.2 High-level cache policy

Given the observation s_t at time t, the cache controller π_{cache} at every time t selects a cached set $C_t \subseteq \mathcal{F}$ of fixed size $|C_t| = C$. This decision defines the cached items at the RSU for the next step.

5.3.1.3 Low-level recommendation policy

Conditioned on C_t and s_t, the recommender π_{rec} produces, for each active vehicle $v \in \mathcal{V}_t$, a per-vehicle list $R_{t,v} \subseteq C_t$ with $|R_{t,v}| \leq K$. This matches the cached supply generated by π_{cache} to the instantaneous personalized vehicle demands.

The joint action at step t is $a_t = \big(C_t, \{R_{t,v}\}_{v\in\mathcal{V}_t}\big)$. It induces the system dynamics including requests, hits/misses, latency, and yields feedback for learning.

Our objective is to learn $(\pi_{\text{cache}}, \pi_{\text{rec}})$ that maximize long-term system utility (e.g., cache hit-ratio under latency constraints), while the structural constraint $R_{t,v} \subseteq C_t$ reduces the recommendation action space and aligns local matching with the global cache decision.

5.3.2 Request Pattern

In the proposed HiCaRe-RL environment, each vehicle $v \in \mathcal{V}$ requests one content $f_t \in \mathcal{F}$ at time t based on both vehicle's personalized baseline preference and current recommended items produced by π_{rec}.

Let $\text{base}_{t,v}$ denote the personalized baseline preference of vehicle v at time t which can be instantiated by the preprocessed user–item preference vector. When a vehicle enters the RSU coverage, it receives a per-vehicle recommendation set $R_{t,v}$ from π_{rec}. A scalar recommendation influence $\lambda \in [0, 1]$ controls how much the recommendation shifts the request distribution of $\text{base}_{t,v}$: a larger λ means the vehicle is more likely to follow the recommended items, and vice versa.

At time t, π_{rec} outputs for vehicle v a set of recommended files $R_{t,v} \subseteq \mathcal{F}$ with size $k = |R_{t,v}|$. We define the *bonus* of vehicle v for item f at time t over recommended items as

$$\text{bonus}_{t,v}(f) = \begin{cases} \dfrac{1}{k}, & k > 0 \text{ and } f \in R_{t,v}, \\ 0, & \text{otherwise.} \end{cases}$$

Given an influence weight $\lambda \in [0,1]$, the (unnormalized) mixed preference is

$$\tilde{p}_{t,v}(f) = (1-\lambda)\,\mathrm{base}_{t,v}(f) \;+\; \lambda\,\mathrm{bonus}_{t,v}(f).$$

Finally, we normalize $\tilde{p}_{t,v}$ into a valid probability distribution by

$$p^{\mathrm{req}}_{t,v}(f) = \begin{cases} \dfrac{\tilde{p}_{t,v}(f)}{\sum_{j\in\mathcal{F}} \tilde{p}_{t,v}(j)}, & \sum_j \tilde{p}_{t,v}(j) > 0, \\ \dfrac{1}{|\mathcal{F}|}, & \text{otherwise.} \end{cases}$$

and draw one request f_t for vehicle v by sampling according to this distribution.

5.3.3 Cache Policy State Space

The cache-policy state at time t is

$$s_t \;=\; [\,P_t,\; b_t,\; m_t\,],$$

where each component is fixed-size and captures complementary aspects of the system.

5.3.3.1 Per-vehicle top-K_{pref} preference summary P_t

Let $I_{t,v} = \big(i^{(1)}, \ldots, i^{(K_{\mathrm{pref}})}\big)$ be the indices of the K_{pref} largest entries of $\mathrm{base}_{t,v}$ (sorted by score). We keep two numbers per selected item—the index normalized by the catalog size and its score:

$$P_t[v,:] \;=\; \Big[\, \tfrac{i^{(1)}}{F},\; \mathrm{base}_{t,v}\big(i^{(1)}\big),\; \tfrac{i^{(2)}}{F},\; \mathrm{base}_{t,v}\big(i^{(2)}\big),\; \ldots,\; \tfrac{i^{(K_{\mathrm{pref}})}}{F},\; \mathrm{base}_{t,v}\big(i^{(K_{\mathrm{pref}})}\big) \,\Big] \;\in\; [0,1]^{2K_{\mathrm{pref}}}.$$

Stacking all vehicle rows yields $P_t \in [0,1]^{V_{\max}\times(2K_{\mathrm{pref}})}$. Empty slots are zero-padded.

Hyperparameter K_{pref}. K_{pref} controls how many top items per vehicle are retained (hence $2K_{\text{pref}}$ features per row). Larger K_{pref} preserves more detail but increases the state dimension $V_{\max}\cdot(2K_{\text{pref}})$. In implementation it is set by `top_k_pref` (e.g., K_{pref}=30).

5.3.3.2 Cache status b_t with Bloom Encoding

Let $C_t \subseteq \mathcal{F}$ be the set of files cached at time t (with $|C_t| = C$). We represent it as a fixed-length binary vector

$$b_t \in \{0,1\}^B.$$

Construction (deterministic) is as follows: initialize b_t to all zeros; for each cached item $c \in C_t$ and for each hash index $h \in \{1,\dots,H\}$, compute the position $j = h(c) \bmod B$ and set $b_t[j]$ to 1.

The resulting b_t is fed to the policy network as a length-B feature vector (stored as floats).

Hyperparameters B and H. B is the length of the Bloom vector (number of bits), i.e., the input dimensionality of b_t. H is the number of hash functions (the number of bit positions set per cached item). Both are configurable in our implementation via `bloom_B` and `bloom_H` (defaults B=256, H=4). For a target cache size C, a common sizing rule is

$$H \;\approx\; \frac{B}{C}\ln 2$$

In practice we keep (B, H) fixed across experiments for consistent state dimensionality.

Directly feeding integer cache IDs would inject vague meaning into the input (Suggesting larger IDs equals more importance), and a one-hot cache vector would be F-dimensional and tightly coupled to the catalog size. Bloom encoding provides a compact, fixed-size representation of the *set* C_t that

1. decouples the state dimension from F,

2. preserves set semantics without imposing an artificial order,

3. improves training stability by avoiding ID memorization.

The environment still maintains the exact cache membership internally when enforcing $R_{t,v} \subseteq C_t$ and computing hits/misses.

5.3.3.3 Presence mask m_t.

We include a binary mask $m_t \in \{0,1\}^{V_{\max}}$ indicating which vehicle slots are active at time t; inactive rows in P_t are aligned with zeros according to m_t.

5.3.3.4 MLP input.

The state fed to the cache policy network is the flattened concatenation

$$x_t \;=\; P_t \,\|\, b_t \,\|\, m_t \;\in\; \mathbb{R}^D,$$

with total dimension $D \;=\; V_{\max}{\cdot}(2K_{\text{pref}}) \;+\; B \;+\; V_{\max}$.

5.3.4 Recommendation Policy State Space

At time t, the recommendation policy uses the same fixed-size components as the cache policy, augmented with an explicit cache indicator. We write

$$s_t^{\text{rec}} \;=\; [\, P_t,\; b_t,\; m_t,\; c_t \,],$$

where P_t (per-vehicle top-K_{pref} summary), b_t (Bloom-encoded cache status), and m_t (presence mask) are as defined in the State Space section, and c_t is:

5.3.4.1 Cache Indicator Mask c_t.

Let $C_t \subseteq \mathcal{F}$ be the cached set with $|C_t| = C$. We include a binary indicator $c_t \in \{0,1\}^F$ with $c_t(f) = 1$ if and only if $f \in C_t$ and 0 otherwise (exactly C ones).

5.3.4.2 MLP input.

The input fed to the recommendation network is the flattened concatenation

$$x_t^{\text{rec}} \;=\; \text{vec}(P_t) \parallel b_t \parallel m_t \parallel c_t \;\in\; \mathbb{R}^{D_{\text{rec}}},$$

with total dimension

$$D_{\text{rec}} \;=\; V_{\max}\cdot(2K_{\text{pref}}) \;+\; B \;+\; V_{\max} \;+\; F.$$

5.3.5 Action Space

5.3.5.1 Cache Policy

At time t, the cache policy outputs a per-file score vector

$$a_t \in \mathbb{R}^F.$$

The cached set is obtained by top-C selection:

$$C_t \;=\; \text{Top-}C\big(a_t\big), \qquad C_t \subseteq \mathcal{F},\; |C_t| = C.$$

Order is ignored and items are distinct. (During training we sample without replacement; at evaluation we take the top-C.)

5.3.5.2 Recommendation Policy

At time t, for each active vehicle $v \in \mathcal{V}_t$, the recommendation policy outputs a per-file score vector

$$a_{t,v} \in \mathbb{R}^F.$$

Scores are restricted to cached items via the cache set C_t. The per-vehicle recommendation list is obtained by top-K selection over the cached subset:

$$R_{t,v} \;=\; \text{Top-}K\big(a_{t,v} \text{ on } C_t\big), \qquad R_{t,v} \subseteq C_t, \;\; |R_{t,v}| \le K.$$

Items are distinct and order is ignored. The overall recommendation action is $\{R_{t,v}\}_{v \in \mathcal{V}_t}$.

5.3.6 Reward Space

5.3.6.1 Cache Policy

At step t, let $\text{req}_t \in \mathbb{N}^F$ be the request count vector and C_t the cached set. The cache reward is simply the hit ratio at current step t:

$$r_t^{\text{cache}} \;=\; \frac{\sum_{f \in C_t} \text{req}_t(f)}{\sum_{f \in \mathcal{F}} \text{req}_t(f) + \varepsilon},$$

where ε is a small float avoiding division by zero

5.3.6.2 Recommendation Policy

At step t, let $f_{t,v}$ be the item actually requested by vehicle v. Define a per-vehicle click indicator

$$r_{t,v} \;=\; \begin{cases} 1, & f_{t,v} \in R_{t,v} \cap C_t, \\ 0, & \text{otherwise}, \end{cases} \qquad v \in \mathcal{V}_t.$$

The step reward for the recommendation policy is the mean over active vehicles:

$$r_t^{\text{rec}} \;=\; \frac{1}{|\mathcal{V}_t|}\sum_{v\in\mathcal{V}_t} r_{t,v} \;=\; \frac{\sum_v m_t[v]\; r_{t,v}}{\sum_v m_t[v] + \varepsilon},$$

where $m_t[v] \in \{0,1\}$ indicates whether slot v (same as Section 5.3.3.3) is active and $\varepsilon > 0$ avoids division by zero.

5.3.7 Model Architecture

We employ two separate actor–critic networks for the cache and recommendation policies. Both are MLPs with the same backbone and task-specific heads.

5.3.7.1 Cache Network Architecture

Let $\mathbf{x}_t^{\text{cache}} \in \mathbb{R}^{D_{\text{cache}}}$ be the flattened input $P_t \,\|\, b_t \,\|\, m_t$(from Section 5.3.3.4). The forward pass uses two fully–connected layers:

$$\mathbf{h}^{(1)} = \sigma\big(\mathbf{W}_1\mathbf{x}_t^{\text{cache}} + \mathbf{b}_1\big), \qquad \mathbf{h}^{(2)} = \sigma\big(\mathbf{W}_2\mathbf{h}^{(1)} + \mathbf{b}_2\big),$$

where $\sigma(\cdot)$ is ReLU and the hidden width is H. W and b are MLP learnable weights and bias introduced in Section 2.2.

The actor head outputs per–file logits and the critic head outputs a scalar state value:

$$\boldsymbol{\ell}_t^{\text{cache}} = \mathbf{W}_a\mathbf{h}^{(2)} + \mathbf{b}_a \in \mathbb{R}^F, \qquad V_{\text{cache}}(s_t) = \mathbf{w}_v^\top\mathbf{h}^{(2)} + b_v \in \mathbb{R}.$$

At training we sample C items without replacement from $\boldsymbol{\ell}_t^{\text{cache}}$ (Gumbel–Top$C$[30]); at evaluation we take $C_t = \text{Top-}C(\boldsymbol{\ell}_t^{\text{cache}})$.

5.3.7.2 Recommendation Network Architecture

Let $\mathbf{x}_t^{\text{rec}} \in \mathbb{R}^{D_{\text{rec}}}$ be $P_t \,\|\, b_t \,\|\, m_t \,\|\, c_t$ (from Section 5.3.4.2), where $c_t \in \{0,1\}^F$ indicates cache membership. The backbone is identical:

$$\mathbf{h}^{(1)} = \sigma\big(\mathbf{W}_1' \mathbf{x}_t^{\text{rec}} + \mathbf{b}_1'\big), \qquad \mathbf{h}^{(2)} = \sigma\big(\mathbf{W}_2' \mathbf{h}^{(1)} + \mathbf{b}_2'\big).$$

The actor head produces per–vehicle, per–file logits and the critic head produces per–vehicle values:

$$\mathbf{L}_t = \text{reshape}\big(\mathbf{W}_a' \mathbf{h}^{(2)} + \mathbf{b}_a',\ V_{\max}, F\big) \in \mathbb{R}^{V_{\max} \times F}, \qquad \mathbf{v}_t = \mathbf{W}_v' \mathbf{h}^{(2)} + \mathbf{b}_v' \in \mathbb{R}^{V_{\max}}.$$

To enforce $R_{t,v} \subseteq C_t$, we mask non–cached items:

$$\tilde{\mathbf{L}}_t[v,f] = \begin{cases} \mathbf{L}_t[v,f], & c_t(f) = 1, \\ -\infty, & c_t(f) = 0. \end{cases}$$

For each active vehicle v, training samples K distinct items via Gumbel–Top$K$ from $\tilde{\mathbf{L}}_t[v,:]$; evaluation uses $R_{t,v} = \text{Top-}K\big(\tilde{\mathbf{L}}_t[v,:]\big)$.

5.4 Algorithm

Algorithm 2: HiCaRe-RL (Stage A): Cache Policy Training

Input: Catalog size (F), cache size (C), vehicle cap ($V_{\max}$), episodes (E_{cache}), discount (γ)

Output: Trained cache policy $\pi^{\star}_{\text{cache}}$

Init. Build dynamic env; derive baseline preferences; init MLP actor–critic (cache);

for $ep = 1$ **to** E_{cache} **do**

  Reset env; disable recommendation (K=0);

  while *not done* **do**

    $x_t \leftarrow \text{vec}(P_t) \parallel b_t \parallel m_t$;

    $(\ell_t^{\text{cache}},\ V_t) \leftarrow \text{CacheNet}(x_t)$;

    $C_t \leftarrow \text{Gumbel-Top}C(\ell_t^{\text{cache}})$;

    Step env with C_t, get r_t^{cache}, next state s_{t+1};

    $\hat{y}_t \leftarrow r_t^{\text{cache}} + \gamma\, V_{\text{cache}}(s_{t+1})$;

    A2C update on cache net with advantage $A_t = \text{norm}(\hat{y}_t) - V_t$;

  end while

end for

return $\pi^{\star}_{\text{cache}}$ (freeze for Stage B)

Algorithm 3: HiCaRe-RL (Stage B): Recommendation Policy Training (Cache Fixed)

Input: Frozen ($\pi^{\star}_{\text{cache}}$), rec budget ($K$), influence ($\lambda$), episodes ($E_{\text{rec}}$)

Output: Trained recommendation policy $\pi^{\star}_{\text{rec}}$

Init. Init MLP actor–critic (rec);

for $ep = 1$ **to** E_{rec} **do**

  Reset env;

  while *not done* **do**

    $x_t \leftarrow \text{vec}(P_t) \parallel b_t \parallel m_t$;

    $C_t \leftarrow \text{Top-}C(\text{CacheNet}(x_t).\text{actor})$;

    $x_t^{\text{rec}} \leftarrow \text{vec}(P_t) \parallel b_t \parallel m_t \parallel c_t$;

    $(\mathbf{L}_t,\ \mathbf{V}_t) \leftarrow \text{RecNet}(x_t^{\text{rec}})$; mask $\mathbf{L}_t$ by c_t;

    for *each active vehicle* v **do**

      $R_{t,v} \leftarrow \text{Gumbel-Top}K(\mathbf{L}_t[v,:])$

    end for

    Step env with $\{C_t, \{R_{t,v}\}\}$, get per-vehicle $r_{t,v} \in \{0,1\}$ and mask m_t;

    A2C update on rec net with masked advantages

    $A_{t,v} = \text{norm}(r_{t,v}) - \mathbf{V}_t[v]$;

  end while

end for

return $\pi^{\star}_{\text{rec}}$

Algorithm 4: HiCaRe-RL (Deployment/Evaluation): Joint Execution

Input: $(\pi^{\star}_{\text{cache}},\ \pi^{\star}_{\text{rec}})$

Output: Cache hit ratio (CHR), average delay

for *eval episodes* **do**
  Reset env;
  while *not done* **do**
    $x_t \leftarrow \text{vec}(P_t) \parallel b_t \parallel m_t$; $C_t \leftarrow \text{Top-}C(\text{CacheNet}(x_t).\text{actor})$;
    $x_t^{\text{rec}} \leftarrow \text{vec}(P_t) \parallel b_t \parallel m_t \parallel c_t$; $\mathbf{L}_t \leftarrow \text{RecNet}(x_t^{\text{rec}}).\text{actor}$ (masked by c_t);
    For active v: $R_{t,v} \leftarrow \text{Top-}K(\mathbf{L}_t[v,:])$;
    Step env with $\{C_t, \{R_{t,v}\}\}$; accumulate CHR & average delay;
  end while
end for
return metrics

5.4.1 Hyper-parameter settings

Notes. Bloom (B, H) are kept fixed to maintain a constant state dimension. Delay parameters are logged for analysis and do not enter the reward. We use stochastic Gumbel–Top$K$[30] sampling during training and deterministic Top-K at evaluation for reproducibility.

5.5 Experiments

This section evaluates the proposed *hierarchical* caching–recommendation scheme with two online RL agents: an upper-level cache policy π_{cache} and a lower-level recommendation policy π_{rec}. We focus on two influential factors—**cache capacity** C and **maximum serviced vehicles** $V_{\max}$—and report two end metrics: **Cache Hit Ratio (CHR)** and **Average Delay**.

Table 5.1: Environment & state hyper-parameters.

Category	Hyper-parameter	Value
Catalog	Number of files F	dataset-dependent
	Cache capacity C	varied
Vehicles	Max vehicles per step $V_{\max}$	varied
	Steps per episode	256
	Spawn rate	1.2
Preferences	Top-K_{pref} per vehicle	30
	Update rate η	0.05
	Rec influence λ	0.35
Recommendation	Max rec per vehicle K	varied
Cache encoding	Bloom bits B	256
	Hash functions H	4
Delay (log only)	Base delay d_{base}	1
	Hit reduction	0.8
	Miss penalty	0.5

Table 5.2: Policy networks & training hyper-parameters (A2C, MLP).

Category	Hyper-parameter	Value
Architecture	Hidden width H	128
	Activation σ	ReLU
	Cache actor output	$\mathbb{R}^F$
	Rec actor output	$\mathbb{R}^{V_{\max} \times F}$ (masked by c_t)
	Cache critic	scalar $V_{\text{cache}}(s_t)$
	Rec critic	per-vehicle $V_{\text{rec}}(s_t)[v]$
Sampling	Selection (train)	Gumbel–Top$C$/Top$K$ (w/o replacement)
	Selection (eval)	Top-C/Top-K
Optimization	Optimizer	Adam
	LR (cache)	1×10^{-4}
	LR (rec)	5×10^{-5}
	Discount γ	0.9
	Value loss coeff v_{coef}	0.5
	Entropy coeff (rec)	1×10^{-3}
	Grad clip (global-norm)	1.0

5.5.1 Setup

The implementation stack (Python/PyTorch) and simulator are identical to the previous chapter; the same filtered MovieLens-1M subset initializes vehicle preference vectors. We do not repeat general hardware or software details here.

5.5.2 Simulation Platform

The proposed vehicular caching environment is also implemented in Python with the same structure, yielding a consistent study on dynamic vehicular environment. However, the reinforcement learning agents and encoders are not based on SB3 since Hierarchical framework is not well integrated in SB3. Neural network components are implemented using PyTorch. We use the same dataset, MovieLens 1M, as in Chapter 4.

5.5.3 Simulation and Training

Each time frame serves up to $V_{\max}$ vehicles; every active vehicle issues one request drawn from the mixture. Unless stated otherwise, the recommendation budget is K=1, an episode has 256 steps, and the preference smoothing rate is η=0.05.

We keep training minimal and two-staged:

1. **Stage 1 (cache).** Recommendations are disabled (k=0 and `auto_fill_rec`=`False`); π_{cache} is trained with A2C using the *per-step CHR* as reward.

2. **Stage 2 (recommendation).** The trained cache policy is frozen, λ is set to 0.35, and π_{rec} is trained with *masked logits*, i.e., recommending only within the cached set.

5.5.4 Baselines and Metrics

We include the DSAC model (Section 3.4.2) as a learning-based and recommendation influenced baseline. However, DSAC was originally designed for a static vehicular group, where the environment does not change over time. As a result, it is suboptimal under the highly dynamic mobility setting considered in our experiments.

All DSAC hyperparameters (learning rate, temperature parameter α, critic update rate, and target network update rate) follow the original paper.

We compare against classical, non-RL cache replacement policies: **LRU**, **FIFO**, and **Random**. Heuristic baselines never issue recommendations (k=0), hence λ is inert for them. In contrast to heuristic baselines, DSAC does issue recommendations (k=1 as in our HiCaRe-RL setup). We report:

- **CHR**: fraction of requests served from the RSU cache, averaged over the episode;
- **Average delay**: the simulator's step-wise average delay, averaged over the episode (cache hits reduce delay per the environment parameters; misses incur the miss penalty).

5.5.5 Evaluation Protocol

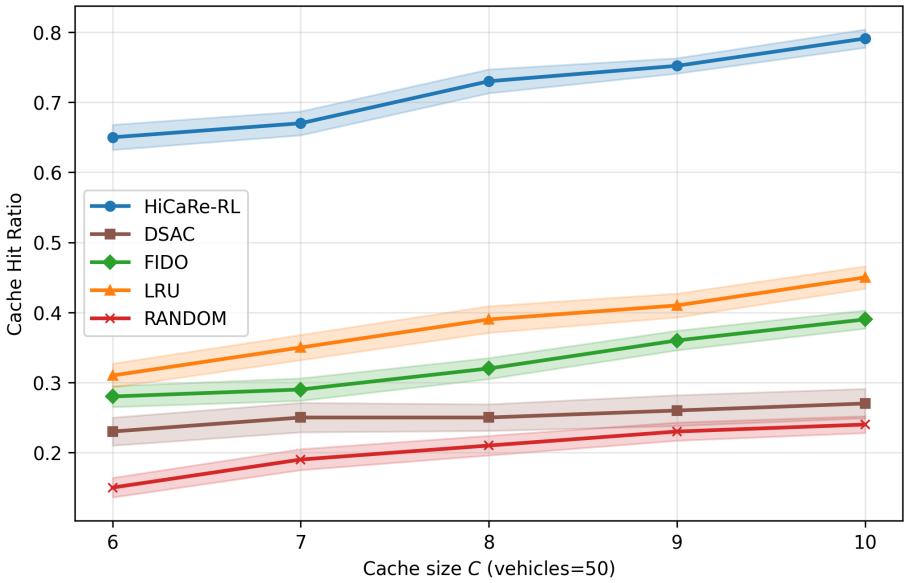
We conduct two controlled sweeps and compare **HiCaRe-RL**, **DSAC**, **LRU**, **FIFO**, and **Random**:

1. **Policy-C**: fix $V_{\max}$=50, vary $C \in \{6, 7, 8, 9, 10\}$; report CHR and average delay (Fig. 5.2).
2. **Policy-V**: fix C=10, vary $V_{\max} \in \{10, 20, 30, 40, 50\}$; report CHR and average delay (Fig. 5.3).

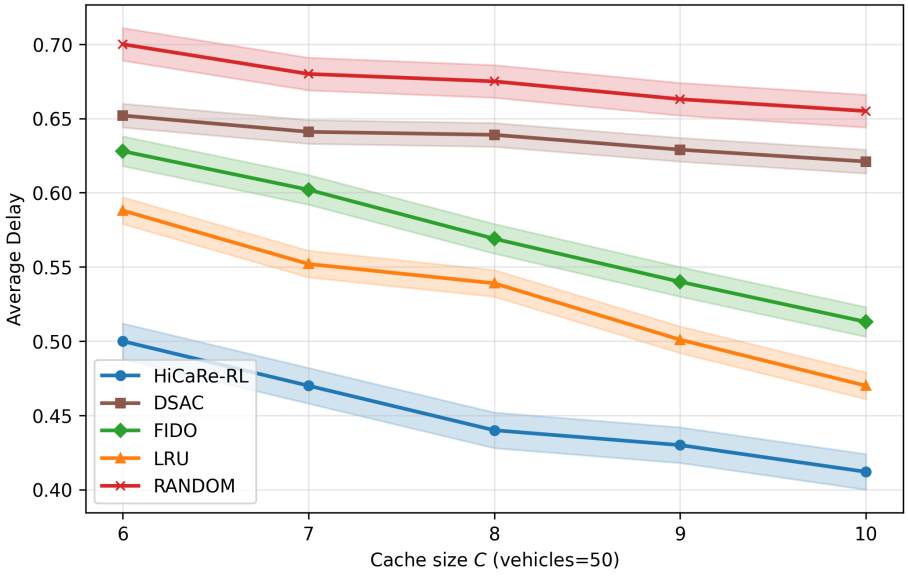
Apart from the variable above, all simulator settings are kept at their defaults.

5.5.6 Results

In Fig. 5.2(a), the cache hit ratio increases for all methods as C grows, with HiCaRe-RL remaining above DSAC, LRU, FIFO, and Random. The improvement tapers off at larger C, indicating diminishing marginal coverage. Fig. 5.2(b) shows that average delay decreases as C increases; HiCaRe-RL attains the lowest delay, consistent with a greater share of local hits.



(a) CHR vs. cache size C ($V_{\max}$=50)



(b) Average delay vs. C ($V_{\max}$=50)

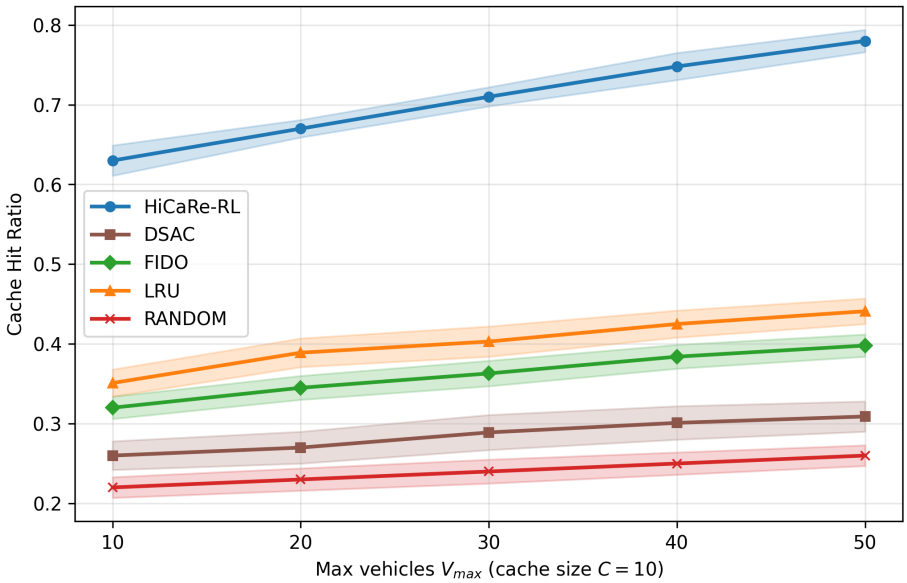
Figure 5.2: Cache Policy (Policy-C): fixed vehicle cap, vary cache capacity.

In Fig. 5.3(a), larger $V_{\max}$ spreads demand and lowers the hit ratio for the baselines, while HiCaRe-RL is more robust due to shaping requests within the cached set. Fig. 5.3(b) shows delay rising with load for the baselines; the HiCaRe-RL curve is flatter, reflecting steadier local service.

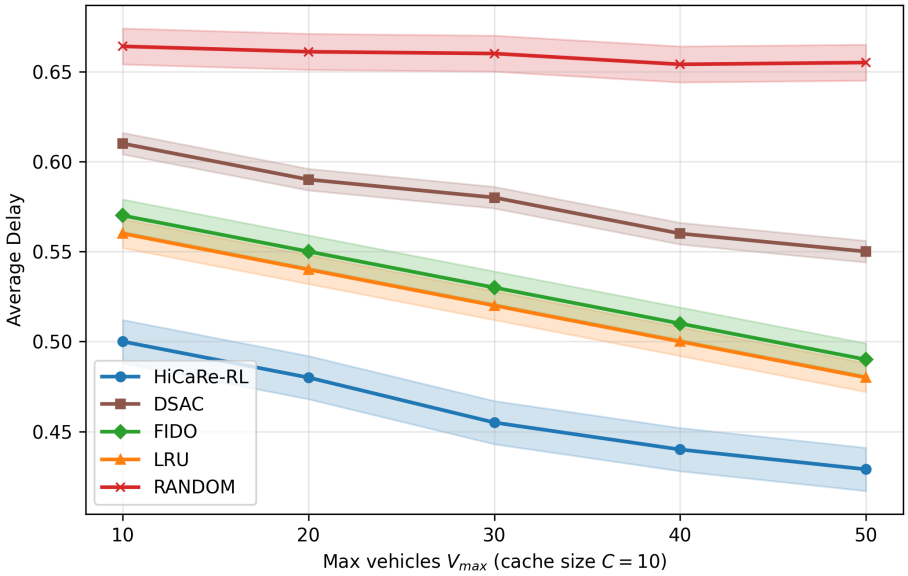
DSAC performs worse than HiCaRe-RL in all scenarios because its environment

formulation assumes a static set of vehicles and fixed channel conditions. When extended to a dynamic IoV environment with continuous vehicle arrivals and departures, the DSAC policy fails to adapt, and its performance becomes comparable to or worse than classical heuristics.

This behavior stems from DSAC's reliance on stable, stationary transition dynamics for soft Q-value estimation. In our dynamic IoV simulator, the composition of vehicles, channel conditions, and request distributions changes every frame, causing DSAC's critic to chase a moving target. The resulting Q-value drift leads to unreliable recommendations and unstable cache decisions, explaining why DSAC underperforms even simple heuristics despite being a learning-based method.



(a) CHR vs. $V_{\max}$ (C=10)



(b) Average delay vs. $V_{\max}$ (C=10)

Figure 5.3: Vehicle-cap Policy (Policy-V): fixed cache capacity, vary $V_{\max}$.

5.5.7 Reproducibility

Hyperparameters follow the previous section. Each stage trains for 300–600 episodes (256 steps/episode); learning rates: $\pi_{cache} = 1 \times 10^{-4}$, $\pi_{rec} = 5 \times 10^{-5}$, entropy coefficient 10^{-3}, value loss coefficient 0.5, and gradient clipping at 1.0. Evaluation

uses the same simulator setup and seeds for all methods. Baselines never recommend (k=0), so comparisons are not confounded by λ.

5.6 Conclusion

This chapter presented HiCaRe-RL, a hierarchical framework for IoV roadside units that couples a high-level cache controller with a low-level per-vehicle recommender. The cache policy selects a size-C set to store at the RSU, while the recommendation policy—constrained to that set ($R_{t,v} \subseteq C_t$)—shapes per-vehicle requests via a simple mixture model with influence weight λ. To remain RSU-friendly, we used fixed-size states (per-vehicle top-K_{pref} summaries P_t, Bloom-encoded cache bitmap b_t, presence mask m_t), lightweight MLP actor–critics with masked logits, and Gumbel–Top$K$ sampling; training proceeds in two stages (cache first, then recommendation).

Under varied cache capacity C and vehicle cap $V_{\max}$, the hierarchical approach consistently outperforms traditional caching strategies (LRU, FIFO, Random), achieving on average **96%** higher cache-hit ratio and **19%** lower latency compared to these heuristics. In addition, HiCaRe-RL remains more robust than the learning-based DSAC baseline. Although DSAC also produces recommendations, its Soft Actor–Critic backbone assumes stationary transition dynamics. Under continuously changing vehicle populations and preference vectors, DSAC's soft Q-values become unstable, leading to degraded caching and recommendation performance that can fall below simple heuristics.

These gains arise from aligning what is recommended with what is actually stored, which shrinks the action space, improves credit assignment, and adapts online to non-stationary preferences without any pretrained, centralized recommender.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we addressed the challenge of intelligent caching in IoV by proposing two reinforcement learning–based approaches that integrate recommendation and caching.

The first contribution, GCNRDRL, combines a pretrained GCN-based recommender with a reinforcement learning cache agent. By leveraging structured user–item interactions and graph-based embeddings, GCNRDRL enables RSUs to adaptively manage their caches under highly dynamic vehicular conditions. Experimental results demonstrated that GCNRDRL not only outperforms traditional caching baselines such as LRU, FIFO, and Random, but also surpasses several state-of-the-art neural architectures, including GAT, Transformer, and MLP. Specifically, GCNRDRL improves cache hit ratio by 5.7%, 92%, 40%, 62%, and 70% compared with TransformerPPO, GAMA-Cache, MLPPPO, LRU, and FIFO, respectively. The GCN backbone consistently achieves the highest cache hit ratio, the lowest average delay, and the most stable reward curves. This advantage stems from the inherent locality bias of GCN aggregation, which yields robust embeddings and facilitates more stable reinforcement learning compared with higher-variance attention-based or fully-connected models.

The second contribution, HiCaRe-RL, introduces a hierarchical framework that couples a high-level cache controller with a low-level recommendation policy. This design aligns the cached set with per-vehicle suggestions, shrinking the effective action space and improving training efficiency. Unlike GCNRDRL, HiCaRe-RL removes the need for offline pretrained embeddings from recommender, making it more suitable for online adaptation to non-stationary user preferences. Experiments confirmed that HiCaRe-RL consistently outperforms traditional baselines, achieving on average a 96% improvement in cache hit ratio and a 19% reduction in latency compared to LRU, FIFO, and Random.

In addition to classical baselines, we also compared both methods against the recent DSAC caching framework [5]. Consistent with findings reported in prior work, DSAC suffers from unstable convergence and reduced sample efficiency under highly dynamic vehicular mobility. Our results confirm that HiCaRe-RL achieves higher cache hit ratios and lower latency across all tested scenarios.

Overall, this thesis demonstrates that integrating recommender systems with reinforcement learning can substantially improve content dissemination efficiency in vehicular networks. GCNRDRL highlights the benefits of topology-aware embeddings in stabilizing RL-based caching, while HiCaRe-RL further enhances adaptability through hierarchical joint optimization. Nevertheless, both methods still rely on semi-online initialization and focus on single-RSU settings, leaving opportunities for further improvement.

6.2 Future Work

Several promising directions can extend this research:

- **Fully online learning.** Current methods rely on semi-online initialization, where user preferences must be known in advance. In real-world IoV scenarios, vehicles may enter coverage without prior information, creating a cold-start

issue. Future work can explore fully online reinforcement learning that continuously infers and adapts user preferences during interaction, eliminating dependence on historical data.

- **Multi-RSU cooperation.** This thesis focused on a single RSU with star topology. However, in practice, neighboring RSUs can cooperate to reduce redundant caching, balance network load, and improve global hit ratios. Extending the framework with multi-agent reinforcement learning (MARL) would enable distributed yet coordinated caching and recommendation strategies.

- **More realistic IoV environments.** Future evaluations can incorporate heterogeneous bandwidth, dynamic mobility patterns, and QoE metrics under large-scale vehicular scenarios. Such extensions would further validate the robustness of the proposed methods in complex real-world settings.

By addressing these directions, future research can build upon GCNRDRL and HiCaRe-RL to achieve more adaptive, scalable, and cooperative caching solutions for next-generation IoV networks.

References

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017. ISSN 1053-5888. doi: 10.1109/msp.2017.2743240. URL http://dx.doi.org/10.1109/MSP.2017.2743240.

[2] Kavosh Asadi and Michael L Littman. An alternative softmax operator for reinforcement learning. In *International Conference on Machine Learning*, pages 243–252. PMLR, 2017.

[3] EN Barron and H Ishii. The bellman equation for minimizing the maximum cost. *Nonlinear Anal. Theory Methods Applic.*, 13(9):1067–1090, 1989.

[4] Dezhi Chen, Qi Qi, Qianlong Fu, Jingyu Wang, Jianxin Liao, and Zhu Han. Transformer-based reinforcement learning for scalable multi-uav area coverage. *Trans. Intell. Transport. Sys.*, 25(8):10062–10077, August 2024. ISSN 1524-9050. doi: 10.1109/TITS.2024.3358010. URL https://doi.org/10.1109/TITS.2024.3358010.

[5] Zhipeng Cheng, Lu Liu, Minghui Liwang, Ning Chen, and Xuwei Fan. Learning-based joint recommendation, caching, and transmission optimization for co-operative edge video caching in internet of vehicles. *Ad Hoc Netw.*, 166(C), January 2025. ISSN 1570-8705. doi: 10.1016/j.adhoc.2024.103667. URL https://doi.org/10.1016/j.adhoc.2024.103667.

[6] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

[7] Chris Dann, Yishay Mansour, Mehryar Mohri, Ayush Sekhari, and Karthik Sridharan. Guarantees for epsilon-greedy reinforcement learning with function approximation. In *International conference on machine learning*, pages 4666–4689. PMLR, 2022.

[8] Getu Fellek, Ahmed Farid, Goytom Gebreyesus, and Shigeru Fujimura. Graph transformer with reinforcement learning for vehicle routing problem. *IEEJ Transactions on Electrical and Electronic Engineering*, 18, 02 2023. doi: 10.1002/tee.23771.

[9] Xiangyu Gao, Yaping Sun, Hao Chen, Xiaodong Xu, and Shuguang Cui. Joint computing, pushing, and caching optimization for mobile-edge computing networks via soft actor–critic learning. *IEEE Internet of Things Journal*, 11(6): 9269–9281, 2023.

[10] Jiayi Guan, Guang Chen, Jin Huang, Zhijun Li, Lu Xiong, Jing Hou, and Alois Knoll. A discrete soft actor-critic decision-making strategy with sample filter for freeway autonomous driving. *IEEE Transactions on Vehicular Technology*, 72 (2):2593–2598, 2023. doi: 10.1109/TVT.2022.3212996.

[11] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[12] Seungyul Han and Youngchul Sung. A max-min entropy framework for reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 25732–25745, 2021.

[13] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4): 19:1–19:19, December 2015. doi: 10.1145/2827872.

[14] Di He, Wei Chen, Liwei Wang, and Tie-Yan Liu. Online learning for auction mechanism in bandit setting. *Decision Support Systems*, 56:379–386, 2013. ISSN 0167-9236. doi: https://doi.org/10.1016/j.dss.2013.07.004. URL https://www.sciencedirect.com/science/article/pii/S0167923613001899.

[15] Qiang He, Yu Wang, Xingwei Wang, Weiqiang Xu, Fuliang Li, Kaiqi Yang, and Lianbo Ma. Routing optimization with deep reinforcement learning in knowledge defined networking. *IEEE Transactions on Mobile Computing*, 23(2):1444–1455, February 2024. ISSN 1536-1233. doi: 10.1109/TMC.2023.3235446. URL https://doi.org/10.1109/TMC.2023.3235446.

[16] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[17] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020. URL https://arxiv.org/abs/2002.02126.

[18] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, CSCW '00, page 241–250, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132220. doi: 10.1145/358916.358995. URL https://doi.org/10.1145/358916.358995.

[19] Jiacheng Hou, Huanzhang Xia, Haoye Lu, and Amiya Nayak. A gnn-based approach to optimize cache hit ratio in ndn networks. In *2021 IEEE Global*

Communications Conference (GLOBECOM), pages 1–6, 2021. doi: 10.1109/GLOBECOM46510.2021.9685872.

[20] Jiacheng Hou, Haoye Lu, and Amiya Nayak. Gnn-gm: A proactive caching scheme for named data networking. In *2022 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2022. doi: 10.1109/ICCWorkshops53468.2022.9882153.

[21] Jiacheng Hou, Huanzhang Xia, Haoye Lu, and Amiya Nayak. A graph neural network approach for caching performance optimization in ndn networks. *IEEE Access*, 10:112657–112668, 2022. doi: 10.1109/ACCESS.2022.3217236.

[22] Jiacheng Hou, Tianhao Tao, Haoye Lu, and Amiya Nayak. An optimized gnn-based caching scheme for sdn-based information-centric networks. In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, pages 401–406, 2023. doi: 10.1109/GLOBECOM54140.2023.10437541.

[23] Jiacheng Hou, Tianhao Tao, Haoye Lu, and Amiya Nayak. Intelligent caching with graph neural network-based deep reinforcement learning on sdn-based icn. *Future Internet*, 15(8), 2023. ISSN 1999-5903. doi: 10.3390/fi15080251. URL https://www.mdpi.com/1999-5903/15/8/251.

[24] Jiacheng Hou, Mourad Elhadef, and Amiya Nayak. Icn-enabled intelligent edge caching strategies for optimizing iot services. *SSRN Electronic Journal*, 2024. doi: 10.2139/ssrn.5031769. URL https://ssrn.com/abstract=5031769.

[25] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 665–674, New York, NY, USA, 2021. Association for Computing Machinery.

ISBN 9781450383325. doi: 10.1145/3447548.3467408. URL https://doi.org/10.1145/3447548.3467408.

[26] Amjad Iqbal, Mau-Luen Tham, and Yoong Chang. Convolutional neural network-based deep q-network (cnn-dqn) resource management in cloud radio access network. *China Communications*, PP:1–14, 10 2022. doi: 10.23919/JCC.2022.00.025.

[27] Sabrina Jiang. A simple neural network, 2021. URL https://www.investopedia.com/terms/n/neuralnetwork.asp.

[28] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

[29] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[30] Wouter Kool, Herke van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement, 2019. URL https://arxiv.org/abs/1903.06059.

[31] Jurij Leskovec. Machine learning with graphs, 2021. URL http://web.stanford.edu/class/cs224w/.

[32] Bingjun Li and Sheida Nabavi. A multimodal graph neural network framework for cancer molecular subtype classification. *BMC Bioinformatics*, 25, 01 2024. doi: 10.1186/s12859-023-05622-4.

[33] Rui Liu, Pengwei Xing, Zichao Deng, Anran Li, Cuntai Guan, and Han Yu. Federated graph neural networks: Overview, techniques, and challenges. *IEEE Transactions on Neural Networks and Learning Systems*, 36(3):4279–4295, 2025. doi: 10.1109/TNNLS.2024.3360429.

[34] Jun Luo, Fenglian Li, and Jiangli Jiao. A dynamic multiobjective recommendation method based on soft actor-critic with discrete actions. *Journal of King Saud University Computer and Information Sciences*, 37(1):1–17, 2025.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. URL https://api.semanticscholar.org/CorpusID:205242740.

[36] Soufiane Oualil, Rachid Oucheikh, Mohamed El Kamili, and Ismail Berrada. A personalized learning scheme for internet of vehicles caching. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 01–06, 2021. doi: 10.1109/GLOBECOM46510.2021.9685308.

[37] David Opeoluwa Oyewola, Sulaiman Awwal Akinwunmi, and Temidayo Oluwatosin Omotehinwa. Deep lstm and lstm-attention q-learning based reinforcement learning in oil and gas sector prediction. *Know.-Based Syst.*, 284 (C), January 2024. ISSN 0950-7051. doi: 10.1016/j.knosys.2023.111290. URL https://doi.org/10.1016/j.knosys.2023.111290.

[38] Hao Peng, Bowen Du, Mingsheng Liu, Mingzhe Liu, Shumei Ji, Senzhang Wang, Xu Zhang, and Lifang He. Dynamic graph convolutional network for long-term traffic flow prediction with reinforcement learning. *Inf. Sci.*, 578(C):401–416, November 2021. ISSN 0020-0255. doi: 10.1016/j.ins.2021.07.007. URL https://doi.org/10.1016/j.ins.2021.07.007.

[39] Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, and Zachary Patterson. Graph neural networks for intelligent transportation systems: A survey. *Trans. Intell.*

Transport. Sys., 24(8):8846–8885, August 2023. ISSN 1524-9050. doi: 10.1109/TITS.2023.3257759. URL https://doi.org/10.1109/TITS.2023.3257759.

[40] Manthena Raju, Natarajan Kumaran, and Chandra Vasamsetty. Remote sensing image data classification using cnn-deep q model. page 020011, 01 2023. doi: 10.1063/5.0128874.

[41] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[43] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent q-network, 2015. URL https://arxiv.org/abs/1512.01693.

[44] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[45] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998. doi: 10.1109/TNN.1998.712192.

[46] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, volume 30, pages 47–56. Barcelona, 2000.

[47] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903.

[48] Jihu Wang, Yuliang Shi, Lin Cheng, Kun Zhang, and Zhiyong Chen. Gre: A gat-based relation embedding model of knowledge graph for recommendation. In *ChineseCSCW (2)*, pages 77–91, 2021. URL https://doi.org/10.1007/978-981-19-4549-6_7.

[49] Qiao Wang and David Grace. Proactive edge caching in vehicular networks: An online bandit learning approach. *IEEE Access*, 10:131246–131263, 2022. doi: 10.1109/ACCESS.2022.3229645.

[50] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

[51] Qiong Wu, Yu Zhao, Qiang Fan, Pingyi Fan, Jiangzhou Wang, and Cui Zhang. Mobility-aware cooperative caching in vehicular edge computing based on asynchronous federated and deep reinforcement learning. *IEEE Journal of Selected Topics in Signal Processing*, 17(1):66–81, January 2023. ISSN 1941-0484. doi: 10.1109/jstsp.2022.3221271. URL http://dx.doi.org/10.1109/JSTSP.2022.3221271.

[52] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Comput. Surv.*, 55(5), December 2022. ISSN 0360-0300. doi: 10.1145/3535101. URL https://doi.org/10.1145/3535101.

[53] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE*

Transactions on Neural Networks and Learning Systems, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

[54] Yanli Xing. Work scheduling in cloud network based on deep q-lstm models for efficient resource utilization. *Journal of Grid Computing*, 22, 02 2024. doi: 10.1007/s10723-024-09746-6.

[55] Liangwei Yang, Shengjie Wang, Yunzhe Tao, Jiankai Sun, Xiaolong Liu, Philip S. Yu, and Taiqing Wang. Dgrec: Graph neural network for recommendation with diversified embedding generation. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, WSDM '23, page 661–669, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394079. doi: 10.1145/3539597.3570472. URL https://doi.org/10.1145/3539597.3570472.

[56] Shantian Yang, Bo Yang, Zhongfeng Kang, and Lihui Deng. Ihg-ma: Inductive heterogeneous graph multi-agent reinforcement learning for multi-intersection traffic signal control. *Neural Netw.*, 139(C):265–277, July 2021. ISSN 0893-6080. doi: 10.1016/j.neunet.2021.03.015. URL https://doi.org/10.1016/j.neunet.2021.03.015.

[57] Enming Yuan, Wei Guo, Zhicheng He, Huifeng Guo, Chengkai Liu, and Ruiming Tang. Multi-behavior sequential transformer recommender. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 1642–1652, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450387323. doi: 10.1145/3477495.3532023. URL https://doi.org/10.1145/3477495.3532023.

[58] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks, 2020. URL https://arxiv.org/abs/1911.06455.

[59] Degan Zhang, Wenjing Wang, Jie Zhang, Ting Zhang, Jinyu Du, and Chun Yang. Novel edge caching approach based on multi-agent deep reinforcement learning for internet of vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 24(8):8324–8338, 2023. doi: 10.1109/TITS.2023.3264553.

[60] Kechen Zheng, Guodong Jiang, Xiaoying Liu, Kaikai Chi, Xinwei Yao, and Jiajia Liu. Drl-based offloading for computation delay minimization in wireless-powered multi-access edge computing. *IEEE Transactions on Communications*, 71:1755–1770, 2023. URL https://api.semanticscholar.org/CorpusID:256142073.

[61] Huan Zhou, Kai Jiang, Shibo He, Geyong Min, and Jie Wu. Distributed deep multi-agent reinforcement learning for cooperative edge caching in internet-of-vehicles. *IEEE Transactions on Wireless Communications*, 22(12):9595–9609, 2023. doi: 10.1109/TWC.2023.3272348.

[62] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. 12 2018. doi: 10.48550/arXiv.1812.08434.