future internet

MDPI

Article

Intelligent Caching with Graph Neural Network-Based Deep Reinforcement Learning on SDN-Based ICN

Jiacheng Hou [1,*], Tianhao Tao [1], Haoye Lu [2] and Amiya Nayak [1]

[1] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON K1N 6N5, Canada; ttao061@uottawa.ca (T.T.); nayak@uottawa.ca (A.N.)
[2] David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada; haoye.lu@uwaterloo.ca
* Correspondence: jhou013@uottawa.ca

Abstract: Information-centric networking (ICN) has gained significant attention due to its in-network caching and named-based routing capabilities. Caching plays a crucial role in managing the increasing network traffic and improving the content delivery efficiency. However, caching faces challenges as routers have limited cache space while the network hosts tens of thousands of items. This paper focuses on enhancing the cache performance by maximizing the cache hit ratio in the context of software-defined networking–ICN (SDN-ICN). We propose a statistical model that generates users' content preferences, incorporating key elements observed in real-world scenarios. Furthermore, we introduce a graph neural network–double deep Q-network (GNN-DDQN) agent to make caching decisions for each node based on the user request history. Simulation results demonstrate that our caching strategy achieves a cache hit ratio 34.42% higher than the state-of-the-art policy. We also establish the robustness of our approach, consistently outperforming various benchmark strategies.

Keywords: information-centric networking; software-defined networking; graph neural network; deep reinforcement learning; intelligent caching

Citation: Hou, J.; Tao, T.; Lu, H.; Nayak, A. Intelligent Caching with Graph Neural Network-Based Deep Reinforcement Learning on SDN-Based ICN. *Future Internet* **2023**, *15*, 251. https://doi.org/10.3390/fi15080251

Academic Editor: Danda B. Rawat

Received: 22 June 2023
Revised: 9 July 2023
Accepted: 24 July 2023
Published: 26 July 2023

1. Introduction

Information-centric networking (ICN) has received significant interest and attention in recent years as a promising paradigm for network communication. ICN introduces a shift in focus from the traditional host-centric model to a content-centric approach. Named-based routing and in-network caching are two key features of ICN that have contributed to its growing popularity and adoption in various domains.

Researchers have explored the potential applications of ICN in diverse areas such as the Internet of Things (IoTs), where the efficient dissemination and retrieval of data is crucial for IoT devices to interact and exchange information [1]. In the context of the Internet of Vehicles (IoVs), ICN can enable efficient content delivery, facilitate real-time communication, and support intelligent transportation systems [2]. Furthermore, in the realm of 5G networks, ICN has been investigated for its potential to enhance content delivery, reduce latency, and improve overall network performance [3]. ICN has also been explored in software-defined networking (SDN) environments, offering flexibility, scalability, and efficient resource management [4].

One of the fundamental challenges in ICN is the effective caching of content across the network. Caching reduces latency, minimizes network congestion, and improves content delivery efficiency. However, it is a complex task due to the limited cache space available at each router and the potentially vast number of items distributed throughout the network. Researchers have been actively investigating caching strategies and policies to optimize cache performance.

In recent years, deep reinforcement learning (DRL) has allowed significant advancements in decision making, particularly in caching decisions. Numerous studies (see [5,6])

have demonstrated the exceptional performance of DRL in solving caching problems. Researchers have adopted deep Q-learning network architectures, such as multi-layer perceptron (MLP) and convolutional neural networks (CNNs), to replace traditional Q-tables. However, MLP and CNN architectures struggle to effectively utilize the neighbourhood information in arbitrary graph data, such as network topologies and knowledge graphs. While CNNs have been extensively optimized for the processing of Euclidean space data such as images and grids, they face challenges when dealing with graph-structured data. This limitation hampers their ability to capture the relational information necessary for efficient caching decision making.

Graph neural networks (GNNs) offer distinct advantages over traditional MLP and CNN architectures, as they are purpose-built to handle graph-structured data and excel in non-Euclidean spaces. This unique capability has made GNNs a popular choice in a wide range of domains that involve data represented as arbitrary graphs [7]. Notably, GNNs have demonstrated remarkable success in network routing optimization [8], where the underlying graph structure captures the intricate relationships between network nodes and facilitates efficient path planning. Additionally, in the domain of traffic prediction [9], GNNs leverage the graph structures of road intersections and their connectivity to forecast traffic flow patterns accurately.

Moreover, recent research has highlighted the remarkable generalization capabilities of GNNs [10]. GNNs can generalize effectively over different network topologies, allowing them to adapt to various environments and scenarios. This has been substantiated by studies such as [11–13], which have showcased the impressive generalization performance of GNNs across diverse network architectures.

The inherent suitability of GNNs for graph-structured data and their exceptional generalization capabilities make them an ideal choice in tackling complex problems in network-related domains. In the context of our research, leveraging the power of GNNs allows us to capture the intricate relationships and dependencies present in network caching scenarios, ultimately enhancing the network caching performance.

This paper aims to enhance the caching performance in the SDN-ICN scenario by leveraging DRL and GNN. Specifically, we introduce a GNN–double deep Q-network [14] (GNN-DDQN) caching agent within the SDN controller. The SDN controller provides a real-time and comprehensive view of the traffic situation in the SDN-ICN environment, while the network nodes are equipped with caching capabilities. The GNN-DDQN agent determines optimal caching decisions for individual nodes by considering the traffic conditions at each time step. The controller then communicates these decisions to the respective nodes, enabling them to update their cache stores accordingly.

The contributions of this paper are as follows.

- We develop a statistical model to generate users' preferences. Initially, we employ matrix factorization based on the Neural Collaborative Filtering Model [15] to learn content and user embeddings using the real-world dataset MovieLens100K [16]. Next, we employ a Gaussian mixture model to cluster users and content based on their embeddings. Subsequently, we employ a statistical model to generate the request behaviour of each user group.
- We introduce a GNN-DDQN agent within the SDN-ICN scenario. Incorporating a GNN in DRL is advantageous as GNNs excel in modelling graph-structured data, enabling nodes to engage in cooperative caching and enhancing the overall caching performance. Additionally, with only a single forward pass through the neural network, the GNN-DDQN agent can make caching decisions for all nodes in the network at each time step.
- We extensively evaluate the proposed caching scheme through simulations across various scenarios. These scenarios include different numbers of items, cache sizes, and network topologies, such as GEANT [17], ROCKETFUEL [18], TISCALI [19], and GARR [19]. Notably, our proposed caching scheme outperforms the state-of-the-art DRL-based caching strategy. Furthermore, it exhibits a significant performance advan-

tage over several benchmark caching schemes (Leave Copy Down (LCD), Probabilistic Caching (PROB_CACHE), Cache Less for More (CL4M), and Leave Copy Everywhere (LCE) [20–23]). The evaluations demonstrate the robustness of our proposed strategy to simulation parameters and variations in network topology.

It is worth noting that GNN-DDQN has several advantages.

- Computational Efficiency: GNN-DDQN is computationally efficient, requiring only one DRL agent to make caching decisions for all network nodes in a single forward pass.
- Multi-Action Capability: GNN-DDQN enables the agent to take multiple actions for each network node at each time step, demonstrating strong performance even with the incorporation of multi-actions.
- Applicability: GNN-DDQN can be applied in various real-world scenarios.
  - Content Delivery Networks (CDNs): Our proposed caching scheme can be employed within CDNs to improve caching decisions at edge nodes.
  - Mobile Edge Computing (MEC): Our caching scheme can benefit MEC environments by strategically caching frequently accessed content at edge servers.
  - Internet Service Providers (ISPs): By deploying our scheme, ISPs can enhance their caching infrastructure, effectively reducing the bandwidth requirements for popular content and providing faster access to frequently accessed data for their subscribers.
  - Video Streaming Platforms: By caching popular videos at appropriate network nodes, our algorithm can reduce buffering times and enhance the overall streaming experience for users.

However, there are also limitations to consider.

- Scalability: GNN-DDQN may face challenges in terms of scalability when dealing with a large number of network nodes. With only one SDN controller monitoring the entire network traffic, it may experience high latency, impacting the overall performance of the caching algorithm.
- Overfitting and Underfitting: GNN-DDQN, as with other deep learning algorithms, may suffer from overfitting or underfitting, depending on various factors.

The rest of this paper is organized as follows. Section 2 overviews related work. Sections 3 and 4 present our system model and proposed methodology. Section 5 shows the experimental results. Section 6 concludes the paper.

2. Related Work

Classical caching placement algorithms commonly used in the literature include LCE, LCD, PROB_CACHE, and CL4M [20–23]. LCE involves copying content at any cache between the serving and receiving nodes, while LCD caches content in the immediate neighbourhood of the serving node in the receiver's direction. PROB_CACHE probabilistically caches content on a path, considering various factors. CL4M aims to place content in nodes with high graph-based centrality. In addition to placement algorithms, traditional caching replacement algorithms such as Least Recently Used (LRU), Least Frequently Used (LFU), and First-In-First-Out (FIFO) are commonly employed [24,25]. LRU discards the least recently accessed content, LFU replaces the least frequently used content first, and FIFO evicts the first item inserted in the cache. However, these traditional algorithms are often considered inefficient, yielding poorer performance than deep learning-based caching algorithms.

DRL-based caching algorithms have demonstrated remarkable achievements in recent years [26,27]. In [5], the authors developed a deep Q-network (DQN)-based caching algorithm designed explicitly for mobile edge networks. The application of DRL in the Internet of Vehicles (IoV) field has also gained substantial attention. As the demand for computation and entertainment in autonomous driving and vehicular scenarios increases, researchers have been actively advancing caching strategies to enhance the user experience.

In [28], the authors propose CoCaRL, a caching strategy leveraging DRL and a multi-level federated learning framework. They utilize a DDQN [14] to optimize the cache hit ratio of local roadside units (RSUs), neighbour RSUs, and cloud data centers in vehicular networks. Their approach also incorporates federated learning to enable decentralized model training. Another study [29] introduces a quality of experience (QoE)-driven RSU caching model based on DRL. Their caching algorithm addresses the growing demand for time-sensitive short videos in a 5G-based IoV scenario. The reward in their DRL model is defined as the ratio of the number of videos interesting to each user to the total number of videos stored in the RSU. Furthermore, in [6], the authors design a DQN-based strategy to optimize joint computing and edge caching in a three-layer IoV-ICN network architecture, encompassing vehicles, edges, and cloud layers. In [30], the authors proposed a social-aware vehicular edge computing architecture to efficiently deliver popular content to end-users in vehicular social networks. They introduce a social-aware graph pruning search algorithm to assign content consumer vehicles to the shortest path with the most relevant content providers. Additionally, they utilize a DRL method to optimize content distribution across the network. In [31], the authors develop an IoV-specific edge caching model that enables collaborative content caching among mobile vehicles and considers varying content popularity and channel conditions. Additionally, the framework empowers each vehicle agent to make caching decisions based on environmental observations autonomously. In [32], the authors proposed a spatial–temporal correlation approach to predict content popularity in the IoV. They introduce a DRL-based multi-agent caching strategy, where each RSU is an independent agent, to optimize caching decisions. In [33], the authors investigate joint computation offloading, data caching, transmission path selection, subchannel assignment, and caching management in the IoV-based environment. Dynamic online algorithms such as the Simulated Annealing Genetic Algorithm (SAGA) and DQN are adopted to minimize the content access latency.

In addition to DRL, GNNs have emerged as another effective approach in addressing caching problems. One notable application of GNNs in caching is presented in [34]. The authors introduce a GNN-based caching algorithm to optimize the cache hit ratio in a named data networking (NDN) context. Their approach involves two key steps. First, they utilize a 1D-CNN to predict the popularity of content in each node. Subsequently, a GNN is employed to propagate the content popularity predictions among neighbouring nodes. Finally, each node makes caching decisions based on node-level caching probability ranking. Leveraging the message-passing capabilities of GNNs, their caching approach outperforms the CNN-based caching algorithm, leading to improved caching performance in the NDN scenario. Moreover, in [35], the authors propose GNN-GM to enhance the caching performance in NDN. In this work, a GNN is utilized to predict users' ratings of unviewed movies within a bipartite graph representation. Leveraging the accurate rating predictions achieved by GNN, the proposed approach achieves a higher cache hit ratio compared to state-of-the-art caching schemes. The successful application of GNNs in these studies highlights their efficacy in addressing caching challenges. By leveraging the GNN's ability to capture complex dependencies and propagate information across nodes, these approaches demonstrate improved caching performance and provide valuable insights for the optimization of cache hit ratios in various network scenarios.

The integration of DRL and GNN has additionally emerged as a growing trend, delivering numerous benefits. The authors in [36] employ dynamic graph convolutional networks (GCNs) and RL for long-term traffic flow prediction. They represent 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 a GCN and long short-term memory (LSTM) to extract spatial and temporal features from the generated dynamic graph sequences, enabling traffic flow prediction in future time steps. Another study [37] 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 an agent. An inductive GNN algorithm is applied to learn the embeddings of the agents and their neighbours. The learned representations are then fed into an actor–critic network to optimize traffic control. Additionally, [38] proposes an innovative approach using a GCN and DQN for the multi-agent cooperative control of connected autonomous vehicles (CAVs). Each CAV is treated as an agent, and a GCN is utilized to extract embeddings for each agent. These representations are then fed into a Q-network to determine the actions of each agent, facilitating effective cooperative control among the CAVs. Furthermore, in an SDN-based scenario, [13] presents a centralized agent that leverages DRL and GNNs to optimize routing strategies. They utilize a GNN to model the network and DRL to calculate the Q-value of an action. By embedding routing paths into node representations and feeding them into the Q-network, they evaluate various routing strategies and select the optimal one when a traffic demand is issued. In [39], the authors propose a method that combines prediction, caching, and offloading techniques to optimize computation in 6G-enabled IoV. The prediction method is based on a spatial–temporal graph neural network (STGNN), the caching decision method is realized using the simplex algorithm, and the offloading method is based on Twin Delayed Deterministic Policy Gradient (TD3).

These studies demonstrate the efficacy of combining DRL and GNNs in tackling various issues, including traffic prediction, traffic signal control, the cooperative control of autonomous vehicles, routing optimization in SDN scenarios, and caching in IoV environments. By leveraging the respective strengths of DRL and GNNs, these approaches enable intelligent decision making and enhance performance in intricate systems. We are confident that the amalgamation of DRL and GNNs can similarly bring advantages to caching in the SDN-ICN context.

3. System Model

In this section, we present the system architecture of our proposed caching scheme and provide a comprehensive overview of the key components. We also define the concept of content popularity. Furthermore, we develop a user preference model based on real-world data from the MovieLens100K dataset [16]. Important notations used throughout the paper are listed in Table 1.

Table 1. Important notations.

Notation	Definition
N	Number of network nodes
C	Number of content items
T	Number of time slots
$\mathcal{N} = \{n_1, \ldots, n_N\}$	Set of network nodes
$\mathcal{C} = \{c_1, \ldots, c_C\}$	Set of content
$\mathcal{U} = \{u_1, \ldots, u_U\}$	Set of users
$\mathcal{T} = \{t_0, t_1, \ldots, t_T\}$	Set of time steps
$b_{t_l}^{c_i,n_k}$	"Cache" or "not cache" content c_i at node n_k at time step t_l (i.e., availability of content c_i at node n_k's cache store during the time interval between t_l and t_{l+1})
$\mathcal{S}_{t_l} = \{s_{t_l}^{n_1}, \ldots, s_{t_l}^{n_N}\}$	Set of all nodes' states at time step t_l
$\mathcal{A}_{t_l} = \{a_{t_l}^{n_1}, \ldots, a_{t_l}^{n_N}\}$	Set of all nodes' caching actions at time step t_l
$a_{t_l}^{n_k} = \{b_{t_l}^{c_1,n_k}, \ldots, b_{t_l}^{c_C,n_k}\}$	Set of node n_k's caching action at time step t_l
$\mathcal{R}_{t_l} = \{r_{t_l}^{n_1}, \ldots, r_{t_l}^{n_N}\}$	Set of all nodes' rewards (i.e., cache hits) at time step t_l
$r_{t_l}^{n_k} = \{r_{t_l}^{c_1,n_k}, \ldots, r_{t_l}^{c_C,n_k}\}$	Set of node n_k's reward for each content item at time step t_l
z	Router's cache size
x_{c_i}	Content c_i's embedding
y_{u_j}	User u_j's embedding

3.1. System Architecture

We propose an intelligent caching strategy in an SDN-based ICN (SDN-ICN) architecture, as depicted in Figure 1. The SDN-ICN architecture separates the control plane from the data plane, and the OpenFlow protocol facilitates the data transfer between them. We introduce the GNN-DDQN [14,40] agent responsible for making caching decisions in the control plane. The data plane comprises network nodes that perform caching actions.

Figure 1 illustrates the control plane, consisting of two modules: (i) the GNN-DDQN agent module, which plays a crucial role in caching decisions for ICN nodes in the data plane; and (ii) the content caching management module, which handles the content caching of each ICN node. We assume that the data plane's network topology consists of N ICN nodes, denoted as $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$.

The data plane encompasses ICN nodes, each fulfilling specific roles: (i) source nodes responsible for content publication without caching capabilities, (ii) receiver nodes accountable for sending requests to source nodes, also without caching capabilities, and (iii) router nodes responsible for forwarding requests and data packets across the network. Instead of assuming that all router nodes have the caching capability, we consider only some of them to have this. These router nodes equipped with caching capabilities have a cache capacity of z, defined as the number of content items.

The network contains C distinct content items, represented by the set $\mathcal{C} = \{c_1, \ldots, c_C\}$. We assume that an experimental time round can be divided into T slots of equal duration, denoted by $\mathcal{T} = \{t_0, t_1, \ldots, t_T\}$. To indicate whether a node n_k caches content c_i at time step t_l, we employ a binary variable $\{b_{t_l}^{c_i,n_k}\}$, where $t_l \in \mathcal{T}$, $c_i \in \mathcal{C}$, and $n_k \in \mathcal{N}$. Specifically, $b_{t_l}^{c_i,n_k} = 1$ if and only if node n_k caches content c_i at time step t_l, implying that content c_i is available at node n_k during the time interval between t_l and t_{l+1}.

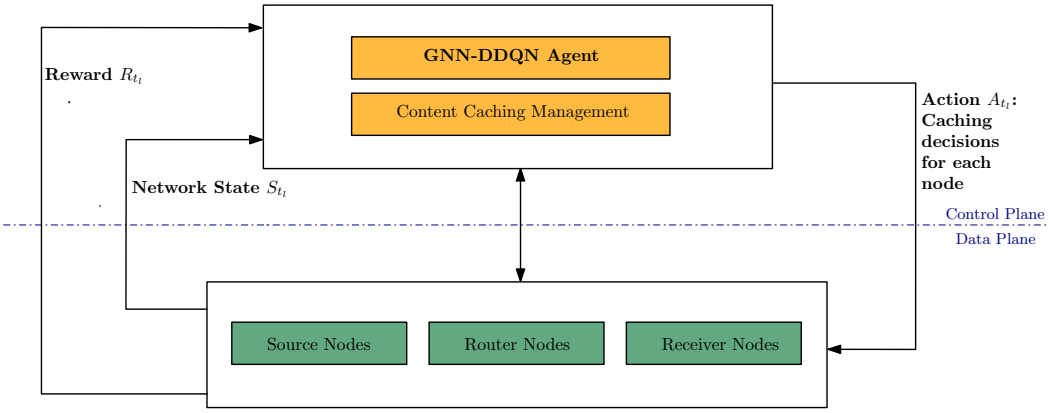

Figure 1. The SDN-ICN architecture. In the controller, the GNN-DDQN agent receives a network state S_{t_l} and generates an action A_{t_l} at each time step t_l. Subsequently, it receives a reward R_{t_l} at the next time step t_{l+1}.

In the SDN-ICN architecture, the controller can see the network's traffic. Therefore, at each time step t_l, the GNN-DDQN agent observes the network state $\mathcal{S}_{t_l}$, which encompasses information about the network's status during the time interval between t_{l-1} and t_l. Subsequently, the GNN-DDQN agent performs content caching predictions at the node level, denoted as $\mathcal{A}_{t_l}$, and communicates the recommended content to be cached to each router node with caching capabilities. When a router node with caching capabilities receives a request packet for content c_i within this period, it can fulfill the request directly if the requested content is cached or forward the request to the source node. At the subsequent time step t_{l+1}, a set of rewards $\mathcal{R}_{t_l}$ is sent to the GNN-DDQN agent. Specifically, $\mathcal{R}_{t_l} = r_{t_l}^{n_1}, r_{t_l}^{n_2}, \ldots, r_{t_l}^{n_N}$ represents the rewards of all nodes at time step t_l. Furthermore, $r_{t_l}^{n_k} = r_{t_l}^{c_1,n_k}, r_{t_l}^{c_2,n_k}, \ldots, r_{t_l}^{c_C,n_k}$ represents the set of rewards for each content item received by node n_k at time step t_l.

3.2. Content Popularity and User Preference

In a realistic computer network, users exhibit preferences for specific types of content, leading to varying request frequencies. We develop a statistical model that incorporates content popularity and user preferences to simulate this network traffic. In our network, receiver nodes correspond to users, and we denote the users as $\mathcal{U} = \{u_1, \ldots, u_U\}$. Our objective is to determine the probability distribution $P(c_i, u_j)$, which represents the likelihood of a request for the i^{th} content by the j^{th} user.

Content popularity refers to the probability distribution of requesting the i^{th} content within the network, represented by $P(c_i)$. Research studies [41] have shown that the content popularity in a network can be modelled using a Zipfian distribution,

$$P(c_i) = \frac{1}{(i)^{\alpha} \sum_{k=1}^{C} (k)^{-\alpha}} \tag{1}$$

where α is a skewness factor with a value of 0.8.

User preference refers to the relationship between users and content items. In our study, we employ collaborative filtering [15,42] to capture user preferences. Collaborative filtering is a popular recommendation system technique that predicts a user's preference by identifying users with similar tastes based on their historical behaviours. Collaborative filtering has two primary approaches: the neighbourhood-based method and the latent factor method. The neighbourhood-based method identifies similar users or items based on their historical preferences and recommends items that similar users or items have liked. On the other hand, the latent factor method discovers latent factors that represent underlying characteristics of users and items and uses these factors to predict user preferences. In our case, we utilize matrix factorization, a latent factor method, to extract the latent factors of users and content items. By decomposing the user–item interaction matrix into lower-dimensional matrices, we can represent users and items in terms of these latent factors. Subsequently, we calculate user preferences by analyzing the relationships between users and content items derived from matrix factorization.

To capture the user–content relation, we construct a matrix M with dimensions $C \times U$, where C represents the number of content items and U represents the number of users. Each element $m_{i,j}$ in the matrix corresponds to the relationship between content c_i and user u_j. This relationship can be based on various factors, such as ratings given by the user, the time spent on the content, or any other relevant metric. We utilize trainable embedding layers to process the user–content matrix further to generate embedding vectors for each content item and user. Specifically, for each content item c_i, we apply an embedding layer that maps it to a continuous vector representation $x_{c_i} \in \mathbb{R}^e$, where e denotes the dimensionality of the embedding. Similarly, for each user u_j, we employ an embedding layer to obtain the embedding vector $y_{u_j} \in \mathbb{R}^e$.

Our research uses the well-known MovieLens100K dataset [16] as a real-world dataset for our experiments. This dataset consists of user ratings for movies and is widely used in evaluating recommendation systems. We focus on learning embedding vectors for 943 users and 1682 content items within this dataset.

To obtain user and content embeddings, we employ a matrix factorization technique combined with a neural network architecture inspired by the works [15,42]. Our model consists of two trainable embedding layers, one for users and another for content items. These layers enable the learning of dense and low-dimensional representations that capture users' and content's underlying characteristics and preferences. The next step in our model involves computing the element-wise product of the content and user embedding vectors. This element-wise product represents the interaction between a specific content item and a user. Subsequently, the resulting products are fed into a linear layer with an activation function. For a given content embedding x_{c_i} and a user embedding y_{u_j}, the output is computed as follows:

$$\hat{m}_{i,j} = \sigma(\mathbf{w}^T (x_{c_i} \odot y_{u_j})) \tag{2}$$

in this equation, σ denotes the sigmoid activation function, **w** represents a trainable matrix, and $\odot$ signifies the element-wise product. To train the matrix factorization model, we minimize the binary cross-entropy (BCE) loss between the ground truth values $m_{i,j}$ and the predicted values $\hat{m}_{i,j}$. It is worth mentioning that we label $m_{i,j}$ as 1 if the j^{th} user has provided a rating for the i^{th} content item, and 0 otherwise. The key training parameters for the Neural Collaborative Filtering (NCF) model are summarized in Table 2.

Table 2. Key NCF model training parameters.

Parameters	Values
Epoch	100
Learning rate	0.001
Batch size	256
Embedding dimension e	8
Optimizer	Adam

In order to fit the number of content items and users in our network, all users and content items in the dataset are divided into groups. If the content embeddings are close to each other, we cluster them into groups, and users are grouped in the same way. We utilize a Gaussian mixture model (GMM) to cluster the embedding vectors, and then compute a representative embedding for each group by taking the element-wise mean.

Since the inner product ${x_{c_i}}^T y_{u_j}$ captures the correlation between content c_i and a user u_j, we apply the softmax function on the inner products to obtain the probability $P(u_j|c_i)$ for given content c_i. This probability represents the preference of user u_j for content c_i. Inspired by the works [43,44], we calculate the joint probability $P(c_i, u_j)$ of content c_i being requested by user u_j as follows:

$$\begin{aligned} P(c_i, u_j) &= P(c_i)P(u_j|c_i) \\ &= P(c_i)\frac{\exp({x_{c_i}}^T y_{u_j})}{\sum_{j=1}^{U}\exp({x_{c_i}}^T y_{u_j})} \end{aligned} \tag{3}$$

where $P(c_i)$ represents the content popularity, while $P(u_j|c_i)$ reflects the preference of user u_j for content c_i. Combining these probabilities establishes a link between the user preference and content popularity.

It is important to note that our approach differs from the methods proposed in [43,44], as we obtain user and content embeddings from a real-world dataset. Furthermore, we consider the inner products of the learned user embeddings and content embeddings to measure their associations, enabling us to capture the relationships between users and content meaningfully.

4. Proposed Methodology

This section presents our GNN-DDQN agent, which incorporates a GNN as the Q-network within the DDQN framework [14]. DDQN improves upon the original DQN algorithm [40] by mitigating Q-value overestimation and enhancing the overall performance.

The GNN-DDQN agent predicts Q-values based on the observed state $\mathcal{S}_{t_l}$ and the chosen action $\mathcal{A}_{t_l}$ at each time step t_l. The predicted Q-value is denoted as $Q(\mathcal{S}_{t_l}, \mathcal{A}_{t_l})$. The objective of the agent is to learn an optimized policy that maximizes the expected Q-value $Q^*(\mathcal{S}_{t_l}, \mathcal{A}_{t_l})$. This section describes the state space, action space, and reward function used in our DDQN. Additionally, we explain the GNN architecture employed to map network states to action rewards for each node. Finally, we provide an overview of the GNN-DDQN agent, including its key components and functionality.

4.1. State Space

The network state $\mathcal{S}_{t_l} = \{s_{t_l}^{n_1}, \ldots, s_{t_l}^{n_N}\}$ captures the state of each network node at time step t_l. Each node's state feature vector $s_{t_l}^{n_k}$ at time step t_l is represented by $s_{t_l}^{n_k} \in \mathbb{R}^{C\times 3}$, where C is the total number of items in the network.

The state $s_{t_l}^{n_k}$ of a network node n_k at time step t_l consists of three components:

- 1st component: The number of requests for each content item c_i that have traversed the node during the previous time interval (t_{l-1} to t_l). This count is stored only for the requested content in receiver nodes, cached content in router nodes, and published content in source nodes.
- 2nd component: The cache storage of the node, represented by a binary variable for each content item c_i. A value of 1 indicates that the node caches the content during the previous time interval, while a value of 0 is used otherwise.
- 3rd component: The content publication of the node is also represented by a binary variable for each content item c_i. A value of 1 indicates that the node has published the content during the previous time interval, while a value of 0 is used otherwise.

4.2. Action Space

At a time step t_l, each node n_k can choose z out of C content items to cache. We record its cache scheme in a binary tuple,

$$a_{t_l}^{n_k} = \{b_{t_l}^{c_1,n_k}, \ldots, b_{t_l}^{c_C,n_k}\}, \tag{4}$$

where 1 means to 'cache' and 0 means to 'not cache', and the sum of all entries cannot exceed the assumed router's cache size z. We also use $\mathcal{A}_{t_l}$ to denote the cache scheme of all nodes such that,

$$\mathcal{A}_{t_l} = \{a_{t_l}^{n_1}, \ldots, a_{t_l}^{n_N}\} \tag{5}$$

and refer to it as the agent's action at the time step t_l. When the agent takes action $\mathcal{A}_{t_l}$ at time step t_l, the node n_k caches content according to $a_{t_l}^{n_k}$, which can be used to satisfy the request in the future.

4.3. Reward Function

Our objective is to maximize the cache hit ratio. Thus, we use cache hits as the agent's reward, denoted as $\mathcal{R}_{t_l} = \{r_{t_l}^{n_1}, \ldots, r_{t_l}^{n_N}\}$, which includes the cache hits of each node. For a node n_k at time step t_l, its cache hits for each content item are $r_{t_l}^{n_k} = \{r_{t_l}^{c_1,n_k}, \ldots, r_{t_l}^{c_C,n_k}\}$. Let us assume that a node n_k's reward sum for all content at time step t_l is $cacheHits_{t_l}^{n_k} = r_{t_l}^{c_1,n_k} + r_{t_l}^{c_2,n_k} + \cdots + r_{t_l}^{c_C,n_k}$; then, the objective function can be formulated as follows:

$$\begin{aligned} &\max \sum_{n_k\in\mathcal{N}} \sum_{t_l\in\mathcal{T}} cacheHits_{t_l}^{n_k} \\ &\text{s.t.} \\ &\sum_{c_i\in\mathcal{C}} b_{t_l}^{c_i,n_k} \leq \begin{cases} z, & \text{node } n_k \text{ has the caching capability} \\ 0, & \text{otherwise} \end{cases}, \forall t_l \in \mathcal{T}, \forall n_k \in \mathcal{N} \end{aligned} \tag{6}$$

this objective aims to maximize the cache hit ratio for the stored content while ensuring that the number of content items stored in a node does not exceed z if it is a router node with caching capability and zero otherwise. We apply this constraint because only router nodes with caching capability can cache content, while other nodes, such as source and receiver nodes, can only distribute or receive content.

4.4. GNN Architecture

Our methodology utilizes a GNN for node-level Q-value predictions. The model architecture, depicted in Figure 2, operates on a network graph consisting of node embeddings and an adjacency matrix.

The GNN takes as input the graph structure data $G = (\mathcal{N}, \mathcal{E}, \mathcal{S})$, where $\mathcal{N}$ represents the set of nodes, $\mathcal{E}$ denotes the set of edges, and $\mathcal{S}$ represents the network state. With this input, the GNN model generates Q-value predictions for each action of every node, allowing us to estimate the outcome of each action through a single forward propagation of the GNN model.

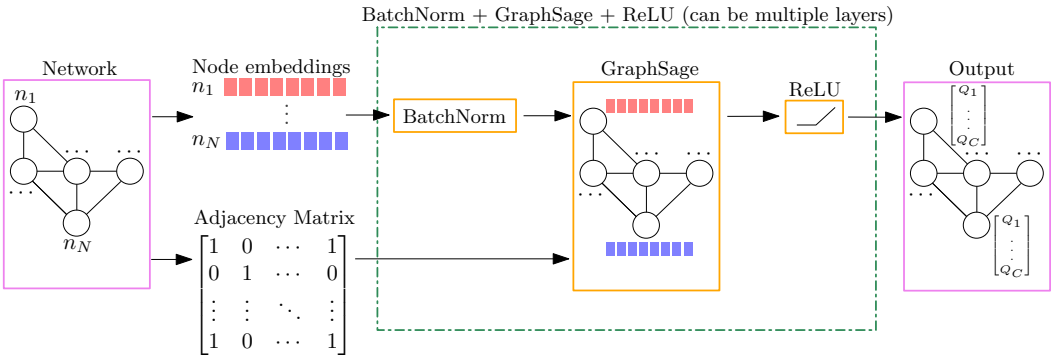

Figure 2. Model architecture.

For the GNN architecture, our approach utilizes four GraphSage layers [45]. Each layer has different hidden embedding dimensions, specifically 1024, 512, 256, and C. GraphSage is an inductive framework that leverages sampling and aggregation techniques to generate node embeddings. It allows for efficient embedding generation even for previously unseen data. By incorporating four GraphSage layers, we can aggregate information from up to four-hop neighbouring nodes at each step. This enables the GNN to capture the network structure and traffic patterns, resulting in more informative node embeddings for Q-value prediction.

The aggregation process in GraphSage is described by the equation

$$h^k_{N(v)} = AGG_k(h^{k-1}_u, \forall u \in N(v)), \tag{7}$$

where $N(v)$ represents the one-hop neighbours of node v, and h^{k-1}_u is the embedding of node u at the previous $(k-1)^{\text{th}}$ step. In each step, the GNN aggregates the embeddings of the one-hop neighbours of a node v from the previous step to obtain $h^k_{N(v)}$. The aggregation function AGG is typically permutation-invariant, meaning that it is not affected by the ordering of the aggregated embeddings. In our approach, we use a mean aggregator, which calculates the element-wise mean of the vectors $h^{k-1}_u, \forall u \in N(v)$.

After the aggregation step, the GNN performs concatenation by combining the embeddings of each central node from the previous $(k-1)^{\text{th}}$ step with the embeddings of its neighbouring nodes from the current k^{th} step. The concatenated embeddings are then fed into a fully connected layer with a nonlinear activation function:

$$h^k_v = \sigma\left[W^k \cdot \text{CONCAT}\left(h^{k-1}_v, h^k_{N(v)}\right)\right], \tag{8}$$

where W^k represents a learned matrix specific to the k^{th} step, σ denotes the rectified linear activation function (ReLU), and h^k_v corresponds to the embedding of the central node v at the k^{th} step. The CONCAT operation refers to the concatenation of the embeddings h^{k-1}_v and $h^k_{N(v)}$.

4.5. The GNN-DDQN Agent

The GNN-DDQN agent operates based on the procedure described in Algorithm 1. In the beginning, we initialize a replay buffer P, a Q-network (Q) implemented as a GNN with randomly generated parameters θ, and a target Q-network ($\hat{Q}$) with the same network architecture and parameters as Q. Each episode corresponds to a complete round of experimentation, and the time is divided into T slots. The GNN-DDQN agent takes actions at each time step, denoted as t_l (starting from t_1, as t_0 represents the initial point of the experimentation).

To balance exploration and exploitation, we utilize an ϵ-greedy exploration strategy [40]. This strategy involves randomly selecting actions with a probability of ϵ and selecting the action with the highest expected Q-value with a probability of $1-\epsilon$. The purpose is to encourage initial exploration and gradually decrease exploration over time. We employ an exponential decay strategy for ϵ, starting with an initial value of $\epsilon_s = 0.9$ and decaying to a minimum value of $\epsilon_e = 0.01$ with a decay rate of 0.01, denoted as $\epsilon_d = 100$.

Since each router with caching capability has a cache size of z, the GNN-DDQN agent selects z actions for each node at each time step. It chooses the top z actions with the highest Q-values for each node during greedy action selection. For random actions, it randomly selects z actions for each node. It is crucial to emphasize that the agent precisely chooses z actions for each node at every time step. However, it only executes these actions for router nodes with caching capabilities, excluding others.

Algorithm 1 GNN-DDQN Agent Operation

Input: number of episodes E, batch size B, target network update step K, replay buffer capacity R, epsilon start ϵ_s, epsilon end ϵ_e, epsilon decay ϵ_d, number of steps k, discount factor γ

Initialize replay buffer P with capacity R
Initialize Q-network with random weights θ
Initialize target $\hat{Q}$-network with weights $\hat{\theta} = \theta$
for *episode* $\in \{1, \ldots, E\}$ **do**
  for $t_l \in \{t_1, \ldots, t_T\}$ **do**
    Randomly pick $\epsilon' \in [0, 1]$
    $\epsilon_t = \epsilon_e \cdot (\epsilon_s - \epsilon_e) \cdot \exp\left(\frac{-k}{\epsilon_d}\right)$
    $k = k + 1$
    for $n_k \in \{n_1, \ldots, n_N\}$ **do**
      if $\epsilon' < \epsilon_t$ **then**
        Randomly select z actions
      else
        Select z actions with the highest $Q(\mathcal{S}_{t_l}, \mathcal{A}_{t_l}|\theta)$
      end if
    end for
    Take action $\mathcal{A}_{t_l}$, get reward $\mathcal{R}_{t_l}$ and next state $\mathcal{S}_{t_{l+1}}$
    Store transition $(\mathcal{S}_{t_l}, \mathcal{A}_{t_l}, \mathcal{R}_{t_l}, \mathcal{S}_{t_{l+1}})$ into P
    Randomly sample B transitions $(\mathcal{S}_{b_j}, \mathcal{A}_{b_j}, \mathcal{R}_{b_j}, \mathcal{S}_{b_{j+1}})$ from P
    Use Equation (10) to compute $\mathcal{Y}_{b_j}$
    Perform a gradient descent step on $L(\theta)$ with respect to the network parameters θ, where $L(\theta)$ is computed in Equation (9)
    Update $\hat{\theta} = \theta$ every K steps
  end for
end for

At time step t_l, the GNN-DDQN agent interacts with the environment by taking action A_{t_l} and receiving a reward R_{t_l} and the subsequent state $S_{t_{l+1}}$ at time step t_{l+1}. The rewards, denoted by $\mathcal{R}_{t_l}$, are node-level rewards, where each node has C rewards corresponding to different actions. The newly generated transition $(\mathcal{S}_{t_l}, \mathcal{A}_{t_l}, \mathcal{R}_{t_l}, \mathcal{S}_{t_{l+1}})$ is then stored in the replay buffer P.

We train the Q-network by randomly sampling a batch of transitions $(\mathcal{S}_{b_j}, \mathcal{A}_{b_j}, \mathcal{R}_{b_j}, \mathcal{S}_{b_{j+1}})$ from the replay buffer P. The Q-network is trained using gradient descent on a loss function $L(\theta)$, which measures the discrepancy between the predicted Q-values and the target Q-values. For the sampled transitions b_j, the loss function is defined as follows:

$$L(\theta) = \sqrt{\mathbb{E}\left[\sum_{b_j}((\mathcal{Y}_{b_j} - Q(\mathcal{S}_{b_j}, \mathcal{A}_{b_j}|\theta))^2 \cdot mask)\right]} \tag{9}$$

where $\mathcal{Y}_{b_j}$ is defined as follows:

$$\mathcal{Y}_{b_j} = \begin{cases} \mathcal{R}_{b_j}, & \text{if episode terminates at } b_{j+1} \\ \mathcal{R}_{b_j} + \frac{1}{z}\gamma \sum_{r \in \hat{Q}(\mathbf{s}_{b_{j+1}}, \arg\max_{\mathcal{A}'_{b_{j+1}}, |\mathcal{A}'_{b_{j+1}}|=z} Q(\mathbf{s}_{b_{j+1}}, \mathcal{A}'_{b_{j+1}}|\theta)|\hat{\theta})} r, & \text{otherwise} \end{cases} \tag{10}$$

and *mask* is defined as follows:

$$mask = \begin{cases} 1, & \text{if } n_k \text{ is a router with the caching capability} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

where $\mathcal{Y}_{b_j}$ represents the ground truth Q-values. If the episode terminates at transition b_{j+1}, $\mathcal{Y}_{b_j}$ is equal to $\mathcal{R}_{b_j}$. Otherwise, it is computed as the sum of $\mathcal{R}_{b_j}$ and the discounted expected reward of the next state. To estimate the expected future reward, the Q-network selects the top z greedy actions based on state $S_{b_{j+1}}$, and the corresponding Q-values are computed using the target Q-network $\hat{Q}$. The discount factor γ determines the importance of long-term rewards and is typically between 0 and 1. To ensure that each action taken at the next time step contributes equally, the sum of the expected long-term rewards is divided by z. To focus the loss contribution on routers with caching capabilities, a mask is applied in Equation (9). Nodes without caching capabilities are assigned a mask value of 0, while routers with caching capabilities have a mask value of 1.

To maintain the training stability, the parameters of the Q-network Q are periodically copied to the target Q-network $\hat{Q}$ every K steps. This helps to reduce the potential for the overestimation of the Q-values during training.

The key training parameters for the GNN-DDQN model are summarized in Table 3.

Table 3. Key GNN-DDQN model training parameters.

Parameters	Value
Number of episodes E	1000
Learning rate	0.001
Batch size B	32
Target network update step K	10
Replay buffer capacity R	1000
Epsilon start ϵ_s	0.9
Epsilon end ϵ_e	0.01
Epsilon decay ϵ_d	100
Discount factor γ	1
Optimizer	Adam

5. Experimentation and Results

In this section, we present simulation results to demonstrate the effectiveness of the proposed caching strategy in various network scenarios. To conduct these experiments, we utilized Icarus[46], a Python-based ICN caching simulator that comprehensively evaluates different caching strategies. Not bound to any specific architecture, such as content-centric

networking (CCN) or named data networking (NDN), Icarus provides functionalities for more generalized ICN.

We employed the LRU strategy as the caching replacement policy for all our experiments. Moreover, the content popularity and user preference distributions mentioned in Section 3.2 were considered. Table 4 lists the key simulation parameters used. We followed the recommendations from a previous study [47] and set the internal and external link delays to 2 milliseconds (ms) and 34 ms, respectively, for all network topologies.

The experiments involved a set of distinct content, ranging from 600 to 1000, uniformly distributed among all source nodes in the network. The router's cache size varied from 1 to 4, denoting the number of content items that it could store. Each experiment consisted of a warm-up phase with 2000 requests, followed by 4000 requests that were measured to evaluate the performance of different caching schemes. User requests followed a Poisson distribution with a mean of 100 requests per second.

We divided each experiment into T segments, each representing 10 s. We conducted 600 experiments for each caching scenario and calculated the average evaluation metrics based on the results of the last 200 experiments.

Table 4. Key simulation parameters.

Parameters	Values
Network topology	GEANT [17], ROCKETFUEL [18], TISCALI [19], and GARR [19]
Internal link delay (all networks)	2 ms
External link delay (all networks)	34 ms
Number of distinct content items	Range: 600–1000 items
Content size	1500 bytes
Request size	150 bytes
Cache size	Range: 1–4 items
Number of warm-up requests	2000
Number of measured requests	4000
Request distribution	Poisson distribution with a mean of 100 requests per second
Time slot	10 s
Number of experimentations	600

The evaluation of different caching strategies relied on four key metrics.

- Cache Hit Ratio (CHR): The cache hit ratio represents the percentage of requests that can be fulfilled by retrieving data packets from the cache in the router nodes,

$$CHR = \frac{cacheHits}{cacheHits + cacheMiss}, \tag{12}$$

  where *cacheHits* refers to the count of *Interest* packets that are successfully satisfied by retrieving the corresponding *Data* packet from the router's cache. On the other hand, *cacheMiss* represents the count of *Interest* packets that cannot be fulfilled by the cache and require fetching from external sources. An *Interest* packet carries the name of the requested content and is transmitted from the receiver node, while a *Data* packet contains the requested content itself and can serve as a response to the corresponding *Interest* packet.
- Average Latency Time (ALT): The average latency time represents the average delay between the moment that a user sends an *Interest* packet and the moment that it receives the corresponding *Data* packet,

$$ALT = \frac{\sum_{i=1}^{I}(i_t + i_r)}{I}, \tag{13}$$

where I denotes the total number of user requests. i_t represents the travel time of the *Interest* packet from the receiver node to the node that fulfills the request, while i_r denotes the travel time of the responding *Data* packet.

- Average Path Stretch (APS): The average path stretch measures the average increase in path length for each user request,

$$APS = \sum_{i=1}^{I} \frac{path_{i,n_u,n_r}}{Path_{i,n_u,n_s}}, \tag{14}$$

where I represents the total number of user requests. n_u denotes the receiver node that sends the request, n_r refers to the node that responds to the request, and n_s represents the source node that publishes the requested content. $path_{i,n_u,n_r}$ denotes the number of hops travelled by the i^{th} request, while $Path_{i,n_u,n_s}$ represents the shortest path from the receiver to the source.

- Average Link Load (ALL): The average link load represents the average ratio of the total link load to the total number of links in the network,

$$ALL = \frac{\sum_{l=1}^{L} \mathcal{L}_l}{L}, \tag{15}$$

where L denotes the total number of links in the network, and $\mathcal{L}_l$ represents the link load of the specific link l.

The caching performance of our proposed GNN-DDQN scheme was evaluated and compared with the state-of-the-art caching scheme MLP-DDQN. MLP-DDQN, which has been extensively studied in various research works [5,6], was used as a baseline for comparison. We adapted the MLP-DQN framework to incorporate the DDQN technique to ensure a fair comparison. The MLP-DDQN agent consisted of four linear layers with dimensions of 1024, 512, 256, and C.

There are some differences between the state representations of the MLP-DDQN agent and our proposed GNN-DDQN approach. In the MLP-DDQN agent, the first component of the state representation includes the number of requests for each content item c_i passed through each node, covering all types of nodes (receivers, routers, and sources). This provides more general traffic-related information to assist the MLP agent in making predictions, as it lacks the ability to gather neighbouring information as in the GNN approach.

Additionally, we compared our caching strategy with classical caching algorithms, including LCD, PROB_CACHE, LCE, and CL4M. These algorithms served as additional baselines to assess the performance of our proposed approach.

5.1. Effect of Content Item Number

This section examines the impact of the number of content items on caching performance. The number of content items ranged from 600 to 1000. Figure 3 illustrates how the caching performance varied with the number of items in the GEANT [17] network, where routers with caching capability had a uniform cache size of one item. The GEANT network is a well-known real-world topology comprising 53 nodes and 74 edges. Within the network are 13 source nodes responsible for content production, 32 router nodes, and 8 receiver nodes that initiate requests. However, it is worth noting that only router nodes with a degree higher than 2 have cache capabilities, which amounts to 19 nodes in this case.

Figure 3 demonstrates that GNN-DDQN consistently outperformed all other caching strategies across different numbers of distinct content items. GNN-DDQN achieved a maximum improvement of 34.42% in CHR, 4.76% in ALT, 3.77% in APS, and 5.21% in ALL compared to MLP-DDQN. On average, GNN-DDQN surpassed LCD and PROB_CACHE by 41.33% and 103.92% in CHR, respectively. It also achieved significantly lower ALT, APS, and ALL than LCD and PROB_CACHE. Furthermore, the performance gap

between GNN-DDQN and LCE and CL4M was even more pronounced regarding all evaluation metrics.

Overall, GNN-DDQN consistently exhibited exceptional caching performance regardless of the number of content items. Its superiority over MLP-DDQN stemmed from its ability to facilitate cooperative caching among neighbouring router nodes. By efficiently utilizing the caching space of all router nodes, GNN-DDQN enhanced the network performance. Additionally, GNN-DDQN outperformed traditional caching algorithms by quickly capturing user preferences and proactively placing popular content on appropriate router nodes. Consequently, the cache hit ratio improved, alleviating network traffic congestion.

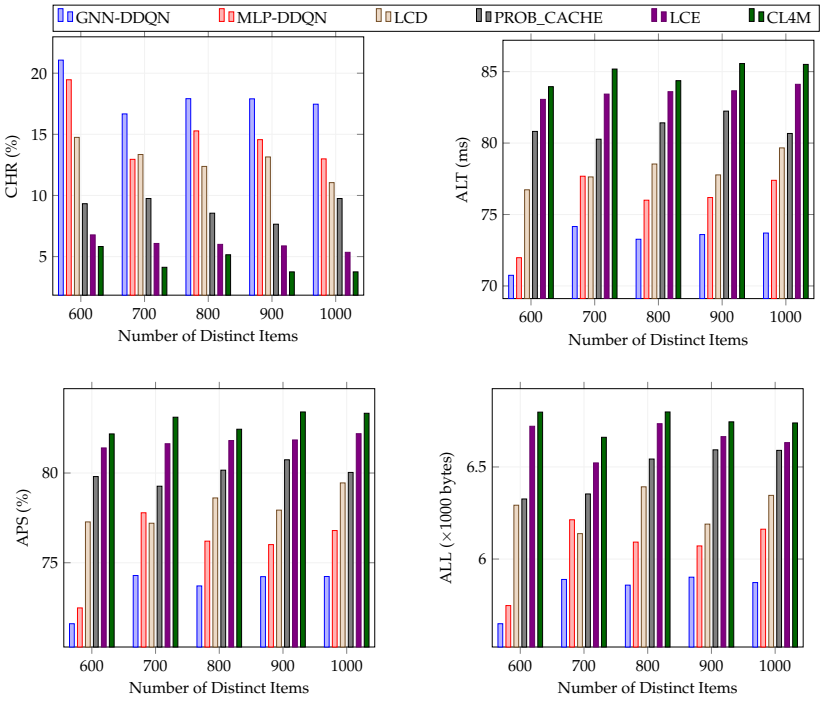

Figure 3. The cache performance of GNN-DDQN, MLP-DDQN, LCD, PROB_CACHE, LCE, and CL4M varied with the number of content items in the GEANT network.

5.2. Effect of Cache Size

This section investigates the performance of different caching schemes across various router cache sizes, defined as the number of content items. Figure 4 presents the caching performance of GNN-DDQN, MLP-DDQN, LCD, PROB_CACHE, LCE, and CL4M under different caching scenarios. The router cache sizes ranged from 1 to 4, while the number of content items was fixed at 1000.

GNN-DDQN exhibited a substantial performance advantage over MLP-DDQN when the cache size was limited to one item. For cache sizes of two and four, GNN-DDQN and MLP-DDQN performed similarly. However, when the cache size was set to three items, GNN-DDQN outperformed MLP-DDQN by achieving an 11.87% higher CHR, 3.57% lower ALT, 1.54% APS, and 2.20% lower ALL.

Significantly, regardless of the router cache size, GNN-DDQN consistently reduced the latency time by at least 14.96%, 29.88%, 92.20%, and 76.37% compared to LCD, PROB_CACHE, LCE, and CL4M, respectively. The advantages of GNN-DDQN stem from its ability to predict popular content in advance and proactively cache them.

5.3. Effect of Network Topology

To further evaluate the effectiveness of the proposed caching scheme, we conducted experiments on different network topologies, namely ROCKETFUEL [18], TISCALI [19], and GARR [19]. The aim was to assess the robustness of the caching scheme in diverse network environments.

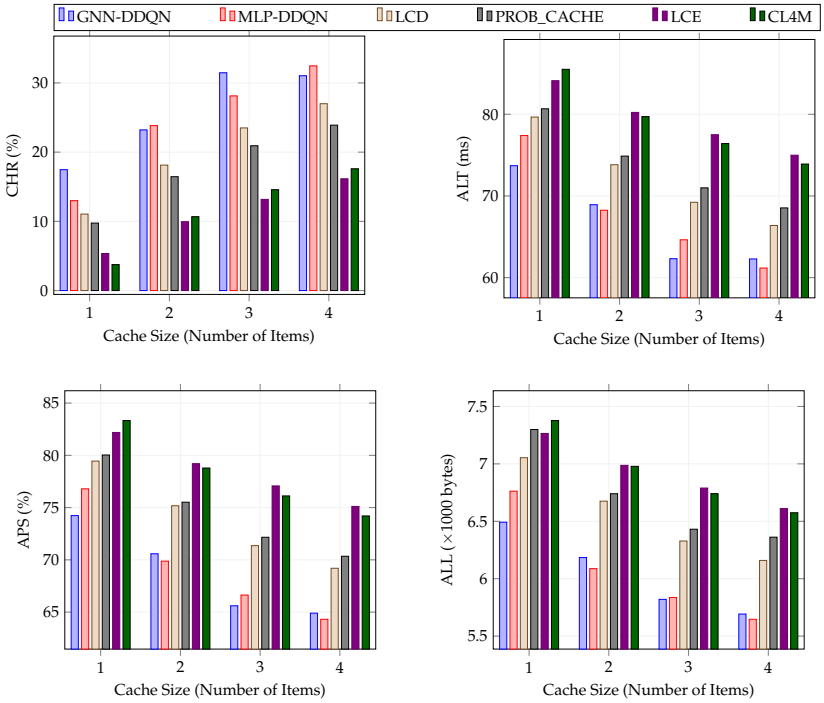

Figure 4. The cache performance of GNN-DDQN, MLP-DDQN, LCD, PROB_CACHE, LCE, and CL4M varied with the router's cache size in the GEANT network.

Table 5 presents the distribution of each network topology's source, router, and receiver nodes. It is important to note that, in the TISCALI network, only router nodes with a degree higher than 6 possess caching capabilities, resulting in 36 router nodes equipped with cache functionality.

Table 5. The number of source, router, and receiver nodes for different network topologies.

Topologies	Source Nodes	Router Nodes	Receiver Nodes
ROCKETFUEL [18]	10	104	104
TISCALI [19]	44	160	36
GARR [19]	13	27	21

This section evaluates the caching performance of different strategies in the ROCKETFUEL, TISCALI, and GARR network topologies. The experiments were conducted with an item number of 1000, and all routers with caching capabilities had a uniform cache size of one item. The results are summarized in Table 6.

Across all network topologies, GNN-DDQN consistently outperformed the other strategies. Specifically, in ROCKETFUEL, GNN-DDQN achieved a 2.89% higher CHR than MLP-DDQN. In TISCALI, the margin became even more significant, with GNN-DDQN achieving a 25.72% higher CHR than MLP-DDQN. These results highlight the superior caching performance of GNN-DDQN, particularly in large networks such as ROCKETFUEL and TISCALI.

Furthermore, GNN-DDQN demonstrated a significant margin over MLP-DDQN and other traditional caching schemes in the GARR network. This further emphasizes the robustness and effectiveness of GNN-DDQN across various network topologies.

Table 6. The caching performance of GNN-DDQN, MLP-DDQN, LCD, PROB_CACHE, LCE and CL4M in ROCKETFUEL, TISCALI, and GARR.

Topology / Strategy	ROCKETFUEL			
	CHR	**ALT**	**APS**	**ALL**
GNN-DDQN	18.41%	75.05 ms	73.78%	4081.90 bytes
MLP-DDQN	17.89%	75.27 ms	74.00%	4104.82 bytes
LCD	13.00%	80.12 ms	78.61%	4410.05 bytes
PROB_CACHE	9.32%	82.67 ms	78.78%	4425.06 bytes
LCE	8.35%	83.16 ms	79.36%	4598.08 bytes
CL4M	10.20%	82.13 ms	79.31%	4530.41 bytes
Topology / Strategy	**TISCALI**			
	CHR	**ALT**	**APS**	**ALL**
GNN-DDQN	15.57%	82.19 ms	82.76%	2498.73 bytes
MLP-DDQN	12.38%	84.41 ms	83.49%	2525.98 bytes
LCD	12.07%	85.14 ms	84.75%	2626.72 bytes
PROB_CACHE	7.90%	88.11 ms	85.55%	2656.99 bytes
LCE	7.22%	88.71 ms	86.03%	2692.65 bytes
CL4M	3.67%	91.22 ms	86.61%	2727.35 bytes
Topology / Strategy	**GARR**			
	CHR	**ALT**	**APS**	**ALL**
GNN-DDQN	12.18%	71.96 ms	74.48%	5559.79 bytes
MLP-DDQN	5.65%	76.31 ms	76.38%	5762.81 bytes
LCD	8.12%	74.77 ms	76.26%	5665.82 bytes
PROB_CACHE	4.02%	77.73 ms	77.48%	5749.57 bytes
LCE	3.67%	77.91 ms	77.76%	5832.152 bytes
CL4M	4.32%	77.42 ms	77.34%	5801.03 bytes

6. Conclusions

In this paper, we introduced GNN-DDQN, an intelligent caching scheme designed for the SDN-ICN scenario. GNNs have gained significant attention recently for their ability to handle graph-structured data. Leveraging this capability, we applied GNNs to process network topologies, enabling cooperative caching among nodes and promoting a wider variety of cached content. By integrating GNNs into DRL, our proposed approach empowered the DRL agent to make caching decisions for all nodes in the network with only one forward pass through the neural network. This integration not only streamlined the caching decision-making process but also harnessed the power of GNN-DRL synergy in optimizing the caching strategies.

Firstly, we generated user preferences for content based on a real-world dataset. This step ensured that the evaluation reflected realistic user behaviour and content demand patterns. Next, we developed a GNN-DDQN agent within the SDN controller, enabling the agent to make intelligent caching decisions for all router nodes equipped with caching capabilities in the ICN network. Finally, we compared the performance of our proposed GNN-DDQN caching scheme with the state-of-the-art MLP-DDQN strategy and several classical benchmark caching schemes, including LCD, PROB_CACHE, CL4M, and LCE. The extensive evaluation revealed that GNN-DDQN consistently outperformed MLP-DDQN in most scenarios. Notably, in the best-case scenario, GNN-DDQN achieved a remarkable 34.42% higher CHR, a 4.76% lower ALT, a 3.77% lower APS, and a 5.21% lower ALL com-

pared to MLP-DDQN. Furthermore, GNN-DDQN demonstrated superior performance compared to classical caching schemes. To assess the robustness of our proposed scheme, we conducted experiments on benchmark network topologies, including GEANT, ROCKET-FUEL, TISCALI, and GARR. GNN-DDQN consistently delivered outstanding performance across these diverse network topologies, reinforcing its reliability and applicability in real-world scenarios.

Some potential directions for future research include the following.

- Latency Consideration: Investigating the latency of the SDN controller and exploring techniques to mitigate the latency issue when dealing with a large number of network nodes.
- IoV-Based Environment: Integrating the proposed caching strategy in an IoV environment. This may involve studying the unique characteristics of vehicular networks and exploring how the methodology can be adapted to optimize content caching and delivery in such dynamic and mobile scenarios.

Author Contributions: Conceptualization, J.H., T.T., H.L. and A.N.; Methodology, J.H., T.T. and H.L.; Software, J.H. and T.T.; Supervision, A.N.; Validation, T.T. and H.L.; Visualization, J.H.; Writing—original draft, J.H. and T.T.; Writing—review and editing, H.L. and A.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data will be available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhang, Z.; Lung, C.H.; Wei, X.; Chen, M.; Chatterjee, S.; Zhang, Z. In-network Caching for ICN-based IoT (ICN-IoT): A Comprehensive Survey. *IEEE Internet Things J.* **2023**. [CrossRef]
2. Musa, S.S.; Zennaro, M.; Libsie, M.; Pietrosemoli, E. Convergence of Information-Centric Networks and Edge Intelligence for IoV: Challenges and Future Directions. *Future Internet* **2022**, *14*, 192. [CrossRef]
3. Gür, G.; Kalla, A.; de Alwis, C.; Pham, Q.V.; Ngo, K.H.; Liyanage, M.; Porambage, P. Integration of ICN and MEC in 5G and Beyond Networks: Mutual Benefits, Use Cases, Challenges, Standardization, and Future Research. *IEEE Open J. Commun. Soc.* **2022**, *3*, 1382–1412. [CrossRef]
4. Aldaoud, M.; Al-Abri, D.; Awadalla, M.; Kausar, F. Leveraging ICN and SDN for Future Internet Architecture: A Survey. *Electronics* **2023**, *12*, 1723. [CrossRef]
5. Sun, S.; Zhou, J.; Wen, J.; Wei, Y.; Wang, X. A DQN-based cache strategy for mobile edge networks. *Comput. Mater. Contin.* **2022**, *71*, 3277–3291. [CrossRef]
6. Li, J.; Tang, J.; Li, J.; Zou, F. Deep reinforcement learning for intelligent computing and content edge service in ICN-based IoV. In Proceedings of the 2021 IEEE International Conference on Communications Workshops (ICC Workshops), Montreal, QC, Canada, 14–23 June 2021; pp. 1–7.
7. Wu, L.; Cui, P.; Pei, J.; Zhao, L.; Song, L. *Graph Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2022.
8. Almasan, P.; Suárez-Varela, J.; Rusek, K.; Barlet-Ros, P.; Cabellos-Aparicio, A. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *Comput. Commun.* **2022**, *196*, 184–194. [CrossRef]
9. Jiang, W.; Luo, J. Graph neural network for traffic forecasting: A survey. *Expert Syst. Appl.* **2022**, *207* , 117921. [CrossRef]
10. Fan, S.; Wang, X.; Shi, C.; Cui, P.; Wang, B. Generalizing Graph Neural Networks on Out-Of-Distribution Graphs. *arXiv* **2021**, arXiv:2111.10657.
11. Rusek, K.; Suárez-Varela, J.; Almasan, P.; Barlet-Ros, P.; Cabellos-Aparicio, A. RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 2260–2270. [CrossRef]
12. Suárez-Varela, J.; Carol-Bosch, S.; Rusek, K.; Almasan, P.; Arias, M.; Barlet-Ros, P.; Cabellos-Aparicio, A. Challenging the generalization capabilities of Graph Neural Networks for network modeling. In Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos, Beijing, China, 19–23 August 2019; pp. 114–115.
13. Almasan, P.; Suárez-Varela, J.; Badia-Sampera, A.; Rusek, K.; Barlet-Ros, P.; Cabellos-Aparicio, A. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *arXiv* **2019**, arXiv:1910.07421.
14. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
15. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.S. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182.

16. Harper, F.M.; Konstan, J.A. The movielens datasets: History and context. *Acm Trans. Interact. Intell. Syst. TIIS* **2015**, *5*, 1–19. [CrossRef]
17. Géant Homepage. 2020. Available online: https://geant3plus.archive.geant.net/Pages/home.html (accessed on 18 August 2022).
18. Spring, N.; Mahajan, R.; Wetherall, D. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Comput. Commun. Rev.* **2002**, *32*, 133–145. [CrossRef]
19. Knight, S.; Nguyen, H.X.; Falkner, N.; Bowden, R.; Roughan, M. The internet topology zoo. *IEEE J. Sel. Areas Commun.* **2011**, *29*, 1765–1775. [CrossRef]
20. Zhang, L.; Estrin, D.; Burke, J.; Jacobson, V.; Thornton, J.D.; Smetters, D.K.; Zhang, B.; Tsudik, G.; Massey, D.; Papadopoulos, C.; et al. Named data networking (ndn) project. *Relat. Téc. NDN-0001 Xerox Palo Alto Res. Cent.-PARC* **2010**, *157*, 158.
21. Laoutaris, N.; Che, H.; Stavrakakis, I. The LCD interconnection of LRU caches and its analysis. *Perform. Eval.* **2006**, *63*, 609–634. [CrossRef]
22. Psaras, I.; Chai, W.K.; Pavlou, G. Probabilistic in-network caching for information-centric networks. In Proceedings of the Second Edition of the ICN Workshop on Information-Centric Networking, Helsinki, Finland, 13–17 August 2012; pp. 55–60.
23. Chai, W.K.; He, D.; Psaras, I.; Pavlou, G. Cache "less for more" in information-centric networks. In Proceedings of the International Conference on Research in Networking, Chennai, India, 1–3 February 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 27–40.
24. Li, Z.; Simon, G.; Gravey, A. Caching policies for in-network caching. In Proceedings of the 2012 21st International Conference on Computer Communications and Networks (ICCCN), Munich, Germany, 30 July–2 August 2012; pp. 1–7.
25. Shailendra, S.; Sengottuvelan, S.; Rath, H.K.; Panigrahi, B.; Simha, A. Performance evaluation of caching policies in ndn-an icn architecture. In Proceedings of the 2016 IEEE Region 10 Conference (TENCON), Singapore, 22–25 November 2016; pp. 1117–1121.
26. Munikoti, S.; Agarwal, D.; Das, L.; Halappanavar, M.; Natarajan, B. Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications. *arXiv* **2022**, arXiv:2206.07922.
27. Nomikos, N.; Zoupanos, S.; Charalambous, T.; Krikidis, I. A Survey on Reinforcement Learning-Aided Caching in Heterogeneous Mobile Edge Networks. *IEEE Access* **2022**, *10* , 4380–4413. [CrossRef]
28. Zhao, L.; Ran, Y.; Wang, H.; Wang, J.; Luo, J. Towards Cooperative Caching for Vehicular Networks with Multi-level Federated Reinforcement Learning. In Proceedings of the ICC 2021-IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6.
29. Song, C.; Xu, W.; Wu, T.; Yu, S.; Zeng, P.; Zhang, N. QoE-driven edge caching in vehicle networks based on deep reinforcement learning. *IEEE Trans. Veh. Technol.* **2021**, *70*, 5286–5295. [CrossRef]
30. Aung, N.; Dhelim, S.; Chen, L.; Lakas, A.; Zhang, W.; Ning, H.; Chaib, S.; Kechadi, M.T. VeSoNet: Traffic-Aware Content Caching for Vehicular Social Networks Using Deep Reinforcement Learning. *IEEE Trans. Intell. Transp. Syst.* **2023** . [CrossRef]
31. Zhang, D.; Wang, W.; Zhang, J.; Zhang, T.; Du, J.; Yang, C. Novel edge caching approach based on multi-agent deep reinforcement learning for Internet of vehicles. *IEEE Trans. Intell. Transp. Syst.* **2023** . [CrossRef]
32. He, P.; Cao, L.; Cui, Y.; Wang, R.; Wu, D. Multi-Agent Caching Strategy for Spatial-Temporal Popularity in IoV. *IEEE Trans. Veh. Technol.* **2023** . [CrossRef]
33. Liu, L.; Yuan, X.; Zhang, N.; Chen, D.; Yu, K.; Taherkordi, A. Joint Computation Offloading and Data Caching in Multi-Access Edge Computing Enabled Internet of Vehicles. *IEEE Trans. Veh. Technol.* **2023** . [CrossRef]
34. Hou, J.; Xia, H.; Lu, H.; Nayak, A. A gnn-based approach to optimize cache hit ratio in ndn networks. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6.
35. Hou, J.; Lu, H.; Nayak, A. GNN-GM: A Proactive Caching Scheme for Named Data Networking. In Proceedings of the 2022 IEEE International Conference on Communications Workshops (ICC Workshops), Seoul, Republic of Korea, 16–20 May 2022; pp. 1–6.
36. Peng, H.; Du, B.; Liu, M.; Liu, M.; Ji, S.; Wang, S.; Zhang, X.; He, L. Dynamic graph convolutional network for long-term traffic flow prediction with reinforcement learning. *Inf. Sci.* **2021**, *578*, 401–416. [CrossRef]
37. Yang, S.; Yang, B.; Kang, Z.; Deng, L. IHG-MA: Inductive heterogeneous graph multi-agent reinforcement learning for multi-intersection traffic signal control. *Neural Netw.* **2021**, *139*, 265–277. [CrossRef]
38. Chen, S.; Dong, J.; Ha, P.; Li, Y.; Labi, S. Graph neural network and reinforcement learning for multi-agent cooperative control of connected autonomous vehicles. *Comput.-Aided Civ. Infrastruct. Eng.* **2021**, *36*, 838–857. [CrossRef]
39. Zhou, X.; Bilal, M.; Dou, R.; Rodrigues, J.J.; Zhao, Q.; Dai, J.; Xu, X. Edge Computation Offloading with Content Caching in 6G-Enabled IoV. *IEEE Trans. Intell. Transp. Syst.* **2023** . [CrossRef]
40. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
41. Breslau, L.; Cao, P.; Fan, L.; Phillips, G.; Shenker, S. Web caching and Zipf-like distributions: Evidence and implications. In Proceedings of the IEEE INFOCOM'99, Conference on Computer Communications, Proceedings, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, The Future is Now (Cat. No. 99CH36320), New York, NY, USA, 21–25 March 1999; Volume 1, pp. 126–134.
42. Rendle, S.; Krichene, W.; Zhang, L.; Anderson, J. Neural collaborative filtering vs. matrix factorization revisited. In Proceedings of the Fourteenth ACM Conference on Recommender Systems, Virtual, 22–26 September 2020; pp. 240–248.
43. Chen, B.; Yang, C. Caching policy optimization for D2D communications by learning user preference. In Proceedings of the 2017 IEEE 85th Vehicular Technology Conference (VTC Spring), Sydney, Australia, 4–7 June 2017; pp. 1–6.

44. Chen, B.; Yang, C. Caching policy for cache-enabled D2D communications by learning user preference. *IEEE Trans. Commun.* **2018**, *66*, 6586–6601. [CrossRef]
45. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive representation learning on large graphs. *arXiv* **2017**, arXiv:1706.02216.
46. Saino, L.; Psaras, I.; Pavlou, G. Icarus: A caching simulator for information centric networking (icn). In Proceedings of the SimuTools, ICST, Lisbon, Portugal, 17–19 March 2014; Volume 7, pp. 66–75.
47. Zhang, B.; Ng, T.E.; Nandi, A.; Riedi, R.; Druschel, P.; Wang, G. Measurement based analysis, modeling, and synthesis of the internet delay space. In Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, Rio de Janeriro, Brazil, 25–27 October 2006; pp. 85–98.