Enhancing Network Efficiency: Empowering Intelligent Caching with Graph Neural Networks in Information-Centric Networking

Jiacheng Hou

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
degree of Doctor of Philosophy in Computer Science

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

Abstract

In the current era of ever-increasing data volumes and network traffic, efficient caching mechanisms play an important role in mitigating latency, managing network workload, and enhancing content delivery efficiency. This thesis explores the application of Graph Neural Networks (GNNs) in intelligent caching within Information-Centric Networking (ICN)-based environments, aiming to optimize content caching, maximize cache hit ratios, and improve overall system performance.

The thesis first introduces a GNN-based caching strategy for ICN networks, leveraging GNNs to predict content popularity and make informed caching replacement decisions. Subsequently, it presents a GNN-based proactive caching placement strategy for ICN networks, capturing user preferences to optimize caching placement decisions in order to enhance the overall user experience. Furthermore, the thesis delves into intelligent caching with GNN-based Deep Reinforcement Learning (DRL) in Software-Defined Networking-based ICN (SDN-ICN) networks, utilizing a centralized GNN-Double Deep Q-Network (GNN-DDQN) agent to make proactive caching placement decisions for all network nodes. Lastly, it presents a fully distributed caching strategy, where each edge node maintains a Spatial-Temporal Graph Attention Network-Soft Actor-Critic (STGAN-SAC) agent to make proactive caching placement decisions in a three-tier edge network.

The thesis aims to develop a comprehensive framework for intelligent caching utilizing GNNs, evaluating the effectiveness of the proposed strategies using synthetic and real-world datasets and simulations across various network topologies. The experimental results demonstrate advancements over state-of-the-art caching approaches.

Acknowledgements

I would like to thank my family and all my friends who supported me physically and mentally. Additionally, I am grateful to my boyfriend, Haolong Zhang, who gave me advice when I needed it most. I also thank my senior colleague, Haoye Lu, who extensively guided me. Finally, I express my most profound appreciation to Prof. Amiya Nayak. He provided guidance, support, advice, and ideas throughout my thesis work. I would not have been able to complete my program if it were not for him.

Contents

List of Tables

List of Figures

Acronyms

1D-CNN One-Dimensional Convolutional Neural Network.

2D Two-Dimensional.

AC Actor-Critic.

ALL Average Link Load.

ALT Average Latency.

APS Average Path Stretch.

AV Autonomous Vehicles.

BCE Binary Cross-Entropy.

BFS Breadth-first Search.

BHR Byte Hit Ratio.

BS Base Station.

CAGR Compound Annual Growth Rate.

CAV Connected Autonomous Vehicle.

CCN Content-Centric Network.

CDN Content Delivery Network.

CHR Cache hit ratio.

CI Confidence Interval.

CL4M Cache Less for More.

CNN Convolutional Neural Network.

CRNN Convolutional Recurrent Neural Network.

D2D Device-to-Device.

DDGARQN Double Deep Graph Attention Recurrent Q-Network.

DDQN Double Deep Q-Network.

DDRQN Double Deep Recurrent Q-Network.

DGCNN Deep Graph Convolutional Neural Network.

DNN Deep Neural Network.

DQL Deep Q-Learning.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

EN Edge Node.

ES Edge Server.

FIFO First-in-first-out.

GAT Graph Attention Network.

GBS Ground Base Station.

GCN Graph Convolutional Network.

GCPN Graph Convolutional Policy Network.

GMM Gaussian Mixture Model.

GNN Graph Neural Network.

GRU Gated Recurrent Unit.

ICN Information-Centric Networking.

IGMC Inductive Matrix Completion.

IHG-MA Inductive Heterogeneous Graph Multi-Agent Actor–Critic.

IOT Internet of Things.

IoV Internet of Vehicles.

ISP Internet Service Providers.

LCD Leave Copy Down.

LCE Leave Copy Everywhere.

LFU Least Frequently Used.

LLM Large Language Model.

LPC Least Popular Content.

LRU Least Recently Used.

LSTM Long Short-Term Memory.

LSTM-ED Long Short-Term Memory - Encoder Decoder.

MAA2C Multi-Agent Advantage Actor-Critic.

MAAC Multi-Agent Actor-Critic.

MARL Multi-Agent Reinforcement Learning.

MEC Mobile Edge Computing.

MLP Multilayer Perceptron.

NCF Neural Collaborative Filtering.

NDN Named Data Networking.

NMF Non-Negative Matrix Factorization.

OPC Optimal Popular Content.

PROB_CACHE Probabilistic Caching.

QoE Quality of Experience.

RCO Randomly Copy On.

ReLU Rectified Linear Unit.

RGCN Relational Graph Convolutional Neural Network.

RNN Recurrent Neural Network.

RS Recommendation System.

RSU Roadside Unit.

SAC Soft Actor-Critic.

SAE Stacked Autoencoders.

SDN Software-Defined Networking.

SGD Stochastic Gradient Descent.

SMAAC Soft Multi-Agent Actor-Critic.

SNN Stimulable Neural Network.

STGAN-SAC spatial-Temporal Graph Attention Network-Soft Actor-Critic.

STGCN Spatio-Temporal Graph Convolutional Networks.

STGNN Spatial-Temporal Graph Neural Network.

TD3 Twin Delayed Deterministic Policy Gradient.

UAV Unmanned Aerial Vehicle.

UAV-BS Unmanned Aerial Vehicle-based Aerial Base Station.

Chapter 1

Introduction

With the surge in network traffic, effective content dissemination plays an important role in ensuring seamless communication and optimal resource utilization. Information-Centric Networking (ICN) emerges as a promising paradigm shifting the focus from traditional host-centric to content-centric communication, thereby fostering efficient content delivery and enhancing network performance.

This thesis explores the enhancement of network efficiency through the integration of intelligent caching mechanisms based on Graph Neural Networks (GNNs) within the ICN framework. GNNs, a subset of deep learning techniques, have a unique capability to model complex relationships and dependencies inherent in graph-based data, making them well-suited for optimizing caching strategies in networks.

This chapter is divided into four parts. Firstly, it provides a concise overview of the main techniques utilized in this thesis. Secondly, it outlines the motivation and objectives of the research regarding the application of GNNs in caching within an ICN-based context. Thirdly, it introduces the contributions of this research. Additionally, the fourth part outlines the list of publications related to the thesis. Finally, the last part presents the organization of the thesis.

1.1 Overview

1.1.1 ICN

ICN [1] represents an emerging networking paradigm that shifts the focus from traditional host-centric to content-centric communication. Unlike IP-based networks, ICN differs in two primary aspects: 1) Distributed cache capability: Each ICN node in the network is equipped with cache storage, allowing it to store content locally. 2) Content-based communication: In contrast to IP-based networks where users send request packets indicating the destination node's IP address, users in the ICN network send request packets specifying the desired content identifier, such as content name. As a result, along the packet forwarding path, any ICN node having the requested content in its cache can satisfy the request.

Specifically, in ICN, users within the network send a request packet, named an "Interest" packet, which specifies the desired content identifier. Each ICN node manages to cache a variety of contents within its cache store, adhering to its cache space limitations. Upon receiving an "Interest" packet, an ICN node searches its local cache first for the requested content and promptly responds with a "Data" packet if available. Otherwise, the ICN node forwards the "Interest" packet to its neighbor along the path to the server node, which publishes the requested content. If most user requests can be satisfied by ICN nodes without reaching the remote server node, network congestion can be alleviated, and users' Quality of Experience (QoE) can be improved. Therefore, developing an efficient caching mechanism in ICN is the key to tackling challenges stemming from the increasing demand for content, the exponential growth of data volumes, and the limitations of traditional IP-based networks.

1.1.1.1 Caching Placement strategy

The caching placement strategy focuses on determining "what" and "where" to cache, i.e., what content to cache and at which network node. Also, caching placement strategies can be extensively categorized into two main types: reactive and proactive.

- Reactive caching placement strategy: In this approach, each ICN node decides whether to cache the corresponding content when a "Data" packet traverses the node. As a result, cache decisions occur after a user sends an "Interest" packet and a "Data" packet is generated to satisfy the "Interest" packet. Specifically, when no user has sent an "Interest" packet, none of the ICN nodes cache any contents, and the "Interest" packet has to be forwarded to the server node to be satisfied. Once the server node produces a "Data" packet, any ICN node along the path from the server node to the user can decide whether to cache the content or not.

- Proactive caching placement strategy: This strategy involves actively caching content at an ICN node before any request for that content is issued. Specifically, even if none of the users have issued an "Interest" packet for a specific content, the content can be proactively cached in an ICN node. Then, along the forwarding path, if an ICN node has the requested content in its cache store, the "Interest" packet can be fulfilled immediately without the need to be forwarded to the server node.

1.1.1.2 Caching Replacement Strategy

The caching replacement strategy addresses the question of "which" content to evict first when the cache space of an ICN node reaches capacity but a new content needs to be cached, based on the decision of the caching placement strategy.

1.1.2 NDN

Named Data Networking (NDN) [2] is a specific ICN architecture that has drawn significant attention in recent years. NDN builds upon the fundamental principles of ICN and introduces a hierarchical naming scheme to facilitate content retrieval and distribution. Similar to ICN, instead of relying on specific IP addresses, NDN utilizes unique content names to identify and access content. Content names in NDN follow a hierarchical structure, allowing efficient content routing and caching at different levels of the network.

In this thesis, we explore both network architectures: the specific architecture NDN and the more general architecture ICN. However, regardless of the architecture, we mainly focus on their distributed caching capability and content-based communication characteristics. As a result, the core focus we delve into is to improve caching performance in various caching-enabled environments.

1.1.3 SDN

Software-Defined Networking (SDN) [3] is a network architecture that separates the control plane from the data plane, enabling centralized network management and programmability. The control plane, located in a centralized controller, makes decisions and orchestrates network operations, while the data plane handles the forwarding of network traffic. This separation provides high flexibility in network management by allowing network administrators to dynamically configure and control network behaviour through software applications. SDN's ability to enable network programmability through open interfaces and standardized protocols, such as OpenFlow [4], empowers operators to define network policies, allocate resources, and implement advanced traffic management strategies. One chapter of the thesis utilizes the concept of a centralized view and

control over the network in the SDN controller, where the central caching agent resides in the SDN control plane and makes caching decisions for all the network nodes in the data plane.

1.1.4 GNN

GNNs [5–7] have emerged as a powerful tool for analyzing and learning from graph-structured data. In contrast to other popular neural network architectures like Convolutional Neural Networks (CNNs), 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.

GNNs operate by propagating information between nodes in a graph through multiple layers of neural networks. Each node in the graph captures both its own features and the features of its neighbouring nodes, allowing GNNs to capture rich structural and relational information. This enables GNNs to learn and extract valuable insights from graph data, and be applied in various graph-related tasks, such as node classification, link prediction, graph-level representation learning and so on.

Recent advancements in GNNs research have further expanded their capabilities by introducing novel architectures, including convolutional-based GNNs [8], attention-based GNNs [9], transformer-based GNNs [10], and federated-based GNNs [11]. These advancements have enhanced the applicability of GNNs in various applications, such as cancer molecular classification [12], intelligent transportation systems [13], etc. This thesis applies GNNs in solving caching difficulties and focuses on the following two main aspects:

1. Applying GNNs to model network architecture to capture spatial dependencies among neighbouring nodes for cooperative caching decisions considering different node positions and relationships.

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

1.1.5 DRL

Deep Reinforcement Learning (DRL) [14] 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 [15], proximal policy optimization algorithm [16], etc. Furthermore, DRL has been applied in diverse applications such as task offloading in wireless networks [17], routing optimization [18], 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 policy accordingly.

1.2 Motivation and Objectives

We find ourselves in the midst of the digital age, witnessing an unprecedented surge in internet traffic. This surge is primarily driven by the widespread adoption of mobile devices, streaming services, cloud computing, and the expanding Internet of Things (IoT) ecosystem. Moreover, the COVID-19 pandemic has further accelerated digital transformation, amplifying the reliance on online connectivity. With remote work, online education, and virtual entertainment becoming commonplace, internet traffic is increasing exponentially, which places immense pressure on existing network infrastructure. According to [19], global internet traffic surpassed 100 billion GB in 2022, with a projected Compound Annual Growth Rate (CAGR) of 24% between 2021 and 2026. As a result, it becomes imperative to develop innovative solutions to enhance network efficiency, alleviate congestion, and ensure seamless content delivery.

ICN emerges as a promising solution to address these challenges. By providing in-network caching capability across the network, ICN offers various benefits, including reduced latency, alleviated network congestion, and enhanced content delivery efficiency. However, caching poses significant challenges due to the limited cache space available at each ICN node and the vast number of contents across the network.

Recent advancements in deep learning-based caching strategies (see papers in related works) have shown promise in optimizing caching performance in ICN. However, a notable gap exists between the current state-of-the-art caching strategies and the application of GNNs. GNNs, as an emerging neural network architecture, offer a unique advantage in modelling graph-based data [20, 21], including diverse aspects such as network topology [22], road intersections [23], etc. By leveraging the power of message-passing between nodes [24], GNNs can effectively capture and comprehend the intricate relationships and spatial dependencies among neighbouring nodes within the network.

This presents an opportunity to apply GNNs in developing caching strategies, employing GNNs to model the network architecture. By utilizing GNN's spatial learning capabilities, our main objective is to foster cooperative caching among neighbouring nodes. Specifically, we aim to employ GNNs to realize the following cooperative caching scenarios:

1. Network nodes collaborate to cache diverse content to maximize caching diversity.
2. Network nodes with low traffic volume can assist in caching content for their busier neighbouring nodes, optimizing resource utilization.
3. Learn packet forwarding paths between users and servers and use this information to infer optimal caching decisions.
4. Integration of caching with recommender systems, where GNNs predict users' ratings of unviewed content, enabling personalized content caching.

In summary, our primary goal is to enhance network efficiency, reduce latency, and optimize content delivery in ICN-based networks through the application of GNNs.

1.3 Contributions

In this thesis, we aim to make significant contributions to the field of caching in ICN by leveraging cutting-edge techniques, including GNNs and DRL. Our contributions are built upon the findings presented in five conference papers and three journal papers:

- Firstly, we introduce a GNN-based caching replacement strategy in NDN. To our knowledge, our work represents the first application of GNNs to address caching challenges in ICN environments. Our GNN-based model integrates One-dimensional

Convolutional Neural Network (1D-CNN) [25] and GraphSAGE [26] layers. These layers capture temporal dependencies of content popularity based on the past content request numbers and spatial dependencies of neighbouring network nodes based on the predicted content popularity across the network, respectively. Through comprehensive evaluations using the Mini-NDN [27] simulator, we demonstrate that our GNN-based caching strategy outperforms other deep learning-based approaches, achieving higher cache hit ratios and lower latency.

- Secondly, in contrast to our previous contribution, where we primarily rely on past content request numbers and the network architecture to predict future content caching popularity and make caching replacement decisions, this contribution focuses on predicting user preferences from another perspective in order to make content placement decisions. We leverage information typically used in recommender systems, such as content features (e.g., movie attributes), user profiles, and user ratings of content. We aim to proactively cache content that users are likely to request and give a high rating in the future. To achieve this goal, we employ a GNN-based model called Inductive Matrix Completion (IGMC) [28] to predict user ratings of unwatched movies. Subsequently, we aggregate the predicted ratings of each movie to determine the gain of caching that movie. Based on these gains, we propose a gain-based caching placement algorithm to decide which movies to cache in advance for each network node. We validate our strategy using a real-world dataset MovieLens 100K [29] in a real-world network topology GEANT [30] in the simulator Mini-NDN [27]. Experimental results demonstrate that our strategy achieves higher cache hit ratios, reduced latency, and lower server load compared to state-of-the-art strategies.

- Thirdly, unlike the previous two contributions that involve training deep neural network models using collected data before deploying them into the network, this contribution focuses on solving caching challenges using online learning. Specifically, we integrate GNN with DRL to make sequential proactive caching placement decisions in the SDN-ICN context. A centralized agent, named GNN-Double Deep Q-network (DDQN) [31], operates within the SDN controller, and has a global view of the entire network. At each step, the agent utilizes a single forward pass of its neural network to make proactive caching decisions for all network nodes. These decisions are based on factors such as the content request number, the content caching status, and the content publication status. To generate realistic user traffic, we develop a statistical model based on the real-world dataset MovieLens 100K [29]. Through extensive simulations conducted on the Icarus simulator [32], across various scenarios and network topologies including GEANT [33], ROCKETFUEL [34], TISCALI [35], and GARR [35], our proposed caching scheme consistently outperforms state-of-the-art DRL-based caching strategies and benchmark schemes.

- Lastly, in contrast to the previous contribution where a centralized agent is responsible for making caching decisions for all network nodes, which can lead to increased workload linearly proportional to the number of network nodes, this contribution introduces a fully distributed proactive caching placement strategy termed "Spatial-Temporal Graph Attention Network-Soft Actor-Critic" (STGAN-SAC) in a three-tier edge network. Specifically, each Base Station (BS) maintains a STGAN-SAC agent and independently makes caching decisions based on its own content request information, as well as its one-hop neighbours, along with their caching statuses. Additionally, we consider dynamic user preferences over time and assume users can move between the coverage areas of neighbouring BSs. We

conduct experiments using the Icarus simulator [32], and simulation results demonstrate that STGAN-SAC efficiently captures user preference dynamics and adapts its caching status to further enhance network performance.

By combining these contributions, we aim to advance the caching field in ICN by leveraging GNNs, fostering cooperative caching, improving cache hit ratios, reducing latency, and enhancing overall network performance. These contributions will pave the way for more efficient and effective caching strategies in ICN-based environments, addressing the challenges posed by the increasing demands of modern network traffic.

1.4 Outline

This thesis is organized as follows: Chapter 1 provides the overview of the main technologies used in this thesis, and the motivation and objectives behind adopting GNNs to design caching strategies in ICN environments. Chapter 2 reviews various caching strategies, including traditional, popularity-based, and state-of-the-art approaches. Chapter 3 introduces a novel caching replacement strategy utilizing 1D-CNN [25] and GraphSage [26]. Chapter 4 proposes an IGMC-based [28] proactive caching placement strategy. Chapter 5 presents a GNN-DRL-based proactive caching placement strategy for the SDN-ICN environment. Chapter 6 shows a GNN-multi-agent-DRL-based proactive caching placement strategy in a three-tire edge network. The last chapter summarizes the work done in the thesis and discusses potential future research directions.

1.5 List of Publications

Conferences Publication:

- Hou, J., Tao, T., Lu, H., Nayak, A. (2023) A Graph-based Spatial-Temporal Deep Reinforcement Learning Model for Edge Caching. In 2023 IEEE Global Communications Conference (GLOBECOM).
- Hou, J., Tao, T., Lu, H., Nayak, A. (2023) An Optimized GNN-based Caching Scheme for SDN-based Information-Centric Networks. In 2023 IEEE Global Communications Conference (GLOBECOM).
- Hou, J., Nayak, A. (2023). Spatial-Temporal Graph Attention-based Multi-Agent Reinforcement Learning in Cooperative Edge Caching. In 2023 IEEE International Conference on Communications (ICC).
- Hou, J., Lu, H., Nayak, A. (2022). GNN-GM: A Proactive Caching Scheme for Named Data Networking. In 2022 IEEE International Conference on Communications Workshops (ICC Workshops).
- Hou, J., Xia, H., Lu, H., Nayak, A. (2021). A gnn-based approach to optimize cache hit ratio in ndn networks. In 2021 IEEE Global Communications Conference (GLOBECOM).

Journals Publications:

- Hou, J., Xia, H., Lu, H., Nayak, A. (2022). A Graph Neural Network Approach for Caching Performance Optimization in NDN Networks. IEEE Access, 10, 112657-112668.

- Hou, J., Tao, T., Lu, H., Nayak, A. (2023). Intelligent Caching with Graph Neural Network-based Deep Reinforcement Learning on SDN-based ICN. Future Internet, 15(8): 251.

- Hou, J., Lu, H., Nayak, A. (2023). A GNN-based proactive caching strategy in NDN networks. Peer-to-Peer Networking and Applications, 16(2), 997-1009.

Chapter 2

Related Works

This chapter provides a comprehensive review of different Graph Neural Network (GNN) architectures, their applications, and various caching strategies. Regarding caching strategies, we first explore traditional strategies, followed by popularity-based strategies that make caching decisions mainly based on content popularities and node positions. These popularities are usualy computed from various network factors and predefined rules. Last but not least, we explore emerging Deep Neural Network (DNN)-based caching strategies from recent research.

2.1 GNNs

GNNs [36] represent an emerging field within deep learning. In recent years, GNN-based models have been used in network problems since a network is naturally a graph with nodes and edges. In particular, GNN has been widely used in traffic prediction problems, such as road traffic flow and speed prediction. Paper [37] suggests that GNN can achieve state-of-the-art performance in traffic forecasting by modelling the graph structures in transportation systems. As suggested in paper [38], GNN is efficient for node level,

edge level, and graph level prediction or classification tasks. Specifically, previous work [8] introduces Graph Convolutional Network (GCN) for semi-supervised learning. The GCN model directly operates on graph data and predicts labels for unlabeled nodes. Also, paper [39] proposes a Spatio-Temporal Graph Convolutional Networks (STGCN) framework, which contains One-dimensional Convolutional Neural Network (1D-CNN) and GCN layers to make node-level traffic flow predictions. The 1D-CNN layer captures temporal dependency, and GCN captures spatial dependency. Also, [40] provides a novel GNN-based link prediction framework, SEAL, to predict whether a link exists between two subgraphs. Additionally, paper [41] proposes a Deep Graph Convolutional Neural Network (DGCNN) architecture to predict labels on the graph level.

2.1.1 GNNs in User-Item Rating Predictions

Recently, GNNs have also shown outstanding performance in Recommendation Systems (RSs) by modelling user-content pairs as a bipartite graph with edge weights representing users' preferences for the content [28, 42]. Paper [43] proposes an inductive node-level Graph Convolutional Neural (GCN) [8] framework to make item recommendations to users. The authors of [28] proposed an Inductive Matrix Completion (IGMC) model to predict the ratings between users and items with encouraging performance. They viewed the users and items rating matrix as a bipartite graph with two types of nodes, user-type and item-type. Edges only exist between users and items with ratings as labels. In this case, the rating prediction problem is converted to an edge-label prediction problem. Experimental results have demonstrated that IGMC can tackle the cold start problem encountered in the Matrix Factorization (MF) approach.

2.1.2 GNNs Meet DRL

DRL has gained popularity in addressing sequential decision-making problems. Traditional DRL approaches like Deep Q-learning [44], policy gradient methods [45], and Actor-Critic (AC) algorithms typically utilize Multilayer Perceptrons (MLPs) as the underlying DNN architecture. However, with the emergence of various advanced DNN models, researchers are increasingly integrating DRL with other neural network architectures such as CNN [46, 47], Long-short Term Memory (LSTM) [48, 49], Transformer [50, 51], and even GNN. Authors in [52] 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 [53] 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. 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, [54] proposes an innovative approach using GCN and Deep Q-network (DQN) for multi-agent cooperative control of Connected Autonomous Vehicles (CAVs). Each CAV is treated as an agent, and 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 Software-defined Networking

(SDN)-based scenario, [55] presents a centralized agent that leverages DRL and GNN to optimize routing strategies. The authors utilized 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, the agent evaluates various routing strategies and selects the most optimal one when a traffic demand is issued. Paper [56] proposes a method that combines prediction, caching, and offloading techniques to optimize computation in 6G-enabled Internet of Vehicles (IoVs). 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 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 SDN scenarios. By leveraging the respective strengths of DRL and GNN, these approaches enable intelligent decision-making and enhance performance in intricate systems.

2.2 Traditional Caching Strategies

This chapter explores traditional caching strategies used in ICN-based environments, where the first section presents traditional reactive caching placement strategies and following that, we introduce traditional caching replacement strategies.

2.2.1 Traditional Reactive Caching Placement Strategies

ICN's default caching placement strategy is Leave Copy Everywhere (LCE) [57], which involves caching the content on all ICN nodes along the forwarding path between the

producer (the node publishing the content) and the consumer (the node sending the request). Another strategy, Randomly Copy One (RCO), [58] randomly caches content along the forwarding path, reducing caching redundancy compared to LCE. Additionally, Probabilistic Cache [59] presents a more advanced approach. It considers both the available cache capacity of each ICN node and its distance from the consumer. Closer nodes to the consumer are more likely to cache the content, and nodes with sufficient caching capacity are also given higher priority for caching. Also, Leave Copy Down (LCD) [60] focuses on caching content in the immediate neighbourhood of the producer in the direction of the consumer. Furthermore, Cache Less for More (CL4M) [61] introduces a graph-based centrality approach. This strategy prioritizes nodes with high graph-based centrality, which leverages the network topology to enhance cache hit ratios and overall content delivery efficiency.

Traditional caching placement strategies are straightforward and broadly applicable in ICN-based environments, requiring minimal computational resources. However, they may not fully exploit network, content, and user information, leading to inefficient caching decisions.

2.2.2 Traditional Caching Replacement Strategies

Traditional caching replacement strategies, including First-in-first-out (FIFO) [62], Least Recently Used (LRU) [63], and Least Frequently Used (LFU) [63], offer distinct approaches for managing content eviction within caches. FIFO operates on a first-come, first-served basis, replacing the content that has been in the cache the longest when the cache space is full. FIFO's simplicity comes at the cost of effectiveness, as it does not consider content popularity. In contrast, LRU outperforms FIFO by factoring in the recency of content access. It evicts the least recently accessed content when the cache

reaches capacity. Compared to LRU and FIFO, LFU typically has better performance. LFU considers the frequency of content access, assuming that content frequently accessed in the past will remain in high demand. Therefore, LFU prioritizes the eviction of the least frequently requested content.

While traditional caching replacement strategies offer basic mechanisms for content eviction, they may not fully exploit the potential benefits of more advanced and context-aware replacement strategies. As a result, the caching performance is not ideal.

2.3 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's 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. Here, we first explore popularity-based reactive and proactive caching placement strategies, and lastly, we delve into popularity-based caching replacement strategies.

2.3.1 Popularity-based Reactive Caching Placement Strategies

Paper [64] maintains a popularity table at each network node, which contains the content name, popularity count, and threshold. Based on this, each node selectively caches only the popular content whose popularity exceeds the threshold. This strategy ensures that caching resources are efficiently utilized for frequently accessed content. Another study [65] integrates popularity-based caching with RCO. In this approach, contents

are cached based on their popularity, and only one network node along the data packet delivery path is selected to cache the content. Also, paper [66] proposes an efficient hybrid content placement strategy. This strategy focuses on caching the most recently downloaded contents at central network nodes along the data packet delivery path, while less frequently downloaded content is cached at edge network nodes. This approach aims to optimize caching efficiency and overall content delivery performance by strategically distributing content across different network nodes. [67] introduces a compound popular content caching strategy. The authors classified content into two types: Optimal Popular Content (OPC) and Least Popular Content (LPC), based on the total number of received "Interest" packets for each content. LPC content is cached only at the network node close to the requested consumers. While OPC content is cached at all mutually connected network nodes along the data packet delivery path. This strategy combines selective caching with content popularity, enhancing the accessibility of both popular and less popular content. Similarly, [68] proposes a compound popularity caching strategy that considers content's global and local popularity. The caching placement decision takes into account the popularity of the content itself as well as the popularity of the network nodes. Also, paper [69] proposes a caching placement strategy in 5G-enabled ICN networks. The authors employed a two-step process for caching placement: they first calculated the content popularity, and then, based on the popularity, determined whether the content should be cached. If caching is deemed necessary, the content is either cached locally or pushed down towards edge nodes. Similarly, the authors in [70] proposed a "Push Down popular, Push Up less-popular" caching placement strategy. This strategy aims to push popular content to edge nodes, which are more likely to be accessed, while less popular content is pushed towards the core network. The authors also developed a one-hop cache notification mechanism to inform neighbouring nodes about their cached contents.

Lastly, the authors in [71] proposed a dynamic popularity window and distance-based efficient caching strategy. This approach employs a dynamic threshold and makes caching decisions based on two factors: (i) a dynamic size popularity window that determines content popularity; and (ii) the distance from the preceding on-path network node that caches a copy of the content. By dynamically adjusting the popularity window and considering the distance to optimize caching decisions, this strategy aims to improve caching effectiveness.

Popularity-based strategies represent an advancement over traditional methods, but they lack the power of DNN-based strategies, which can effectively capture complex relationships, environmental dynamics, and iteratively learn from them.

2.3.2 Popularity-based Proactive Caching Placement Strategies

Paper [72] proposes a proactive caching strategy that considers the popularity and chunks of large content objects. The objective is to minimize the number of forwarding nodes and content replications while adhering to cache capacity constraints. Another proactive caching strategy [73] adopts an ant colony process to make caching decisions. The authors demonstrated that their approach effectively places contents in proximity to users. In the context of ICN-IoV networks, [74] proposes a mobility-aware proactive caching algorithm. The authors leveraged a Markov model to capture the mobility patterns of vehicles and their connections with Roadside Units (RSUs). However, these strategies do not fully exploit the potential of deep learning techniques.

2.3.3 Popularity-based Caching Replacement Strategies

Paper [75] introduces a universal caching strategy that considers multiple factors for caching replacement decisions. These factors include the frequency of content fetching,

the distance from the content publisher, and the number of outgoing links at the intermediate network node. Similarly, [76] presents a dynamic fine-grained popularity-based caching replacement strategy. This scheme caches incoming content when there is available storage, prioritizing the most popular content when space is limited. To quantify content popularity, each network node maintains a popularity table containing information such as the content name, content counter, and timestamp. This information aids in dynamically updating the cache to ensure efficient content replacement decisions. However, once again, these strategies do not utilize the potential of DNN.

2.4 DNN-based Caching Strategies

This section explores various DNN-based caching strategies, which have gained significant popularity recently with the development of DNN. Firstly, we present DNN-based reactive and proactive caching placement strategies. Lastly, we discuss DNN-based caching replacement strategies.

2.4.1 DNN-based Reactive Caching Placement Strategies

Paper [77] explores a DRL-based personalized edge caching approach for IoVs, determining content caching at RSUs based on each received content. Similarly, Paper [78] develops 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 reactive caching placement decisions based on environmental observations autonomously.

2.4.2 DNN-based Proactive Caching Placement Strategies

Compared to DNN-bsed reactive caching placement strategies, more researchers are focusing on DNN-based proactive caching placement strategies. [79] introduces the DeepMEC strategy, which utilizes deep learning models such as Recurrent Neural Networks (RNNs), CNNs, and Convolutional Recurrent Neural Networks (CRNNs) to predict future content popularity. By proactively caching content with high predicted popularity scores, DeepMEC improves caching efficiency and enhances content delivery performance. Similarly, [80] proposes IntellCache, a technique that employs deep learning models including MLP, LSTM, and a combination of LSTM and CNNs. IntellCache actively caches content that is predicted to be popular in the future.

In addition to applying DNNs to predict content popularity, some researchers have explored user preference predictions using advanced deep learning technologies. [81] proposes a proactive caching strategy in the 5G-ICN scenario, applying Non-Negative Matrix Factorization (NMF) to predict user ratings for unwatched movies. Building upon this work, [82] combines the caching approach with predictions of vehicle mobility in the context of Autonomous Vehicles (AV) on highways. Furthermore, [83] introduces a cooperative caching scheme incorporating caching locations, content popularities, and predicted future content ratings in ICN-based vehicle networks. The authors also employed NMF to predict future content ratings and used this information to make informed caching decisions. Additionally, in the realm of ICN-Internet of Things (IoTs), [84] optimizes edge caching using a collaborative filtering-based strategy. The authors divided the cache space of each edge node into two halves, dedicating one half to caching contents based on local popularity and the other half to caching contents based on predicted future requests.

Furthermore, DRL-based caching strategies [85, 86] have significantly advanced in recent years, demonstrating their potential to improve caching performance. Paper [87] introduces a DQN-based caching strategy for mobile edge networks. The study [88] proposes a caching strategy that leverages DRL and a multi-level federated learning framework. The authors utilized a Double DQN (DDQN) [31] to optimize the cache hit ratio of local RSUs, neighbouring RSUs, and Cloud Data Centers in vehicular networks. Their approach incorporates federated learning to facilitate decentralized model training. Another study [89] introduces a Quality of Experience (QoE)-driven RSU caching model based on DRL, which addresses the increasing demand for time-sensitive short videos in a 5G-based IoV scenario. The authors defined the reward in their DRL model as the ratio of the number of videos interesting to each user to the total number of videos stored in the RSU, aiming to optimize the QoE for users. Furthermore, [90] designs a DQN-based strategy to optimize joint computing and edge caching in a three-layer IoV-ICN network architecture, including vehicles, edge nodes, and cloud layers. The strategy predicts the popularity of service requests from vehicles, and based on these predictions, joint computing and caching decisions are made on the edge nodes, aiming to improve overall system performance. Also, paper [91] proposes a spatial-temporal correlation approach to predict content popularity in the IoV and make proactive caching decisions. It introduces a DRL-based multi-agent caching strategy, where each RSU is an independent agent to optimize caching decisions. Furthermore, paper [92] proposes a proactive caching strategy based on AC-based DRL. To reduce the complexity of estimating the policy function due to the large dimensionality of the action space, the authors utilized a branching neural network [93] in the actor. Also, paper [94] proposes MacoCache, a fully decentralized Multi-Agent Advantage AC (MAA2C)-based proactive caching strategy. The authors used LSTM in actor and critic networks to extract temporal features of user requests.

Additionally, they considered cooperative caching by incorporating adjacent BSs' states in making caching decisions. Paper [95] combines an edge federated learning mode into a heterogeneous Multi-Agent AC (MAAC) and realizes joint caching and offloading decisions. The authors considered cache-enabled Unmanned Aerial Vehicles (UAVs), and each has an actor and critic network to make decisions about movement, execution and offloading. In addition, the cloud maintains a central actor and critic to make resource allocation decisions. Authors in [96] combined AC to solve joint caching, computing, and radio resource allocation problems in fog-enabled IoT to minimize the network's average end-to-end delay. Their actor and critic networks are deployed in the cloud. Paper [97] applies a multi-agent Double Deep Recurrent Q Network (DDRQN) caching strategy. The authors employed Gated Recurrent Unit (GRU) and fully-connected layers as the DQN [44]. It is worth noting that they demonstrated that the decentralized DDRQN outperforms a centralized DDRQN and a neural network-based decentralized DQN [98].

2.4.3 DNN-based Caching Replacement Strategies

In the context of SDN-based ICN, [99] introduces a deep learning-based content popularity prediction approach. The authors utilized Stacked Autoencoders (SAE) to extract features from network nodes and predict content popularity, which guides caching decisions. Also, paper [100] employs an LSTM-Encoder Decoder (LSTM-ED) model to predict content popularity and make caching decisions accordingly. Additionally, the authors combined the predicted content popularity with traditional caching replacement strategies such as LRU to maximize cache hit ratios. Moreover, [101] introduces the Stimulable Neural Network (SNN) model, which analyzes the inter-relationships among sequenced requests and considers factors like content size and data retrieval costs to make caching decisions. Furthermore, [102] proposes a caching replacement strategy

that utilizes a 1D-CNN. This model captures seasonal patterns in user request streams and predicts future hits for each content. When cache space is limited, only contents with high predicted future hits are cached, optimizing cache utilization and improving overall caching performance. Also, paper [103] develops a MAAC caching replacement strategy. Notably, the actor-network selects a caching scheme rather than makes a caching replacement decision. In this way, the action space is discrete and shrinks a lot. Similarly, in the paper [104], the actor-network also selects a caching replacement policy rather than making a replacement decision. Furthermore, the authors in [89] proposed an AC-based caching replacement strategy in the IoVs. Both actor and critic networks operate in the cloud. In each time step, each RSU sends its state to the cloud, and the cloud makes caching decisions for it. Paper [105] proposes a centralized and decentralized multi-agent discrete Soft AC (SAC)-based [106] caching replacement strategy. SAC was initially designed for continuous action space, while the authors developed a SAC variant to handle discrete action spaces. Paper [107] develops centralized and decentralized AC-based DRL caching replacement strategies. In the decentralized case, each Base Station (BS) maintains an actor-network, and the cloud maintains a centralized critic network to evaluate actions taken by each actor. Paper [108] proposes a multi-agent attention-based critic [109] caching strategy. The attention mechanism helps each BS realize adaptive cooperation with neighbouring BSs. Paper [110] applies Graph Attention Networks (GAT) [111] and a Soft MAAC (SMAAC) [106] in caching problems in Device-to-Device (D2D) communications. The approach is fully decentralized and can realize joint inter-agent caching coordination. Paper [112] proposes a UAV-assisted MAAC replacement strategy in a wireless network. The authors considered a three-layer networking architecture, including a Ground BS (GBS) layer, a UAV-based aerial BS (UAV-BS) layer and the cloud server layer. GBSs and UAV-BSs have caching capabilities, each of which maintains

an actor-network, and the cloud maintains a centralized critic network. The authors used the Random Waypoint model [113] to simulate user movement, and assumed each UAV-BS can reallocate itself to satisfy users' requests.

Chapter 3

GNN-based Caching Replacement Approach in NDN

3.1 Introduction

This chapter focuses on improving caching performance (i.e., maximizing the cache hit ratio) in NDN [57]. A higher cache hit ratio indicates that NDN nodes can cache contents that users are interested in. In this case, more user requests are satisfied by NDN nodes' cache stores rather than the content producer (i.e., the server). It improves network performance regarding caching space utilization and data delivery latency.

There has been a large amount of research on caching strategies, and one of the most promising is the work by the authors in [102], utilizing a 1D-Convolutional Neural Network (1D-CNN) for Information-Centric Networking (ICN) caching and achieving a high cache hit ratio. They applied 1D-CNN to predict the future hits of each content chunk and used the future hits as the cache replacement algorithm's priority, meaning content chunks with lower future hits were replaced by content chunks with higher future hits.

The method we propose in this chapter offers further improvements to [102]. As part of our approach, we also utilize a deep learning-based strategy. In particular, we predict the cache probability of content rather than the future hits of each content at each NDN node. Since user preferences change over time, we first employ 1D-CNN [114] to capture the dynamics of each user preference in the temporal dimension. It gives us an idea of how the content popularity at each NDN node changes over time. Moreover, due to the nature of packet forwarding, content requests between neighbouring nodes usually affect each other. Therefore, we apply a Graph Neural Network (GNN) to capture the spatial dependencies between NDN nodes. It allows us to understand how each NDN node's requests affect its neighbouring nodes. Specifically, we apply GNN to node-level classification problems. For this type of problem, GNN usually requires two inputs, where the first is a feature matrix representing node information, and the second is an adjacency matrix representing edge information.

This chapter also applies the Software Defined Networking (SDN) concept, where an SDN controller comprehensively views the entire network and thus, it can collect the status of all the network nodes and make caching decisions accordingly. In our implementation, an NDN controller knows the traffic of each NDN node and controls them. Our GNN-based model works in the controller. Periodically, the NDN controller sends the collected traffic information and the network topology to the GNN-based model, and then the GNN-based model predicts content caching probability for each node. After that, the NDN controller informs each NDN node about the prediction, and each performs cache replacement based on it.

The contributions of this chapter are as follows:

- We propose a GNN-based cache replacement policy in NDN. To the best of our knowledge, we are the first to apply GNN to the caching problem. In particular, our

GNN-based model contains 1D-CNN layers and GraphSAGE layers [26], which are utilized to extract the input features' temporal dependence and spatial dependence, respectively. It is worth mentioning that the GNN-based model makes content caching probability predictions for all NDN nodes with a single forward pass and requires constant inference time.

- To train a GNN-based probabilistic prediction model, we propose a heuristic search algorithm that labels the collected traffic data with a binary ground truth, where 0 means no cache and 1 means cache.

- We deploy our GNN-based, state-of-the-art deep learning-based strategies, including 1D-CNN [102], Long Short-Term Memory Encoder-Decoder (LSTM-ED) [100] and Staked Auto Encoder (SAE) [100], and classical benchmark strategies, including Least Frequently Used (LFU), Least Recently Used (LRU) and First-in-first-out (FIFO) on Mini-NDN [27]. Extensive studies show that our GNN-based caching strategy achieves a much higher cache hit ratio and lower latency than other approaches.

The organization of this chapter is as follows: The proposed GNN-based caching replacement strategy is first introduced. Following it, a heuristic algorithm used to give labels to our collected data is presented. Next are the experimental results and finally, we conclude the chapter.

3.2 GNN-based Caching Replacement Method

In GNN, a graph is represented as $G = (V, E)$ where V is the set of nodes and E is the set of edges. In this section, each NDN node is denoted as $v_i \in V$. An edge $e_{i,j} \in E$ denotes a directed connection from node v_i to node v_j. Assume there are K distinct contents

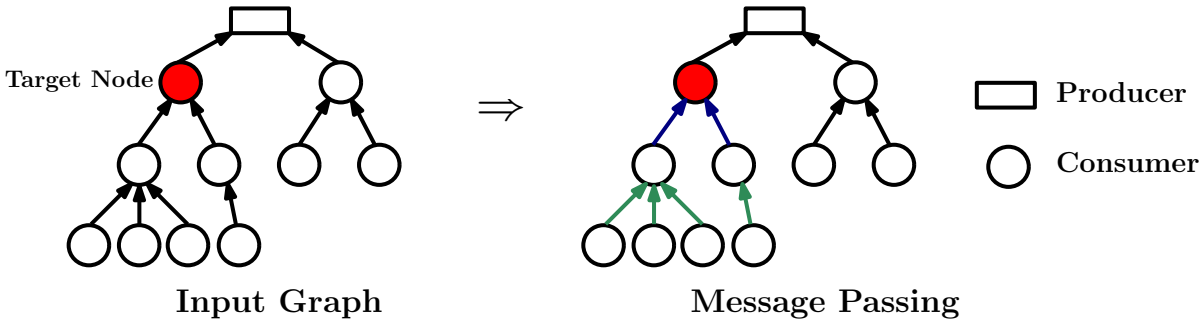


Figure 3.1: GNN input graph and message passing in a tree network topology

across the network, denoted by $C = \{c_1, c_2, ..., c_K\}$. For each content c_k, there is a graph, and each node v_i has an n-dimensional feature vector $x_{i,k}$ and a ground truth label $y_{i,k}$, where $y_{i,k} = 1$ indicates that the node v_i should cache the content c_k and $y_{i,k} = 0$ means the node should not cache the content.

As suggested in paper [102], we collect the number of content requests in the previous n timesteps as node features, where $n = 8$, and the time interval between two adjacent timesteps is ten minutes. For each content c_k, we feed the node feature matrix and the network adjacency matrix to GNN to predict the caching probability of that content for each node at the next time slot.

In this chapter, we focus on two types of network topologies. The first type is a tree network topology, and the second is an arbitrary network topology.

3.2.1 Tree Network

In a tree network, each node has a parent node except the root node. We assume the root node is the producer and is responsible for publishing the content. All other nodes are consumers and generate *Interest* packets tagged with the desired content name. Each

consumer has a cache store which can cache contents to fulfill future *Interest* packets. If the requested content is not cached at a node, then the node needs to forward the *Interest* packet to other nodes until the packet can be satisfied. In a tree structure, the child node always forwards an *Interest* packet to its parent node. Therefore, we consider that the content requests of the child nodes affect the parent nodes, but the content requests of the parent nodes do not impact the child nodes. For this case, we convert the tree network structure into a directed graph, where $e_{i,j} \in E$ if and only if v_j is the parent of v_i and $e_{j,i} \notin E$.

3.2.2 Arbitrary Network

In arbitrary network topology, one node is randomly selected as the producer according to a uniform distribution, and all other nodes are consumers. Unlike the tree structure, there are multiple packet forwarding paths from the consumer to the producer. The content requests from each node may affect some or all of its neighbouring nodes. Therefore, we convert the arbitrary network topology into an undirected graph (i.e., $e_{i,j} \in E$ and $e_{j,i} \in E$ if there is an connection between v_i and v_j).

3.2.3 GNN-based Content Caching Probability Prediction

The proposed GNN-based model contains 1D-CNN layers and GraphSAGE [26] layers. 1D-CNN captures the temporal dependence of the content prevalence dynamics at each node. For each content, the input to the 1D-CNN is a feature vector of each node. The feature vector contains the number of requests for that content from that node in the previous eight time intervals. 1D-CNN's output is the number of requests for that content from that node in the next time interval. The 1D-CNN architecture consists of two convolutional layers with kernel sizes 2 and 6, respectively. A Rectified Linear Unit

(ReLU) layer is applied after each convolutional layer. Following the second convolutional layer is a fully connected layer with the activation function ReLU. The goal of 1D-CNN is to predict the number of requests per node for each content in the next time slot.

After the 1D-CNN architecture, we use GNN to capture the node features' spatial dependence to predict each node's caching probability for each content. A GNN captures spatial dependence by passing messages between nodes [24]. In a tree network topology, the parent node gathers node features from its children since we consider the tree network as a directed graph with the children pointing to the parent node. Figure 3.1 shows the input graph to our GNN model and the message passing process in a tree network. The figure shows that the input graph is directed, and the red node is the target node. It also shows two message passing layers, meaning the target node can aggregate information from 2-hop child nodes. The blue arrows indicate that the target node aggregates the features of its 1-hop child nodes in the first message passing layer. Message passing denoted as green arrows happen simultaneously at the target node's child nodes. Once the aggregation is done, each node's feature vector will be updated. In the second message passing layer, the target node can aggregate its 2-hop child nodes' information because its 1-hop children have already aggregated its child nodes' features in the first message passing layer. After the final message passing layer, node features are updated and are used to predict the nodes' labels.

Unlike the tree structure, each pair of 1-hop neighbouring nodes carries a bidirectional message passing in the random graph because we consider the random graph as an undirected graph. Besides the message passing directions, the random graph's neighbouring information aggregation process is the same as the tree structure. Regardless of the network topology, the purpose of GNN is to capture the network topology and aggregate the features of neighbouring nodes to each central node for cooperative caching.

The GraphSAGE framework [26] generates node embeddings of previously unknown graphs using inductive methods. Our study separates training and testing graphs, and testing graphs are unseen during training. As a result, our GNN architecture is developed using GraphSAGE layers. GraphSAGE generates node embeddings by sampling and aggregating neighbourhood node representations:

$$h^k_{N(v)} = AGG_k(\{h^{k-1}_u, \forall u \in N(v)\}), \tag{3.1}$$

where $N(v)$ represents node v's neighbours and h^{k-1}_u denotes node u's representation at the $(k-1)^{th}$ step. AGG_k is an aggregation function at the current k^{th} step. The aggregator aggregates node v's 1-hop neighbouring nodes' representations and generates a hidden vector $h^k_{N(v)}$. According to the GraphSAGE paper, there are three types of aggregators: mean aggregator, pooling aggregator, and LSTM aggregator. A pooling aggregator has been shown to be more efficient than an LSTM aggregator in that paper. In addition, the pooling aggregator generally performs better than the mean aggregator. For both performance and efficiency reasons, we aggregate the features of neighbouring nodes via the pooling aggregator:

$$AGG^{pool}_k = max(\{\sigma(W_{pool}h^k_{u_i} + b), \forall u_i \in N(v)\}), \tag{3.2}$$

where $\sigma(W_{pool}h^k_{u_i} + b)$ performs a nonlinear transformation on the neighbourhood representation $h^k_{u_i}$ of the central node v, and then a max pooling operator is performed on the transformed neighbourhood representation vector.

After aggregating neighbour nodes' representations, the central node representation and neighbour node representations are concatenated, and then fed into a fully connected

layer with a nonlinear activation function, as shown below:

$$h_v^k = \sigma\left[W^k \cdot \text{CONCAT}\left(h_v^{k-1}, h_{N(v)}^k\right)\right], \tag{3.3}$$

where h_v^{k-1} is the node v's representation vector at the $(k-1)^{th}$ step and $h_{N(v)}^k$ is node v's neighbourhood representation vector at k^{th} step. The two vectors are concatenated and fed into a fully connected layer with a nonlinear activation function σ. We perform a ReLU activation function, and h_v^k is the updated representation of node v at k^{th} step.

3.2.4 Caching Replacement Decision

The GNN-based model works in the network controller, and after the model predicts the probability of caching each content at each NDN node, the prediction is sent to each node. In NDN, each node caches data packets in its cache store, and we name the cached data packets in the cache store as cache store entries. In the implementation, we extend a cache probability field for each cache store entry, and each NDN node maintains a local hash map with the content name as the key and the content's cache probability as the value. When an NDN node receives a data packet, it will insert the data packet into its cache store when there is available cache space. Otherwise, the data packet will only be inserted if its caching probability is higher than the lowest caching probability of the cache store entries, evicting the cache store entry with the least caching probability. In addition, the cache probability of each cache store entry is updated periodically to avoid content with a high predicted cache probability in the past remaining in the cache store forever.

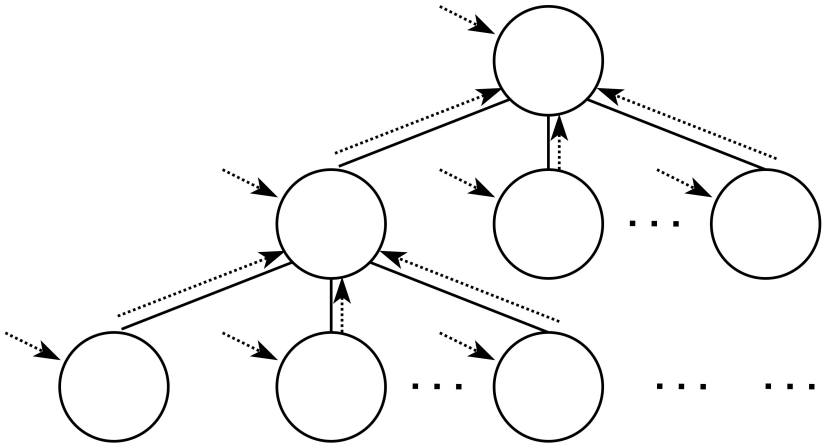


Figure 3.2: The visualization of the algorithm

3.3 The Derivation of Ground Truth

An iterative algorithm is applied to approximate the ground truth of content caching. Assume we have a graph $G = (V, E)$, a set of contents C, and a set of requests R. There is a tuple for each request $r_z \in R$: $r_z = (c_k, v_i, p_h)$, with c_k being the requested content, v_i being the node receiving the request, and p_h being the popularity of the request. If a node fails to cache the content requested, we call that a request missed. A node v_i that misses a request $r_1 = (c_k, v_i, p_h)$ will pass on a new request $r_2 = (c_k, v_j, p_h)$ to its parent v_j. A node that receives two requests for the same content considers them one request combined with their popularity. The goal of the algorithm is to minimize the accumulated popularity of all missing requests (i.e. if both the child and the parent fail to cache the content c_k with popularity p_h, the accumulated popularity will be $2p_h$).

It is worth mentioning that we consider the producer as the root node in the random network and apply the Djikstra algorithm to find the shortest path tree from all other nodes to the producer node. In this case, both tree and random network structures have a root node (i.e., the producer node) and child-parent node relationships.

In general, the graph G can be arbitrarily large, making it difficult for local nodes to know requests that are far away. Choosing cached content that is beneficial for reducing global accumulated popularity is thus a difficult decision for local nodes. As a result, the algorithm seeks a locally optimal solution.

Our algorithm is a greedy algorithm, and its visualization is shown in Figure 3.2. At the beginning of each iteration, our algorithm checks whether the *current_node* is a *leaf*. 1) If this is the case, the *current_node* will attempt to cache all of the requested contents received from R. The node prefers to cache more popular content. If the number of requests for a particular piece of content exceeds the popularity of that node's cached content, the node will choose to skip caching the less popular content. The missed requests will then be forwarded to the parent, and the program will terminate. 2) If the *current_node* is not a *leaf*, the program will iteratively call itself on all children of the *current_node*. Following the completion of those iterative calls, the *current_node* will merge all requests received from R with all requests passed by its direct children. Afterwards, it will try to cache all requested contents, prioritizing popular contents. Finally, it will forward any missed requests to its parent, after which the program will terminate.

The same content might be requested on different nodes by two requests $r_1, r_2 \in R$ (i.e. $r_1 = (c_k, v_1, p_1)$, $r_2 = (c_k, v_2, p_2)$). As soon as r_2 meets r_1, the node v_1 will consider both to be the same request: $r_3 = (c_k, v_1, p_1 + p_2)$. It will result in higher popularity of c_k and an increased likelihood of it being cached. As a result, our greedy algorithm may not provide the best solution. However, when each $r_z \in R$ requests different pieces of content, our algorithm successfully minimizes the total popularity of all missing requests. The strong induction can demonstrate this. In the future, we can improve our algorithm by integrating some heuristic search strategies.

3.4 Experimental Results

In this section, we use Mini-NDN [27] to perform all experiments. Mini-NDN is an emulation tool that runs real instances of NDN packages. We deploy our proposed GNN-based model, 1D-CNN model mentioned in paper [102], LSTM-ED model mentioned in paper [100], and SAE model mentioned in paper [99] on Mini-NDN. We compare the performances of our GNN-based caching strategy with other deep learning-based caching strategies, 1D-CNN, LSTM-ED, SAE, and three classical caching strategies, LFU, LRU and FIFO.

3.4.1 Network Topology

We use tree and arbitrary network topologies for evaluations. In both topologies, there is a producer, and all other nodes are consumers. We put the producer at the root node in the tree network topology. In arbitrary network topology, the producer is chosen randomly by following a uniform distribution. All nodes in the network, except the producer, have uniform caching capability with a cache size of 1 to 6 content chunks.

3.4.2 Traffic Generation

We employ NDN Traffic Generator [115] to generate *Interest* and *Data* packets. By default, each consumer requests 2 unique contents with the content popularity following a Zipf-like distribution [116] with parameter $\alpha = 0.8$. We also evaluate other experimentation parameters where each consumer requests the number of unique contents from the set {2, 3, 4, 5, 6} and the α value from the set {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}. By default, all contents are 2049 bytes each. We also conduct experiments with randomly uniform content sizes ranging from 1049 to 8049 bytes. We assume consumers

have different request rates, with six or sixty requests per minute, respectively. Table 3.1 shows key parameters used to generate traffic.

Table 3.1: Key Experimentation Parameters

Parameters	Default Value	Values
Network topology	Tree topology	Tree or Arbitrary topology
Number of nodes	55	3-95
Number of producers	1	
Number of consumers	54	2-94
Number of unique contents each consumer requested	2	2 or 10
Number of distinct contents	200	200 or 600
Content size	2049 bytes	1049-8049 bytes
Node cache capacity	1 chunk	1-6 chunks
Number of requests per consumer	600 or 6000	
Consumer requests rate	6 req/min or 60 req/min	
Content popularity Zipf distribution	$\alpha = 0.8$	0.1-1.0

3.4.3 Dataset Collection

We run each experiment for 100 minutes without applying any caching algorithm and collect the number of content requests of each node. Since the number of requested contents per node varies greatly, we transform the dataset to have a mean of 0 and a standard deviation of 1. We used 80% of the dataset as the training dataset and 20% as the testing dataset. All the deep learning-based models are trained and tested using the same dataset.

3.4.4 Evaluation Metrics

The following three metrics are adopted to evaluate various caching strategies' performances:

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

$$CHR = \frac{cacheHits}{cacheHits + cacheMiss}, \tag{3.4}$$

  where $cacheHits$ refers to the count of $Interest$ packets that are successfully satisfied by retrieving the corresponding $Data$ packet from the 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 user, while a $Data$ packet contains the requested content itself and can serve as a response to the corresponding $Interest$ packet.

- BHR (Byte Hit Ratio): It defines the number of bytes satisfied by the cached data packets divided by the total number of bytes that consumers requested.

$$BHR = \frac{cacheHitsBytes}{cacheHitsBytes + cacheMissesBytes}, \tag{3.5}$$

  where $cacheHitsBytes$ is the total object sizes for cache hits and $cacheMissesBytes$ is the total object sizes for cache misses.

- Average Latency (ALT): The average latency represents the average delay between the moment a user sends an $Interest$ packet and the moment it receives the cor-

responding *Data* packet,

$$ALT = \frac{\sum_{i=1}^{I}(i_t + i_r)}{I}, \tag{3.6}$$

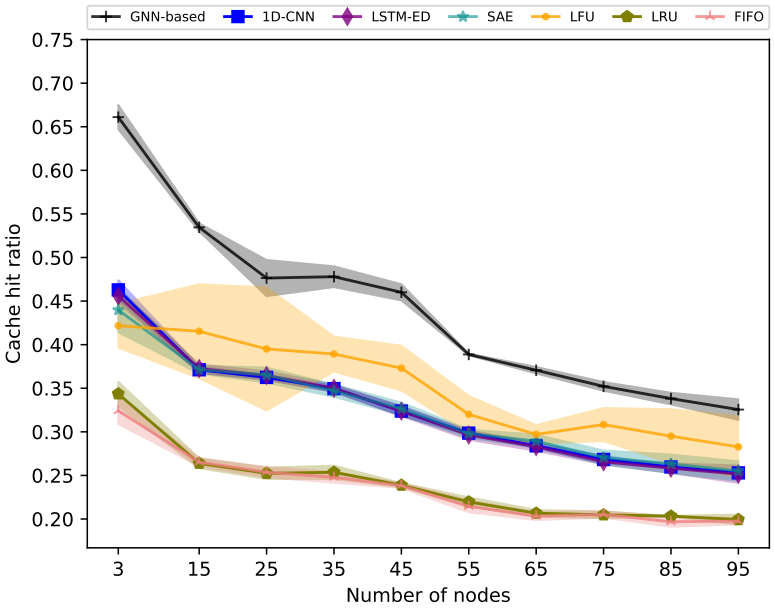
where I denotes the total number of user requests. i_t represents the travel time of the *Interest* packet from the user to the node that fulfills the request, while i_r denotes the travel time of the responding *Data* packet.

3.4.5 Results

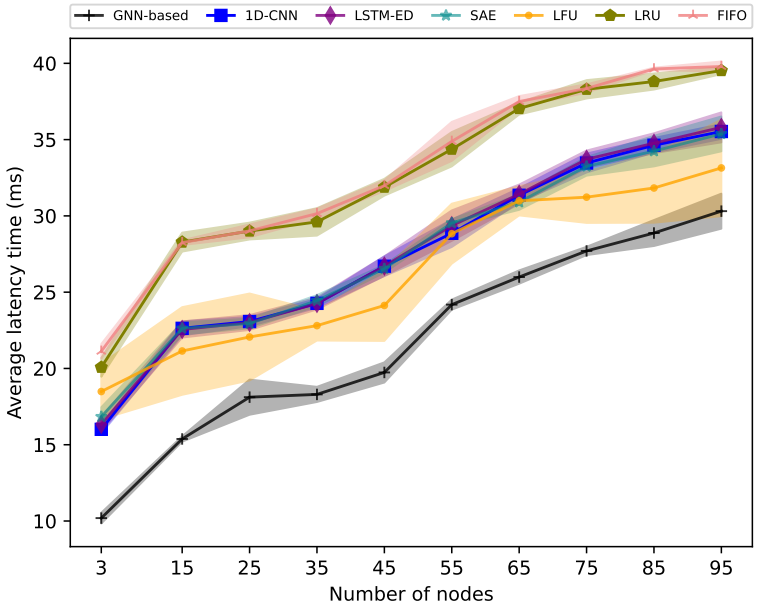
This section shows the experimental results of our GNN-based caching algorithm, 1D-CNN caching algorithm, LSTM-ED caching algorithm, SAE caching algorithm, LFU, LRU and FIFO. By default, the cache placement strategy is LCE. For deep learning-based cache replacement strategies, the models are trained and tested using the same dataset, and the best model is selected with early stopping on the testing dataset. We minimize the binary cross entropy [117] for the GNN model and the mean square error [118] for the other models. All models are optimized using the Adam optimizer [117]. After training and testing, we deploy each selected best model to the Mini-NDN to make real-time content caching decisions. All deep learning models use the node features of the previous eight time slots to predict the content caching probability for the next time slot. By default, a time slot is 10 minutes. Therefore, CHR, BHR, and ALT are computed 80 minutes after the start of the experiment. In order to control variables, the performances of classical caching algorithms, LFU, LRU and FIFO, are also measured in the same way. Each network scenario is performed ten times, and the results are averaged.

3.4.5.1 Effect of Network Size

In this section, we explore the effect of network size on different caching algorithms, GNN-based, 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO. Each network scenario contains



(a) 95% CI for the CHR of all caching strategies



(b) 95% CI for the ALT of all caching strategies

Figure 3.3: Performance of GNN-based, 1D-CNN, LSTM-ED, SAE, LFU LRU and FIFO caching strategies in tree network topologies with different numbers of nodes. Note that each network scenario contains 200 different contents, and each consumer requests 2 different contents and has a cache size of 1 content chunk.

200 distinct contents, and each consumer requests 2 different contents and has a cache size of 1 content chunk. Each consumer sends requests following a Zipf distribution with an α of 0.8. All the networks are tree topologies, and the number of network nodes is from the set {3, 15, 25, 35, 45, 55, 65, 75, 85, 95}. The depth of all topologies is 4, except for the topologies with 3 and 15 nodes, which have depths of 2 and 3, respectively. Regarding the GNN-based caching strategy, we utilize 2 GraphSAGE layers for all topologies except for the topology with 3 nodes that applies 1 GraphSAGE layer. The number of the node's representation dimension is {128}, {128, 64}, corresponding with the number of GraphSAGE layers 1 and 2, respectively.

Figure 3.3 shows the caching performance of all strategies. We can observe that no matter the number of nodes, the GNN-based caching strategy performs the best with a rather narrow Confidence Interval (CI). Moreover, the performance of 1D-CNN, LSTM-ED and SAE is pretty similar because each node can only cache the most popular content requested by itself. Unlike them, the GNN-based caching strategy uses aggregated neighbourhood information to obtain neighbouring nodes' status and thus helps achieve cooperative caching. In addition, the classical strategy LFU performs a little bit better than the deep learning-based caching strategies but with a rather wide CI. The other two classical strategies, LRU and FIFO, perform the worst compared with others.

Figure 3.3a shows the 95% CI for the CHR of all strategies. In the best case, our GNN-based caching strategy performs a 44%, 45%, 50%, 56%, 102%, 104% higher CHR than 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO, respectively. Furthermore, regardless of the number of nodes, the CHR of the GNN-based caching strategy performs at least 27% better than the 1D-CNN, LSTM-ED and SAE, 14% better than LFU and 60% better than the LRU and FIFO. It demonstrates that our GNN-based caching algorithm can accurately predict content popularity and increase caching diversity across the network.

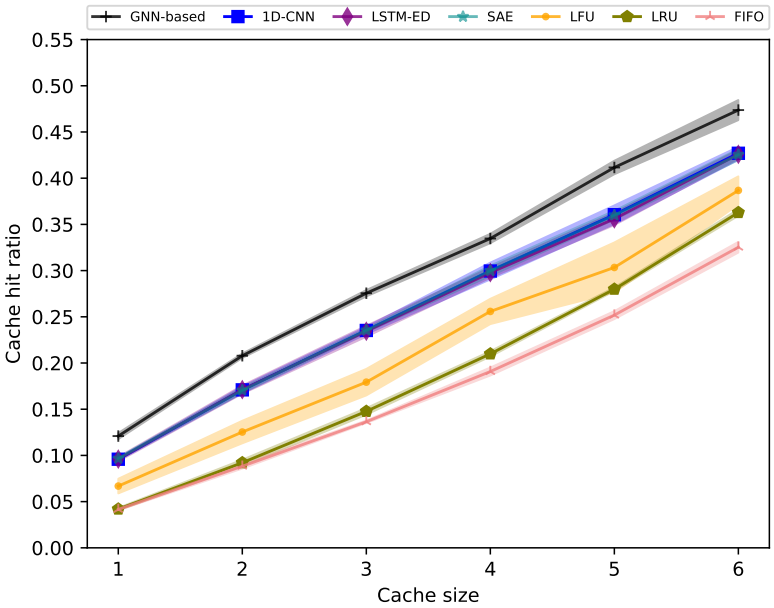
Figure 3.3b shows the 95% CI for the ALT of all strategies. In the best case, our GNN-based caching algorithm achieves 36%, 37%, 39%, 45%, 49% and 51% lower ALT than 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO, respectively. On average, our GNN-based caching strategy can achieve around 22% lower ALT than 1D-CNN, LSTM, and SAE. In addition, LRU and FIFO have the highest ALT among all the caching strategies. It shows that the GNN-based caching strategy can decrease consumer access latency by a large margin by caching more popular content near the consumer.

3.4.5.2 Effect of Cache Size

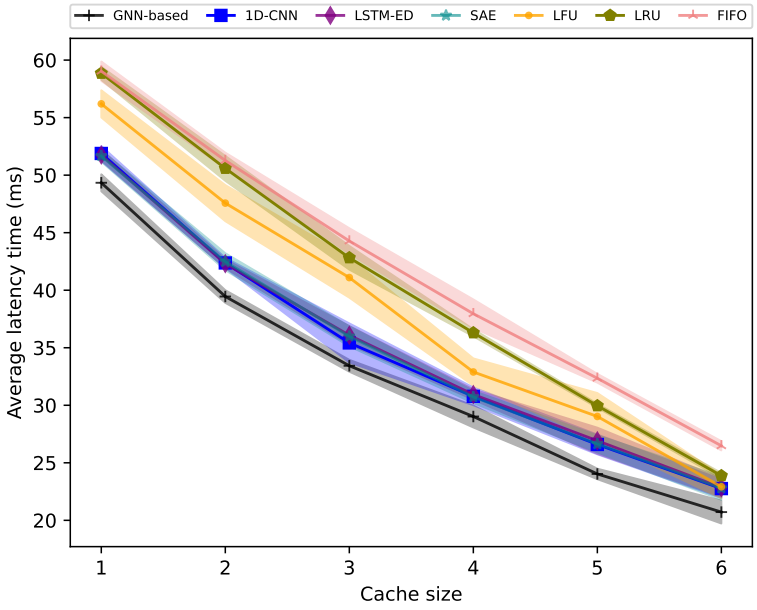
This section explores the impact of node's cache sizes on various caching strategies. Cache size is defined as the number of content chunks each node can cache. By default, each NDN node except the content producer has the caching capability with uniform cache size. In this section, the producer advertises 600 different contents, and each consumer requests 10 distinct contents with a cache size ranging from 1 content chunk to 6 content chunks. Figure 3.4 plots the 95% CI for the CHR and ALT of GNN-based, 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO under different network scenarios in a 55-node tree network topology.

Figure 3.4a shows that the CHR increases with increasing cache sizes for all strategies. In particular, the GNN-based caching strategy outperforms 1D-CNN, LSTM and SAE by around 17% on average. The GNN-based caching strategy generally outperforms LFU with a more significant margin. Besides, GNN-based has an average advantage of more than 89% and 102% over LRU and FIFO, respectively.

Figure 3.4b shows that the ALT decreases with increasing cache sizes for all strategies. Overall, GNN has about 7% lower ALT than 1D-CNN. LSTM and SAE have almost the same performance as 1D-CNN, and both can achieve lower ALT than LFU, LRU and



(a) 95% CI for the CHR of all caching strategies



(b) 95% CI for the ALT of all caching strategies

Figure 3.4: Performance of GNN-based, 1D-CNN, LSTM-ED, SAE, LFU LRU and FIFO caching strategies varies with node cache sizes in a 55-node tree network topology. Note that each network scenario contains 600 different contents, and each consumer requests 10 different contents.

FIFO. On average, GNN can achieve a latency that is 14%, 18%, and 22% lower than LFU, LRU and FIFO, respectively.

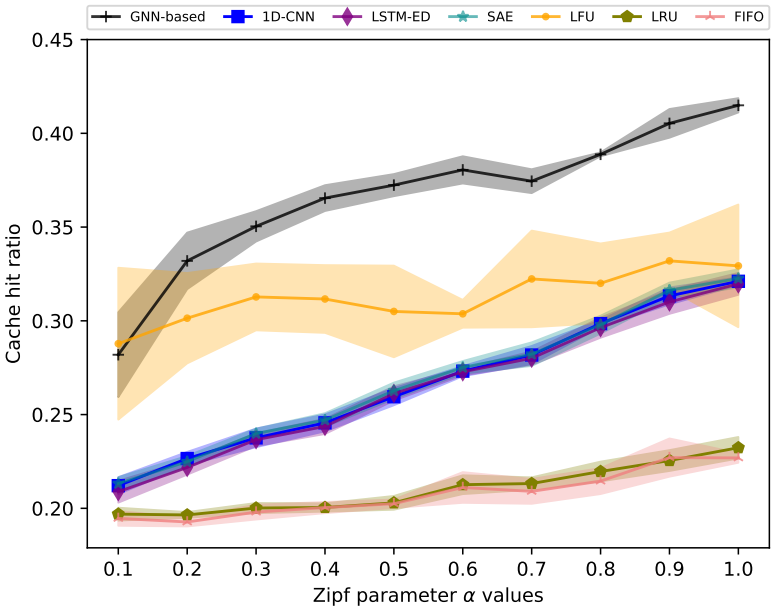
3.4.5.3 Effect of Zipf Parameter α Values

This section explores the CHR and ALT varies with the Zipf parameter α with various values. Zipf parameter α defines the content popularity distribution each consumer requested. Figure 3.5 shows the caching performance with different α values ranging from 0.1 to 1.0 for GNN-based, 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO in a 55 nodes tree network topology.

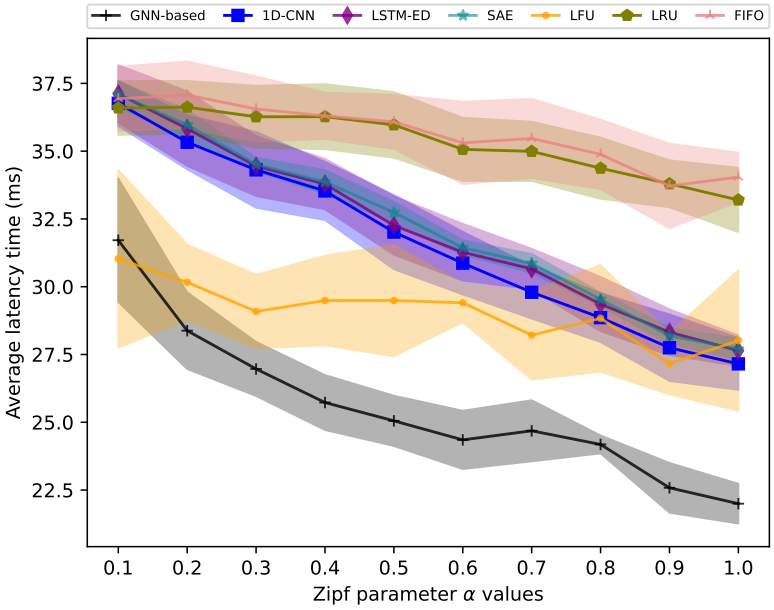
As shown in Figure 3.5a, for all deep learning-based caching algorithms, CHR increases as the α value increases due to a more noticeable difference in content popularity. When $\alpha = 0.1$, 1D-CNN, LSTM-ED, SAE, LFU, LRU, and FIFO perform similarly, but the GNN-based caching strategy can achieve much better performance. The GNN-based caching algorithm has an averagely of around 40% higher CHR than 1D-CNN, LSTM-ED and SAE, 17% higher CHR than LFU and 75% higher CHR than LRU and FIFO.

Figure 3.5b shows that our GNN-based caching algorithm achieves the lowest ALT among all caching schemes. On average, the GNN-based caching algorithm has around 20% lower ALT than 1D-CNN, LSTM-ED and SAE, 12% lower ALT than LFU, and 27% lower ALT than LRU and FIFO.

In summary, 1D-CNN, LSTM-ED, and SAE perform similarly across Zipf parameters as they only predict content popularity based on individual node requests. In contrast, GNN captures content popularity considering both the node and its neighbors' requests.



(a) 95% CI for the CHR of all caching strategies



(b) 95% CI for the ALT of all caching strategies

Figure 3.5: Performance of GNN-based, 1D-CNN, LSTM-ED, SAE, LFU LRU and FIFO caching strategies varies with Zipf alpha values in a 55-node tree network topology. Note that each network scenario contains 200 different contents, and each consumer requests 2 different contents and has a cache size of 1 content chunk.

Table 3.2: 95% CI for the CHR and ALT of the GNN-based, 1D-CNN, LSTM-ED, LFU, LRU and FIFO caching strategies in a 55-node arbitrary network topology and a 55-node tree network topology.

	Arbitrary topology		**Tree topology**	
Caching strategies	95% CI for CHR (%)	95% CI for ALT (ms)	95% CI for CHR (%)	95% CI for ALT (ms)
GNN-based	**33.59 (32.77 to 34.41)**	**26.64 (26.08 to 27.20)**	**38.87 (38.77 to 38.98)**	**24.18 (23.85 to 24.51)**
1D-CNN	26.82 (18.36 to 35.28)	31.14 (25.57 to 36.72)	29.85 (29.66 to 30.03)	28.85 (27.96 to 29.75)
LSTM-ED	22.03 (21.24 to 22.82)	34.58 (33.92 to 35.24)	29.63 (29.09 to 30.16)	29.35 (28.36 to 30.35)
SAE	22.15 (21.74 to 22.56)	34.37 (33.80 to 34.95)	29.79 (29.31 to 30.27)	29.53 (29.27 to 29.78)
LFU	20.89 (17.72 to 24.06)	37.57 (34.24 to 40.91)	32.00 (29.87 to 34.12)	28.83 (26.86 to 30.80)
LRU	14.85 (14.29 to 15.41)	47.45 (46.83 to 48.07)	21.96 (21.43 to 22.48)	34.37 (33.24 to 35.49)
FIFO	14.81 (14.45 to 15.16)	48.04 (47.72 to 48.36)	21.45 (20.76 to 22.14)	34.88 (33.60 to 36.16)

3.4.5.4 Effect of Network Topology

We also compare the caching performance of all strategies in arbitrary network topologies. Except for the network topology, the key network parameters in this section follow the default values provided in Table 3.1. Table 3.2 shows the performance of the GNN-based, 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO in a 55-node arbitrary network topology and a 55-node tree network topology. Regarding the GNN-based model, we utilize 7 and 2 message passing layers for the arbitrary and tree networks, respectively. The hidden node representation dimension is {128, 64, 64, 64, 64, 64, 32} and {128, 64}, respectively.

Regardless of the network topology, our GNN-based caching strategy performs the best among all strategies. The GNN-based caching algorithm outperforms 1D-CNN by

about 20% in CHR and 15% in terms of ALT in the arbitrary network topology. The GNN-based strategy can also achieve around 28% higher CHR and 24% lower ALT than the LSTM-ED and SAE. In addition, the GNN-based caching algorithm has a more remarkable improvement in CHR and ALT than classical caching algorithms, LFU, LRU and FIFO.

The results also show that all caching strategies' performances decrease in the arbitrary network compared with the tree network. The reason is that the tree network topology has a depth of 4. It means that if an *Interest* packet cannot be satisfied by any node's cache store along the forwarding path, it will be forwarded at most 4 hops before it reaches the producer, which results in up to 4 cache misses. However, in the arbitrary network topology, the node farthest from the producer has a distance of 9 hops. If network nodes' cache store cannot satisfy an *Interest* packet, there will be more cache misses.

In summary, benefiting from aggregating neighbour nodes' information, the GNN-based caching algorithm can realize cooperative caching and thus achieve a higher CHR and lower ALT than 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO caching algorithms in both arbitrary and tree network topologies.

3.4.5.5 Effect of Content Size

Instead of using a uniform content size, we also evaluate different caching policies on various content sizes. The content sizes are randomly sampled using a uniform distribution of 1049 to 8049 bytes. Other network simulation parameters are also from default values of Table 3.1. This section introduces one more metric, BHR, to show how much bandwidth the cache has saved. Table 3.3 shows the 95% CI for the CHR, BHR and ALT of the GNN-based, 1D-CNN, LSTM-ED, SAE, LFU, LRU and FIFO caching

Table 3.3: 95% CI for the CHR, BHR and ALT of the GNN-based, 1D-CNN, LSTM-ED, LFU, LRU and FIFO caching strategies in a 55-node tree network topology.

Caching strategies	95% CI for CHR (%)	95% CI for BHR (%)	95% CI for ALT (ms)
GNN-based	**40.50 (39.63 to 41.38)**	**40.61 (39.46 to 41.75)**	**23.03 (22.39 to 23.67)**
1D-CNN	30.41 (29.69 to 31.13)	29.97 (29.18 to 30.77)	28.91 (28.32 to 29.51)
LSTM-ED	30.17 (29.46 to 30.89)	29.65 (28.90 to 30.40)	29.10 (28.52 to 29.68)
SAE	30.31 (29.50 to 31.13)	29.82 (29.00 to 30.64)	28.96 (28.26 to 29.65)
LFU	34.80 (33.64 to 35.97)	34.35 (30.73 to 37.98)	26.52 (25.60 to 27.45)
LRU	21.55 (21.08 to 22.02)	21.31 (20.91 to 21.71)	35.74 (35.33 to 36.15)
FIFO	21.68 (21.52 to 21.84)	21.44 (21.28 to 21.61)	35.70 (35.59 to 35.82)

strategies in a 55 nodes tree network topology. We can observe that the GNN-based strategy can save much more bandwidth than the other caching algorithms. Compared to 1D-CNN, LSTM-ED and SAE, the GNN-based strategy obtains about 35% higher BHR. The GNN-based strategy can also achieve 18% higher BHR than the LFU. In addition, the GNN-based strategy provides a more significant gain in BHR than LRU and FIFO. Benefiting from the higher BRH, the CHR improvement and ALT reduction of the GNN-based caching algorithm are significant.

Table 3.4: 95% CI for the CHR and ALT of the GNN-based caching strategy with different information aggregator types in a 55-node tree network topology.

Aggregator types	95% CI for CHR (%)	95% CI for ALT (ms)
Pooling	38.87 (38.77 to 38.98)	24.18 (23.85 to 24.51)
Mean	40.30 (39.83 to 40.78)	23.54 (23.29 to 23.78)
LSTM	35.76 (34.74 to 36.78)	26.21 (25.34 to 27.08)

3.4.5.6 Effect of Information Aggregator Types

As mentioned, we utilize GraphSAGE [26] layers to realize message passing in the proposed GNN-based caching strategy. GraphSAGE introduces three aggregation functions,

which are pooling, mean and lstm. The aggregation function updates the representation of each central node with the representation of itself and its neighbouring nodes. It is the key to realizing information passing between nodes to realize cooperative caching. This section explores the performance of the proposed GNN-based caching strategy with different aggregators. Table 3.4 shows the 95% CI for the CHR and ALT of GraphSAGE's three aggregators in a tree network topology with 55 nodes. The tree topology has a depth of 4, and we utilize 2 GraphSAGE layers. Key network simulation parameters are the default values from Table 3.1.

The results show that the pooling and mean aggregation functions perform better than the lstm aggregators in general. In addition, all aggregators have relatively steady performance. We use the pooling aggregator as the default aggregation function.

3.4.5.7 Effect of the Number of Message Passing Layers

The number of message passing layers is essential in the GNN-based model because it affects how much information each node knows about its neighbouring nodes. If n message passing layers are adopted, each node knows its distance n neighbours. In this section, we explore the performance of the GNN-based caching policy in two different network topologies with different network sizes regarding the different number of message passing layers, where the hidden node dimension of each message passing layer is chosen from {128, 64, 32}.

Table 3.5 shows the 95% CI for the CHR and ALT of the GNN-based caching policy with the different number of message passing layers in a 15-node arbitrary network topology. In this topology, the node farthest from the producer has 4 hops, and we explore GNN's performance with 2 to 5 message passing layers. We can see that the GNN model with 2 or 3 message passing layers performs worse than the model with 4

Table 3.5: 95% CI for the CHR and ALT of the GNN-based caching strategy with different numbers of message passing layers in a 15-node arbitrary network topology.

Number of message passing layers	Nodes feature size (First-last layer)	95% CI for CHR (%)	95% CI for ALT (ms)
2	[128, 64]	25.09 (22.77 to 27.41)	34.19 (32.73 to 35.65)
3	[128, 64, 64]	27.79 (21.48 to 34.10)	31.24 (28.45 to 34.02)
4	[128, 64, 64, 32]	54.15 (53.57 to 54.72)	15.86 (15.58 to 16.14)
5	[128, 64, 64, 64, 32]	54.02 (53.39 to 54.65)	15.92 (15.69 to 16.14)

or 5 message passing layers. The difference is that in the case of 2 or 3 message passing layers, each node can only aggregate the information of its 2-hop or 3-hop neighbouring nodes, but more neighbouring nodes' information can be aggregated with 4 or 5 message passing layers. It shows that the GNN-based caching strategy has difficulty capturing the content popularity across the network with an insufficient number of message passing layers. However, the GNN-based model can realize cooperative caching and achieve outstanding performance with a sufficient number of message passing layers.

To more thoroughly demonstrate the impact of the number of message passing layers on GNN's performance, we also demonstrate the performance of GNN-based policy with different numbers of message passing layers in a larger network topology. Table 3.6 shows the caching performance of the GNN-based strategy with different message passing layers in an arbitrary network topology with 55 nodes. We can observe that, in general, more message passing layers contribute to a higher CHR and a lower ALT. In the 55-node arbitrary network topology, each consumer traverse at most 9 hops to arrive at the producer. Therefore, with 6 to 9 message passing layers, the GNN model can capture the network topology and traffic information more accurately, which helps to achieve excellent performance.

Table 3.6: 95% CI for the CHR and ALT of the GNN-based caching algorithm with different numbers of message passing layers in a 55-node arbitrary network topology.

Number of message passing layers	Nodes feature size (First-last layer)	95% CI for CHR (%)	95% CI for ALT (ms)
2	[128, 64]	16.55 (15.09 to 18.01)	38.63 (37.14 to 40.12)
3	[128, 64, 64]	17.40 (16.11 to 18.68)	38.62 (37.55 to 39.69)
4	[128, 64, 64, 32]	22.91 (22.31 to 23.52)	33.27 (32.56 to 33.99)
5	[128, 64, 64, 64, 32]	22.66 (22.21 to 23.12)	33.34 (33.09 to 33.59)
6	[128, 64, 64, 64, 64, 32]	33.12 (32.61 to 33.62)	26.56 (26.10 to 27.01)
7	[128, 64, 64, 64, 64, 64, 32]	33.59 (32.77 to 34.41)	26.64 (26.08 to 27.20)
8	[128, 64, 64, 64, 64, 64, 64, 32]	32.72 (31.75 to 33.69)	26.96 (26.33 to 27.58)
9	[128, 64, 64, 64, 64, 64, 64, 64, 32]	32.35 (31.25 to 33.45)	26.90 (26.36 to 27.43)

We can conclude that the number of message passing layers is critical for the GNN-based caching strategy. In larger network topology, each node needs to aggregate more neighbour information to learn the network structure and traffic status so that GNN can accurately predict the content caching probability. Therefore, more message passing layers are required in a complex network topology compared with a small network topology.

3.5 Conclusion

In this chapter, we propose a GNN-based caching algorithm and deploy it on Mini-NDN. We compare our caching algorithm with three other popular deep learning-based caching algorithms, 1D-CNN, LSTM-ED, and SAE, and three traditional caching algorithms, LFU, LRU and FIFO. We evaluate the performance of all caching strategies regarding various network parameters, including network topologies, network sizes, content popularity distributions, node cache sizes, and content sizes. Regardless of network parameters, our GNN-based caching algorithm substantially improves the CHR, BHR and ALT compared to all the other caching strategies. In the best case, our GNN-based caching algorithm outperforms the 1D-CNN, LSTM-ED, and SAE caching algorithms by about 50% in terms of the CHR. In addition, at best, the ALT of our GNN-based strategy is around 30% lower than the other three deep-learning caching algorithms. The outstanding performance of the GNN-based caching strategy is that GNN can represent the network topology, and each node can aggregate information from its neighbouring nodes; thus, it helps to capture the structure and traffic information of the whole network and makes cooperative caching decisions.

Chapter 4

GNN-based Proactive Caching Placement Approach in NDN

4.1 Introduction

In contrast to the previous chapter focuses on the caching replacement strategy primarily relying on predicted content caching probability derived from past content request numbers and network architecture, this chapter introduces a proactive caching placement strategy centred on user preference predictions, leveraging features including content attributes, user profiles, user-content ratings. After user preference is predicted, we propose a gain-based caching algorithm to strategically place contents of interest closer to users as well as increase caching diversity across the network.

This chapter tries to address the caching delimiter, where there are hundreds of thousands of different contents but limited cache space across the network. Recently, several papers [79, 80, 119, 120] have addressed this challenge by applying deep learning-based models to predict the number of future content requests and actively cache these popular contents on nodes. However, these approaches solely rely on expected content

request numbers without considering user preferences, which can be inferred from content features, user profiles, and so on. Yet, user preference is a critical factor in caching as it reflects user request patterns and aids in predicting future content interests even for content not previously accessed by users.

The authors of the paper [82] predicted users' future demand through user preferences. They adopted the Non-Negative Matrix Factorization (NMF) [121] technique in the Recommender System (RS) to predict user ratings to videos. Following that, they proactively cached popular videos and achieved promising results. However, the problem with the NMF technique is that it is transductive, and thus it cannot generalize to unseen users or videos during the training stage. To address the NMF's problem, they also considered the previous popularity of videos to help make caching decisions. However, the popularity of videos in the past does not strongly correlate with their popularity in the future. Usually, users who have watched videos in the past are likely not to watch them again in the future. To address these problems, we utilize an Inductive Matrix Completion (IGMC) [28] technique, which is based on Graph Neural Network (GNN), to predict user ratings to videos that have not been watched. Furthermore, we consider the total predicted ratings of a video as the gain of caching the video. Following this, videos are cached according to their ranking of gains in descending order.

The contributions of this chapter are as follows:

- We utilize an inductive GNN-based model to predict user ratings of movies that have not been watched and use the total predicted movies' ratings as the gains in the caching framework. We are the first to apply a GNN model to the caching problem to the best of our knowledge.

- We propose a gain-based caching placement algorithm utilizing gains of caching the movies to make caching decisions.

- We deploy our proposed scheme and state-of-the-art caching algorithms on Mini-NDN. We evaluate various caching algorithms using the real-world dataset and different network topologies. Our proposed caching strategy achieves a 25% higher cache hit ratio, 5% lower latency and 7% lower server load than the state-of-the-art algorithm in a real-world network topology GEANT [30].

The organization of this chapter is as follows: The proposed GNN-based proactive caching placement strategy is first introduced. Following this, experimental results are presented, and finally, we conclude the chapter.

4.2 Proposed Methodology

A. System Model

We consider a NDN network consisting of F forwarders and C user communities, denoted by $\mathcal{F} = \{f_1, f_2, ..., f_F\}$ and $\mathcal{C} = \{c_1, c_2, ..., c_C\}$, respectively. Each user community is placed at a different forwarder. There are U users and M movies in our model, denoted by $\mathcal{U} = \{u_1, u_2, ..., u_U\}$ and $\mathcal{M} = \{m_1, m_2, ..., m_M\}$, respectively. All users are divided into C user communities, and each $u_i \in c_i$ but $u_i \notin \mathcal{C} \backslash c_i$, where $c_i \in \mathcal{C}$. Users give a rating to movies they have watched, denoted as $r_{m_i}^{u_i}$, where $u_i \in \mathcal{U}$ and $m_i \in \mathcal{M}$. For movies they have not watched, the ratings are empty. We consider each rating as a request in NDN. We assume that user communities send interest packets follows a Uniform Distribution with λ_1 requests per second or a Poisson Distribution with λ_2 requests per minute.

In our model, all forwarders have the caching ability with a uniform cache size N, which is defined as number of movies. We apply a binary variable $\{b_{m_i,f_i}\}$, for $m_i \in \mathcal{M}$ and $f_i \in \mathcal{F}$, to indicate forwarder f_i's caching status. We define $b_{m_i,f_i} = 1$ if and only if movie m_i is cached at forwarder f_i. For each forwarder $f_i \in \mathcal{F}$, $\sum_{m_i \in \mathcal{M}} b_{m_i,f_i} \leq N$. Our

proposed caching strategy aims to predict ratings u_i to movie m_i, and make optimized caching decisions within limited caching space.

B. IGMC Ratings Prediction Model

Unlike traditional matrix factorization techniques [121, 122], IGMC [28] trains a GNN model. We take an approach similar to IGMC to make rating predictions. Given that a matrix contains ratings from users to movies, we build an undirected bipartite graph $G = (U, M, E)$, where U denotes sets of users, M denotes sets of movies, and E denotes set of edges. Edges exist between a user u_i and a movie m_i instead of two users or movies. Each edge has a label $r_{m_i}^{u_i}$.

The first component is enclosing subgraph extraction. From a (u_i, m_i) pair, a Breadth-first Search (BFS) strategy is applied to extract u_i's and m_i's h-hop enclosing subgraph. We select a 1-hop subgraph, and each subgraph includes: (i) the target user u_i, (ii) the target movie m_i, (iii) all users that have watched the movie m_i, (iv) all movies that the user u_i has watched, (v) known edges and corresponding labels between users and movies. The subgraph is fed into a GNN model and mapped to the target rating $r_{m_i}^{u_i}$.

The second part of the rating prediction is node labelling. It is to distinguish the target user, target movie, user-type nodes and movie-type nodes in the enclosing subgraph extracted in the first step. We give label 0 to the target user and label 1 to the target movie. Other nodes' labels are given according to the hop count included in the subgraph. For example, a user-type node is included at the n^{th} hop, and then it will be given a label $2n$, while a movie-type node will be given a label $2n + 1$ if it is included at the same hop. One crucial point is that node labels depend on the local subgraph rather than the global bipartite graph. Therefore, we can predict ratings even for a subgraph from an entirely different bipartite graph. After labelling each node in the subgraph, we consider the one-hop encoding of each label as the initial feature of that node.

The third part trains a GNN model to predict ratings from the (u_i, m_i)'s 1-hop subgraph. We utilize a graph-level GNN strategy and aim to map the subgraph to the target rating $r_{m_i}^{u_i}$. The IGMC paper applies a Relational Graph Convolutional Neural Network (RGCN) operator [123] to implement the message passing layers in GNN. RGCN is an extension of [8], and the main difference is that the former one is to handle heterogeneous graphs where there are different edge types in a graph, while the latter one not. In our dataset, the ratings range from 1 to 5, each with an edge type. Therefore, RGCN is adopted to handle the five edge types. It works as follows: 1) for a central node, aggregates its 1-hop neighbouring nodes features; 2) Update the central node's feature based on the neighbouring nodes' features and edge types. The procedure is as below:

$$x_i^{l+1} = tanh\left(W_0^l x_i^l + \sum_{r\in R}\sum_{j\in\mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|} W_r^l x_j^l\right), \tag{4.1}$$

where x_i^l is node i's feature at layer l, and W_0^l is a learnable weight matrix that applies to the node's self-loop connection. Note that each edge has a rating feature, and edges with the same feature have the same edge type. We use R to denote the set of all edge types. For each edge type $r \in R$, $\mathcal{N}_r(i)$ is the set of 1-hop neighbour nodes of node i connected through edge type r, and W_r^l is a learnable weight matrix corresponding to the edge type r and massage passing layer l. For each node i and a message passing layer l, we compute a feature vector x_i^l. After computing L feature vectors, each corresponding to a message passing layer, we concatenate the feature vectors computed for each node and consider this as the node's final representation. Next, we concatenate final representations of the target user and target movie and treat it as a graph representation. Finally, the Rectified Linear Unit (ReLU) activation function and Multilayer Perceptron (MLP) are applied on the graph representation to predict the rating $\hat{r}_{m_i}^{u_i}$.

It is worth noting that the model only leverages subgraph patterns and ignores user or movie features, which are difficult to achieve due to information privacy and label cost. Besides, it is inductive as it learns GNN parameters rather than user or movie embeddings. Therefore, the model generalizes well to unseen users or movies during the training stage. Furthermore, the model can transfer to new tasks since different datasets may share similar subgraph rating structures.

After predicting users' ratings of unwatched movies, we apply our gain-based caching decisions across the network. The proposed caching decision algorithm is described in the next section.

C. Caching Decision

This section introduces the caching decisions for each forwarder in the network. We consider each movie's total predicted ratings in a user community $c_i \in \mathcal{C}$ as the gain of caching the movie:

$$g_{m_i}^{c_i} = \frac{\sum_{u_i \in c_i} r_{m_i}^{u_i}}{\max_{m_j \in \mathcal{M}} \sum_{u_i \in c_i} r_{m_j}^{u_i}}, \tag{4.2}$$

where $\sum_{u_i \in c_i} r_{m_i}^{u_i}$ is the sum of movie m_i's predicted ratings in the user community c_i, and $\max_{m_j \in \mathcal{M}} \sum_{u_i \in c_i} r_{m_j}^{u_i}$ is the maximum sum of a movie's ratings in c_i.

We normalize the gains of each movie by the maximum gain of a movie in the same user community. The total ratings for each movie reflect the movie's popularity across all users in that community. We aim to maximize the total gain G of caching movies in

the network, which is mathematically formulated as follows:

$$\begin{aligned} G &= \sum_{m_i \in \mathcal{M}} \sum_{c_i \in S_{f_i}} g_{m_i}^{c_i} b_{m_i,f_i} \\ \text{s.t.} &\sum_{m_i \in \mathcal{M}} b_{m_i,f_i} \leq N, f_i \in \mathcal{F} \quad , \\ & b_{m_i,f_i} \in \{0,1\}, m_i \in \mathcal{M}, f_i \in \mathcal{F} \end{aligned} \tag{4.3}$$

where S_{f_i} is the set of user communities whose requests pass through the forwarder f_i. Besides, the number of cached movies in each forwarder f_i does not exceed the maximum cache size N.

Given a network topology and $g_{\mathcal{M}}^{\mathcal{C}}$, our task is to make caching decisions for each forwarder in order to optimize Equation 4.3. It is worth mentioning that our network topology is static, and the routing policy is the shortest path routing. We firstly apply Dijkstra's algorithm to find the shortest path from each f to the server and optimize the content caching along the shortest path tree. In the shortest path tree, the server node is considered the root. Let V denotes a node (i.e., a server or a forwarder). Each V is associated with attributes { *id*, *gain_t*, *gain_arr*, *u_set*, *cache_size*, *n*, *child_arr*, *par*, *local_arr*, *global_arr* }, where *id*: a unique id; *gain_t*: a hash table with (item, gain) pair; *gain_arr*: a two dimensional array stores [item, gain] pair in its *gain_t*; *u_set*: a set storing user communities' id whose requests pass through the current node; *cache_size*: an integer scalar indicates the cache size, $cache\_size = 0$ for the server and $cache\_size = N$ for forwarders; n : an integer scalar indicates the node should cache the item with n^{th} highest gain, $n = 0$ by default; *child_arr*: an array stores a node's direct children; *par*: the node's direct parent, each node has at most one parent node due to the extraction of the shortest path tree; *local_arr*: an array that stores cached items by the node; *global_arr*: an array stores the node and its ancestors' cached items.

Algorithm 1 Gain-based caching algorithm

Input: all user communities set C
Output: caching decision for all $f \in \mathcal{F}$

for c in C **do**
    $c.u\_set.add(c.id)$
    $root$ = NODE-INITIALIZATION(c)
end for
CACHING-DECISION($root$)

Algorithm 1 illustrates our proposed gain-based caching strategy where all user communities are provided as input. Each user community has a unique id and is placed at a different forwarder in the network topology. For each user community, we update its user community set (u_set) with its own id and then call the "Node-Initialization" function with a parameter c.

Algorithm 2 Node-Initialization

Input: c, a user community

if $c.par$ not None **then**
    for key in $c.gain\_t$ **do**
        if key in $c.par.gain\_t$ **then**
            $c.par.gain\_t[key] \leftarrow c.par.gain\_t[key] + c.gain\_t[key]$
        else
            $c.par.gain\_t[key] \leftarrow c.gain\_t[key]$
        end if
    end for
    $c.par.u\_set \leftarrow c.par.u\_set \cup c.u\_set$
    if $c.par.u\_set$!= $c.u\_set$ **then**
        $c.par.n \leftarrow 0$
    else
        $c.par.n \leftarrow c.n + 1$
    end if
    NODE-INITIALIZATION($c.par$)
else
    return c
end if

Algorithm 2 represents the process of node initialization, where a user community c is provided as input. In this function, we traverse the shortest path tree from c to the server. If c has a parent node, we update its parent node's $gain\_t$ by merging the parent node's and child node's $gain\_t$ using plus operator. Besides, we update the parent node's u_set by unioning the child node's u_set. If the parent node receives the exact user community requests as its child node, we assign its child node's n plus 1 to the parent node's n (e.g., assume $n = 0$, then the child node caches the item with the highest gain, but the parent node caches the item with the second-highest gain). The idea is to put the popular item near the user community and optimize the caching diversity. Otherwise, we assign 0 to the parent node's n, indicating the parent node caches the item with the highest gain in the parent node's $gain\_t$. The process is repeated until the server node is reached.

After updating nodes' information, the Algorithm 3 is executed, where the server node is the input. We traverse the shortest path tree from the server to the forwarder to make caching decisions. Firstly, each forwarder's $gain\_t$ is updated by removing items cached by its ancestors. The idea is to make downstream forwarders not cache items cached by upstream forwarders. The following is to append (item, gain) pairs in $gain\_t$ to the $gain\_arr$ and sort $gain\_arr$ by gains in descending order. Next, the loop (i.e., $cache\_size$) indicates the cache space of the current forwarder. For each iteration, the forwarder caches the item with the gain in the $(n * cache\_size)^{\text{th}}$ index in $gain\_arr$. The cached item is inserted into the forwarder's $local\_arr$ and $global\_arr$. Besides, the item is removed from $gain\_t$ and $gain\_arr$. The process is repeated until reaching the user community.

Once the caching decisions for each forwarder are made, we proactively load items in the forwarder's $local\_arr$ to its cache store. The proactive caching process makes sure

Algorithm 3 Caching-Decision

```
    Input: s, a server node
 1: if s.par is None then
 2:     for child in s.child_arr do
 3:         CACHING-DECISION(child)
 4:     end for
 5: else
 6:     for m in s.par.global_arr do
 7:         del s.gain_t[m]
 8:     end for
 9:     for m in s.gain_t do
10:         gain_arr.append([m, gain_t[m]])
11:     end for
12:     gain_arr ← sort(gain_arr, (a1, a2) → (a2[1] − a1[1]))
13:     for i ← 1...s.cache_size do
14:         item_index ← s.n ∗ s.cache_size
15:         item ← gain_arr[item_index][0]
16:         s.local_arr.append(item)
17:         s.global_arr.append(item)
18:         del s.gain_t[item]
19:         del gain_arr[item_index]
20:     end for
21:     for child in s.child_arr do
22:         CACHING-DECISION(child)
23:     end for
24: end if
```

that, before users send *Interest* packets, contents are already available in the forwarder's cache store to satisfy user requests. Regarding the cache replacement policy, when the forwarder's cache store is full, the content with the lowest cache yield will be evicted first.

4.3 Experimental Results

To evaluate our caching algorithm, we use Mini-NDN[27] to perform all experiments. Mini-NDN is an emulation tool, and it runs real instances of NDN packages. We deploy

our GNN-GM and the NMF-based proactive caching strategy proposed in paper [82] on Mini-NDN. The authors of [82] also took user mobility into account when making caching decisions due to the highway simulation environment, which is different from our experimental environment. Therefore, we do not consider user mobility. Except for the user mobility, [82] employs the same caching scheme as [81], which is the method we compare. We utilize their caching decision module to calculate the gain of caching movies in each forwarder. After that, we apply our proposed gain-based caching placement algorithm to make caching decisions for each forwarder. Besides, we also compare our proposed caching algorithm with GNN-CPP [124], which employs the GNN model to make item caching probability predictions. It is worth noting that GNN-CPP makes predictions only based on the item's requested numbers in the past. For GNN-GM, NMF-based caching strategy, and GNN-CPP, we preload items that need to be cached into the forwarder's cache store before users send requests. In addition, GNN-GM and NMF-based caching strategies employ cache replacement policies based on content caching gains, and GNN-CPP employs caching replacement policies based on predicted content popularities. Furthermore, we compare two traditional reactive caching strategies, Leave Copy Everywhere + Least Recently Used (LCE+LRU) and LCE + First-in-first-out (LCE+FIFO).

4.3.1 Experimentation Setup

This section presents network topologies, traffic generation, dataset collection and metrics we used to evaluate caching algorithms GNN-GM, NMF-based caching strategy, CNN-CPP, LCE+LRU and LCE+FIFO.

Table 4.1: Experimentation Parameters

Parameters	Default Value	Values
Network topology	GEANT [30]	GEANT [30], Tree or Arbitrary Topology
Number of nodes	45	10-60
Number of producers	1	
Number of forwarders	44	9-59
Number of user communities	2	
Number of users	943	
Number of distinct movies	1682	
Requests rate	Uniform Distribution with 1 req/sec	Uniform Distribution with 1 req/sec or Poisson Distribution with 50 req/min

4.3.1.1 Network Topology

Similar to papers [125, 126], we employ a real-world network topology GEANT [30], which has 45 nodes associated with 71 edges. The server is placed at the "UK" node, and all other nodes are forwarders. In addition, we explore a tree network topology with 50 nodes. We also explore random topologies with various numbers of nodes {10, 20, 30, 40, 50, 60}. There are one content producer and two user communities for all topologies, and each user community randomly accesses a forwarder. In particular, the root node is the content producer, and user communities can only access leaf nodes in the tree topology. It is worth noting that all forwarders have uniform caching capability.

4.3.1.2 Traffic Generation

We employ NDN Traffic Generator[115] to generate *Interest* and *Data* packets. We assume each user community sends interest packets in a Uniform Distribution with one request per second or a Poisson Distribution with 50 requests per minute. Table 4.1 shows key parameters and values used in this chapter.

4.3.1.3 Dataset Collection

We use the public benchmark dataset MovieLens 100K [29], which includes 943 users and 1682 movies. We sort the dataset by timestamp and use 80% of it as the training dataset and 20% as the testing dataset to compare the performances of various caching strategies. Similar to papers [81, 82], we consider the user rating for a movie as the user request for that movie. We randomly divide 943 users into two user communities.

4.3.1.4 Evaluation Metrics

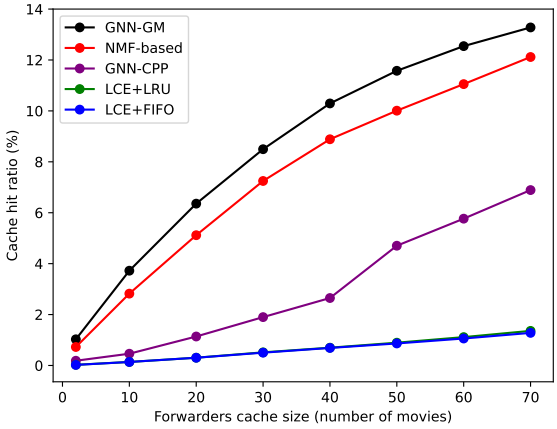
The following three metrics are adopted to evaluate various caching algorithms:

- CHR (Cache Hit Ratio): It defines the percentage of requests that can be satisfied by the cached data packets. The CHR is defined in Eqn. 3.4.
- 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. 3.6.
- Server Load: It defines the number of *Interest* packets served by the server.

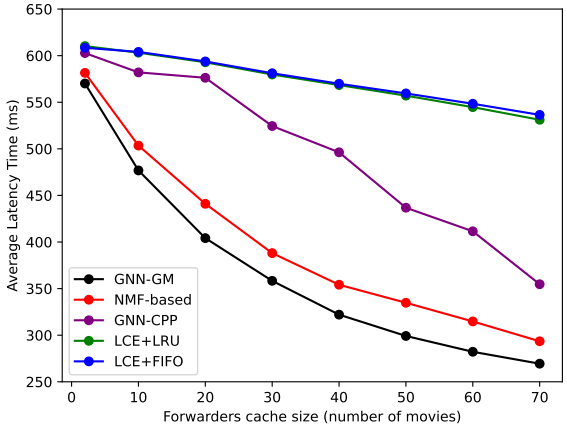
4.3.2 Results

This section describes the experimental results for GNN-GM, NMF-based caching strategy, GNN-CPP, LCE+LRU, and LCE+FIFO. We utilize 4 RGCN layers in the GNN-based rating prediction model. Both GNN and NMF are trained and tested using the same dataset, and they are trained with the Adam optimizer and Stochastic Gradient Descent (SGD) optimizer, respectively. The loss function is the mean square error. The GNN-CPP model requires time-series data. Therefore, we divide the training dataset into four time periods, with 20,000 requests within one time period. The testing dataset

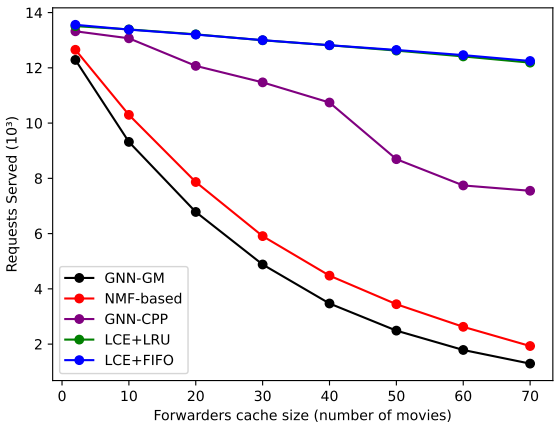
has 20000 requests, and thus it can be considered a single period. We use the content requests number in the previous two time periods to predict content caching probability in the next period. The GNN-CPP model includes 3 GNN layers and is trained using the Adam optimizer, and the loss function is binary cross-entropy. All experiments are run multiple times, and the results have been averaged.



(a) Cache hit ratio



(b) Average latency



(c) Server Load

Figure 4.1: GNN-GM, NMF-based, GNN-CPP, LCE + LRU, and LCE + FIFO caching algorithms' performances with different forwarders' cache sizes in a 50 nodes tree network topology.

4.3.2.1 Effect of Node Cache Sizes in Tree Topology

Figure 4.1a shows the cache hit ratio of the five caching algorithms with various forwarders' caching abilities in a 50 nodes tree network topology. The cache size of a forwarder is {2, 10, 20, 30, 40, 50, 60, 70}. We can observe that the cache hit ratio increases with the increase of forwarders caching size for all caching strategies. GNN-GM achieves the best performance among the five caching strategies. On average, the GNN-GM caching algorithm has a 20% higher cache hit ratio than the NMF-based one. Benefit from accurate user rating predictions and applying total predicted ratings to make caching decisions, GNN-GM caching algorithm has a significant performance improvement (40% higher) over the NMF-based caching algorithm when each forwarder can cache two movies. The GNN-CPP algorithm performs worse than the other two proactive caching strategies because it only considers previous content requests when making predictions. However, in reality, users will not be likely to watch movies they have watched before. Besides, GNN-GM caching algorithm can perform nearly 200% better on average than the other two traditional reactive caching algorithms, LCE+LRU and LCE+FIFO.

Figure 4.1b shows the average latency of the five caching algorithms. At best, the GNN-GM caching algorithm achieves around 11% and 35% lower latency than the NMF-based caching algorithm and GNN-CPP, respectively. In addition, GNN-GM consistently achieves the lowest latency regardless of the cache size. LCE+LRU and LCE+FIFO have the worst performance, with a notable margin compared with the other three proactive caching strategies. In the best case, GNN-GM caching algorithm can achieve 50% lower latency than LCE+LRU and LCE+FIFO.

Figure 4.1c shows that the server load decreases as the forwarders' cache sizes increase. Overall, GNN-GM caching algorithm can achieve a 20% lower server load than the NMF-

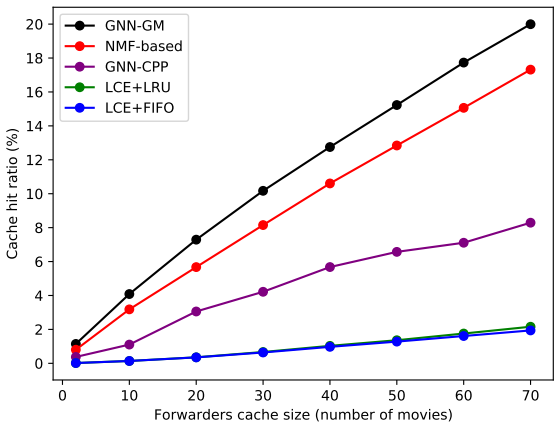
based caching algorithm. The proactive caching algorithm GNN-CPP has a heavier server load than GNN-GM and NMF-based caching algorithms. Besides, LCE+LRU and LCE+FIFO have the heaviest server load among the five caching algorithms. On average, GNN-GM caching algorithm can perform almost 60% lower server load than the LCE+LRU and LCE+FIFO.

The results indicate that our GNN-GM caching algorithm has an outstanding performance in a tree network topology. GNN-GM can catch user preferences and put movies that most users will likely watch near the user in advance. In addition, our GNN-GM caching strategy improves cache diversity by ensuring that different movies are cached on the path.

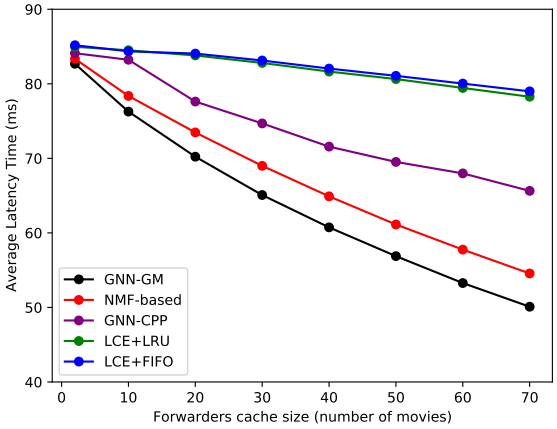
4.3.2.2 Effect of Node Cache Sizes in GEANT

Figure 4.2a shows the cache hit ratio of the five caching algorithms with various forwarders' caching abilities in GEANT. Similar to the Section 4.3.2.1, the cache size of a forwarder varies from 2 to 70 movies. GNN-GM performs best among the five caching methods in the GEANT network topology. The cache hit ratio of GNN-GM is, on average, about 25% higher than that of the NMF-based caching algorithm. When each forwarder can only cache two movies, the GNN-GM can achieve a 40% higher cache hit ratio than the NMF-based caching algorithm. In addition, GNN-CPP performs worse than the GNN-GM and NMF-based caching algorithms because it only considers user content requests number in previous time steps. LCE+LRU and LCE+FIFO still have the worst performance because they are reactive caching strategies that do not capture users' future preferences.

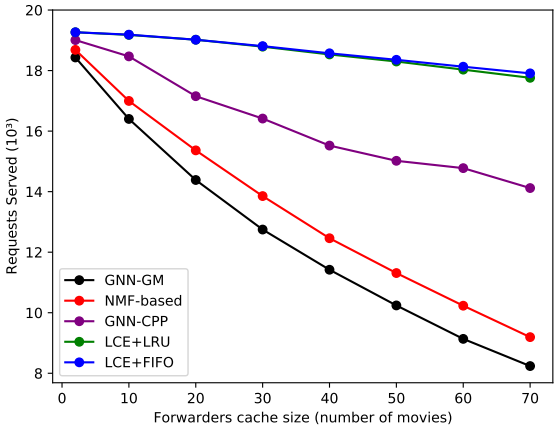
Figure 4.2b shows the average latency of the five caching algorithms. At best, GNN-GM achieves around 8% and 25% lower latency than the NMF-based caching algorithm



(a) Cache hit ratio



(b) Average latency



(c) Server Load

Figure 4.2: GNN-GM, NMF-based, GNN-CPP, LCE + LRU, and LCE + FIFO caching algorithms' performances with different forwarders' cache sizes in GEANT.

and GNN-CPP, respectively. GNN-GM consistently achieves the lowest latency regardless of the cache size. The other two traditional reactive caching algorithms have the worst performance, with a notable margin with the other three proactive caching strategies. In the best case, GNN-GM can achieve around 30% lower latency than LCE+LRU and LCE+FIFO.

Figure 4.2c shows the server load of the five caching algorithms. The server load of all caching algorithms decreases as the forwarders' cache sizes increase. Overall, GNN-GM can achieve a 7% lower and a 22% lower server load than the NMF-based caching algorithm and GNN-CPP, respectively. Because LCE+LRU and LCE+FIFO have the lowest cache hit ratio, more number of *Interest* packets are forwarded to the server. It results in the server load of LCE+LRU and LCE+FIFO being much higher than the other three proactive caching strategies. On average, GNN-GM can perform 30% lower server load than the two traditional caching algorithms.

In short, our GNN-GM can catch user preferences, resulting in a higher cache hit ratio. Besides, the lower latency demonstrates that interest packets can be satisfied along the forwarding path before reaching the server. GNN-GM can also put more popular content nearer to the user in order to improve user experiences. Our GNN-GM can definitely decrease the traffic workload and provide better QoS.

4.3.2.3 Effect of User Requests Distribution in GEANT

Table 4.2 shows the cache hit ratio, average latency and server load for GNN-GM, NMF-based, GNN-CPP, LCE+LRU and LCE+FIFO in GEANT when the user requests follow a Poisson distribution with a request rate of 50 requests per minute. All forwarders have a uniform cache size of 30. The table shows that GNN-GM achieves the best performance, with significant improvements compared to other caching algorithms. In particular,

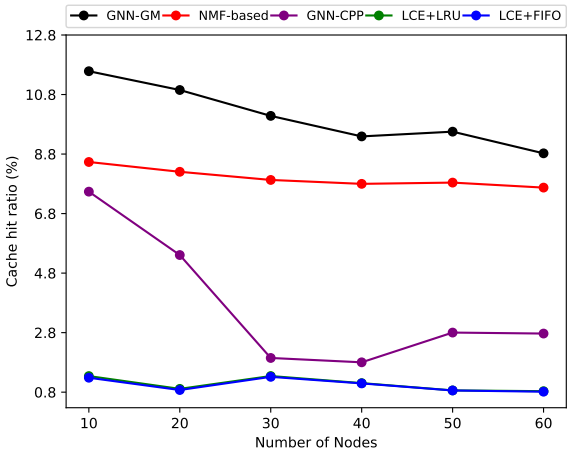
GNN-GM achieves a 27% cache hit rate, 6.3% latency, and 9.2% server load compared to the NMF-based caching algorithm. GNN-GM achieves a tremendous advantage in all three performance metrics compared to GNN-CPP. In addition, the other two reactive caching algorithms, LCE+LFU and LCE+FIFO, perform the worst and possess a large gap with the other three proactive caching algorithms.

Table 4.2: Cache hit ratio, average latency and server load for GNN-GM, NMF-based, GNN-CPP, LCE + LRU, and LCE + FIFO when user requests follow a Poisson Distribution in GEANT.

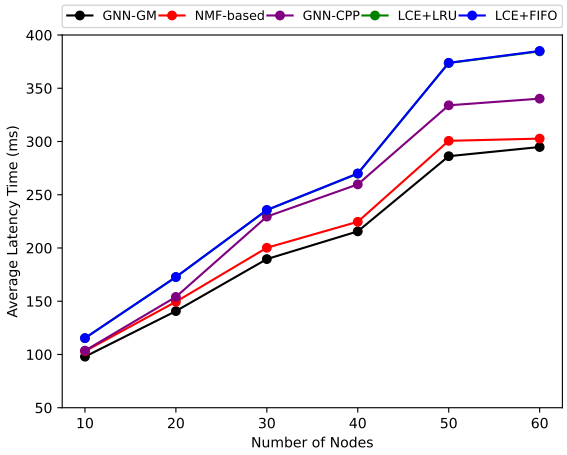
Algorithms	Cache hit ratio (%)	Average latency (ms)	Server load (10^3)
GNN-GM	10.875	60.851	6.174
NMF-based	8.554	64.986	6.803
GNN-CPP	4.544	70.55	8.11
LCE+LFU	0.76	79.093	9.357
LCE+FIFO	0.734	79.166	9.365

4.3.2.4 Effect of Network Sizes for Arbitrary Topologies

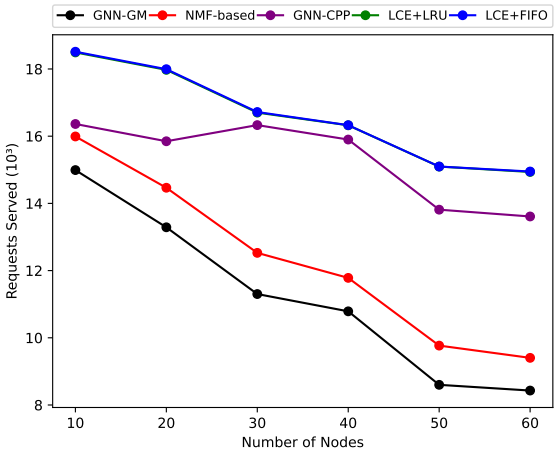
Figure 4.3 shows the cache hit ratio, average latency and server load for five caching algorithms with a different number of nodes {10, 20, 30, 40, 50, 60}. In each network topology, all forwarders have a uniform cache size of 30. From Figure 4.3a, we can notice that the cache hit ratio decreases as the number of nodes increases. The reason is that the user community is far away from the content provider in a large network topology. In this case, an *Interest* packet has to be forwarded through more nodes to reach the content provider. It results in a much larger denominator than the numerator in Equation 3.4. We can easily find that GNN-GM always performs best regardless of the number of nodes. In the best case, GNN-GM can achieve about 36% higher cache hit ratio than the NMF-based caching strategy. From the figure, we can see that GNN-CPP performs worse when the network size is large. GNN-CPP is sensitive to the model structure, i.e. the number of GNN layers used to train the model. Since all experiments utilize



(a) Cache hit ratio



(b) Average latency



(c) Server Load

Figure 4.3: GNN-GM, NMF-based, GNN-CPP, LCE + LRU, and LCE + FIFO caching algorithms' performances with a different number of nodes in random topologies. The performances of LCE+LRU and LCE+FIFO overlap each other.

only 3 GNN layers, each node in the GNN can only know information about its 3-hop nodes at most. It is not sufficient in larger network topologies. On average, the cache hit rate of GNN-GM is 200% higher than that of the GNN-CPP algorithm. In addition, LCE+LRU and LCE+FIFO have similar performances and are the worst among the five caching strategies.

Figure 4.3b shows the average latency for the five caching algorithms. We can see that GNN-GM has the lowest latency for all network sizes. It demonstrates that our GNN-GM caching algorithm can catch user preferences and put contents that most users will be interested in near the user. More number of *Interest* packets can be satisfied along the path, and thus users can receive the videos without much time waiting. At best, GNN-GM performs a 5.8% lower and a 17% lower latency than the NMF-based caching algorithm and GNN-CPP, respectively. The figure also shows that LCE+LRU and LCE+FIFO have almost exactly the same average latency and perform significantly worse than the other three deep learning-based proactive caching strategies.

Figure 4.3c shows the number of requests served in the server node for the five caching algorithms. If more number of *Interest* packets are satisfied along the path, then less number of them will be served by the server node. It makes sense that GNN-GM's server serves the lowest number of user requests. Following that is the NMF-based caching algorithm, GNN-CPP, LCE+LRU and LCE+FIFO. It is worth noting that GNN-GM alleviates a significant amount of server load than the other four caching strategies. The other two traditional caching algorithms, LCE+LRU and LCE+FIFO, almost overlap in the figure and have the highest server load.

We can conclude that our GNN-GM caching strategy has the best performance for a different number of nodes in arbitrary network topologies. GNN-GM can capture user preferences and increase the diversity of caches. In addition, it can cache popular videos

near users. The GNN-GM caching strategy can significantly ease the traffic load and enhance user experiences.

4.4 Conclusion

In this chapter, we propose GNN-GM, a GNN-based active cache placement policy. The cache placement strategy is implemented based on the ranking of cached movies in terms of gains, and movies with high cache gain replace movies with low cache gain. We can predict user ratings more accurately than NMF by using a GNN-based model to predict ratings. We can also cache popular movies and place movies near users by considering the total ratings of predicted movies as gain and applying our gain-based caching decision. We compared our caching strategy with two state-of-the-art, NMF-based caching algorithms, GNN-CPP, and two traditional reactive caching algorithms, LCE+LRU and LCE+FIFO. We deployed these five caching algorithms on Mini-NDN and evaluated them using real-world datasets. We evaluated the five caching algorithms' performances among a tree network topology with various cache sizes, a real-word network topology GEANT with various cache sizes, a GEANT with user requests following a Poisson Distribution, and random network topologies with a different number of nodes. The evaluation results show that our GNN-GM can consistently achieve the highest cache hit ratio, lowest latency and lowest server load. More notably, our proposed caching algorithm has a 25% higher cache hit ratio, 5% lower latency and 7% lower server load on average than the NMF-based caching algorithm in GEANT. In addition, our proposed caching algorithm performs much better than GNN-CPP, which utilizes only previous user movie requests for prediction. Note that GNN-CPP is an end-to-end algorithm, while our algorithm is not. The experimental results show that GNN-GM can perform much better than the end-to-end algorithm GNN-CPP. In particular, the average latency

and server load of GNN-GMM are about 13% and 25% lower than GNN-CPP, respectively, in GEANT. In addition, GNN-GMM provides more significant improvements than the other two traditional caching strategies, namely LCE+LRU and LCE+FIFO.

Chapter 5

GNN-DRL-based Proactive Caching Placement Approach in SDN-based ICN

5.1 Introduction

In contrast to the previous two chapters, which primarily focus on supervised learning strategies to train GNN-based models using collected datasets, this chapter delves into the realm of DRL. DRL, a form of online learning, involves continuous interaction between an agent and the environment, where the agent receives rewards from the environment for each action taken, allowing for sequential caching decisions aimed at maximizing long-term rewards. Specifically, in this chapter, we propose a GNN-DRL-based proactive caching placement strategy within an SDN-ICN context.

In recent years, DRL has made significant advancements in decision-making, particularly in caching decisions. Numerous studies (see [87, 90]) have demonstrated the exceptional performance of DRL in solving caching problems. Researchers have adopted

various 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 processing Euclidean space data like 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.

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 [42]. Notably, GNNs have demonstrated remarkable success in network routing optimization [127], where the underlying graph structure captures the intricate relationships between network nodes and facilitates efficient path planning. Additionally, in the domain of traffic predictions [37], GNNs leverage the graph structure of road intersections and their connectivity to forecast traffic flow patterns accurately.

Moreover, recent research has highlighted the remarkable generalization capabilities of GNNs [128]. 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 [55, 129, 130], 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 for 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 chapter aims to enhance caching performance in the SDN-ICN scenario by leveraging DRL and GNN. Specifically, we introduce a GNN-Double Deep Q-network [31] (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 chapter are as follows:

- We develop a statistical model to generate users' preferences. Initially, we employ matrix factorization based on the Neural Collaborative Filtering Model [131] to learn content and user embeddings using the real-world dataset MovieLens100K [29]. Next, we employ a Gaussian Mixture Model to cluster users and contents 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 GNN in DRL is advantageous as GNN excels at modelling graph-structured data, enabling nodes to engage in cooperative caching and enhancing overall caching performance. Additionally, with just 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 contents, cache

sizes, and network topologies such as GEANT [33], ROCKETFUEL [34], TISCALI [35], and GARR [35]. Notably, our proposed caching scheme outperforms the state-of-the-art DRL-based caching strategy. Furthermore, it exhibits a significant performance advantage over several benchmark caching schemes (Leave Copy Down (LCD), Probabilistic Caching (PROB_CACHE), Cache Less for More (CL4M), and Leave Copy Everywhere (LCE) [57, 59–61]). 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 time 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, like other deep learning algorithms, may suffer from overfitting or underfitting, depending on various factors.

The structure of this chapter is as follows: We start by introducing the system model, followed by an explanation of the proposed methodology. Subsequently, we delve into the experimental results, and finally, draw the conclusion of the chapter.

5.2 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 [29]. Important notations used throughout the chapter are listed in Table 5.1.

Table 5.1: Important Notations

Notation	Definition
N	Number of network nodes
C	Number of contents
T	Number of time slots
$\mathcal{N} = \{n_1, ..., n_N\}$	Set of network nodes
$\mathcal{C} = \{c_1, ..., c_C\}$	Set of contents
$\mathcal{U} = \{u_1, ..., u_U\}$	Set of users
$\mathcal{T} = \{t_0, t_1, ..., t_T\}$	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})
$\mathcal{S}_{t_l} = \{s_{t_l}^{n_1}, ..., s_{t_l}^{n_N}\}$	Set of all nodes' states at time step t_l
$\mathcal{A}_{t_l} = \{a_{t_l}^{n_1}, ..., a_{t_l}^{n_N}\}$	Set of all nodes' caching actions at time step t_l
$a_{t_l}^{n_k} = \{b_{t_l}^{c_1,n_k}, ..., b_{t_l}^{c_C,n_k}\}$	Set of node n_k's caching action at time step t_l
$\mathcal{R}_{t_l} = \{r_{t_l}^{n_1}, ..., r_{t_l}^{n_N}\}$	Set of all nodes' rewards (i.e., cache hits) at time step t_l
$r_{t_l}^{n_k} = \{r_{t_l}^{c_1,n_k}, ..., r_{t_l}^{c_C,n_k}\}$	Set of node n_k's reward for each content 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

5.2.1 System Architecture

We propose an intelligent caching strategy in an SDN-ICN architecture, as depicted in Figure 5.1. 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 [31, 44] agent responsible for making caching decisions in the control plane. The data plane comprises network nodes that perform caching actions.

Figure 5.1 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 the data plane's network topology consists of N ICN nodes, denoted as $\mathcal{N} = \{n_1, n_2, ..., n_N\}$. 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 request and data packets across the network. Instead of assuming all router nodes have the caching capability, we consider only part of them to have. Those router nodes equipped with caching capabilities have a cache capacity of z, defined as the number of contents.

The network contains C distinct contents, represented by the set $\mathcal{C} = \{c_1, ..., c_C\}$. We assume an experimental time round can be divided into T slots of equal duration, denoted by $\mathcal{T} = \{t_0, t_1, ..., t_T\}$. To indicate whether a node n_k caches a 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}.

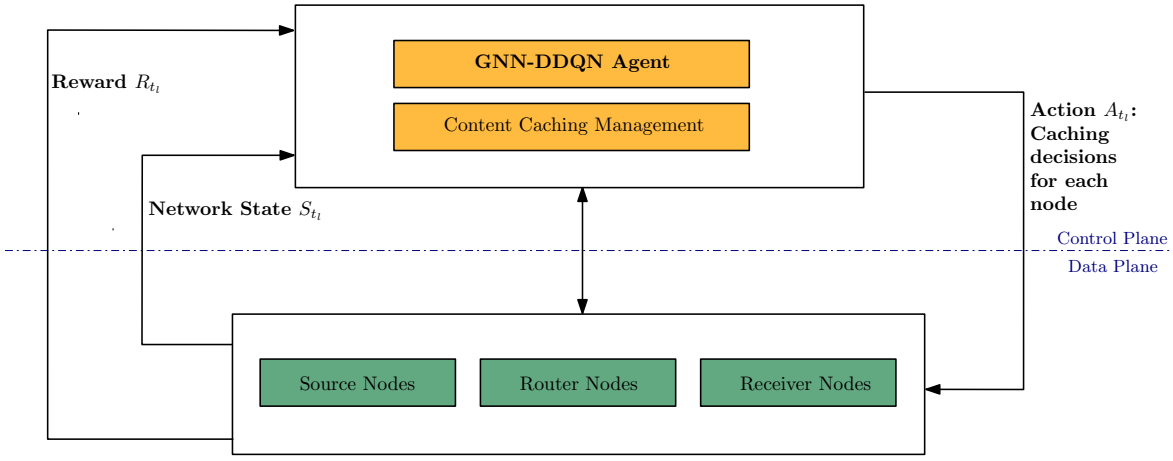


Figure 5.1: The SDN-ICN architecture. In the controller, the GNN-DDQN Agent receives a network state S_{t_l} and generates an action A_{t_l} at each time step t_l. Subsequently, it receives a reward R_{t_l} at the next time step t_{l+1}.

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 that 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, $\mathcal{R}_{t_l} = r_{t_l}^{n_1}, r_{t_l}^{n_2}, ..., r_{t_l}^{n_N}$ represents the rewards of all nodes at time step t_l. Furthermore, $r_{t_l}^{n_k} = r_{t_l}^{c_1,n_k}, r_{t_l}^{c_2,n_k}, ..., r_{t_l}^{c_C,n_k}$ represents the set of rewards for each content received by node n_k at time step t_l.

5.2.2 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 preference to simulate this network traffic. In our network, receiver nodes correspond to users, and we denote the users as $\mathcal{U} = \{u_1, ..., u_U\}$. 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 [116] have shown that the content popularity in the network can be modelled using a Zipfian distribution,

$$P(c_i) = \frac{1}{(i)^{\alpha} \sum_{k=1}^{C} (k)^{-\alpha}} \tag{5.1}$$

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 [131, 132] 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 those 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 preference 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 and user. Specifically, for each content 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 [29] as a real-world dataset for our experiments. This dataset consists of user ratings for movies and is widely used for 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 [131, 132]. 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:

$$\hat{m}_{i,j} = \sigma(\mathbf{w}^T(x_{c_i} \odot y_{u_j})) \tag{5.2}$$

In this equation, σ denotes the sigmoid activation function, $\mathbf{w}$ 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 5.2.

In order to fit the number of contents and users in our network, all users and contents in the dataset are divided into groups. If the content embeddings are close to each other,

Table 5.2: Key NCF model training parameters.

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

we cluster them into groups, and users are grouped in the same way. We utilize the 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 a 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 a given content c_i. This probability represents the preference of user u_j for content c_i. Inspired by the works [133, 134], we calculate the joint probability $P(c_i, u_j)$ of content c_i being requested by user u_j as follows:

$$\begin{aligned} P(c_i, u_j) &= P(c_i)P(u_j|c_i) \\ &= P(c_i)\frac{\exp({x_{c_i}}^T y_{u_j})}{\sum_{j=1}^{U} \exp({x_{c_i}}^T y_{u_j})} \end{aligned} \tag{5.3}$$

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 user preference and content popularity.

Note that our approach differs from [133, 134], as we obtain user and content embeddings from a real-world dataset. Furthermore, we consider the inner product of the learned user embeddings and content embeddings to measure their association, enabling us to capture the relationship between users and content meaningfully.

5.3 Proposed Methodology

This section presents our GNN-DDQN agent, which incorporates a GNN as the Q-network within the DDQN framework [31]. DDQN improves upon the original DQN algorithm [44] by mitigating Q-value overestimation and enhancing 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.

5.3.1 State Space

The network state $\mathcal{S}_{t_l} = \{s_{t_l}^{n_1}, ..., s_{t_l}^{n_N}\}$ 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 contents 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:

- 1^{st} component: The number of requests for each content 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 contents in receiver nodes, cached contents in router nodes, and published contents in source nodes.

- 2^{nd} component: The cache storage of the node, represented by a binary variable for each content c_i. A value of 1 indicates that the node caches the content during the previous time interval, while 0 indicates otherwise.

- 3^{rd} component: The content publication of the node, is also represented by a binary variable for each content c_i. A value of 1 indicates that the node has published the content during the previous time interval, while 0 indicates otherwise.

5.3.2 Action Space

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

$$a_{t_l}^{n_k} = \{b_{t_l}^{c_1,n_k}, ..., b_{t_l}^{c_C,n_k}\}, \tag{5.4}$$

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,

$$\mathcal{A}_{t_l} = \{a_{t_l}^{n_1}, \ldots, a_{t_l}^{n_N}\} \tag{5.5}$$

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 contents according to $a_{t_l}^{n_k}$, which can be used to satisfy the request in the future.

5.3.3 Reward Function

Our objective is to maximize the cache hit ratio. Thus we use cache hits as the agent's reward, denoted as $\mathcal{R}_{t_l} = \{r_{t_l}^{n_1}, ..., r_{t_l}^{n_N}\}$, which includes cache hits of each node. For a node n_k at time step t_l, its cache hits for each content is $r_{t_l}^{n_k} = \{r_{t_l}^{c_1,n_k}, ..., r_{t_l}^{c_C,n_k}\}$. Let's assume a node n_k's reward sum for all contents at time step t_l is $cacheHits_{t_l}^{n_k} =$

$r_{t_l}^{c_1,n_k} + r_{t_l}^{c_2,n_k} + ... + r_{t_l}^{c_C,n_k}$, then the objective function can be formulated as follows:

$$\begin{aligned} &\max \sum_{n_k \in \mathcal{N}} \sum_{t_l \in \mathcal{T}} cacheHits_{t_l}^{n_k} \\ &\text{s.t.} \\ &\sum_{c_i \in \mathcal{C}} b_{t_l}^{c_i,n_k} \leq \begin{cases} z, & \text{node } n_k \text{ has the caching capability} \\ 0, & \text{otherwise} \end{cases}, \forall t_l \in \mathcal{T}, \forall n_k \in \mathcal{N} \end{aligned} \tag{5.6}$$

This objective aims to maximize the cache hit ratio for the stored contents while ensuring that the number of contents 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 contents, while other nodes, such as source and receiver nodes, can only distribute or receive contents.

5.3.4 GNN Architecture

Our methodology utilizes a GNN for node-level Q-value predictions. The model architecture, depicted in Figure 5.2, 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.

For the GNN architecture, our approach utilizes four GraphSage layers [26]. Each layer has different hidden embedding dimensions, specifically 1024, 512, 256, and C. GraphSage is an inductive framework that leverages sampling and aggregation tech-

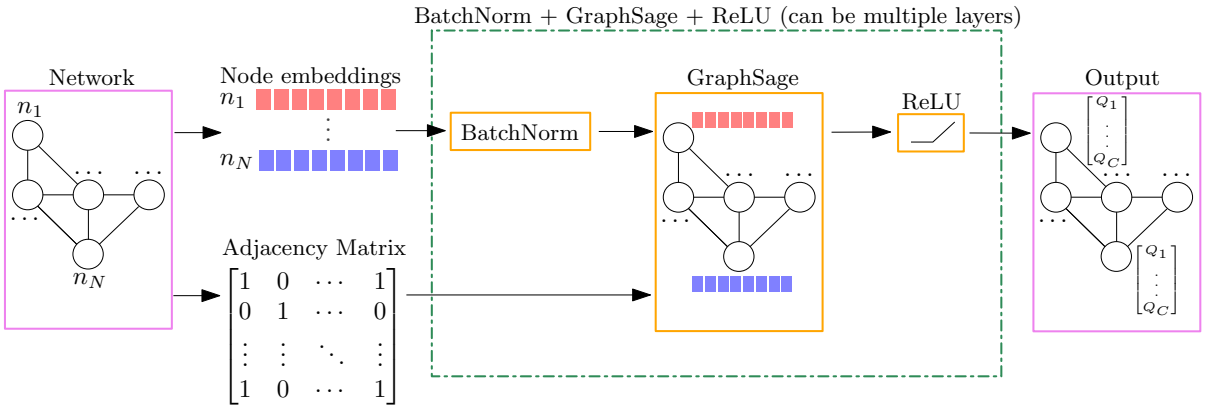


Figure 5.2: Model architecture.

niques 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:

$$h^k_{N(v)} = AGG_k(h^{k-1}_u, \forall u \in N(v)), \tag{5.7}$$

where $N(v)$ represents the one-hop neighbors 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 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:

$$h_v^k = \sigma\left[W^k \cdot \text{CONCAT}\left(h_v^{k-1}, h_{N(v)}^k\right)\right], \tag{5.8}$$

where W^k represents a learned matrix specific to the k^{th} step, σ denotes the Rectified Linear (ReLU) activation function, and h_v^k 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_v^{k-1} and $h_{N(v)}^k$.

5.3.5 The GNN-DDQN Agent

The GNN-DDQN agent operates based on the procedure described in Algorithm 4. 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 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 [44]. 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

Algorithm 4 GNN-DDQN Agent Operation

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 $n_k \in \{n_1, ..., n_N\}$ **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 5.10a 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 5.9
        Update $\hat{\theta} = \theta$ every K steps
    end for
end for

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.

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:

$$L(\theta) = \sqrt{\mathbb{E}\left[\sum_{b_j}((\mathcal{Y}_{b_j} - Q(\mathcal{S}_{b_j}, \mathcal{A}_{b_j}|\theta))^2 \cdot mask)\right]}, \tag{5.9}$$

where $\mathcal{Y}_{b_j}$ and $mask$ are defined as follows:

$$\mathcal{Y}_{b_j} = \begin{cases} \mathcal{R}_{b_j}, & \text{if episode terminates at } b_{j+1} \\ \mathcal{R}_{b_j} + \frac{1}{z}\gamma \sum_{r \in \hat{Q}(\mathbf{S}_{b_{j+1}}, \arg\max_{\mathcal{A}'_{b_{j+1}}, |\mathcal{A}'_{b_{j+1}}|=z} Q(\mathbf{S}_{b_{j+1}}, \mathcal{A}'_{b_{j+1}}|\theta)|\hat{\theta})} r, & \text{otherwise} \end{cases}, \tag{5.10a}$$

$$mask = \begin{cases} 1, & \text{if } n_k \text{ is a router with the caching capability} \\ 0, & \text{otherwise} \end{cases} \tag{5.10b}$$

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 5.9. Nodes without caching capabilities are assigned a mask value of 0, while routers with caching capabilities have a mask value of 1.

To maintain 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 overestimation of Q-values during training.

The key training parameters for the GNN-DDQN model are summarized in Table 5.3.

Table 5.3: Key GNN-DDQN model training parameters.

Parameters	Values
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

5.4 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* [32], a Python-based ICN caching simulator that comprehensively evaluates different caching strategies. Unlike being bound to any specific architecture such as Content-centric Network (CCN) or Named Data Network (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 5.2.2, were considered. Table 5.4 lists the key simulation parameters used. We followed the recommendations from a previous study [135] 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 contents, 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 contents it can 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 seconds. We conducted 600 experiments for each caching scenario and calculated the average evaluation metrics based on the results of the last 200 experiments.

The evaluation of different caching strategies relies 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. The formula for CHR is given in Eqn. 3.4.

Table 5.4: Key simulation parameters.

Parameters	Values
Network topology	GEANT [33], ROCKETFUEL [34], TISCALI [35] and GARR [35]
Internal link delay (all networks)	2 ms
External link delay (all networks)	34 ms
Number of Distinct Contents	Range: 600-1000 contents
Content Size	1500 bytes
Request Size	150 bytes
Cache size	Range: 1-4 contents
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 seconds
Number of Experimentations	600

- Average Latency (ALT): The average latency represents the average delay between the moment a user sends an *Interest* packet and the moment it receives the corresponding *Data* packet. The formula for ALT is given in Eqn. 3.6.

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

$$APS = \sum_{i=1}^{I} \frac{path_{i,n_u,n_r}}{Path_{i,n_u,n_s}}, \tag{5.11}$$

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,

$$ALL = \frac{\sum_{l=1}^{L} \mathcal{L}_l}{L}, \tag{5.12}$$

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 is evaluated and compared with the state-of-the-art caching scheme MLP-DDQN. MLP-DDQN, which has been extensively studied in various research works [87, 90], is used as a baseline for comparison. We have adapted the MLP-DQN framework to incorporate the DDQN technique to ensure a fair comparison. The MLP-DDQN agent consists 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 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 neighboring information like the GNN approach.

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

5.4.1 Effect of Content Number

This section examines the impact of the number of contents on caching performances. The number of contents ranges from 600 to 1000. Figure 5.3 illustrates how caching performances vary with the number of contents in the GEANT [33] network, where routers with the caching capability have a uniform cache size of 1 content. 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 the cache capability, which amounts to 19 nodes in this case.

Figure 5.3 demonstrates that GNN-DDQN consistently outperforms all other caching strategies across different numbers of distinct contents. GNN-DDQN achieves 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 surpasses LCD and PROB_CACHE by 41.33% and 103.92% in CHR, respectively. It also achieves significantly lower ALT, APS, and ALL than LCD and PROB_CACHE. Furthermore, the performance gap between GNN-DDQN and LCE and CL4M is even more pronounced regarding all evaluation metrics.

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

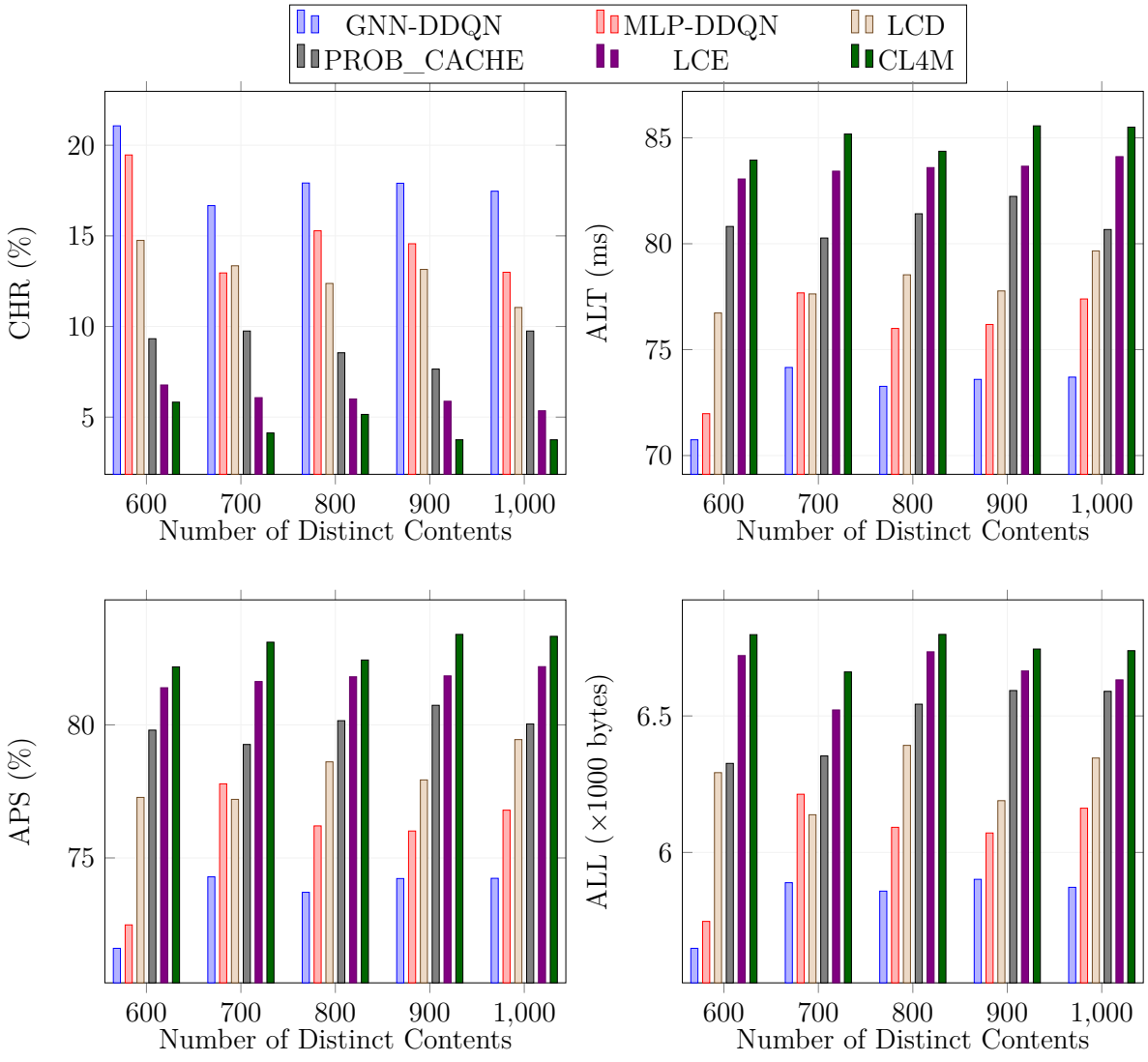


Figure 5.3: The cache performances of GNN-DDQN, MLP-DDQN, LCD, PROB_CACHE, LCE, and CL4M vary with the number of contents in the GEANT network.

5.4.2 Effect of Cache Size

This section investigates the performances of different caching schemes across various router cache sizes, defined as the number of contents. Figure 5.4 presents the caching performances of GNN-DDQN, MLP-DDQN, LCD, PROB_CACHE, LCE, and CL4M under different caching scenarios. The router cache sizes range from 1 to 4, while the number of contents is fixed at 1000.

GNN-DDQN exhibits a substantial performance advantage over MLP-DDQN when the cache size is limited to 1 content. For cache sizes of 2 and 4 contents, GNN-DDQN and MLP-DDQN perform similarly. However, when the cache size is set to 3 contents, GNN-DDQN outperforms MLP-DDQN by achieving an 11.87% higher CHR, 3.57% lower ALT, 1.54% APS, and 2.20% lower ALL.

Regardless of router's cache sizes, GNN-DDQN consistently reduces latency 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 contents in advance and proactively cache them.

5.4.3 Effect of Network Topology

To further evaluate the effectiveness of the proposed caching scheme, we conducted experiments on different network topologies, namely ROCKETFUEL [34], TISCALI [35], and GARR [35]. The aim was to assess the robustness of the caching scheme in diverse network environments.

Table 5.5 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.

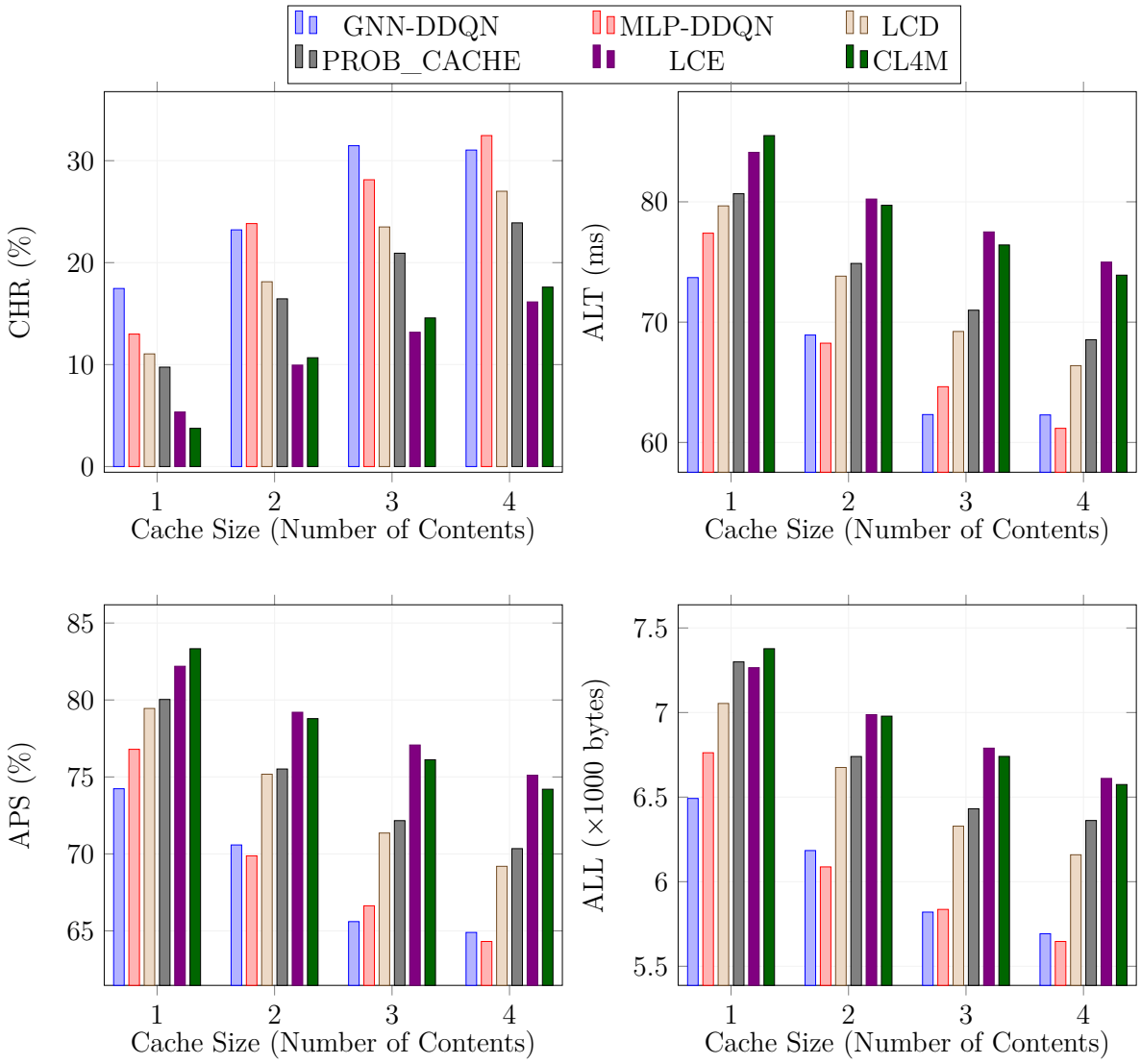


Figure 5.4: The cache performances of GNN-DDQN, MLP-DDQN, LCD, PROB_CACHE, LCE, and CL4M vary with the router's cache size in the GEANT network.

Table 5.5: The number of source, router, and receiver nodes for different network topologies.

Topologies	Source Nodes	Router Nodes	Receiver Nodes
ROCKETFUEL [34]	10	104	104
TISCALI [35]	44	160	36
GARR [35]	13	27	21

Table 5.6: The caching performances 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

This section evaluates the caching performances of different strategies in ROCKETFUEL, TISCALI, and GARR network topologies. The experiments are conducted with a content number of 1000, and all routers with caching capabilities have a uniform cache size of 1 content. The results are summarized in Table 5.6.

Across all network topologies, GNN-DDQN consistently outperforms the other strategies. Specifically, in ROCKETFUEL, GNN-DDQN achieves a 2.89% higher CHR than MLP-DDQN. In TISCALI, the margin becomes 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 ROCKET-FUEL and TISCALI.

Furthermore, GNN-DDQN demonstrates 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.

5.5 Conclusion

In this chapter, we introduce 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 apply GNNs to process network topologies, enabling cooperative caching among nodes and promoting a wider variety of cached contents. By integrating GNNs into DRL, our proposed approach empowers the DRL agent to make caching decisions for all nodes in the network with just one forward pass through the neural network. This integration not only streamlines the caching decision-making process but also harnesses the power of GNN-DRL synergy in optimizing caching strategies.

Firstly, we generate user preferences for content based on a real-world dataset. This step ensures that the evaluation reflects realistic user behaviour and content demand patterns. Next, we develop 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 compare 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 shows that GNN-DDQN consistently outperforms MLP-

DDQN in most scenarios. Notably, in the best-case scenario, GNN-DDQN achieves a remarkable 34.42% higher CHR, a 4.76% lower ALT, a 3.77% lower APS, and a 5.21% lower ALL compared to MLP-DDQN. Furthermore, GNN-DDQN demonstrated superior performance compared to classical caching schemes. To assess the robustness of our proposed scheme, we conduct experiments on benchmark network topologies, including GEANT, ROCKETFUEL, TISCALI, and GARR. GNN-DDQN consistently delivers outstanding performance across these diverse network topologies, reinforcing its reliability and applicability in real-world scenarios.

Chapter 6

GNN Multi-agent-DRL-based Proactive Caching Placement Approach in ICN-based Three-tier Edge Network

6.1 Introduction

In contrast to the preceding chapter, which uses a centralized agent responsible for caching decisions across all network nodes, this chapter introduces a fully distributed proactive caching placement strategy within a three-tier edge network. Furthermore, the focus shifts to mobile users operating in a wireless network environment, departing from the static user requests in a wired network environment featured in the previous three chapters.

The exponential growth in mobile device usage among business and consumer users worldwide has become a compelling phenomenon. Smartphones and tablets have become indispensable tools for various communication needs. According to a report by Ericsson ([136]), the global number of smartphone mobile network subscriptions reached approximately 6.6 billion in 2022, with forecasts predicting a further increase to over 7.8 billion by 2028. This tremendous surge in mobile users, devices, and connections poses significant challenges to network performance.

To address the escalating demand for internet traffic, edge caching has emerged as a promising solution to alleviate the strain on backhaul links and reduce transmission latency in mobile networks. By caching files closer to users in edge devices such as Base Stations (BSs), user requests can be fulfilled without the need to reach the cloud. However, edge devices' limited cache storage capacity due to cost constraints presents a crucial dilemma. Caching numerous unpopular files on an edge device wastes cache storage and fails to mitigate backhaul transmissions. Hence, it becomes essential for edge devices to intelligently cache frequently accessed files while staying within the cache storage limits.

However, several challenges exist to achieving optimal caching: (i) Users' preferences may change over time, requiring each BS to capture users' preference dynamics all the time quickly. (ii) Users may move from one cell to another covered by a different BS. Neighbouring BSs must realize cooperative caching to minimize caching waste due to user movement. (iii) Users' activity levels may differ in different cells, and BSs must consider them to make cooperative caching decisions. For example, if a BS covers a set of users that send Internet requests frequently, at the same time, its neighbouring BS covers a set of highly inactive users. Then the neighbour of that BS should cache popular files requested by users covered by that BS, not those requested by users covered by itself.

Traditional caching strategies have difficulties in dealing with above challenges. Leave Copy Everywhere (LCE) [57] is a popular traditional caching placement strategy where a data packet is cached at the edge device as long as it passes through. Due to the lack of consideration for file popularity and caching variety, LCE does not use cache space efficiently. Once the caching space is full, caching replacement strategies are introduced. Popular classic caching replacement strategies are First-in-first-out (FIFO) [137], Least Recently Used (LRU) [137], and Least Frequently Used (LFU) [138]. FIFO does not consider file popularities and replaces files by arrival orders only. In LRU, a file is replaced first if it has not been requested for a long time. LFU considers long-term file popularity, and a file is first replaced if requested the least number of times. In general, LRU and LFU perform better than FIFO, and in this chapter, we compare our proposed caching strategy with them. However, their significant drawbacks are that LRU does not consider a file's popularity over a long period, while LFU cannot quickly capture changes in file popularities over a short period.

With the development of artificial intelligence, more and more researchers are using cutting-edge deep learning-based caching strategies to address shortcomings of traditional algorithms. Papers [102, 139, 140] proposed supervised learning-based caching strategies, where deep learning models are trained offline using labelled datasets and then deployed online to make real-time caching decisions. However, a supervised-learning-based caching strategy has three main drawbacks: (i) Labelled data is expensive to obtain. (ii) Caching is an NP-hard problem without an optimal solution; thus, getting ground truth labels to the caching dataset is impossible. (iii) Such models perform poorly if the real-time network traffic distribution differs from the training data distribution.

This is where DRL comes into play. DRL is a framework for sequential decision-making. Unlike supervised learning, no labelled data is required, and the DRL agent is

trained online and learns through interacting with the environment. Through trial and error tests, the DRL agent aims to perform actions that maximize its reward. Recently, papers [89, 141, 142] have developed DRL-based caching strategies in edge networks. However, these works employ a centralized DRL agent to make caching decisions for all BSs. It is impractical because each BS has to send its state to the cloud periodically, increasing the backhaul link load and consuming more transmission costs. As a result, decentralized edge caching is of increasing interest. Researchers have been leveraging caching strategies based on Multi-Agent RL (MARL), known for collaborative decision-making and solving complex problems [143].

This chapter proposes a MARL-based caching strategy named "Spatial-Temporal Graph Attention Network-Soft Actor-Critic" (STGAN-SAC). Soft Actor-Critic (SAC) [106] is one of the popular DRL frameworks and has been demonstrated [106] to perform better than Deep Q-Learning (DQL) [44], policy gradient [45] and AC. DQL trains a value approximation function and computes the Q-value for each state-action pair. When the action space is large, DQL results in slow convergence and brittleness to scalability [105]. Different from DQL, policy gradient trains and optimizes a policy directly. However, it suffers from high variance, slow convergence, and sample inefficiency [144]. AC combines DQL and policy gradient and mitigates the disadvantages of both frameworks. It uses an actor and a critic network to learn a policy and a state-action value function, respectively. The actor is responsible for making caching decisions, and the critic evaluates actions taken by the actor. AC does better than DQL because it learns a policy directly and takes actions based on the probability distribution over all actions. In addition, AC enhances policy gradient with a trained value approximation function, significantly improving sample efficiency. Compared with AC, the SAC agent explores more, and SAC models learn faster [106]. Therefore, we use SAC as our DRL framework.

Our actor and critic networks have the same network structure but different weights. Each network is a spatial-temporal Graph Attention Network (GAT), including a multi-headed GAT layer and a Gated Recurrent Unit (GRU) layer. Spatial-temporal GAT has been demonstrated to be influential in many areas, such as traffic forecasting [145], traffic light control [146] and so on. The spatial block, multi-head GAT, is a famous GNN [5] architecture and can operate on graph-structured data. The temporal block, GRU, is a prevalent Recurrent Neural Network (RNN) architecture and can operate on time-series data. In reality, user requests have been shown to have spatial-temporal dependencies [94]. Therefore, we apply spatial-temporal GAT to capture such dependencies. This chapter applies a neighbour-awareness caching [108], i.e., each BS knows its neighbours. As a result, we first model each BS's state and its one-hop BSs' states as a graph and then feed it into a multi-head GAT layer to learn the attention coefficient between neighbouring BSs. In this way, BSs can realize adaptive cooperative caching by considering diverse users' activity levels and preferences covered by different BSs. We then utilize a GRU layer to extract user requests' temporal dynamics.

STGAN-SAC differs from other state-of-the-art caching strategies for the following reasons: (i) STGAN-SAC makes caching decisions without knowing file popularities in advance. (ii) STGAN-SAC considers user mobility. (iii) STGAN-SAC is fully decentralized, where each BS maintains an agent rather than a central agent maintained in the cloud, which leads to communication overhead. (iv) STGAN-SAC realizes adaptive cooperative caching, where each BS cooperates with its neighbouring BS. (v) STGAN-SAC applies spatial-temporal models to extract user requests' spatial-temporal dependencies. Other papers either use a simple network architecture, such as Multilayer Perceptron (MLP) or lack spatial or temporal modules.

The contributions of this chapter are as follows:

- We formulate the edge caching problem as a problem of maximizing transmission cost savings through caching.
- We propose a "Spatial-Temporal Graph Attention Network-Soft Actor-Critic (STGAN-SAC) caching strategy. As far as we know, we are the first to apply spatial-temporal GNNs and MARL in caching problems.
- With extensive experiments, we demonstrate the importance of the spatial block (i.e., multi-head GAT) and the temporal block (i.e., GRU) in our model. We also show that STGAN-SAC performs much better than state-of-the-art and traditional caching strategies.

The structure of this chapter is as follows: Firstly, the system architecture is introduced, followed by the presentation of the proposed STGAN-SAC agent. Next, experimental results are discussed, and finally, we conclude the chapter.

6.2 System Architecture

In this section, the system model and user mobility model are introduced first. Then the optimization objective of this chapter is presented. Important notations are listed in Table 6.1.

6.2.1 System Model

The system architecture is illustrated in Figure 6.1. There are four BSs (i.e., $B = 4$), denoted as $\mathcal{B} = \{b_1, b_2, b_3, b_4\}$, and they are evenly deployed within a 600 m $\times$ 600 m space. Each BS is connected to the cloud via a backhaul link, and neighbouring BSs

Table 6.1: Important Notations

Notation	Definition
B	Number of BSs
F	Number of file chunks
T	Number of time instants
N	Cache size of each BS
L	Length of temporal sequences
$\mathcal{B} = \{b_1, ..., b_B\}$	Set of BSs
$\mathcal{F} = \{f_1, ..., f_F\}$	Set of file chunks
$\mathcal{T} = \{t_0, ..., t_T\}$	Set of time instants
$\mathcal{U} = \{u_1, ..., u_U\}$	Set of users
$\mathcal{UG} = \{ug_1, ..., ug_{UG}\}$	Set of user groups
$x^n_{u_i}$	User u_i's x coordinate at n^{th} time interval
$y^n_{u_i}$	User u_i's y coordinate at n^{th} time interval
$F_{ug_i} \leq F$	Number of file chunks requested by the user group ug_i
$\mathcal{F}_{ug_i} \subseteq \mathcal{F}$	Set of file chunks requested by the user group ug_i
R_{b_i}	Number of requests arriving at b_i
$\mathcal{R}_{b_i} = \{r_1, ..., r_{R_{b_i}}\}$	Requests stream arriving at b_i
$C_{b_i,cloud}$	Transmission cost of a file chunk between b_i and the cloud
C_{b_i,b_j}	Transmission cost of a file chunk between b_i and b_j
C_{b_i,u_j}	Transmission cost of a file chunk between b_i and u_j
$cache^{t_k}_{f_i,b_j}$	"Cache" or "Not cache" f_i in b_j at t_k

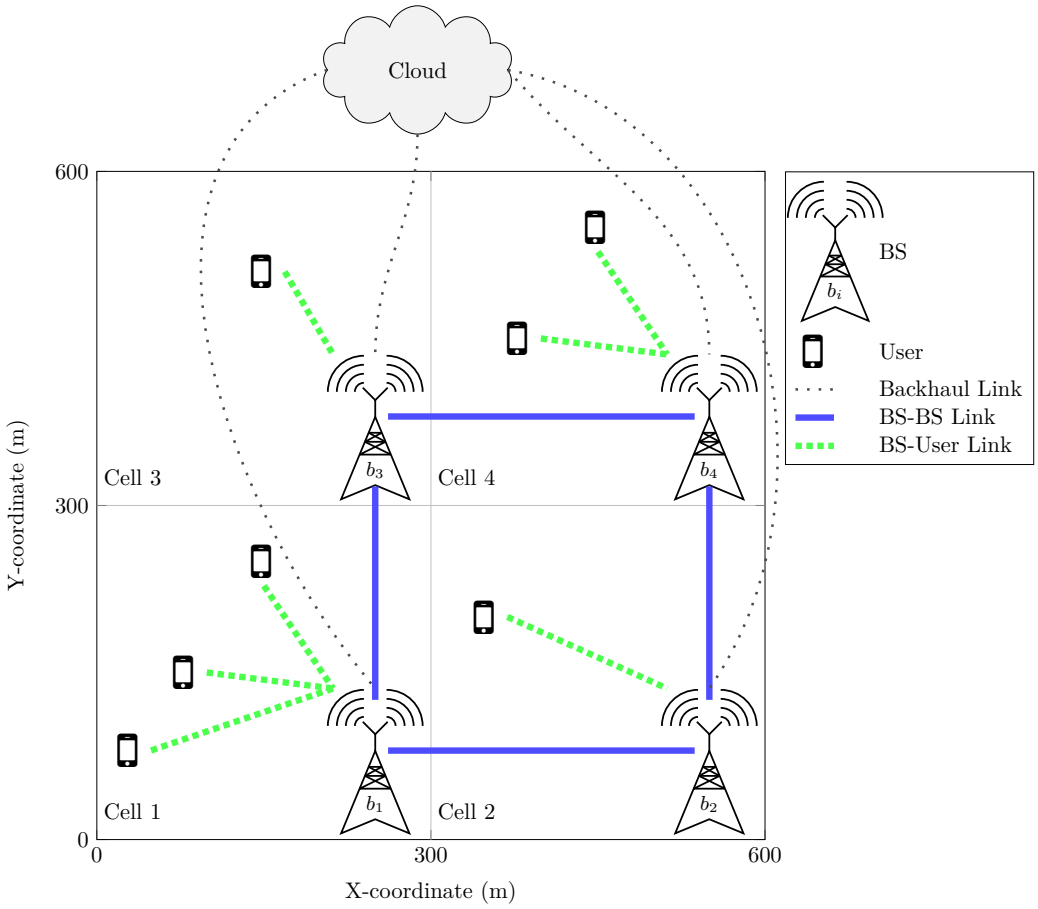


Figure 6.1: System Architecture. Users move within a 600 m × 600 m area, where 4 BSs are evenly distributed in the corresponding cells.

are connected via a BS-BS link. Users are denoted as $\mathcal{U} = \{u_1, ..., u_U\}$, and each user connects to one BS through a BS-User link. We refer to the user's connected BS as the user's direct BS. In this chapter, we assume users are moving within a rectangular area. We use $(x^n_{u_i}, y^n_{u_i})$ to denote user u_i's position at n^{th} time interval, where $x^n_{u_i}$ and $y^n_{u_i}$ are the x and y coordinates in the Two-Dimensional (2D) space, respectively. Initially, $x^0_{u_i}$ and $y^0_{u_i}$ are chosen from [0, 600 m] following a uniform distribution. We also assume users are grouped in different user groups, denoted as $\mathcal{UG} = \{ug_1, ..., ug_{UG}\}$. Note that each user belongs to one user group only.

Each BS has the same cache size N, defined as the number of file chunks, and each BS knows its one-hop neighbours. When a user's direct BS cannot satisfy that user's request, it can get the requested file from the neighbouring BS if the neighbour caches the file or from the cloud otherwise.

The network contains F file chunks, and we use $\mathcal{F} = \{f_1, ..., f_F\}$ to denote the set of all file chunks. As in the paper [97], we assume that each file chunk has the same size and that all file chunks are available in the cloud. We also assume that different user groups have distinct preferences. We use $F_{ug_i} \leq F$ to denote the number of file chunks requested by user group ug_i, and $\mathcal{F}_{ug_i} \subseteq \mathcal{F}$ to denote the set of file chunks requested by the user group. Note that $F_{ug_i} = F_{ug_j}$ and $\mathcal{F}_{ug_i} \neq \mathcal{F}_{ug_j}, \forall ug_i \in \mathcal{UG}, ug_j \in \mathcal{UG}$ and $i \neq j$.

We divide each experiment's time into T uniform periods, and all time instants are denoted as $\mathcal{T} = \{t_0, t_1, ..., t_T\}$, where t_0 is the experiment's start time instant. We use $cache^{t_k}_{f_i,b_j}$ to denote whether b_j caches file chunk f_i at time instant t_k. If yes, f_i is available in b_j's cache store between time interval t_k and t_{k+1}.

We use $\mathcal{R}_{b_i}$ to denote the number of requests arriving at b_i in one experiment. The request stream arriving at each BS b_i is denoted as $\mathcal{R}_{b_i} = \{r_1, ..., r_{R_{b_i}}\}$. Note that each request $r_j \in \mathcal{R}_{b_i}$ is associated with a file chunk f_k, where $b_i \in \mathcal{B}$ and $f_k \in \mathcal{F}$.

6.2.2 User Mobility Model

Papers [147–149] have demonstrated the importance of considering user mobility when designing a caching strategy. In this chapter, we let each user moves following the Gauss-Markov Mobility Model [147, 150, 151]. The Gauss-Markov Mobility Model can yield more realistic movements because it allows a user's moving speed and direction in the past to influence its future speed and direction, which can avoid sudden stops and sharp turns in Random Walk Mobility Model and Random Waypoint Mobility Model [147].

We initialize each user's speed and direction based on a Gaussian distribution with a mean equal to zero and a standard deviation equal to one, respectively. Each user's speed and direction change after a fixed time interval. In this chapter, we set the interval duration as 1 second ($1s$) and update each user's speed and direction after each time interval. For a user u_i, its speed and direction at n_{th} time interval, $Spe^n_{u_i}$ and $Dir^n_{u_i}$ are updated as follows:

$$Spe^n_{u_i} = \alpha \times Spe^{n-1}_{u_i} + (1-\alpha)\overline{Spe_{u_i}} + G_{spe_{u_i}}\sqrt{(1-\alpha^2)} \tag{6.1.1}$$

$$Dir^n_{u_i} = \alpha \times Dir^{n-1}_{u_i} + (1-\alpha)\overline{Dir_{u_i}} + G_{dir_{u_i}}\sqrt{(1-\alpha^2)} \tag{6.1.2}$$

where $Spe^{n-1}_{u_i}$ and $Dir^{n-1}_{u_i}$ are the user's speed and direction at $(n-1)^{th}$ time interval. α is a hyperparameter with a range of [0, 1]. We set $\alpha = 0.99$ in this chapter. $\overline{Spe_{u_i}}$ and $\overline{Dir_{u_i}}$ are the mean values of the user's speed and direction when n tends to infinity. We assume $\overline{Spe_{u_i}}$ is a constant with a value of 1 $m\ s^{-1}$, and $\overline{Dir_{u_i}}$ is 90 degrees initially but changes over time according to how close the user is near the edges. $G_{spe_{u_i}}$ and $G_{dir_{u_i}}$ are chosen from a Gaussian distribution with $mean = 0$ and $standard\ deviation = 1$, respectively. After each time interval, the user's position is updated. Each user's current position is calculated based on its previous position, speed and direction,

$$x^n_{u_i} = x^{n-1}_{u_i} + Spe^{n-1}_{u_i}\cos Dir^{n-1}_{u_i} \tag{6.2.1}$$

$$y^n_{u_i} = y^{n-1}_{u_i} + Spe^{n-1}_{u_i}\sin Dir^{n-1}_{u_i} \tag{6.2.2}$$

where $(x^n_{u_i}, y^n_{u_i})$ and $(x^{n-1}_{u_i}, y^{n-1}_{u_i})$ are u_i's coordinates in a 2D space at the n^{th} and $(n-1)^{th}$ time interval, respectively. $Spe^{n-1}_{u_i}$ and $Dir^{n-1}_{u_i}$ are u_i's speed and direction at the $(n-1)^{th}$ time interval, respectively. Note that if a move puts the user over the rectangular boundary, the user's direction will be flipped 180 degrees.

6.2.3 Optimization Objective

In this chapter, we aim to minimize the system's transmission costs. As shown in Figure 6.1, the system contains three kinds of links: backhaul, BS-BS, and BS-User links. We use $C_{b_i,cloud}$ to represent the transmission cost of a file chunk between b_i and the cloud,

$$C_{b_i,cloud} = 0.3, \forall b_i \in \mathcal{B} \tag{6.3}$$

The transmission cost between any two BSs, denoted as C_{b_i,b_j}, where $i \neq j$, is much smaller than $C_{b_i,cloud}$, and it is related to the distance between two BSs,

$$C_{b_i,b_j} = w \times Dist_{b_i,b_j}, b_i \in \mathcal{B}, b_j \in \mathcal{B}, i \neq j, \tag{6.4}$$

where w is a positive constant and $Dist_{b_i,b_j}$ is the distance between b_i and b_j. Since all BSs are evenly distributed and thus neighbouring BSs have the same distance. We set $C_{b1,b2} = C_{b2,b3} = C_{b3,b4} = C_{b4,b1} = 0.05$. The transmission cost between a user u_j and its direct BS b_i is the lowest,

$$C_{b_i,u_j} = 0.01, u_j \in \mathcal{U}, b_i \in \mathcal{B} \tag{6.5}$$

We convert the objective of minimizing the transmission cost of the system into maximizing cost savings through caching. Assuming a user's request r_j arrives at its direct BS b_i between the time interval t_k and t_{k+1}, and the requested file chunk is f_l, then the gain of the request r_j is

$$g_{r_j} = \begin{cases} C_{b_i,cloud} & cache^{t_k}_{f_l,b_i} = 1 \\ C_{b_i,cloud} - C_{b_i,b_n} & cache^{t_k}_{f_l,b_i} = 0, cache^{t_k}_{f_l,b_n} = 1 \\ 0 & otherwise \end{cases} \tag{6.6}$$

The first case arises when the direct BS b_i can satisfy the request r_j (i.e., b_i caches the file f_l at time t_k), and the second case arises when b_i's neighbour b_n caches the file f_l at t_k but b_i does not. The third case happens when f_l is obtained from the cloud.

Then, our objective is to maximize the total gain G, which is mathematically formulated as follows,

$$\begin{aligned} G = \sum_{b_i \in \mathcal{B}} \sum_{r_j \in \mathcal{R}_{b_i}} g_{r_j} \quad \text{s.t.} \\ \sum_{f_l \in \mathcal{F}} cache_{f_l,b_i}^{t_k} \leq N, \forall b_i \in \mathcal{B}, \forall t_k \in \mathcal{T}, \end{aligned} \tag{6.7}$$

where the number of files cached in a BS cannot exceed N.

6.3 Multi-Agent Spatial-Temporal Graph Attention-based Soft Actor-Critic

Paper [109] developed a multi-agent SAC framework, and we applied this framework to caching problems. However, [109] requires each agent to maintain the state of all other agents, defeating this chapter's fully decentralized goal. We have made some changes to support this strategy to work in a decentralized manner. Also, different from [109], we apply a spatial-temporal GAT as the actor and critic architecture. The spatial-temporal GAT is similar to paper [145], except that the temporal block is a GRU rather than an Long-Short Term Memory (LSTM) layer. The reason for using GRU is that it has less number of parameters and is easier to train than LSTM. This section first defines an agent's state, action, and reward and then explains the spatial-temporal GAT architecture. Following that, the proposed STGAN-SAC agent is described.

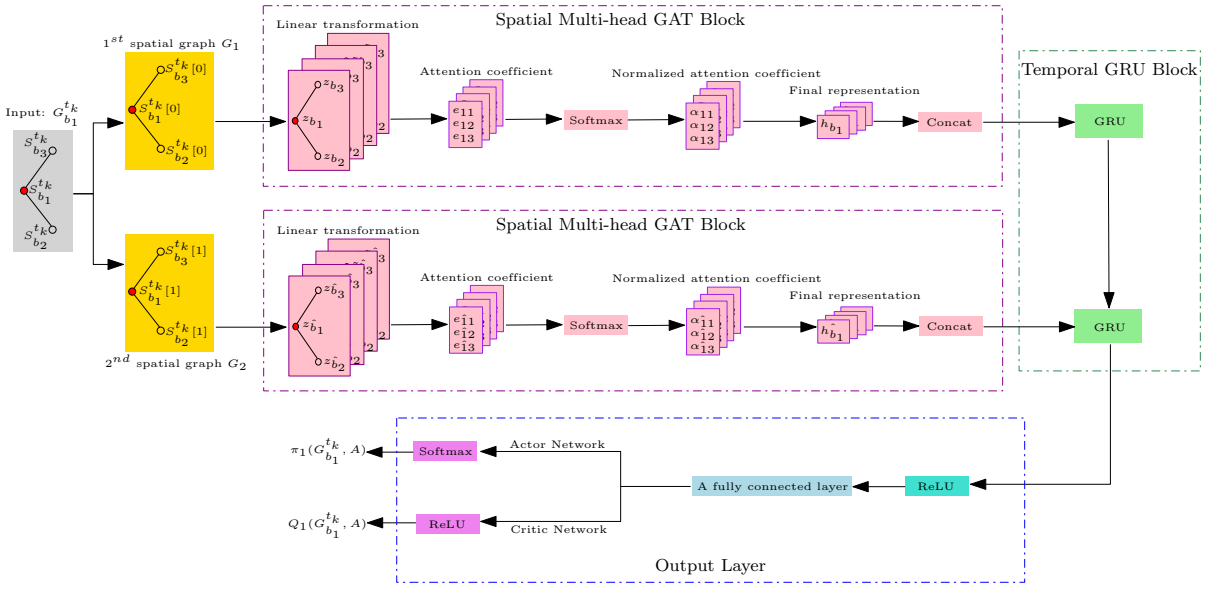


Figure 6.2: Spatial-temporal GAT architecture of the actor and the critic network. In this example, b_1 is the target BS. $S_{b_i}^{t_k}$ is b_i's state at t_k. Multi-head GAT performs graph attention to extract spatial dependencies between BSs. h_{b_1} and h'_{b_1} denote b_1's updated feature representations at each temporal sequence. GRU conducts a recurrent mechanism to extract temporal dependencies on the updated features of b_1.

6.3.1 Multi-Agent SAC

6.3.1.1 State Space

Each BS b_j has a state at a time instant $t_k \in \mathcal{T}$, denoted as $S_{b_j}^{t_k} \in \mathbb{R}^{L \times 2F}$, where L is the length of temporal sequences and F is the number of file chunks. We set $L = 2$. The first and second temporal sequences contain the number of requests received by b_j for each file chunk $f_i \in \mathcal{F}$ and the cache storage of b_j (i.e., a binary variable for each f_i) during the time interval between t_{k-2} and t_{k-1}, and between t_{k-1} and t_k, respectively.

6.3.1.2 Action Space

Each agent has F available actions, and the available action set is denoted as A. The number of actions each agent takes at each time instant equals its corresponding BS's cache size. Suppose $N = 1$ and an agent j (its corresponding BS is b_j) decides to take an action a_{f_i} (i.e., cache the file chunk f_i) at time instant t_k, then $cache_{f_i,b_j}^{t_k} = 1$. The file f_i will stay in the BS's cache store if it is already there. Otherwise, b_j will fetch f_i from the cloud and evict the file chunk that does not need to be cached.

6.3.1.3 Reward Function

Once an agent takes action, it will receive a reward the next instant. In this chapter, each action corresponds to a reward computed from Formula 6.6. Suppose a BS takes action a_{f_i} at t_k, then this action's reward is the sum of gains of requests that request for the file f_i and arrive at the BS during t_k and t_{k+1}.

6.3.2 Spatial-Temporal GAT

In this chapter, the actor and critic have the same network architecture but different parameters. The network architecture is a spatial-temporal GAT, including a multi-

head GAT [9] layer, a GRU layer and an output layer, shown in Figure 6.2.

As we mentioned earlier, each BS knows its neighbours' states. We first use GAT to extract the spatial dependence of neighbouring BSs. For each BS b_i at t_k, there is a spatial-temporal graph $G_{b_i}^{t_k}$. $G_{b_i}^{t_k} = (V_{b_i},\ V_{b_i,state}^{t_k},\ E_{b_i})$, where V_{b_i} is the set of vertices, including b_i and its neighbours. $V_{b_i,state}^{t_k} \in \mathbb{R}^{|V|\times L\times 2F}$ is the set of vertices' states. E_{b_i} is the set of edges, including the connection between two vertices and the self-loop of each vertex.

Figure 6.2 shows the action and q-value prediction process of agent 1 (its corresponding BS is b_1) at t_k. The input is a spatial-temporal graph $G_{b_1}^{t_k}$, which is then divided into two graphs, G_1 and G_2, each containing the states of b_1 and its neighbours at one time instant. Next, G_1 and G_2 are fed into a multi-head GAT layer. Since they have the same transformation process in the multi-head GAT block, we only explain the transformation of G_1 in the following.

The first step is to perform a linear transformation. For each vertex $b_i \in V_{b_1}$, $z_{b_i} = WS_{b_i}^{t_k}[0]$, where W is a weight matrix and $W \in \mathbb{R}^{1024\times 2F}$, and $S_{b_i}^{t_k}[0] \in \mathbb{R}^{2F}$. Attention coefficients between b_1 and its neighbours and its own are then computed,

$$e_{1j} = LeakyReLU(a(z_{b_1}||z_{b_j})) \tag{6.8}$$

where $||$ denotes concatenation, a is a learnable weight vector, and LeakyReLU is an activation function. $b_j \in \mathcal{N}_{b_1} \cup \{b_1\}$, where $\mathcal{N}_{b_1}$ is the neighbour set of b_1. The computed attention scores are then sent to a Softmax layer. Next, b_i and its neighbour's embeddings are aggregated and weighted by the normalized attention score. The updated

representation of b_1 is given by

$$h_{b_1} = \sigma\left(\sum_{b_j \in \mathcal{N}_{b_1} \cup \{b_1\}} \alpha_{1j} z_{b_j}\right) \tag{6.9}$$

where α_{1j} is the normalized attention score between b_1 and b_j. $\mathcal{N}_{b_1}$ is the set of b_1's neighbours and σ is a LeakyReLU. Note that this chapter uses four-head attention, which means there are four different weight matrix W. The multi-head GAT layer uses each W to compute h_{b_1} and concatenates them.

The output of the multi-head GAT block in each temporal sequence is then fed into a GRU layer, which is used to capture temporal features. The hidden dimension of the GRU layer is 512, and its output is fed into a ReLU and a fully connected layer.

Next, the output of the fully connected layer is fed into a Softmax layer if it is the actor network. The actor network outputs a vector $\pi_1(G_{b_1}^{t_k}, A) \in \mathbb{R}^F$, which is the probability distribution over F actions for b_1. For the critic network, there is a ReLU layer following the fully connected layer, and the output is a vector $Q_1(G_{b_1}^{t_k}, A) \in \mathbb{R}^F$, which is the predicted q-value for each action for b_1.

6.3.3 STGAN-SAC

In this chapter, each BS b_i maintains an independent STGAN-SAC agent. At each time instant t_k, an agent i has a spatial-temporal graph as we mentioned in Section 6.3.2,

$$G_{b_i}^{t_k} = \{\{S_j^t\}_{j \in b_i \cup \mathcal{N}_{b_i}, t \in \{t_{k-1}, t_k\}}, Adj_{b_i}\}, \tag{6.10}$$

where $\mathcal{N}_{b_i}$ is the set of b_i's one-hop neighbours. $\{t_{k-1}, t_k\}$ indicates two consecutive time instants. Adj_{b_i} is the adjacency matrix of the graph.

At each time instant, $G_{b_i}^{t_k}$ is fed into the actor network (i.e., a spatial-temporal GAT) shown in Figure 6.2. Then, the agent i takes an action set $A_{b_i}^{t_k}$ based on the output, $\pi_i(G_{b_i}^{t_k}, A)$, which is the probability distribution over all available actions. In the next time instant, the agent receives a reward set $R_{b_i}^{t_k}$ from the environment. Note that both $A_{b_i}^{t_k}$ and $R_{b_i}^{t_k}$ have length N (i.e., BS's cache size), and each reward the agent receives (i.e., $r_{b_i}^{t_k} \in R_{b_i}^{t_k}$) corresponds to an action taken by that agent (i.e., $a_{b_i}^{t_k} \in A_{b_i}^{t_k}$).

The STGAN-SAC agent works by interacting with the environment, shown in Algorithm 5. For an agent i, it initializes a critic network Q_i with parameters ϕ, a target critic network with parameters $\hat{\phi} = \phi$, an actor network π_i with parameters θ and a target actor network with parameters $\hat{\theta} = \theta$. Also, a replay buffer U_i is initialized with a length of 2000. For each episode and each time instant $t_k \in \mathcal{T}$, the agent i feeds a graph $G_{b_i}^{t_k}$ into π_i to make probability predictions over all available actions. The agent then chooses the top N actions [94] with the highest probability. Once an agent performs an action set $A_{b_i}^{t_k}$ at t_k, it will receive a reward set, $R_{b_i}^{t_k}$ and a spatial-temporal graph $G_{b_i}^{t_{k+1}}$ at the next time instant. The transition tuple $(G_{b_i}^{t_k}, A_{b_i}^{t_k}, \mathcal{R}_{b_i}^{t_k}, G_{b_i}^{t_{k+1}})$ is then inserted into U_i.

In order to train the critic Q_i, the agent randomly samples C transitions from U_i at each time instant. Q_i is then updated to minimize the squared soft Bellman residual,

$$J_{Q_i}(\phi) = \mathbb{E}_{(G_{b_i}^{t_k},\ a_i:a_i\in A_{b_i}^{t_k},\ r_i:r_i\in R_{b_i}^{t_k},\ G_{b_i}^{t_{k+1}})\sim U_i}[(Q_i^{\phi}(G_{b_i}^{t_k}, a_i) - y_i)^2], \tag{6.11}$$

where the target is given by,

$$y_i = r_i + \gamma\, \mathbb{E}_{a_i'\sim\pi_i^{\hat{\theta}}(G_{b_i}^{t_{k+1}}, A)}[Q_i^{\hat{\phi}}(G_{b_i}^{t_{k+1}},\ a_i') - \alpha\ log(\pi_i^{\hat{\theta}}(a_i'|G_{b_i}^{t_{k+1}})], \tag{6.12}$$

which is the sum of the immediate reward, the expected discounted return, and the

Algorithm 5 STGAN-SAC Agent i's Operation

Input: number of episodes E, batch size C, target network update step K, number of steps e, discount factor γ

Initialize the critic network Q_i^{ϕ} and the target critic network $Q_i^{\hat{\phi}}$ with weights $\hat{\phi} = \phi$
Initialize the actor network π_i^{θ} and the target actor network $\pi_i^{\hat{\theta}}$ with weights $\hat{\theta} = \theta$
Initialize replay buffer U_i and set e to 0
for *episode* $\in \{1, \ldots, E\}$ **do**
    for $t_k \in \{t_1, \ldots, t_T\}$ **do**
        $e = e + 1$
        Select N actions with the highest probability in $\pi_i^{\theta}(G_{b_i}^{t_k}, A)$
        Take selected action $A_{b_i}^{t_k}$
        Get reward $R_{b_i}^{t_k}$ and a spatial-temporal graph $G_{b_i}^{t_{k+1}}$ at the next time instant
        Store transition $(G_{b_i}^{t_k}, A_{b_i}^{t_k}, \mathcal{R}_{b_i}^{t_k}, G_{b_i}^{t_{k+1}})$ into U_i
        Randomly sample B transitions from U_i
        Use Equation 6.11 to update Q_i^{ϕ}

        Use Equation 6.13 to update π_i^{θ}

        if e mod K == 0 **then**
            Update target network parameters:
            $\hat{\phi} = \tau\hat{\phi} + (1 - \tau)\phi$
            $\hat{\theta} = \tau\hat{\theta} + (1 - \tau)\theta$
        end if
    end for
end for

entropy of the action probability distribution. γ is a discount factor in the range of [0, 1]. α is a temperature parameter determining the balance between expected return and entropy. Note that $G_{b_i}^{t_k}$, a_i, r_i and $G_{b_i}^{t_{k+1}}$ are sampled from U_i, but the action a_i' taken at the $(k+1)^{th}$ time instant is sampled from the current target policy $\pi_i^{\hat{\theta}}$.

The agent's policy network π_i^{θ} is updated to maximize the advantage and the entropy of actions probability distribution,

$$J_{\pi_i}(\theta) = \mathbb{E}_{G_{b_i}^{t_k} \sim U_i,\ a_i \sim \pi_i^{\theta}(G_i^t, A)}[log(\pi_i^{\theta}(a_i|G_{b_i}^{t_k})) \times (-\alpha\, log(\pi_i^{\theta}(a_i|G_{b_i}^{t_k})) + Q_i^{\phi}(G_{b_i}^{t_k}, a_i) - b(G_{b_i}^{t_k}, A))], \tag{6.13}$$

where $Q_i^{\phi}(G_{b_i}^{t_k}, a_i) - b(G_{b_i}^{t_k}, A)$ is the advantage function, and it shows the degree of dominance of the selected action a_i compared to a random action when the state is $G_{b_i}^{t_k}$. $b(G_i^t, A)$ is computed by,

$$b(G_i^{t_k}, A) = \mathbb{E}_{a_i \sim \pi_i^{\theta}(G_{b_i}^{t_k}, A)}[Q_i^{\phi}(G_{b_i}^{t_k}, a_i)], \tag{6.14}$$

which is the baseline, and it calculates the expected return of a random action chosen according to the policy π_i^{θ} at a given state $G_{b_i}^{t_k}$.

Key training parameters for actor and critic networks are shown in Table 6.2.

Table 6.2: Training Parameters for Actor and Critic Networks

Parameters	Values
Number of episodes E	50
Learning rate of actor and critic	0.001
Batch size C	32
Target network update step K	30
Replay buffer capacity	2000
Discount factor γ	0.999
Temperature parameter α	0.01
Target update rate τ	0.005
Optimizer	Adam

6.4 Experimental Results

A comparison of various caching schemes is shown in this section. We simulate all experiments using *Icarus* [32], a Python-based caching simulator. Key experimentation parameters are shown in Table 6.3. We perform 50 experiments for each network scenario and compute evaluation metrics using the last 40 experiments. Each experiment contains 8000 warm-ups and 160,000 measured requests for measuring cache performance. Each request packet has a size of 150 bytes, and each data packet has a size of 1500 bytes. The interval between two consecutive time instants is 10 s. The total number of file chunks is set to $F = 400$, and all BSs have a uniform cache size N, which ranges from 1 to 4 file chunks. We assume users move in groups, and each user group is randomly distributed in the rectangle area shown in Figure 6.1. There are four user groups in total (i.e., $UG = 4$), and users in the same group move in the same direction and speed. The number of file chunks requested by each user group, F_{ug_i}, ranges from 50 to 90 file chunks. Each user group sends requests following a Poisson distribution with a mean of 1 request per second (1 req/s) or 50 req/s, or 100 req/s. The content popularity requested by each user group follows a Zipfian distribution [116] with an α of 0.8, 0.9, 1.0 or 1.1.

Inspired by papers [107, 108], we consider the following heterogeneous services for different BSs: (i) Different user groups have distinct preferences. $\mathcal{F}_{ug_i}$ is randomly sampled from $\mathcal{F}$, where each file has a uniform probability of being sampled. (ii) Different user groups have different activity levels. The requests from ug_1, ug_2, ug_3 and ug_4 follow a Poisson distribution with a mean of 100 req/sec, 1 req/sec, 50 req/sec and 1 req/sec, respectively. (iii) The file popularity of each user group's requests follows a Zipfian distribution [116], denoted as $P(ug_i, f_j) = ((j)^{\alpha} \sum_{m=1}^{F_{ug_i}} (m)^{-\alpha})^{-1}$, where F_{ug_i} is the number of requested file chunks from ug_i and j is the index of file chunk f_j in the set $\mathcal{F}_{ug_i}$. α is a skewness factor, and the α of ug_1, ug_2, ug_3 and ug_3 are 0.8, 0.9, 1.0 and 1.1, respectively.

Table 6.3: Key Experimentation Parameters

Parameters	Values
Number of experimentations	50
Number of warm-up requests	8000
Number of measured requests	160,000
Request packet size	150 bytes
Data packet size	1500 bytes
Time interval	10 s
Number of file chunks F	400
BS cache size N	1-4 file chunks
Number of user groups UG	4
Number of file chunks requested by a user group F_{ug_i}	50-90 file chunks
User group request distribution	A Poisson distribution with a mean of 1 req/s, 50 req/s or 100 req/s
User group requested content popularity distribution	A Zipf distribution with an α of 0.8, 0.9, 1.0 or 1.1

(iv) We simulate dynamic file popularity. Each user group's requested file popularity is shuffled every 300 s in each experiment.

We compare STGAN-SAC with the following benchmark caching schemes:

- **DDRQN** proposed in paper [97]: it is a DDQN-based [31] multi-agent caching policy that makes proactive caching decisions based on q-value predictions, where the q-network is a GRU.
- **DDGARQN**: it is similar to DDRQN, but the q-network is a spatial-temporal GAT.
- **LFU** [138]: it is a caching replacement policy that evicts the least frequently used file chunk.
- **LRU** [137]: it is also a caching replacement strategy, and it evicts the least recently requested file chunk.

DDRQN and DDGARQN are trained using the same parameters. For LFU and LRU, their caching placement algorithm is LCE.

We use the following three metrics to evaluate different caching strategies:

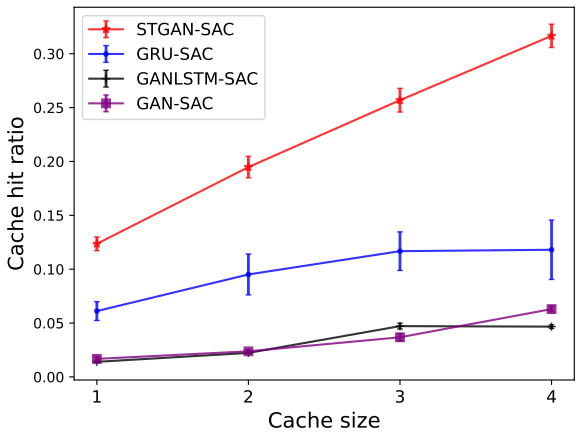
- **Cache hit ratio**: the percentage of requests that can be satisfied by the cached file chunks in BSs. The CHR has been defined in Eqn. 3.4.
- **Average latency**: the average delay between the time a user sends a request packet and the time the user receives a data packet. The formula for ALT has been given in Eqn. 3.6.
- **Average link load**: the average link load between the cloud and all BSs.

6.4.1 Ablation Studies

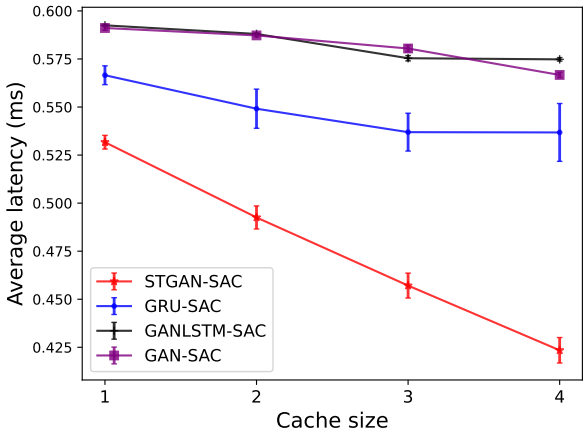
In this section, we perform ablation studies to demonstrate two critical layers of STGAN-SAC. First, we test the importance of the multi-head GAT layer by comparing STGAN-SAC against the GRU-SAC model, which is the STGAN-SAC without the multi-head GAT layer. Secondly, we study the significance of the GRU layer by comparing STGAN-SAC against GANLSTM-SAC and GAN-SAC. GANLSTM-SAC replaces the GRU layer in STGAN-SAC with an LSTM layer, and GAN-SAC is the STGAN-SAC without the GRU layer.

6.4.1.1 Ablation Study of Multi-head GAT

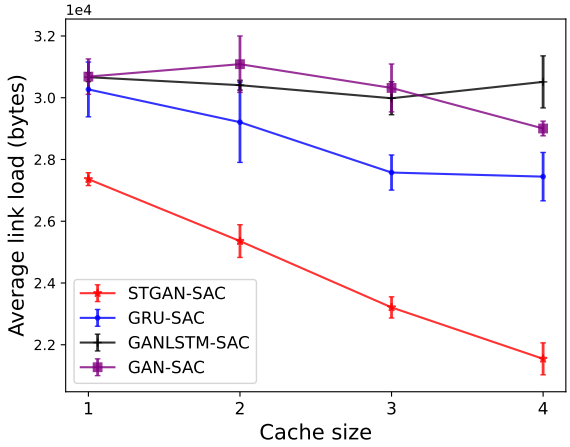
This section studies the impact of the multi-head GAT layer. Figure 6.3 shows the caching performance of STGAN-SAC and GRU-SAC when $F_{ug_i} = 50$ and each BS's cache size is 1 to 4 file chunks. Although the number of parameters of GRU-SAC is less than STGAN-SAC, its performance is much worse than STGAN-SAC. In the best



(a) Cache hit ratio



(b) Average latency



(c) Average link load

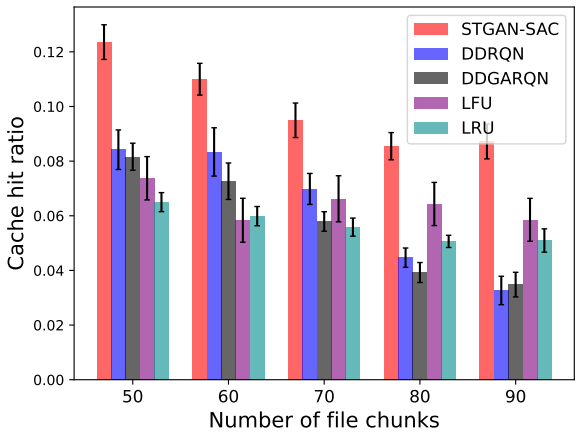
Figure 6.3: The caching performances of STGAN-SAC, GRU-SAC, GANLSTM-SAC and GAN-SAC vary with each BS's cache size. Error bars are a 95% confidence interval across 40 runs.

case, STGAN-SAC improves GRU-SAC by 168% cache hit ratio, 21.1% average latency and 24% average link load, respectively. The significant margin demonstrates that the multi-head GAT layer is essential in extracting spatial features of user requests.

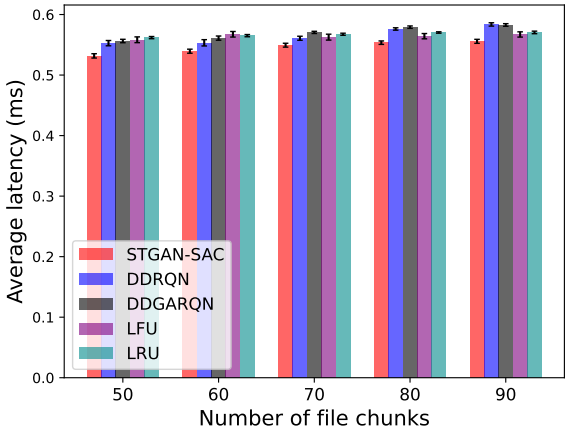
6.4.1.2 Ablation Study of GRU

This section studies the importance of the GRU layer. Figure 6.3 shows the caching performance of STGAN-SAC, GANLSTM-SAC and GAN-SAC when $F_{ug_i} = 50$ and each BS's cache size is 1 to 4 file chunks. STGAN-SAC has the best performance, and GANLSTM-SAC performs much worse than STGAN-SAC. Although GANLSTM-SAC contains the spatial block (i.e., multi-headed GAT) and the temporal block (i.e., LSTM) to extract user requests' features, it performs poorly. The reason is that LSTM is more complex than GRU, resulting in more challenging training and slower convergence of GANLSTM-SAC. In addition, GRU is preferred for small datasets, and it fits our case because we use a batch size of 32. We can also observe that GAN-SAC performs similarly to GANLSTM-SAC in all scenarios. GAN-SAC cannot extract temporal dependencies from user requests, which causes it performs much worse than STGAN-SAC. In short, this section demonstrates that GRU is necessary for our model. With less number of parameters, GRU can train faster than LSTM. Moreover, GRU helps capture user requests' temporal dynamics, dramatically improving the caching performance.

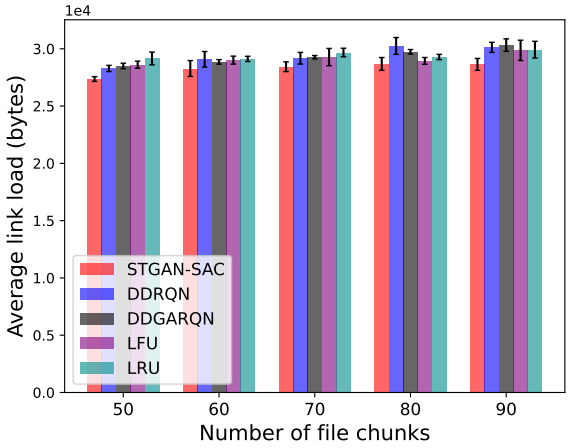
In addition to the muti-head GAT and GRU ablation studies, Figure 6.3 shows that GRU-SAC performs better than GANLSTM-SAC and GAN-SAC. The reason is that GRU-SAC has the least number of parameters, which is easy to train. Also, GRU can capture temporal features of user requests, which is easier to capture than spatial features due to user mobility.



(a) Cache hit ratio



(b) Average latency



(c) Average link load

Figure 6.4: The caching performances of STGANSAC, DDRQN, DDGARQN, LFU and LRU vary with the number of file chunks requested by each user group. Error bars are a 95% confidence interval across 40 runs.

6.4.2 Comparison with Benchmarks

In this section, we compare the performances of STGAN-SAC with other benchmark models in various network scenarios.

6.4.2.1 Effect of Number of File Chunks

This section explores the effect of the number of file chunks requested by each user group, i.e., F_{ug_i}. Each BS has a uniform cache size of 1 file chunk. Figure 6.4 shows caching performances of different strategies with different F_{ug_i}. We can see that STGAN-SAC consistently achieves the best performance regardless of the number of F_{ug_i}.

In the best case, STGAN-SAC outperforms DDGARQN with a 150% higher cache hit ratio, a 4.6% lower average latency and a 5.5% lower average link load. STGAN-SAC outperforms DDGARQN for the following reasons: (i) DDQN, utilized in DDGARQN, suffers from poor convergence, as a slight change in the q-value estimation may result in a substantial change in policy space. On the contrary, AC works in the policy space directly and learns much more smoothly. (ii) SAC helps the agent explore more widely [106] than the ϵ-greedy exploration strategy [44] used in DDGARQN.

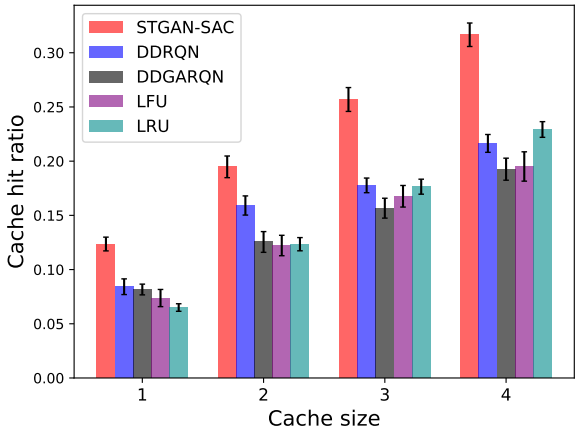
STGAN-SAC also outperforms DDRQN by a significant margin, where the cache hit ratio is improved by 166%, the average latency is lowered by 4.7%, and the average link load is lowered by 5.1% at best. It is also worth noting that the difference between STGAN-SAC and the two deep learning-based strategies, DDRQN and DDGARQN, increases when the number of file chunks increases. STGAN-SAC outperforms DDRQN because, first, SAC wins in smooth convergence and broad exploration. Secondly, the spatial module (i.e., the multi-head GAT layer) in STGAN-SAC helps each BS cooperates, while DDRQN misses this part.

Additionally, DDRQN performs better than DDGARQN in general. The reason is that both of them apply DDQN, which has difficulties in dealing with large discrete action space as it tries to generate the state-action value for each action. In this chapter, the action dimension is 400, which is rather unfriendly to DDQN. On top of that, DDGARQN contains more parameters than DDRQN, which results in slower convergence.

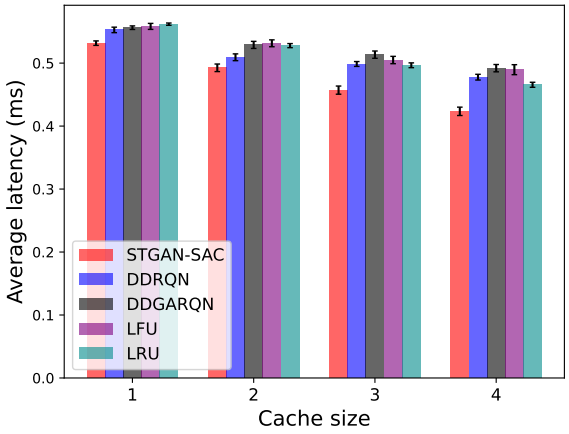
Furthermore, STGAN-SAC improves cache hit ratio by 90%, average latency by 5.3% and average link load by 6.1% compared to LRU at best. In general, STGAN-SAC has a more significant margin than LFU. Interestingly, DDRQN and DDGARQN perform worse than LFU and LRU when F_{ug_i} is higher than 80, which verifies that DDQN does not fit well with large discrete action space. In addition, LRU outperforms LFU in most cases, indicating that short-term file popularity is much more important than long-term file popularity in our experiments.

6.4.2.2 Effect of Cache Size

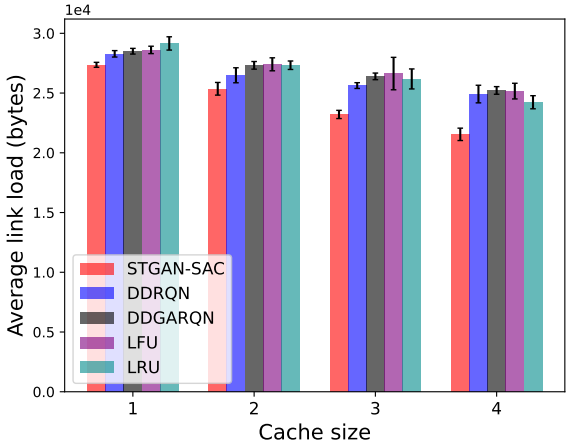
This section demonstrates the impact of each BS's cache size on different caching algorithms. We set $F_{ug_i} = 50$, and the cache size of each BS is 1, 2, 3 and 4 file chunks. Figure 6.5 shows 95% confidence intervals for the cache hit ratio, average latency and average link load for all strategies. Notably, STGAN-SAC has substantial advantages over other strategies. When each BS has a limited cache size of 1 file chunk, STGAN-SAC improves cache hit ratio by 46.7%, 51.4%, 67.5% and 90.1% compared with DDRQN, DDGAQN, LFU and LRU, respectively. Additionally, STGAN-SAC improves average latency by 3.8% and 4.4% and average link load by 3.2% and 3.9% compared to DDRQN and DDGARQN, respectively. STGAN-SAC performs significantly better than other strategies when the cache size increases. In particular, STGAN-SAC outperforms DDRQN, DDGARQN, LFU and LRU by 46.7%, 64.3%, 67.6% and 90% on the cache hit ratio



(a) Cache hit ratio



(b) Average latency



(c) Average link load

Figure 6.5: The caching performances of STGANSAC, DDRQN, DDGARQN, LFU and LRU vary with each BS's cache size. Error bars are a 95% confidence interval across 40 runs.

at best. The results indicate that STGAN-SAC can precisely capture user requests' dynamics and realize cooperative caching between neighbouring BSs.

6.5 Conclusion

We propose STGAN-SAC, an intelligent caching strategy in edge networks. The proposed strategy is fully decentralized, where each BS maintains an independent STGAN-SAC agent to make sequential caching decisions. Most importantly, the actor and critic networks in the STGAN-SAC framework is a spatial-temporal GAT, where the spatial block (i.e., multi-head GAT) is used to extract spatial dependencies among neighbouring BSs, and the temporal block (i.e., GRU) is used to capture temporal dynamics of user requests. In order to simulate realistic edge network traffic, we consider user mobility and assume users have different preferences and activity levels. We conduct ablation studies to demonstrate that the multi-head GAT layer is essential for extracting user requests' spatial dependencies and realizing cooperative caching between neighbouring BSs. Also, we show that the GRU layer is important for capturing user preferences' dynamics over time. The experimental results show that STGAN-SAC performs the best among all benchmark caching strategies.

Chapter 7

Conclusion and Future Work

In this chapter, we provide a comprehensive summary of the main contributions of the thesis and outline potential direcitons for future research. .

7.1 Conclusion

In this thesis, we conduct in-depth research on efficient caching strategies for ICN-based networks:

- Firstly, we introduce a novel GNN-based caching replacement strategy designed for a specific ICN architecture, NDN. Our approach integrates a 1D-CNN to capture content popularity trends over time, coupled with GraphSAGE layers to predict the caching probability for each content item at each network node. Subsequently, caching replacement decisions are made through predicted probabilities across the network. When a network node's cache is at capacity but new content arrives, our strategy prioritizes the eviction of the content with the lowest predicted caching probability to accommodate the new content if its predicted caching probability exceeds that of the evicted content. Extensive analyses are conducted against al-

ternative deep learning-based caching strategies such as 1D-CNN, LSTM-ED, and SAE, as well as classical benchmark strategies including LFU, LRU, and FIFO. Simulation results underscore the superiority of our GNN-based approach, achieving a 50% increase in CHR and a 30% reduction in ALT compared to the three deep learning-based caching strategies in optimal scenarios.

- Secondly, we present GNN-GM, an innovative proactive caching placement strategy that utilizes a cutting-edge GNN model to predict users' ratings of unviewed videos in NDN. The cumulative predicted rating of each video serves as a key indicator, representing its caching potential or "caching gain". GNN-GM is designed to proactively cache videos with the highest caching gain. Comparative analyses are conducted against NMF-based caching, GNN-CPP, and traditional strategies. Leveraging real-world datasets and diverse network topologies, we evaluate the performance of our approach. The experimental findings demonstrate a 25% increase in CHR, a 5% reduction in ALT, and a 7% decrease in server load compared to GNN-CPP in the GEANT network topology.

- Thirdly, we introduce GNN-DDQN, a GNN-based DRL agent designed to make sequential proactive caching placement decisions within the SDN-ICN context. Initially, we present a statistical model aimed at generating users' content preferences, incorporating key characteristics observed in real-world user traffic patterns. Subsequently, we present the GNN-DDQN agent, which operates within the SDN controller and has a comprehensive view of the entire network. At each time step, leveraging a single forward pass of a GNN, it can efficiently make proactive caching decisions for all network nodes. Through extensive simulations, we validate the superior performance of the proposed caching strategy, achieving a CHR that surpasses the state-of-the-art strategy by up to 34.42%.

- Fourthly, we introduce STGAN-SAC, a fully decentralized proactive caching strategy implemented in a three-tier ICN-based edge network. In this setup, each BS has caching capabilities, with each maintaining an STGAN-SAC agent. Each agent incorporates a spatial-temporal GAT to capture user preferences within its corresponding BS coverage area and the spatial dependencies between neighbouring BSs. At each time step, STGAN-SAC makes caching decisions. Additionally, we integrate user mobility and preference dynamics to simulate the dynamic changes in traffic patterns. Experimental results demonstrate the superiority of STGAN-SAC, exhibiting significant enhancements in CHR, ALT reduction, and mitigation of link load between BSs and the cloud, with improvements of at least 22.4%, 2.1%, and 2.1%, respectively over two state-of-the-art caching strategies, DDRQN and DDGARQN.

In summary, integrating the caching capabilities inherent in ICN-based networks with the spatial learning capabilities of GNNs advances network performance to the next level.

7.2 Future Work

Caching stands as a compelling avenue for exploration with promising prospects for the future. Below are several directions that warrant attention:

1. Enhancing DRL-based Caching Strategies: DRL demonstrates significant potential in addressing caching challenges. However, the DRL-based caching strategies proposed in this dissertation encounter two primary limitations. Firstly, scalability concerns arise as the state and action space increases linearly with the growing number of content items. Secondly, neural network reconstruction and retraining become necessary upon the emergence of new content.

The following are two potential directions to tackle these challenges:

(a) Reactive caching strategy: By considering the state space as the caching state of individual agent and the feature representation of the received content request, and utilizing a binary action space to guide caching decisions, this approach provides scalability and network adaption benefits. However, there may be a trade-off in performance compared to proactive caching strategies.

(b) Integration with recommender systems: By leveraging user preferences, proactive caching of content recommended to a large number of users becomes feasible. Instead of mainly relying on past request frequencies for caching decisions, incorporating content features and user profiles allows for the prediction of future content popularity. Consequently, proactively caching these predicted popular contents can potentially enhance network performance. However, the issue of user information privacy may emerge in this context. Federated learning, an emerging paradigm, offers promise in addressing this concern.

2. Expansion with heterogeneous GNNs: While the current work primarily utilizes GNNs to address caching challenges, it focuses solely on homogeneous GNNs. Exploring heterogeneous GNNs is an intriguing direction for future expansion. Heterogeneous GNNs are applicable in graphs with diverse node and edge types, where a single node or edge feature vector might not sufficiently capture all node or edge features of the whole network due to differences in dimensionality or component meanings. Therefore, utilizing heterogeneous GNNs in networks with different types of nodes and edges to address caching challenges could be a promising area.

3. Caching in IoVs: With the advancements in autonomous driving, users are increasingly inclined towards various entertainment options, including video streaming

and gaming, while on the move. The integration of caching plays an important role in significantly enhancing network performance within vehicular environments. Although the proposed STGAN-SAC caching strategy in this dissertation addresses user mobility to some extent, its scope remains limited, primarily focusing on localized user movement without accounting for handovers between RSUs or broader vehicular mobility patterns. Future endeavours in IoV-based caching demand comprehensive frameworks that consider vehicular mobility dynamics, ensuring seamless content delivery.

4. Collaboration with cache-enabled aerial BSs: Caching faces significant challenges in scenarios involving mobile users or vehicles. With the rise of cache-enabled aerial BSs, also known as UAV-mounted BSs, exploring collaborative caching strategies could be beneficial. By incorporating cache storage within each of aerial BSs, they could dynamically adjust their positions to cater to areas experiencing high user request densities. This dynamic nature offers advantages over static BSs. Investigating collaborative caching approaches with aerial BSs could provide a promising avenue for further improving caching efficiency in wireless network environments.

5. Integration with semantic communication: The rise of Large Language Models (LLMs) has driven semantic communication into the spotlight. This innovative communication paradigm extracts meaningful information from the source to optimize information transmission throughput in wireless networks, surpassing theoretical capacity limits. However, existing caching strategies primarily focus on raw content caching, neglecting potential synergies with semantic communication paradigms. Future research endeavours could delve into integrating caching mechanisms with semantic communication frameworks, potentially improving content delivery efficiency further.

References

[1] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, 2012.

[2] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[3] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[4] Wolfgang Braun and Michael Menth. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336, 2014.

[5] 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, 2008.

[6] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu,

Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[7] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[8] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[10] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[11] 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*, 2024.

[12] Bingjun Li and Sheida Nabavi. A multimodal graph neural network framework for cancer molecular subtype classification. *BMC bioinformatics*, 25(1):27, 2024.

[13] Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, and Zachary Patterson. Graph neural networks for intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[14] 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, 2017.

[15] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent q-network. *arXiv preprint arXiv:1512.01693*, 2015.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[17] 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(3):1755–1770, 2023.

[18] 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*, 2023.

[19] Gitnux. Internet traffic statistics and trends in 2023 • gitnux. https://blog.gitnux.com/internet-traffic-statistics/, 2023. Accessed on 10 Jun 2023.

[20] Zhun Ge, Jiacheng Hou, and Amiya Nayak. Gnn-based end-to-end delay prediction in software defined networking. In *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 372–378. IEEE, 2022.

[21] Zhun Ge, Jiacheng Hou, and Amiya Nayak. Forecasting sdn end-to-end latency using graph neural network. In *2023 International Conference on Information Networking (ICOIN)*, pages 293–298. IEEE, 2023.

[22] Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet-fermi: Network modeling with graph neural networks. *IEEE/ACM Transactions on Networking*, 2023.

[23] Fuxian Li, Jie Feng, Huan Yan, Guangyin Jin, Fan Yang, Funing Sun, Depeng Jin, and Yong Li. Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution. *ACM Transactions on Knowledge Discovery from Data*, 17(1):1–21, 2023.

[24] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[27] named-data/mini-ndn. https://github.com/named-data/mini-ndn, 2020. Accessed on 10 Mar 2021.

[28] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058*, 2019.

[29] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[30] GÉant topology map. https://eapconnect.eu/wp-content/uploads/GEANT_Topology_Map_December_2018.pdf, 2018. Accessed on 20 Sep 2021.

[31] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[32] Lorenzo Saino, Ioannis Psaras, and George Pavlou. Icarus: a caching simulator for information centric networking (icn). In *SimuTools*, volume 7, pages 66–75. ICST, 2014.

[33] Géant homepage. https://geant3plus.archive.geant.net/Pages/home.html, 2020. Accessed on 18 August 2022.

[34] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[35] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[36] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[37] Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, page 117921, 2022.

[38] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B Wiltschko. A gentle introduction to graph neural networks. *Distill*, 6(9):e33, 2021.

[39] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[40] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

[41] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[42] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. *Graph neural networks*. Springer, 2022.

[43] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[45] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[46] Amjad Iqbal, Mau-Luen Tham, and Yoong Choon Chang. Convolutional neu-

ral network-based deep q-network (cnn-dqn) resource management in cloud radio access network. *China Communications*, 19(10):129–142, 2022.

[47] Manthena Narasimha Raju, Kumaran Natarajan, and Chandra Sekhar Vasamsetty. Remote sensing image data classification using cnn-deep q model. In *AIP Conference Proceedings*, volume 2724. AIP Publishing, 2023.

[48] 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. *Knowledge-Based Systems*, 284:111290, 2024.

[49] Yanli Xing. Work scheduling in cloud network based on deep q-lstm models for efficient resource utilization. *Journal of Grid Computing*, 22(1):1–20, 2024.

[50] Dezhi Chen, Qi Qi, Qianlong Fu, Jingyu Wang, Jianxin Liao, and Zhu Han. Transformer-based reinforcement learning for scalable multi-uav area coverage. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[51] Getu Fellek, Ahmed Farid, Goytom Gebreyesus, Shigeru Fujimura, and Osamu Yoshie. Graph transformer with reinforcement learning for vehicle routing problem. *IEEJ Transactions on Electrical and Electronic Engineering*, 18(5):701–713, 2023.

[52] 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. *Information Sciences*, 578:401–416, 2021.

[53] 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 networks*, 139:265–277, 2021.

[54] Sikai Chen, Jiqian Dong, Paul Ha, Yujie Li, and Samuel Labi. Graph neural network and reinforcement learning for multi-agent cooperative control of connected autonomous vehicles. *Computer-Aided Civil and Infrastructure Engineering*, 36(7):838–857, 2021.

[55] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *arXiv preprint arXiv:1910.07421*, 2019.

[56] Xuanhong Zhou, Muhammad Bilal, Ruihan Dou, Joel JPC Rodrigues, Qingzhan Zhao, Jianguo Dai, and Xiaolong Xu. Edge computation offloading with content caching in 6g-enabled iov. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[57] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D Thornton, Diana K Smetters, Beichuan Zhang, Gene Tsudik, Dan Massey, Christos Papadopoulos, et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 157:158, 2010.

[58] Suyong Eum, Kiyohide Nakauchi, Masayuki Murata, Yozo Shoji, and Nozomu Nishinaga. Catt: potential based routing with content caching for icn. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 49–54, 2012.

[59] Ioannis Psaras, Wei Koong Chai, and George Pavlou. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 55–60, 2012.

[60] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. The lcd interconnection of lru caches and its analysis. *Performance Evaluation*, 63(7):609–634, 2006.

[61] Wei Koong Chai, Diliang He, Ioannis Psaras, and George Pavlou. Cache "less for more" in information-centric networks. In *International conference on research in networking*, pages 27–40. Springer, 2012.

[62] Samar Shailendra, Senthilmurugan Sengottuvelan, Hemant Kumar Rath, Bighnaraj Panigrahi, and Anantha Simha. Performance evaluation of caching policies in ndn-an icn architecture. In *2016 IEEE Region 10 Conference (TENCON)*, pages 1117–1121. IEEE, 2016.

[63] Zhe Li, Gwendal Simon, and Annie Gravey. Caching policies for in-network caching. In *2012 21st International conference on computer communications and networks (ICCCN)*, pages 1–7. IEEE, 2012.

[64] César Bernardini, Thomas Silverston, and Olivier Festor. Mpc: Popularity-based caching strategy for content centric networks. In *IEEE International Conference on Communications (ICC)*, pages 3619–3623, 2013.

[65] Meiju Yu, Ru Li, Yingqi Liu, and Yingqi Li. A caching strategy based on content popularity and router level for ndn. In *2017 7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 195–198. IEEE, 2017.

[66] Yahui Meng, Muhammad Ali Naeem, Rashid Ali, and Byung-Seo Kim. Ehcp: An efficient hybrid content placement strategy in named data network caching. *IEEE Access*, 7:155601–155611, 2019.

[67] Muhammad Ali Naeem, Shahrudin Awang Nor, Suhaidi Hassan, and Byung-Seo Kim. Compound popular content caching strategy in named data networking. *Electronics*, 8(7):771, 2019.

[68] Yiqi Gui and Yongkang Chen. A cache placement strategy based on compound popularity in named data networking. *IEEE Access*, 8:196002–196012, 2020.

[69] Oussama Serhane, Khadidja Yahyaoui, Boubakr Nour, and Hassine Moungla. Cns: A cache and split scheme for 5g-enabled icn networks. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[70] Boubakr Nour, Hakima Khelifi, Hassine Moungla, Rasheed Hussain, and Nadra Guizani. A distributed cache placement scheme for large-scale information-centric networking. *IEEE Network*, 34(6):126–132, 2020.

[71] Sumit Kumar and Rajeev Tiwari. Dynamic popularity window and distance-based efficient caching for fast content delivery applications in ccn. *Engineering Science and Technology, an International Journal*, 2021.

[72] Muhammad Faran Majeed, Matthew N Dailey, Riaz Khan, and Apinun Tunpan. Pre-caching: A proactive scheme for caching video traffic in named data mesh networks. *Journal of Network and Computer Applications*, 87:116–130, 2017.

[73] Shatarupa Dash, Suvendu Kumar Dash, and Bharat JR Sahu. Proactive content caching for streaming over information-centric network. In *Intelligent and cloud computing*, pages 165–172. 2021.

[74] Salahadin Seid Musa, Marco Zennaro, Mulugeta Libsie, and Ermanno Pietrosemoli. Mobility-aware proactive edge caching optimization scheme in information-centric iov networks. *Sensors*, 22(4):1387, 2022.

[75] Bighnaraj Panigrahi, Samar Shailendra, Hemant Kumar Rath, and Anantha Simha. Universal caching model and markov-based cache analysis for information centric networks. In *2014 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pages 1–6, 2014.

[76] Mau Dung Ong, Min Chen, Tarik Taleb, Xiaofei Wang, and Victor CM Leung. Fgpc: Fine-grained popularity-based caching design for content centric networking. In *Proceedings of the 17th ACM int. conference on modeling, analysis and simulation of wireless and mobile systems*, pages 295–302, 2014.

[77] 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. IEEE, 2021.

[78] 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*, 2023.

[79] Kyi Thar, Nguyen H Tran, Thant Zin Oo, and Choong Seon Hong. Deepmec: Mobile edge caching using deep learning. *IEEE Access*, 6:78260–78275, 2018.

[80] Nishat Tasnim Niloy and Md Shariful Islam. Intellcache: An intelligent web caching scheme for multimedia contents. In *2020 Joint 9th International Conference on Informatics, Electronics & Vision (ICIEV) and 2020 4th International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pages 1–6. IEEE, 2020.

[81] Zhe Zhang, Chung-Horng Lung, Marc St-Hilaire, and Ioannis Lambadaris. Smart

caching: Empower the video delivery for 5g-icn networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[82] Zhe Zhang, Chung-Horng Lung, Marc St-Hilaire, and Ioannis Lambadaris. Smart proactive caching: Empower the video delivery for autonomous vehicles in icn-based networks. *IEEE Transactions on Vehicular Technology*, 69(7):7955–7965, 2020.

[83] Divya Gupta, Shalli Rani, Syed Hassan Ahmed, Sahil Garg, Md Jalil Piran, and Mubarak Alrashoud. Icn-based enhanced cooperative caching for multimedia streaming in resource constrained vehicular environment. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4588–4600, 2021.

[84] Divya Gupta, Shalli Rani, Syed Hassan Ahmed, Sahil Verma, Muhammad Fazal Ijaz, and Jana Shafi. Edge caching based on collaborative filtering for heterogeneous icn-iot applications. *Sensors*, 21(16):5491, 2021.

[85] Sai Munikoti, Deepesh Agarwal, Laya Das, Mahantesh Halappanavar, and Balasubramaniam Natarajan. Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications. *arXiv preprint arXiv:2206.07922*, 2022.

[86] Nikolaos Nomikos, Spyros Zoupanos, Themistoklis Charalambous, and Ioannis Krikidis. A survey on reinforcement learning-aided caching in heterogeneous mobile edge networks. *IEEE Access*, 2022.

[87] Siyuan Sun, Junhua Zhou, Jiuxing Wen, Yifei Wei, and Xiaojun Wang. A dqn-based cache strategy for mobile edge networks. *Computers, Materials & Continua*, 71(2):3277–3291, 2022.

[88] Lei Zhao, Yongyi Ran, Hao Wang, Junxia Wang, and Jiangtao Luo. Towards cooperative caching for vehicular networks with multi-level federated reinforcement learning. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.

[89] Chunhe Song, Wenxiang Xu, Tingting Wu, Shimao Yu, Peng Zeng, and Ning Zhang. Qoe-driven edge caching in vehicle networks based on deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 70(6):5286–5295, 2021.

[90] Jingsong Li, Junhua Tang, Jianhua Li, and Futai Zou. Deep reinforcement learning for intelligent computing and content edge service in icn-based iov. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–7. IEEE, 2021.

[91] Peng He, Li Cao, Yaping Cui, Ruyan Wang, and Dapeng Wu. Multi-agent caching strategy for spatial-temporal popularity in iov. *IEEE Transactions on Vehicular Technology*, 2023.

[92] Wei Jiang, Daquan Feng, Yao Sun, Gang Feng, Zhenzhong Wang, and Xiang-Gen Xia. Proactive content caching based on actor-critic reinforcement learning for mobile edge networks. *IEEE Transactions on Cognitive Communications and Networking*, 2021.

[93] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[94] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun. Intelligent video caching at network edge: A multi-agent deep reinforcement learning

approach. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2499–2508. IEEE, 2020.

[95] Zheqi Zhu, Shuo Wan, Pingyi Fan, and Khaled B Letaief. Federated multiagent actor–critic learning for age sensitive mobile-edge computing. *IEEE Internet of Things Journal*, 9(2):1053–1067, 2021.

[96] Yifei Wei, F Richard Yu, Mei Song, and Zhu Han. Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor–critic deep reinforcement learning. *IEEE Internet of Things Journal*, 6(2):2061–2073, 2018.

[97] Haitao Xu, Yuejun Sun, Jingnan Gao, and Jianbo Guo. Intelligent edge content caching: A deep recurrent reinforcement learning method. *Peer-to-Peer Networking and Applications*, pages 1–14, 2022.

[98] Dan Zhang, Zehua Ye, Gang Feng, and Hongyi Li. Intelligent event-based fuzzy dynamic positioning control of nonlinear unmanned marine vehicles under dos attack. *IEEE Transactions on Cybernetics*, 2021.

[99] Wai-Xi Liu, Jie Zhang, Zhong-Wei Liang, Ling-Xi Peng, and Jun Cai. Content popularity prediction and caching for icn: A deep learning approach with sdn. *IEEE access*, 6:5075–5089, 2017.

[100] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*, pages 48–53, 2018.

[101] Youngbin Im, Prasanth Prahladan, Tae Hwan Kim, Yong Geun Hong, and Sangtae Ha. Snn-cache: A practical machine learning-based caching system utilizing the

inter-relationships of requests. In *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6. IEEE, 2018.

[102] Kelvin HT Chiu, Jun Zhang, and Brahim Bensaou. Cache management in information-centric networks using convolutional neural network. In *IEEE Global Communications Conference*, pages 1–6, 2020.

[103] Yuming Zhang, Bohao Feng, Wei Quan, Aleteng Tian, Keshav Sood, Youfang Lin, and Hongke Zhang. Cooperative edge caching: A multi-agent deep learning based approach. *IEEE Access*, 8:133212–133224, 2020.

[104] Tongyu Zong, Chen Li, Yuanyuan Lei, Guangyu Li, Houwei Cao, and Yong Liu. Cocktail edge caching: Ride dynamic trends of content popularity with ensemble learning. *IEEE/ACM Transactions on Networking*, 2022.

[105] Xiongwei Wu, Xiuhua Li, Jun Li, PC Ching, Victor CM Leung, and H Vincent Poor. Caching transient content for iot sensing: Multi-agent soft actor-critic. *IEEE Transactions on Communications*, 69(9):5886–5901, 2021.

[106] 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.

[107] Chen Zhong, M Cenk Gursoy, and Senem Velipasalar. Deep reinforcement learning-based edge caching in wireless networks. *IEEE Transactions on Cognitive Communications and Networking*, 6(1):48–61, 2020.

[108] Yiwei Zhao, Ruibin Li, Chenyang Wang, Xiaofei Wang, and Victor CM Leung. Neighboring-aware caching in heterogeneous edge networks by actor-attention-

critic learning. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.

[109] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*, pages 2961–2970. PMLR, 2019.

[110] Yan Yan, Baoxian Zhang, Cheng Li, and Changqing Su. Cooperative caching and fetching in d2d communications-a fully decentralized multi-agent reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 69(12):16095–16109, 2020.

[111] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

[112] Sadman Araf, Adittya Soukarjya Saha, Sadia Hamid Kazi, Nguyen H Tran, and Md Golam Rabiul Alam. Uav assisted cooperative caching on network edge using multi-agent actor-critic reinforcement learning. *IEEE Transactions on Vehicular Technology*, 2022.

[113] Esa Hyytiä, Henri Koskinen, Pasi Lassila, A Penttinen, J Roszik, and J Virtamo. Random waypoint model in wireless networks. *Networks and algorithms: Complexity in physics and computer science*, pages 16–19, 2005.

[114] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[115] named-data/ndn-traffic-generator. https://github.com/named-data/ndn-traffic-generator, 2020. Accessed on 10 May 2021.

[116] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *IEEE INFOCOM*, pages 126–134, 1999.

[117] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[118] David M Allen. Mean square error of prediction as a criterion for selecting variables. *Technometrics*, 13(3):469–475, 1971.

[119] Hoon-Geun Song, Seong Ho Chae, Won-Yong Shin, and Sang-Woon Jeon. Predictive caching via learning temporal distribution of content requests. *IEEE Communications Letters*, 23(12):2335–2339, 2019.

[120] Saidur Rahman, Md Golam Rabiul Alam, and Md Mahbubur Rahman. Deep learning-based predictive caching in the edge of a network. In *2020 International Conference on Information Networking (ICOIN)*, pages 797–801. IEEE, 2020.

[121] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2):1273–1284, 2014.

[122] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[123] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[124] 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.

[125] Muhammad Ali Naeem, Tu N Nguyen, Rashid Ali, Korhan Cengiz, Yahui Meng, and Tahir Khurshaid. Hybrid cache management in iot-based named data networking. *IEEE Internet of Things Journal*, 2021.

[126] Dapeng Man, Yao Wang, Hanbo Wang, Jiafei Guo, Jiguang Lv, Shichang Xuan, and Wu Yang. Information-centric networking cache placement method based on cache node status and location. *Wireless Communications and Mobile Computing*, 2021, 2021.

[127] Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case. *Computer Communications*, 196:184–194, 2022.

[128] Shaohua Fan, Xiao Wang, Chuan Shi, Peng Cui, and Bai Wang. Generalizing graph neural networks on out-of-distribution graphs. *arXiv preprint arXiv:2111.10657*, 2021.

[129] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet: Leveraging graph neural networks for network modeling and optimization in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.

[130] José Suárez-Varela, Sergi Carol-Bosch, Krzysztof Rusek, Paul Almasan, Marta Arias, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Challenging the generaliza-

tion capabilities of graph neural networks for network modeling. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pages 114–115, 2019.

[131] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[132] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Fourteenth ACM conference on recommender systems*, pages 240–248, 2020.

[133] Binqiang Chen and Chenyang Yang. Caching policy optimization for d2d communications by learning user preference. In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pages 1–6. IEEE, 2017.

[134] Binqiang Chen and Chenyang Yang. Caching policy for cache-enabled d2d communications by learning user preference. *IEEE Transactions on Communications*, 66(12):6586–6601, 2018.

[135] Bo Zhang, TS Eugene Ng, Animesh Nandi, Rudolf Riedi, Peter Druschel, and Guohui Wang. Measurement based analysis, modeling, and synthesis of the internet delay space. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 85–98, 2006.

[136] Petroc Taylor. Number of smartphone mobile network subscriptions worldwide from 2016 to 2022, with forecasts from 2023 to 2028. 2023.

[137] Asit Dan and Don Towsley. An approximate analysis of the lru and fifo buffer replacement schemes. In *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 143–152, 1990.

[138] John Dilley and Martin Arlitt. Improving proxy cache performance: Analysis of three replacement policies. *IEEE Internet Computing*, 3(6):44–50, 1999.

[139] Jiacheng Hou, Huanzhang Xia, Haoye Lu, and Amiya Nayak. A graph neural network approach for caching performance optimization in ndn networks. *IEEE Access*, 2022.

[140] 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. IEEE, 2022.

[141] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165, 2019.

[142] Min Zhang, Yanxiang Jiang, Fu-Chun Zheng, Mehdi Bennis, and Xiaohu You. Cooperative edge caching via federated deep reinforcement learning in fog-rans. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2021.

[143] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2):895–943, 2022.

[144] Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. Stein variational policy gradient. *arXiv preprint arXiv:1704.02399*, 2017.

[145] Chenhan Zhang, JQ James, and Yi Liu. Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting. *IEEE Access*, 7:166246–166256, 2019.

[146] Yanan Wang, Tong Xu, Xin Niu, Chang Tan, Enhong Chen, and Hui Xiong. Stmarl: A spatio-temporal multi-agent reinforcement learning approach for cooperative traffic light control. *IEEE Transactions on Mobile Computing*, 2020.

[147] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

[148] Sunitha Safavat, Naveen Naik Sapavath, and Danda B Rawat. Recent advances in mobile edge computing and content caching. *Digital Communications and Networks*, 6(2):189–194, 2020.

[149] Rui Wang, Xi Peng, Jun Zhang, and Khaled B Letaief. Mobility-aware caching for content-centric wireless networks: Modeling and methodology. *IEEE Communications Magazine*, 54(8):77–83, 2016.

[150] Yih-Chun Hu and David B Johnson. Caching strategies in on-demand routing protocols for wireless ad hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 231–242, 2000.

[151] Viswanath Tolety. Load reduction in ad hoc networks using mobile servers. *1990-1999-Mines Theses & Dissertations*, 1999.