A GNN-based Proactive Caching Strategy in NDN Networks

Jiacheng Hou (jhou013@uottawa.ca)
University of Ottawa
Haoye Lu
University of Waterloo
Amiya Nayak
University of Ottawa

Research Article

Keywords: named data networking, deep learning, predict rating, caching decision, content placement, proactive caching

Posted Date: June 8th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1713271/v1

A GNN-based Proactive Caching Strategy in NDN Networks

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

[1]School of Electrical Engineering and Computer Science, University of Ottawa, Canada.
[2]David R. Cheriton School of Computer Science, University of Waterloo, Canada.

*Corresponding author(s). E-mail(s): jhou013@uottawa.ca; Contributing authors: haoye.lu@uwaterloo.ca; nayak@uottawa.ca;

Abstract

As people spend more time watching and sharing videos online, it is critical to provide users with a satisfactory quality of experience (QoE). Leveraging the in-network caching and named-based routing features in Named Data Networks (NDNs), our paper aims to improve user experience through caching. We propose a graph neural network-gain maximization (GNN-GM) cache placement algorithm. First, we use a GNN model to predict users' ratings of unviewed videos. Second, we consider the total predicted rating of a video as the gain of the cached video. Third, we propose a cache placement algorithm to maximize the caching gain and actively cache videos. Cache replacement is implemented based on the cache gain ranking of videos, with higher cache gain videos replacing lower cache gain videos. We compare GNN-GM with two state-of-the-art caching strategies, namely the NMF-based caching strategy and GNN-CPP. GNN-GM is also compared with two traditional caching strategies, LCE and LRU, LCE and FIFO. We evaluate the five caching strategies using real-world datasets in a tree network topology, a real-world network topology GEANT, and various random topologies. The experimental results show that our caching policy significantly improves cache hit ratio, latency and server load. Notably, GNN-GM achieves a 25% higher cache hit rate, 5% lower latency and 7% lower server load than GNN-CPP in GEANT.

Keywords: named data networking, deep learning, predict rating, caching decision, content placement, proactive caching

1 Introduction

With name-based routing and in-network caching, NDN [1] offers many advantages. Unlike IP packets which require host addresses to forward packets to specific locations, NDN packets are associated with a unique content name identifier, and all nodes have caching ability in NDN. Users send an *Interest* packet, and any node receiving the interest packet and having the content can reply with a *Data* packet.

Suppose a network has a video provider far away from the user. In this case, the IP network requires the user to get the video from that content provider, which causes a significant delay. However, NDN's name-based routing and in-network caching capabilities enable the NDN node to cache the video requested by the user. Then, if the user-requested video is cached at a node near the user, the video can be sent back to the user as soon as the node receives the user's request. It can undoubtedly improve the user experience and reduce the traffic load of the whole network.

However, due to the limitation of node cache size, the best we can do is to cache only popular content. Recently, several papers [2–5] 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. They all aim to make caching decisions based on the expected number of content requests without considering user preferences. However, user preference is an essential factor in caching because it reflects user request patterns and can be used to predict what content users will be interested in in the future.

The authors of the paper [6] predicted users' future demand through user preferences. They adopted the Non-Negative Matrix Factorization (NMF) [7] technique in the recommender system 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) [8] 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 paper 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.

A preliminary version of this work [9] was presented at the 2022 IEEE ICC Workshop on Research Advancements in Future Networking Technologies (RAFNET). The rest of this paper is organized as follows. Section II overviews related work. Section III presents our proposed caching strategy. Section IV presents the experimental results. Section V concludes the paper.

2 Related work

In NDN, caching can generally be divided into two main categories: reactive caching and proactive caching. In reactive caching, content is cached only as it passes through the node. Unlike reactive caching, proactive caching actively caches content at the node. If a node caches the requested content in advance, it can immediately satisfy the interest without forwarding it to the server, even if the content has never been requested before. This section reviews various reactive and proactive caching strategies. In addition, papers that make caching decisions based on user preferences are presented.

2.1 Reactive Caching

A traditional cache placement algorithm: leave copy everywhere (LCE)[1] aims to cache packets as soon as they pass through the node. However, a significant disadvantage is that it reduces cache diversity. Another method, Leave copy down (LCD) [10] is to cache the content in the immediate neighbourhood of the original producer. However, since the cached content is only one hop away from the producer, it is still non-optimal. On the other hand, cache replacement is essential to evict the undesired content and make room for more popular content. The traditional methods of cache replacement are least recently used (LRU), least frequently used (LFU), and first-in-first-out (FIFO), [11, 12]. LRU works by discarding the least recently accessed content, while LFU discards the least frequently used content first. The caching method FIFO is a less efficient caching strategy compared to LFU and LRU. It discards the oldest content when there is no cache space available, regardless of the popularity of the content. In our paper, we compare the performance of our scheme with LCE and LRU, LCE and FIFO schemes.

Recently, paper [13] proposed a cache placement and replacement strategy named CnS in 5G-enabled Information-Centric Networking (ICN) networks. The authors performed cache placement as two steps: (i) calculating the content popularity (ii) based on the popularity, determining whether the content needs to be cached. If so, the content will be cached locally or pushed down towards the edge nodes. Once the nodes' cache store is full, the cache replacement policy will be executed based on content popularities. Paper [14] proposed a Push Down popular, Push Up less-popular (PDPU) cache placement strategy in ICN. It aims to push popular content to the edge nodes while pushing less popular content to the core network. The authors also developed a one-hop cache notification to notify neighbouring nodes of their cached contents. Besides, the cache replacement was done through the content popularity. However, the two papers do not apply a powerful tool - deep learning - to predict the popularity of content. Moreover, they utilize the reactive caching strategy that is less powerful than proactive because interest packets can only be satisfied if the content was requested before.

The authors in paper [15] proposed a deep Q-learning caching algorithm in the ICN-based intelligent Internet of Vehicles (IoV) scenario. They focused on providing integrated computing and caching services at the edge server. The deep Q-learning-based algorithm was used to predict the popularity of service requests of vehicles, and joint computing and caching decision were made on edge nodes. Paper [16] utilized multi-level federated Reinforcement Learning (named CoCaRL) to cache contents in vehicular networks. The reinforcement learning (RL) is used to optimize cooperative caching, and the federated learning is used to reduce the communication and computation latency as well as cost. Paper [17] proposed a deep reinforcement learning (DRL) caching strategy to realize the QoE-driven roadside units (RSUs) caching update strategy in IoV. The QoE-driven RSUs caching model was established based on their innovative user interest model.

2.2 Proactive Caching

Recently, paper [18] proposed a proactive caching strategy based on the popularity and chunks of large content objects. They considered the caching problem as an optimization problem. They aimed to minimize the number of forwarding nodes and the number of content replications from consumers to cache nodes while obeying the cache capacity constraint. Paper [19] described their proactive caching strategy as an Ant Colony Process. They demonstrated that their approach could put contents close to the user and reduce access latency. Paper [20] proposed a mobility-aware proactive caching algorithm in ICN-IoV networks. They modelled the mobility of vehicles and their connection with RSUs using a Markov Model. These papers did not apply a deep learning strategy to predict the popularity of content.

The authors in [2] proposed a DeepMEC strategy that applies deep learning to predict future requests count (popularity) of contents and proactively cache contents with high popularity scores. They utilized various types of deep learning models, Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and Convolutional Recurrent Neural Networks (CRNN), to predict future content requests and then compare their performance. The authors in [5] proposed a technique, named IntellCache, to increase the caching efficiency by predicting future content popularity using deep learning models, multilayer perceptron (MLP), Long short-term memory (LSTM), and a combination of LSTM and CNN. The authors of [21] proposed a proactive sequence-aware content caching strategy (PSAC), which is based on a convolutional neural network and an attention mechanism to make caching decisions.

2.3 User Preferences-based Content Caching

Some papers make caching decisions by predicting user preferences. The authors of [22] proposed a proactive caching method in the 5G-ICN scenario. They applied NMF [7] to predict user ratings of movies that have not been watched. Furthermore, they considered content's historical popularity and achieved better performance than traditional reactive caching strategies. Following this, they combined the caching approach with autonomous vehicle (AV) user mobility predictions in highway scenario [6]. Paper [23] proposed a cooperative caching scheme that jointly considers caching locations, content popularities, and predicted future content ratings to make caching decisions in ICN-based vehicle networks. They also use the NMF technique to predict content ratings in the future. Paper [24] applied the collaborative filtering-based caching strategy to optimize edge caching in ICN-Internet of Things (IoT) architecture. They divided the cache space of each edge node into two halves, where the first half is to cache contents based on the content's local popularity, and another half is to cache contents based on the highest possible content to be requested in the future. They utilized collaborative filtering, which calculated cosine similarity between any pair of contents and predicted the content requested probability in each edge node in the future.

Machine learning techniques Matrix Factorization (MF)[25], Singular Value Decomposition (SVD)[25], and NMF are used to predict users' preferences, i.e., to predict users' ratings of videos they have not watched. These techniques characterize items and users by vectors, and the factorization between the user and item vectors is the corresponding rating. However, a big problem of matrix factorization is the cold start problem. We cannot make predictions for items and users that are neven been seen in the training stage because their embeddings are not available. It means that the MF approach is transductive.

Recently, GNNs have been used for user-item rating prediction. Recent research [26] applied inductive node-level graph convolutional neural (GCN)[27] framework to make item recommendations to users. The authors of [8] proposed an 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. The IGMC can tackle the cold start problem encountered in the MF approach, and our paper uses this model to predict users' ratings of movies.

3 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 [7, 25], IGMC [8] 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. In order to distinguish two kinds of nodes in a graph, user-type and movie-type nodes, as well as to identify the target user and target movie, we give label 0 to the target user and label 1 to the target movie first. 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 current subgraph rather than the complete bipartite graph. Therefore, we can predict ratings even for a subgraph from an entirely different bipartite graph.

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 operator (R-GCN) [28] to implement the message passing layers in GNN. R-GCN is an extension of [27], 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, R-GCN 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:

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

where x_i^l is the 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. R represents the set of all edge types, and $\mathcal{N}_r(i)$ is the set of 1-hop neighbourhood nodes of node i. For each edge type r, there is a different learnable weight matrix W_r^l. Therefore, the model can learn different relation patterns between users and movies in the graph. After that, the model concatenates node i's feature vectors from L message passing layers as the final representation. The following step is to apply a pooling layer to consider the concatenation of the final representations of the target user and movie feature vectors as the graph representation. Finally, the MLP is applied on the graph representation to get a predicted rating $\hat{r}_{m_i}^{u_i}$.

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. In our paper, we consider each movie's total predicted ratings as the gain of caching the movie:

$$g_{m_i}^{c_i} = \frac{totalRat_{m_i}^{c_i}}{MaxRat^{c_i}} \tag{2}$$

where $m_i \in \mathcal{M}, c_i \in \mathcal{C}$, $totalRat_{m_i}^{c_i}$ is the sum of movie m_i's predicted ratings in the user community c_i, and $MaxRat^{c_i}$ is the maximum total predicted ratings of the movie set $\mathcal{M}$ 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. We aim to maximize the total gain G of caching movies in the network, which is mathematically formulated as follows:

$$\begin{aligned} G &= \sum_{m_i \in \mathcal{M}} \sum_{c_i \in S_{f_i}} g_{m_i}^{c_i} b_{m_i,f_i} \\ \text{s.t.} & \sum_{m_i \in \mathcal{M}} b_{m_i,f_i} \leq N, f_i \in \mathcal{F} \\ & b_{m_i,f_i} \in \{0,1\}, m_i \in \mathcal{M}, f_i \in \mathcal{F} \end{aligned} \tag{3}$$

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 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 $f \in \mathcal{F}$
1: for $c$ in $C$ do
2:     $c.u\_set.add(c.id)$
3:     $root$ = NODE-INITIALIZATION($c$)
4: end for
5: 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
1: if $c.par$ not None then
2:     for $key$ in $c.gain\_t$ do
3:         if $key$ in $c.par.gain\_t$ then
4:             $c.par.gain\_t[key] \leftarrow c.par.gain\_t[key] + c.gain\_t[key]$
5:         else
6:             $c.par.gain\_t[key] \leftarrow c.gain\_t[key]$
7:         end if
8:     end for
9:     $c.par.u\_set \leftarrow c.par.u\_set \cup c.u\_set$
10:    if $c.par.u\_set$ != $c.u\_set$ then
11:        $c.par.n \leftarrow 0$
12:    else
13:        $c.par.n \leftarrow c.n + 1$
14:    end if
15:    NODE-INITIALIZATION($c.par$)
16: else
17:    return c
18: 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.

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:     $gain\_arr \leftarrow sort(gain\_arr, (a1, a2) \rightarrow (a2[1] - a1[1]))$
13:     for $i \leftarrow 1...s.cache\_size$ do
14:         $item\_index \leftarrow s.n * s.cache\_size$
15:         $item \leftarrow gain\_arr[item\_index][0]$
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
```

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 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 Experimental results

To evaluate our caching algorithm, we use Mini-NDN[29] 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 [6] on Mini-NDN. The authors of [6] also took user mobility into account when making caching decisions due to the highway simulation environment, which is different from our paper. Therefore, we do not consider user mobility. Except for the user mobility, [6] employs the same caching scheme as [22], which is the method we compare in our paper. 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 [30], 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, LCE+LRU and LCE+FIFO.

4.1 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.

4.1.1 Network Topology

Similar to papers [32, 33], we employ a real-world network topology GEANT [31], 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

Table 1: Experimentation Parameters

Parameters	Default Value	Values
Network topology	GEANT [31]	GEANT, 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

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.

4.1.2 Traffic Generation

We employ NDN Traffic Generator[34] 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 1 shows key parameters and values used in our paper.

4.1.3 Dataset Collection

We use the public benchmark dataset MovieLens 100K [35], 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 [6, 22], we consider the user rating for a movie as the user request for that movie. We randomly divide 943 users into two user communities.

4.1.4 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 calculated as follows:

$$\text{CHR} = \frac{\text{cache\_hits\_num}}{\text{cache\_hits\_num} + \text{cache\_misses\_num}} \tag{4}$$

  where cache_hits_num is the number of cache hits and cache_misses_num is the number of cache misses.

- ALT (Average Latency Time): It defines the average delay between the time the consumer sends an *Interest* packet and the time it receives a *Data* packet.

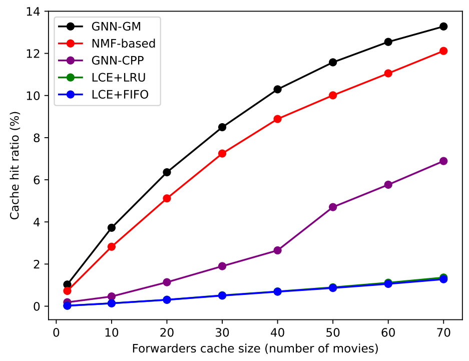
- Server Load: It defines the number of *Interest* packets served by the server.

4.2 Results

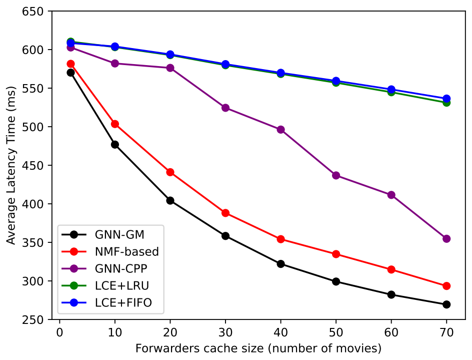
This section describes the experimental results for GNN-GM, NMF-based caching strategy, GNN-CPP, LCE+LRU, and LCE+FIFO. We utilize 4 R-GCN layers in the GNN-based rating prediction model. Both GNN and NMF are trained and tested using the same dataset, and they are trained with the Adam optimizer and 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.

4.2.1 Effect of Node Cache Sizes in Tree Topology

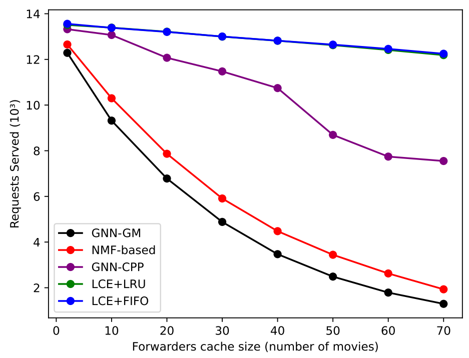
Figure 1a shows the cache hit ratio of the five caching algorithms with various forwarders' caching abilities in a 50 nodes tree network topology. The cache size of a forwarder is {2, 10, 20, 30, 40, 50, 60, 70}. We can observe that the cache hit ratio increases with the increase of forwarders caching 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



(a) Cache hit ratio



(b) Average latency time



(c) Server Load

Fig. 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.

perform nearly 200% better on average than the other two traditional reactive caching algorithms, LCE+LRU and LCE+FIFO.

Figure 1b shows the average latency time 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.

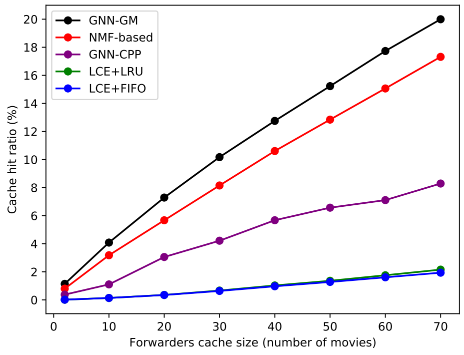
Figure 1c shows that the server load decreases as the forwarders' cache sizes increase. Overall, GNN-GM caching algorithm can achieve a 20% lower server load than the NMF-based caching algorithm. The proactive caching algorithm GNN-CPP has a heavier server load than GNN-GM and NMF-based caching algorithms. 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.

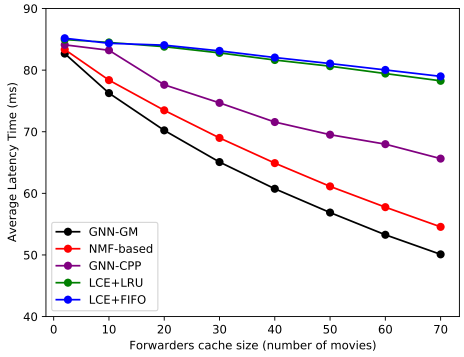
4.2.2 Effect of Node Cache Sizes in GEANT

Figure 2a shows the cache hit ratio of the five caching algorithms with various forwarders' caching abilities in GEANT. Similar to the Section 4.2.1, 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.

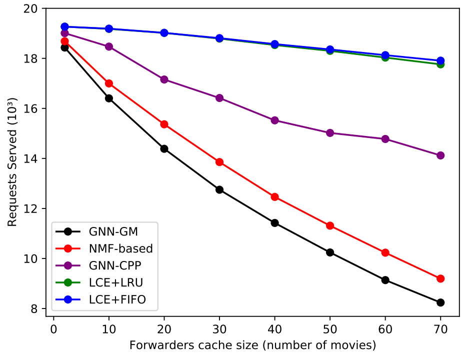
Figure 2b shows the average latency time of the five caching algorithms. At best, GNN-GM achieves around 8% and 25% lower latency than the NMF-based caching algorithm and GNN-CPP, respectively. GNN-GM consistently achieves the lowest latency regardless of the cache size. The other two traditional reactive caching algorithms have the worst performance, with a



(a) Cache hit ratio



(b) Average latency time



(c) Server Load

Fig. 2: GNN-GM, NMF-based, GNN-CPP, LCE + LRU, and LCE + FIFO caching algorithms' performances with different forwarders' cache sizes in GEANT.

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 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.

4.2.3 Effect of User Requests Distribution in GEANT

Table 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 2: Cache hit ratio, average latency time 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

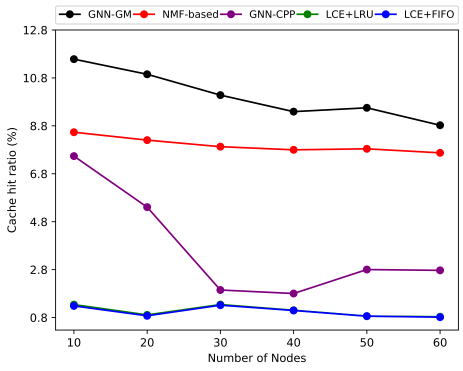
4.2.4 Effect of Network Sizes for Arbitrary Topologies

Figure 3 shows the cache hit ratio, average latency time 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 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 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 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.

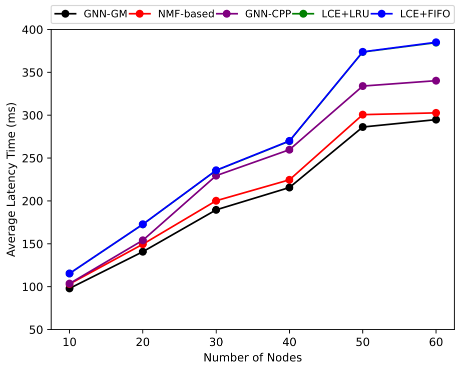
Figure 3b shows the average latency time for the five caching algorithms. We can see that GNN-GM has the lowest latency for all network sizes. It demonstrates that our GNN-GM caching algorithm can catch user preferences and put contents that most users will be interested in near the user. More number 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 time and perform significantly worse than the other three deep learning-based proactive caching strategies.

Figure 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.

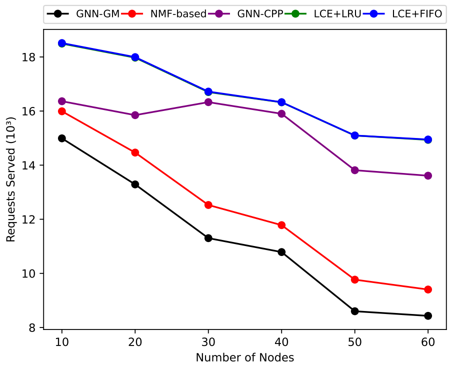
We can conclude that our GNN-GM caching strategy has the best performance for a different number of nodes in arbitrary network topologies. GNN-GM can capture user preferences and increase the diversity of caches.



(a) Cache hit ratio



(b) Average latency time



(c) Server Load

Fig. 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.

In addition, it can cache popular videos near users. The GNN-GM caching strategy can significantly ease the traffic load and enhance user experiences.

5 Conclusion

In this paper, 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. 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.

Deep reinforcement learning is a technique that is widely used in caching decisions. We plan to use GNN with reinforcement learning to optimize cache hit rate and content access latency in future work.

Declarations

Ethical Approval and Consent to participate

The authors consent to participate.

Human and Animal Ethics

The authors declare they follow human and animal ethics.

Consent for publication

The authors agree to publication.

Availability of supporting data

Not applicable.

Competing interests

The authors declare no competing interests.

Funding

Not applicable.

Authors' contributions

All authors designed this work.

Acknowledgements

This work was supported by a grant from Natural Sciences and Engineering Research Council of Canada (NSERC).

Authors' information

School of Electrical Engineering and Computer Science, University of Ottawa, Canada
Jiacheng Hou, Amiya Nayak
David R. Cheriton School of Computer Science, University of Waterloo, Canada
Haoye Lu

References

[1] Zhang, L., Estrin, D., Burke, J., Jacobson, V., Thornton, J.D., Smetters, D.K., Zhang, B., Tsudik, G., Massey, D., Papadopoulos, C., *et al.*: Named data networking (ndn) project. Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC **157**, 158 (2010)

[2] Thar, K., Tran, N.H., Oo, T.Z., Hong, C.S.: Deepmec: Mobile edge caching using deep learning. IEEE Access **6**, 78260–78275 (2018)

[3] Song, H.-G., Chae, S.H., Shin, W.-Y., Jeon, S.-W.: Predictive caching via learning temporal distribution of content requests. IEEE Communications Letters **23**(12), 2335–2339 (2019)

[4] Rahman, S., Alam, M.G.R., Rahman, M.M.: Deep learning-based predictive caching in the edge of a network. In: 2020 International Conference on Information Networking (ICOIN), pp. 797–801 (2020). IEEE

[5] Niloy, N.T., Islam, M.S.: 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), pp. 1–6 (2020). IEEE

[6] Zhang, Z., Lung, C.-H., St-Hilaire, M., Lambadaris, I.: Smart proactive caching: Empower the video delivery for autonomous vehicles in icn-based networks. IEEE Transactions on Vehicular Technology **69**(7), 7955–7965 (2020)

[7] Luo, X., Zhou, M., Xia, Y., Zhu, Q.: An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. IEEE Transactions on Industrial Informatics **10**(2), 1273–1284 (2014)

[8] Zhang, M., Chen, Y.: Inductive matrix completion based on graph neural networks. arXiv preprint arXiv:1904.12058 (2019)

[9] Hou, J., Lu, H., Nayak, A.: 'gnn-gm: A proactive caching scheme for named data networking. In: IEEE ICC Workshop on Research Advancements in Future Networking Technologies (RAFNET) (2022)

[10] Laoutaris, N., Che, H., Stavrakakis, I.: The lcd interconnection of lru caches and its analysis. Performance Evaluation **63**(7), 609–634 (2006)

[11] Li, Z., Simon, G., Gravey, A.: Caching policies for in-network caching. In: 2012 21st International Conference on Computer Communications and Networks (ICCCN), pp. 1–7 (2012). IEEE

[12] Shailendra, S., Sengottuvelan, S., Rath, H.K., Panigrahi, B., Simha, A.: Performance evaluation of caching policies in ndn-an icn architecture. In: 2016 IEEE Region 10 Conference (TENCON), pp. 1117–1121 (2016). IEEE

[13] Serhane, O., Yahyaoui, K., Nour, B., Moungla, H.: Cns: A cache and split scheme for 5g-enabled icn networks. In: ICC 2020-2020 IEEE International Conference on Communications (ICC), pp. 1–6 (2020). IEEE

[14] Nour, B., Khelifi, H., Moungla, H., Hussain, R., Guizani, N.: A distributed cache placement scheme for large-scale information-centric networking. IEEE Network **34**(6), 126–132 (2020)

[15] Li, J., Tang, J., Li, J., Zou, F.: Deep reinforcement learning for intelligent computing and content edge service in icn-based iov. In: 2021 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–7 (2021). IEEE

[16] Zhao, L., Ran, Y., Wang, H., Wang, J., Luo, J.: Towards cooperative caching for vehicular networks with multi-level federated reinforcement learning. In: ICC 2021-IEEE International Conference on Communications, pp. 1–6 (2021). IEEE

[17] Song, C., Xu, W., Wu, T., Yu, S., Zeng, P., Zhang, N.: Qoe-driven edge caching in vehicle networks based on deep reinforcement learning. IEEE Transactions on Vehicular Technology **70**(6), 5286–5295 (2021)

[18] Majeed, M.F., Dailey, M.N., Khan, R., Tunpan, A.: Pre-caching: A proactive scheme for caching video traffic in named data mesh networks. Journal of Network and Computer Applications **87**, 116–130 (2017)

[19] Dash, S., Kumar Dash, S., Sahu, B.J.: Proactive content caching for streaming over information-centric network. In: Intelligent and Cloud Computing, pp. 165–172 (2021)

[20] Musa, S.S., Zennaro, M., Libsie, M., Pietrosemoli, E.: Mobility-aware proactive edge caching optimization scheme in information-centric iov networks. Sensors **22**(4), 1387 (2022)

[21] Zhang, Y., Li, Y., Wang, R., Lu, J., Ma, X., Qiu, M.: Psac: Proactive sequence-aware content caching via deep learning at the network edge. IEEE Transactions on Network Science and Engineering **7**(4), 2145–2154 (2020)

[22] Zhang, Z., Lung, C.-H., St-Hilaire, M., Lambadaris, I.: Smart caching: Empower the video delivery for 5g-icn networks. In: ICC 2019-2019 IEEE International Conference on Communications (ICC), pp. 1–7 (2019). IEEE

[23] Gupta, D., Rani, S., Ahmed, S.H., Garg, S., Piran, M.J., Alrashoud, M.: Icn-based enhanced cooperative caching for multimedia streaming in resource constrained vehicular environment. IEEE Transactions on Intelligent Transportation Systems **22**(7), 4588–4600 (2021)

[24] Gupta, D., Rani, S., Ahmed, S.H., Verma, S., Ijaz, M.F., Shafi, J.: Edge caching based on collaborative filtering for heterogeneous icn-iot applications. Sensors **21**(16), 5491 (2021)

[25] Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)

[26] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 974–983 (2018)

[27] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)

[28] Schlichtkrull, M., Kipf, T.N., Bloem, P., Van Den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: European Semantic Web Conference, pp. 593–607 (2018). Springer

[29] named-data/mini-ndn. https://github.com/named-data/mini-ndn

[30] Hou, J., Xiu, H., Lu, H., Nayak, A.: A gnn-based approach to optimize cache hit ratio in ndn networks. In: IEEE GLOBECOM (2021)

[31] GÉANT topology map. https://www.geant.org/Networks/Pan-European_network/Pages/GEANT_topology_map.aspx

[32] Naeem, M.A., Nguyen, T.N., Ali, R., Cengiz, K., Meng, Y., Khurshaid, T.: Hybrid cache management in iot-based named data networking. IEEE Internet of Things Journal (2021)

[33] Man, D., Wang, Y., Wang, H., Guo, J., Lv, J., Xuan, S., Yang, W.: Information-centric networking cache placement method based on cache node status and location. Wireless Communications and Mobile Computing **2021** (2021)

[34] named-data/ndn-traffic-generator. https://github.com/named-data/ndn-traffic-generator

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