IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 September 2022, accepted 17 October 2022, date of publication 26 October 2022, date of current version 1 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3217236

RESEARCH ARTICLE

A Graph Neural Network Approach for Caching Performance Optimization in NDN Networks

JIACHENG HOU[1], HUANZHANG XIA[2], HAOYE LU[3], (Member, IEEE), AND AMIYA NAYAK[1], (Senior Member, IEEE)

[1]School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON K1N 6N5, Canada
[2]Department of Computer Science, University of Toronto, Toronto, ON M5S 2E4, Canada
[3]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada

Corresponding author: Jiacheng Hou (jhou013@uottawa.ca)

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

ABSTRACT Named Data Networking (NDN) is a new architecture with in-network caching ability. NDN nodes can cache data packets in their cache store to satisfy further requests. Accurately caching popular content across the network is essential for NDN to reduce the traffic workload and improve network efficiency. However, traditional caching algorithms are not good at predicting future dynamic content popularity. In our paper, we propose a Graph Neural Network-based (GNN-based) caching strategy to optimize the caching performance in NDN. First, our paper utilizes a Convolutional Neural Network (CNN) to extract time-series features for each NDN node. Secondly, GNN is applied to make content caching probability predictions for each NDN node. Third, at each NDN node, a cache replacement decision is made based on its content caching probability ranking, and content with high caching probability replaces content with low probability. We compare our GNN-based caching strategy with three deep learning-based caching techniques, which are 1D-Convolutional Neural Network (1D-CNN), Long Short-Term Memory Encoder-Decoder (LSTM-ED), and Staked Auto Encoder (SAE), and three classical benchmark caching strategies, which are Least Frequently Used (LFU), Least Recently Used (LRU) and First-in-first-out (FIFO). All caching scenarios are simulated in the Mini-NDN platform and evaluated on the tree and arbitrary network topologies. Empirical results suggest that the GNN-based caching approach can achieve around a 50% higher cache hit ratio and a 30% lower latency in the best case than other deep learning-based caching strategies.

INDEX TERMS Named data networking, deep learning, graph neural network, content caching probability, cache replacement.

I. INTRODUCTION

The internet architecture Named Data Networking (NDN) [1] is emerging these days. With NDN, data exchange does not require IP addresses as it does with traditional IP-based networks. Rather than specifying the IP address of the potential recipient, the consumer of NDN sends the *Interest* packet that contains the name tag of the desired data. The *Interest* packet can be replied with a *Data* packet by any NDN node holding the required content along its forwarding path. The ability to cache in-network is crucial in NDN since it dramatically reduces server load and prevents network congestion.

NDN [1] communications are not based on the content's locations but their names. As NDN packets and forwarding locations are independent, in-network caching can be developed. With this capability, suppose a user sends an *Interest* packet, then the NDN node closest to the user and caching the corresponding content can reply to it with an *Data* packet. In this case, network traffic is reduced, and the user experience is significantly improved. However, in reality, each NDN node has a limited cache space; simultaneously, there are many different contents throughout the network.

The associate editor coordinating the review of this manuscript and approving it for publication was Qingchun Chen.

Caching popular content and placing them as close to users as possible are ongoing research questions. This paper focuses on improving caching performance (i.e., maximizing the cache hit ratio) in NDN. The high cache hit ratio indicates that NDN nodes are able to 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. Therefore, achieving performance improvements in NDN requires a higher cache hit ratio.

In the context of Information-Centric Networking (ICN), which is closely related to NDN, there has been a large amount of research on caching strategies (see [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], and [17]). One of the most promising is the work by the authors in [17], utilizing a 1D-Convolutional Neural Network (1D-CNN) for 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 paper offers further improvements to [17]. 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 [18] 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.

GNN [19] is an emerging field of deep learning. With its powerful non-Euclidean graph representation capabilities, GNN has demonstrated impressive results in multiple graph-related tasks in many applications. GNN uses graph representation learning to transform graph data into a low-dimensional dense vector representation, ensuring that specific properties of graph data can be retained in the vector space. 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 [20] 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 [21], GNN is efficient for node level, edge level, and graph level prediction or classification tasks. In this paper, we apply GNN to node-level classification problems. For node-level classification problems, 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 paper 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 paper 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 [22], 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 [17], Long Short-Term Memory Encoder-Decoder (LSTM-ED) [15] and Staked Auto Encoder (SAE) [15], and classical benchmark strategies, including LFU, LRU and FIFO on Mini-NDN [23]. Extensive studies show that our GNN-based caching strategy achieves a much higher cache hit ratio and lower latency than other approaches.

The rest of this paper is organized as follows: Section II overviews related work. Section III presents the GNN-based caching strategy. Section IV presents the derivation of the ground truth label. Section V shows experimental results. Section VI concludes the paper. Preliminary results of this paper appear in [24], and the differences between the journal paper and the conference paper are listed below:

- Compared to the conference paper, the journal paper demonstrates a more comprehensive introduction, related work and methodology.
- The journal paper generalizes the GNN-based caching approach to handle arbitrary network topologies, not just tree networks. The conference paper only focuses on the tree network structure.
- The journal paper compares the performance of the GNN-based caching strategy with two more deep learning-based caching strategies, LSTM-ED and SAE,

and one more classical benchmark strategy, LFU, which is unavailable in the conference paper.
- The journal paper explores the performance of GNN-based strategy for different types of information aggregators of the GraphSAGE layer; this part is not included in the conference paper.
- The journal paper conducts extensive experiments on the effects of network size, node's cache size, content popularity distribution, content size, GraphSAGE's aggregation function and the number of GraphSAGE layers on the GNN-based caching. In addition, the journal paper considers one more evaluation metric: byte hit ratio.

II. RELATED WORK

Caching algorithms include cache placement strategies and cache replacement strategies. The cache placement strategy is responsible for deciding which content to cache in each network node, and the cache replacement strategy is responsible for choosing which content to evict when a network node's cache space is full. This section reviews both cache placement and replacement strategies.

A. TRADITIONAL CACHE PLACEMENT STRATEGIES

Leave Copy Everywhere (LCE) [1] is NDN's default cache placement policy. A packet is cached on all NDN nodes on the forwarding path between the producer and the consumer, where the producer is an NDN node that publishes the content, and the consumer is an NDN node that sends the *Interest* packet. Randomly Copy One (RCO) [2] is to cache the content randomly along the forwarding path between the producer and the consumer. This strategy reduces the caching redundancy compared to LCE. Another method named Probabilistic Cache [3] is a more advanced strategy. It considers the available cache capacity of each NDN node and its distance from the consumer - the closer it is to the consumer, the content will be cached on that node with a higher probability. In addition, nodes with sufficient caching capacity are more likely to cache the content.

B. POPULARITY-BASED CACHE PLACEMENT STRATEGIES

Researchers [4] proposed a strategy where each network node holds a popularity table containing information about the content name, the popularity count, and the threshold. Each network node caches only popular content whose popularity exceeds the threshold. Researchers [5] integrated popularity-based caching with RCO such that contents are cached according to their popularity, and only one network node is selected to cache the content along the data packet delivery path. Paper [6] proposed an efficient hybrid content placement strategy, where the most recently downloaded contents are cached at central network nodes along the data packet delivery path, and most minor downloaded ones are cached at edge network nodes. The authors aimed to increase content caching diversity across the network. Paper [7] proposed a compound popular content caching strategy, where all the contents are divided into two types, Optimal Popular Content (OPC) and Least Popular Content (LPC), according to the total number of received *Interest* packets for each content. The LPC content is cached only at the mutual network node near the requested consumers; on the other hand, the OPC content is cached at all mutually connected network nodes along the data packet delivery path. Authors in [8] also proposed a compound popularity caching strategy, where content popularity is divided into two types: global and local popularity. The cache placement decision is made considering both content and network node popularity. Paper [9] proposed a dynamic popularity window and distance-based efficient caching strategy. The scheme considers a dynamic threshold and makes caching decisions based on two aspects: a novel dynamic size popularity window to determine content popularity and the distance from the preceding on-path network node cached the content copy.

C. TRADITIONAL CACHE REPLACEMENT STRATEGIES

Traditional cache replacement strategies are First-in-first-out (FIFO) [11], Least Recently Used (LRU) [10] and Least Frequently Used (LFU) [10], and we compare our GNN-based cache replacement strategy with them in this paper. FIFO is a less effective caching replacement policy in many applications because it does not consider content popularity or other features. It replaces content according to the arriving order, where the newly arriving content replaces the oldest content in the cache. LRU usually performs better than FIFO because LRU caches the most recently requested content by assuming that the content will highly be requested again in the future. The least recently accessed content is discarded when the cache space is full. LFU usually performs much better than LRU and FIFO. This scheme assumes that the frequently requested content in the past will highly be requested in the future, so the least frequently requested contents are discarded first when there is no available cache space.

D. POPULARITY-BASED CACHE REPLACEMENT STRATEGIES

Paper [12] proposed a universal caching strategy, where it makes caching replacement decisions based on the frequency of fetching the content, distance from the content publisher, and the number of outgoing links at the intermediate network node. Researchers in [13] proposed a dynamic fine-grained popularity-based cache replacement strategy. The scheme always caches incoming content when there is available cache storage; otherwise, it keeps only the most popular content. In order to measure the popularity of each content, each network node maintains a popularity table containing the content name, content counter, and timestamp.

E. DEEP LEARNING-BASED CACHE REPLACEMENT STRATEGIES

With the development of deep learning, many deep learning-based cache replacement strategies are emerging. Recently, researchers have proposed many deep learning-based cache schemes to optimize caching performance in ICN.

The authors in [14] made use of a deep learning-based content popularity prediction (DLCPP) to perform cache decisions in the SDN-based ICN. They utilized SAE to extract network node features and predict the content's popularity. The authors in [15] proposed DeepCache, utilizing an LSTM-ED model to predict content popularity and make caching decisions accordingly. The authors also combined the predicted content popularity with a traditional cache replacement strategy (e.g., LRU) to prefetch the content to maximize the cache hit ratio. The researchers [16] proposed a Stimulable Neural Network (SNN) model to analyze the inter-relationships among sequenced requests and utilized previously unknown information and other factors, such as content size and data retrieval costs, to make caching decisions. The authors in [17] utilized 1D-CNN to capture the seasonal patterns in the user requests stream and predict each content's future hits. When the cache space is limited, only contents with high future hits are cached in the cache store.

F. GNN

Unlike the deep learning-based cache replacement strategy mentioned in Section II-E, our paper applies 1D-CNN and GNN to predict the probability of content caching. The papers in Section II-E only treat the caching problem as a time-series problem, i.e., the content popularity is highly correlated over time. However, caching can also be considered a spatial problem, and network nodes can cooperate to cache the most popular content as close to users as possible and increase the diversity of cached content across the network. Therefore, we utilize GNN to learn the spatial dependency of collected traffic data. Recently, GNN has played an important role in representing graph structure data. Like the 2D-Convolutional Neural Network (2D-CNN), GNN also utilizes neighbourhood information to learn spatial features. However, unlike 2D-CNN, GNN can capture data patterns on non-Euclidean structures rather than only 2D images or grids. Previous work [25] introduced Graph Convolutional Network (GCN) for semi-supervised learning. The GCN model directly operates on graph data and predicts labels for unlabeled nodes. The article [26] proposed Spatio-Temporal Graph Convolutional Networks (STGCN) framework, which contains 1D-CNN and GCN layers to make node-level traffic flow predictions. The 1D-CNN layer captures temporal dependency, and GCN captures spatial dependency. The authors in [27] provided a novel GNN-based link prediction framework, SEAL, to predict whether a link exists or not between two subgraphs. Paper [28] proposed a Deep Graph Convolutional Neural Network (DGCNN) architecture to predict labels on the graph level.

III. GNN-BASED CACHE 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 our paper, 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 across the network, denoted by $C = \{c_1, c_2, \ldots, c_K\}$. 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 [17], 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 paper, we focus on two types of network topologies. The first type is a tree network topology, and the second is an arbitrary network topology.

A. 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 $e_{j,i} \notin E$.

B. 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).

C. GNN-BASED CONTENT CACHING PROBABILITY PREDICTION

The proposed GNN-based model contains 1D-CNN layers and GraphSAGE [22] 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

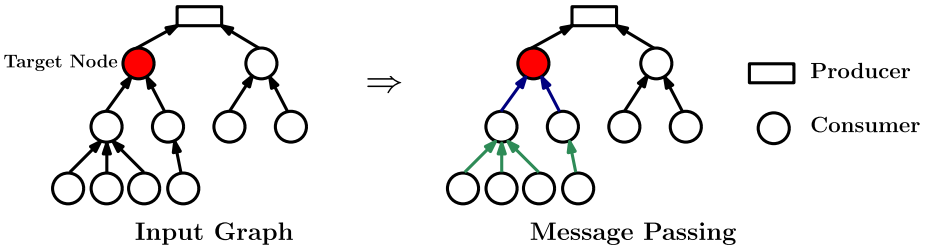


FIGURE 1. GNN Input graph and message passing in a tree network topology [24].

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 [29]. 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 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 out 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 to achieve cooperative caching.

The GraphSAGE framework [22] 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:

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

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:

$$AGG^{pool}_k = max(\{\sigma(W_{pool} h^k_{u_i} + b), \forall u_i \in N(v)\}) \tag{2}$$

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:

$$h^k_v = \sigma\left[W^k \cdot CONCAT\left(h^{k-1}_v, h^k_{N(v)}\right)\right] \tag{3}$$

where h^{k-1}_v is the node v's representation vector at the $(k-1)^{th}$ step and $h^k_{N(v)}$ 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. In our paper, we perform a ReLU activation function, and h^k_v is the updated representation of node v at k^{th} step.

D. CACHE 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.

IV. 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 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.

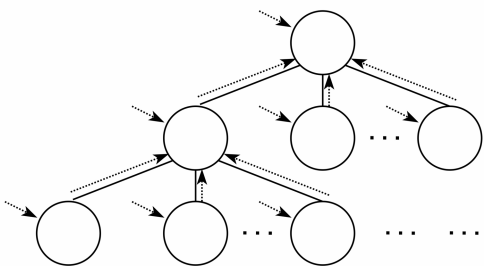


FIGURE 2. The visualization of the algorithm [24].

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.

V. EXPERIMENTAL RESULTS

In this section, we use Mini-NDN [23] 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 [17], LSTM-ED model mentioned in paper [15], and SAE model mentioned in paper [14] 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.

A. 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.

B. TRAFFIC GENERATION

We employ NDN Traffic Generator [30] to generate *Interest* and *Data* packets. By default, each consumer requests 2 unique contents with the content popularity follows a Zipf-like distribution [31] 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 per minute or sixty requests per minute, respectively. Table 1 shows key parameters used in our paper to generate traffic.

TABLE 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

C. 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.

D. EVALUATION METRICS

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

- 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:

$$CHR = \frac{cache\_hits\_num}{cache\_hits\_num + cache\_misses\_num} \quad (4)$$

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

$$BHR = \frac{cache\_hits\_bytes}{cache\_hits\_bytes + cache\_misses\_bytes} \quad (5)$$

  where *cache_hits_bytes* is the total object sizes for cache hits and *cache_misses_bytes* is the total object sizes for 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.

E. RESULTS

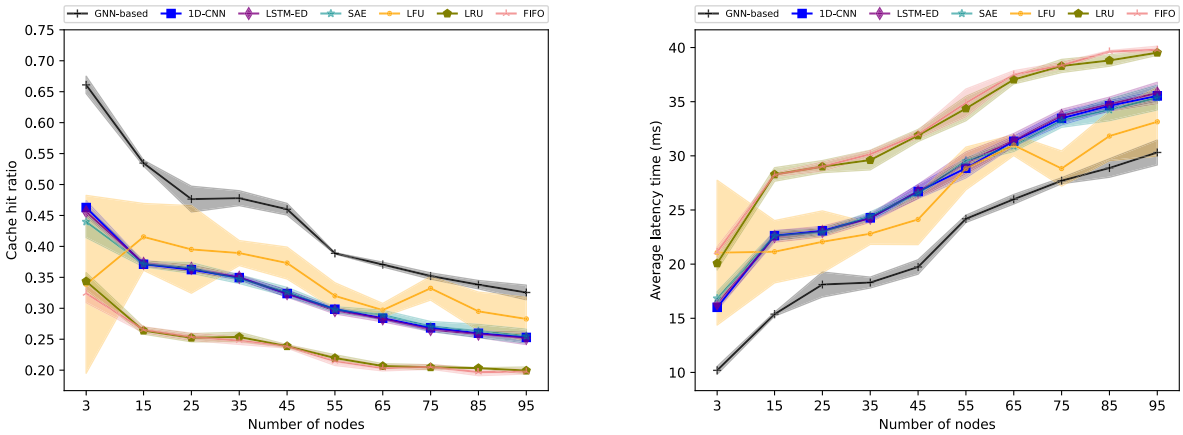
This section shows the experimental results of our GNN-based caching algorithm, 1D-CNN caching algorithm, LSTM-ED caching algorithm, SAE caching algorithm, LFU, LRU and FIFO. By default, the cache placement strategy is LCE. For deep learning-based cache replacement strategies, the models are trained 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 [32] for the GNN model and the mean square error [33] for the other models. All models are optimized using the Adam optimizer [32]. 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.

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 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 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 3a shows the 95% CI for the CHR of all strategies. In the best case, our GNN-based caching strategy performs



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

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

FIGURE 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.

a 44%, 45%, 50%, 95%, 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, 6% 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 3b shows the 95% CI for the ALT of all strategies. In the best case, our GNN-based caching algorithm achieves 36%, 37%, 39%, 51%, 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.

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

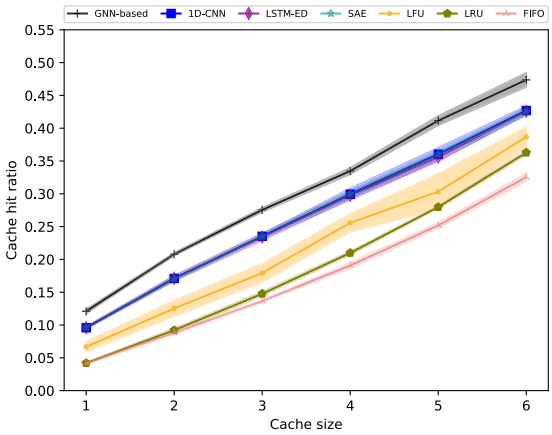
3) EFFECT OF ZIPF PARAMETER α VALUES

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 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.

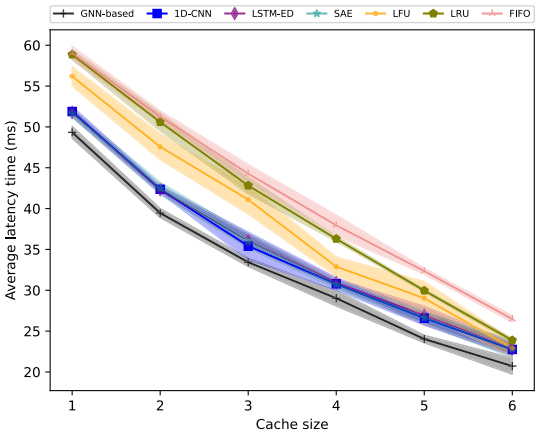
As shown in Figure 5a, for all deep learning-based caching algorithms, CHR increases as the α value increases due to a more noticeable difference in content popularity. When $\alpha = 0.1$, 1D-CNN, LSTM-ED, SAE, LFU, LRU, and FIFO perform similarly, but the GNN-based caching strategy can achieve much 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 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, the performance of 1D-CNN, LSTM-ED and SAE are almost the same regardless of the Zipf parameters because they can only predict the content popularity of each

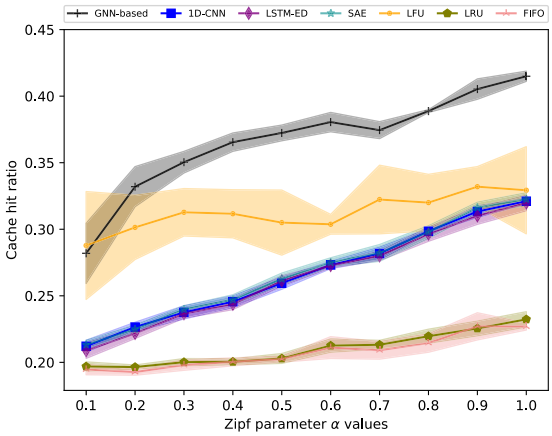


(a) 95% CI for the CHR of all caching strategies

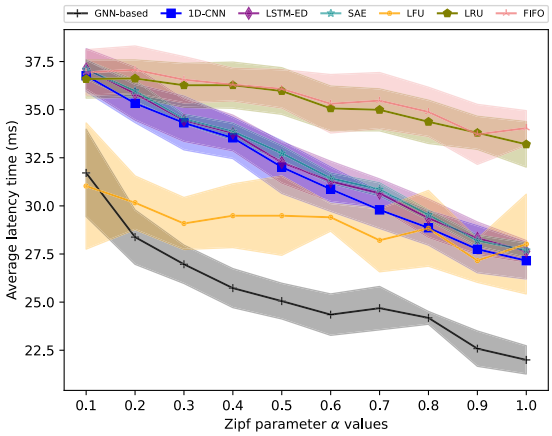


(b) 95% CI for the ALT of all caching strategies

FIGURE 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.



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



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

FIGURE 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.

node based on its requests. GNN, on the other hand, can capture the content popularity based on each node and its neighbouring nodes' requests.

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 1. Table 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.

TABLE 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)

TABLE 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)

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.

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 1. This section introduces one more metric, BHR, to show how much bandwidth the cache has saved. Table 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 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.

6) EFFECT OF INFORMATION AGGREGATOR TYPES

As mentioned, we utilize GraphSAGE [22] 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 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 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. This paper uses the pooling aggregator as the default aggregation function.

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 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 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

TABLE 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)

TABLE 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)

TABLE 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)

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 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.

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.

VI. CONCLUSION

In this paper, we proposed a GNN-based caching algorithm and deployed it on Mini-NDN. We compared 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 evaluated 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. In future work, we plan to combine our GNN-based caching algorithm with reinforcement learning to solve the data drift problem caused by caching.

REFERENCES

[1] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, and C. Papadopoulos, "Named data networking (NDN) project," Xerox Palo Alto Res. Center-PARC, Palo Alto, CA, USA, Tech. Rep. NDN-0001, 2010, vol. 157, p. 158.

[2] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga, "CATT: Potential based routing with content caching for ICN," in *Proc. 2nd, Ed., ICN Workshop Inf.-Centric Netw. (ICN)*, 2012, pp. 49–54.

[3] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. 2nd, Ed., ICN Workshop Inf.-Centric Netw. (ICN)*, 2012, pp. 55–60.

[4] C. Bernardini, T. Silverston, and O. Festor, "MPC: Popularity-based caching strategy for content centric networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 3619–3623.
[5] M. Yu, R. Li, Y. Liu, and Y. Li, "A caching strategy based on content popularity and router level for NDN," in *Proc. 7th IEEE Int. Conf. Electron. Inf. Emergency Commun. (ICEIEC)*, Jul. 2017, pp. 195–198.
[6] Y. Meng, M. A. Naeem, R. Ali, and B.-S. Kim, "EHCP: An efficient hybrid content placement strategy in named data network caching," *IEEE Access*, vol. 7, pp. 155601–155611, 2019.
[7] M. A. Naeem, S. A. Nor, S. Hassan, and B.-S. Kim, "Compound popular content caching strategy in named data networking," *Electronics*, vol. 8, no. 7, p. 771, Jul. 2019.
[8] Y. Gui and Y. Chen, "A cache placement strategy based on compound popularity in named data networking," *IEEE Access*, vol. 8, pp. 196002–196012, 2020.
[9] S. Kumar and R. Tiwari, "Dynamic popularity window and distance-based efficient caching for fast content delivery applications in CCN," *Eng. Sci. Technol., Int. J.*, vol. 24, no. 3, pp. 829–837, Jun. 2021.
[10] Z. Li, G. Simon, and A. Gravey, "Caching policies for in-network caching," in *Proc. 21st Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2012, pp. 1–7.
[11] S. Shailendra, S. Sengottuvelan, H. K. Rath, B. Panigrahi, and A. Simha, "Performance evaluation of caching policies in NDN—An ICN architecture," in *Proc. IEEE Region 10 Conf. (TENCON)*, Nov. 2016, pp. 1117–1121.
[12] B. Panigrahi, S. Shailendra, H. K. Rath, and A. Simha, "Universal caching model and Markov-based cache analysis for information centric networks," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommuncations Syst. (ANTS)*, Dec. 2014, pp. 1–6.
[13] M. D. Ong, M. Chen, T. Taleb, X. Wang, and V. C. M. Leung, "FGPC: Fine-grained popularity-based caching design for content centric networking," in *Proc. 17th ACM Int. Conf. Model., Anal. Simul. Wireless Mobile Syst. (MSWiM)*, 2014, pp. 295–302.
[14] W.-X. Liu, J. Zhang, Z.-W. Liang, L.-X. Peng, and J. Cai, "Content popularity prediction and caching for ICN: A deep learning approach with SDN," *IEEE Access*, vol. 6, pp. 5075–5089, 2017.
[15] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "DeepCache: A deep learning based framework for content caching," in *Proc. Workshop Netw. Meets AI ML (NetAI)*, 2018, pp. 48–53.
[16] Y. Im, P. Prahladan, T. H. Kim, Y. G. Hong, and S. Ha, "SNN-cache: A practical machine learning-based caching system utilizing the inter-relationships of requests," in *Proc. 52nd Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2018, pp. 1–6.
[17] K. H. T. Chiu, J. Zhang, and B. Bensaou, "Cache management in information-centric networks using convolutional neural network," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–6.
[18] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," 2017, *arXiv:1703.04691*.
[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Mar. 2021.
[20] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Exp. Syst. Appl.*, vol. 207, Nov. 2022, Art. no. 117921.
[21] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. Wiltschko, "A gentle introduction to graph neural networks," *Distill*, vol. 6, no. 8, Aug. 2021.
[22] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2017, *arXiv:1706.02216*.
[23] *Named-Data/Mini-NDN*. [Online]. Available: https://github.com/named-data/mini-ndn
[24] J. Hou, H. Xia, H. Lu, and A. Nayak, "A GNN-based approach to optimize cache hit ratio in NDN networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 1–6.
[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[26] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," 2017, *arXiv:1709.04875*.
[27] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. NIPS*, vol. 31, 2018, pp. 5165–5175.
[28] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.
[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.
[30] *Named-Data/NDN-Traffic-Generator*. [Online]. Available: https://github.com/named-data/ndn-traffic-generator
[31] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM. Conf. Comput. Commun. 18th Annu. Joint Conf. IEEE Comput. Commun. Societies. Future Now*, Mar. 1999, pp. 126–134.
[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[33] D. M. Allen, "Mean square error of prediction as a criterion for selecting variables," *Technometrics*, vol. 13, no. 3, pp. 469–475, 1971.

JIACHENG HOU received the B.Sc. degree in computing degree from the Macao Polytechnic Institute, Macao, in 2019. She is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science, University of Ottawa, Canada. Her research interests include information-centric networking, edge computing, and deep learning.

HUANZHANG XIA is currently pursuing the undergraduate degree with the University of Toronto, specializing in computer science and focusing on artificial intelligence. Her research interests include machine learning and artificial intelligence.

HAOYE LU (Member, IEEE) received the B.Sc. joint degree in computer science and mathematics, in 2017, and the master's degree in computer science, in 2019. He is currently pursuing the Ph.D. degree with the David R. Cheriton School of Computer Science, University of Waterloo. He joined at the University of Ottawa, Canada, in 2013. His research interests include deep learning and convex and non-convex optimization.

AMIYA NAYAK (Senior Member, IEEE) received the B.Math. degree in computer science and combinatorics and optimization from the University of Waterloo, Canada, in 1981, and the Ph.D. degree in systems and computer engineering from Carleton University, Canada, in 1991.

Currently, he is a Full Professor with the School of Electrical Engineering and Computer Science, University of Ottawa. He has over 17 years of industrial experience in software engineering, avionics and navigation systems, and simulation and system level performance analysis. His research interests include software-defined networking, mobile computing, wireless sensor networks, and vehicular ad hoc networks. He was on the Editorial Board of several journals, including IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *International Journal of Parallel, Emergent and Distributed Systems*, *Journal of Sensor and Actuator Networks*, and *EURASIP Journal on Wireless Communications and Networking*. Currently, he is on the Editorial Board of IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE OPEN JOURNAL OF THE COMPUTER SOCIETY, *Future Internet*, and *International Journal of Distributed Sensor Networks*.