IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INNOVATION AND APPLICATION?OF INTELLIGENT PROCESSING OF DATA, INFORMATION AND KNOWLEDGE AS RESOURCES IN EDGE COMPUTING

Received September 24, 2019, accepted October 4, 2019, date of publication October 11, 2019, date of current version October 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2947067

Toward Edge-Assisted Video Content Intelligent Caching With Long Short-Term Memory Learning

CONG ZHANG[1], HAITIAN PANG[2,3], JIANGCHUAN LIU[4], (Fellow, IEEE), SHIZHI TANG[3], RUIXIAO ZHANG[3], DAN WANG[2], AND LIFENG SUN[3]

[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei 230052, China
[2]Department of Computing, The Hong Kong Polytechnic University, Hong Kong
[3]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[4]School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada

Corresponding author: Haitian Pang (pht14@tsinghua.org.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61902369, in part by the Fundamental Research Funds for the Central Universities under Grant WK2150110015, in part by the National Natural Science Foundation of China under Grant 61936011, and in part by the Hong Kong under Grant ITF UIM/363, Grant CRF C5026-18G, and Grant PolyU ZVPZ.

ABSTRACT Nowadays video content has contributed to the majority of Internet traffic, which brings great challenge to the network infrastructure. Fortunately, the emergence of edge computing has provided a promising way to reduce the video load on the network by caching contents closer to users.But caching replacement algorithm is essential for the cache efficiency considering the limited cache space under existing edge-assisted network architecture. To investigate the challenges and opportunities inside, we first measure the performance of five state-of-the-art caching algorithms based on three real-world datasets. Our observation shows that state-of-the-art caching replacement algorithms suffer from following weaknesses: 1) the rule-based replacement approachs (e.g., LFU,LRU) cannot adapt under different scenarios; 2) data-driven forecast approaches only work efficiently on specific scenarios or datasets, as the extracted features working on one dataset may not work on another one. Motivated by these observations and edge-assisted computation capacity, we then propose an edge-assisted intelligent caching replacement framework *LSTM-C* based on deep Long Short-Term Memory network, which contains two types of modules: 1) four basic modules manage the coordination among content requests, content replace, cache space, service management; 2) three learning-based modules enable the online deep learning to provide intelligent caching strategy. Supported by this design, LSTM-C learns the pattern of content popularity at long and short time scales as well as determines the cache replacement policy. Most important, LSTM-C represents the request pattern with built-in memory cells, thus requires no data pre-processing, pre-programmed model or additional information. Our experiment results show that LSTM-C outperforms state-of-the-art methods in cache hit rate on three real-traces of video requests. When the cache size is limited, LSTM-C outperforms baselines by 20%∼32% in cache hit rate. We also show that the training and predicting time of one iteration are 8.6 *ms* and 300 μs on average respectively, which are fast enough for online operations.

INDEX TERMS Edge-assisted caching replacement, intelligent content caching, long short term memory.

I. INTRODUCTION

In the past decade, we are witnessing the explosive growth of Internet video, which accounts for 60% of the Internet traffic in 2016, and is forecasted to double by 2021 according to Cisco's recent report [1]. The quality of individual video streams is rapidly improving as well, with a majority of them have become 1080p high resolution (HD) or even 4K ultra HD (UHD), posing significant challenges towards delivering high Quality of Service (QoS) to video consumers.

Caching near the end users has become an indispensable component in Internet content distribution systems for decades. Given the sheer volume of video traffic, caching is critical to Internet video streaming, too [2]. For instance, iQiyi [3], a major video service providers in China, has nearly 6 billion hours of viewing per month [4], and the

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

corresponding data size is in an EB scale; As such, each 1% hit ratio can easily translate into PBs of data savings. Improving the cache hit ratio however is easy said than done, particularly for video, which has much large size and longer playback duration as compared to conventional data objects. The user preferences on videos also vary over time and geo-locations. This will further be aggregated in future 5G networks with massive small cells, where base stations will be equipped with storage capacity and serve local users with caching [5], [6].

Another important caching component is to employ a proper cache replacement algorithm in content distribution systems. Most of today's caching systems rely on such *rule-based* cache replacement algorithms as FIFO [7], Least Recently Used (LRU), Least Frequently Used (LFU), or their variants [8]–[10]. These algorithms follow simplified rules and are easy to implement in reality, but the fixed rules can hardly adapt to the dynamic video access patterns. With the advancement in data analytics, *forecast-based* cache replacement algorithms have recently been suggested [11]. Using a data-driven machine learning design, they integrate three cascaded components: (1) extracting key features from content request dataset; (2) training a model with the dataset, and (3) forecasting the objects to be evicted with the model. A well-trained model with proper feature engineering can achieve high hit ratio. Yet it heavily depends on the specific data for training, and is hardly adaptive, neither in temporal nor in spatial domains. Recent studies have suggested the use of multi-source input (e.g., location-based [11], [12] and social-based information [13]) to improve accuracy. They however further complicates the algorithm design and limits the applicability given the extra demands on input data.

In this paper, using real-world datasets for video accesses, we examine the representative state-of-the-art algorithms for video caching and systematically analyze their limits, shedding lights into the design of adaptive video caching to approximate the optimal. Our observation shows that state-of-the-art caching replacement algorithms suffer from following weaknesses: 1) the algorithms using rule-based replacement approach (e.g., LFU, LRU) cannot adapt under different scenarios, while the algorithms using data-driven forecast approach only work efficiently on specific scenarios or datasets, as the extracted features working on one dataset may not work on another one.

Recently, edge computing has emerged as a promising solution to shift a part of computation-intensive workloads at edge network in lots of practical applications [14]. Besides, more and more edge nodes with light-weight computing capacity have been deployed in a large number of scenarios to enable learning-based tasks [15]. Motivated by the aforementioned observations and the enhanced computation capacity from edge nodes, We accordingly propose LSTM-C, an edge-assisted intelligent caching framework that learns the caching strategy automatically from the request sequence in real-time, without using any data pre-processing or feature engineering. Given the dynamics of video access patterns in both short- and long-terms, LSTM-C employs a deep LSTM network [16], an advanced neural network architecture with memory to characterize the sequential pattern. Approaching the optimal caching requires the knowledge of future information, which is not available in reality. We develop an approximate method in the training process, together with an asynchronous operation to reduce the computation cost for training. As such, LSTM-C automatically learns the cache replacement strategy that adapts to a wide range of scenarios.

Our experiment results show that LSTM-C outperforms state-of-the-art methods in cache hit rate on three real-traces of video requests (including one small cell caching trace) across different cache sizes. In particular, when the cache size is limited, LSTM-C outperforms baselines by 20%∼32% in cache hit rate. Moreover, both the training and predicting processes are executed with fast speeds. Each iteration is confined to 8.6 *ms* (training) and 300 μs (predicting) only, which are fast enough for online operations. Using only the request sequence data as the input, LSTM-C can be applied to many other contexts as well, especially to such human-in-the-loop applications as web browsing and picture viewing, where the access patterns are highly dynamic and heterogeneous.

The remainder of the paper is organized as follows. We illustrate our motivation toward a better cache algorithm for video caching in Section II. The deep-learning-based cache system model and the design of LSTM-C is introduced in Section III. Experimental results are provided in Section IV. We show the related work in Section V and conclude the paper in Section VI.

II. MOTIVATION FOR A BETTER CACHING ALGORITHM

In this section, we investigate the performance of state-of-the-art algorithms. We first introduce the caching mechanism of these algorithms and optimal method and the real-world datasets used in our work. Then, we illustrate the caching model and measurement methods. The metrics we focus on are the hit rate of algorithms on a specific dataset, the gap in hit rate between the algorithms and the optimal policy, and the scalability of algorithms across multiple datasets. Finally, we provide the observations, which also motivate us to design a better algorithm for caching replacement under current edge-assisted network architecture.

A. EXISTING CACHING REPLACEMENT METHODS

1) ORDER-BASED METHOD—FIFO

The cache replacement follows the Fisrt-In-First-Out principle: the earliest stored content is replaced by the new content when the cache space is full.

2) RECENCY-BASED METHOD—LRU

An ordered list is maintained to track the most recent request of all cached contents. The least recently requested one is replaced by the new content when the cache is full.

3) FREQUENCY-BASED METHOD—LFU

An ordered list is maintained to track the requested numbers of all contents. The content which is least frequently used in a time window is replaced by the new content when the cache is full.

4) FREQUENCY-BASED METHOD—LFUDA

Note that LFU may have very poor long-term performance due to a cache pollution problem [9]: if a previously popular content becomes unpopular, LFU may still hold it for a long time, resulting in inefficient utilization of the cache. Furthermore, we introduce a method called LFUDA [17], which maintains the cache age count to solve the cache pollution problem.

5) FORECAST-BASED METHOD—POP

Recently, some works collect the request data and use machine learning based methods to forecast the popularity of the contents [11], [12], [18], [19]. PopCaching [18] is a recent representative algorithm, which learns the relationship between the future popularity of a content and its recent request pattern. Using the popularity forecasting result, PopCaching makes proper cache replacement decisions to maximize the cache hit rate.

6) OPTIMAL METHOD—OPT

Belady's MIN algorithm [20] can achieve theoretically optimal performance with complete information. The key idea is to evict the content which has the largest next-access time. Note that Belady's algorithm is not implementable in a real system due to the fact that it needs future information.

B. DATASETS

We present how we collect the datasets used in our study. In order to provide convincing experiment results and test the algorithm scalability across different scenarios, we implement the above algorithms on the three datasets.

1) IQIYI DATASET

We collected the mobile video request dataset from iQiyi, which is one of the largest video providers in China [3]. How users view videos in the mobile video streaming app has been recorded. The dataset spans 2 weeks and covers 2 million users watching 0.3 million unique videos in Beijing City. In each trace item, the following information is recorded: (1) The device identifier, which is unique for different devices and can be used to track users; (2) The timestamp when the user starts to watch the video; (3) The location where the user watches the video: the video player reports the location collected from the device's built-in GPS function; (4) The title of the video, which is unique for different video contents.

2) IQIYI SMALL CELL DATASET

With the emerging technologies like femto-caching [6], an important trend in content caching is to cache contents in edge networks covering small regions. Most important, due to the advantages of mobile networking communication, e.g., 4G/5G, a large number of mobile users prefer watching high-definition videos using 4G/5G technologies. We therefore study how the algorithms perform for content delivery in today's edge network.

We jointly utilize the iQiyi Dataset and the base station dataset. The base station dataset contains the locations, IDs, and location area codes (LACs) of over 70 thousand base stations in Beijing. With the knowledge of the location and signal radius of a base station, we can filter user requests sent from the coverage, which is used to obtain the small cell dataset.

3) MOVIELENS DATASET

MovieLens [21] is a website providing user ratings on movies. To simulate the content request process, we take each comment on a video content by a user as the request for this content. The idea of using video comments to simulate the video requests is provided in [12], [18], where the authors believe users usually give a comment when they finish watching a movie. By analyzing the timestamp of submitting a review, we can know the sequence of video content requests.

C. CACHING MODEL

Consider one content provider (CP) who provides a set of contents $\mathcal{C} = \{1, 2, \ldots, C\}$ to end users. User requests arrive in a sequential fashion in discrete time slots. The content provider employs a cache system, aiming to offload the user requests to its local cache at its best effort. In this paper, we focus on a single cache node in such a system. But the location of the cache node is different under different scenario. The cache node in the first dataset is located at a datacenter of a video streaming service provider, it serves all of the viewers in Beijing area under this dataset. The cache node in the second dataset is located at a base station, it serves the viewers in a small area. The cache node in the third dataset is controlled by a CDN service provider, it serves all of the viewers from lots of different areas. So the networking setting in this work covers the large scale and small scale streaming services in various scenarios.

Let $s < C$ be the capacity of the node. The videos in the Internet are divided into a sequence of contents with similar sizes and delivered to users. For simplicity, we assume that the contents have the same size, so the node can cache up to s contents [18] or we can cache the prefix of video contents [22], [23] with the same data size. We denote the users' request sequence that the CP receives as $Seq = \{c_1, c_2, \ldots c_t, \ldots, c_T\}$, where t denotes the time slot of the request, and $c_t \in \mathcal{C}, \ \forall t \in [1, T]$. In each time slot, there may exist multiple contents being cached. We use one-hot encoding to represent the cached contents in time slot t: $Y_t = [Y_t(1), Y_t(2), \ldots, Y_t(C)]$, where $Y_t(c) \in \{0, 1\}, \ \forall c \in [1, C]$. Specifically, $Y_t(c_t) = 1$ means that request c_t can be delivered by the local cache, and $Y_t(c_t) = 0$ means that request for c_t should be delivered from the remote server.

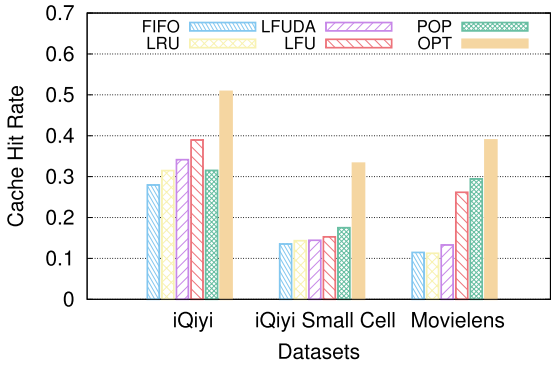


FIGURE 1. Hit rate of the algorithms.

When a new content request c_t arrives, we first check if the requested content is cached locally. The cache algorithm works in a passive way, i.e., when a requested content is not cached, we should decide whether to cache the content and if so, which content to replace from the cache server. When the requested content is cached in advance, the cache does not update. In this way, we use the *cache hit rate* as the system objective. Formally, the cache hit rate is defined as:

$$\frac{1}{T}\sum_{t\in[1,T]} Y_t(c_t). \tag{1}$$

Generally speaking, the cache hit rate is defined as follows: the cache hit number of requested contents divide the number of requested contents. When the size is changed, the cache hit rate will not change. In our work, we define the caching model according to the research work in [18]. The cache hit rate reflects the percentage of requests that are served from the local cache up to the user request in the T-th time slot.

D. METHODS PERFORMANCE MEASUREMENT

For convenience, we set the cache size as 10, i.e., at most 10 video contents can be cached on one caching node. Then we test the above caching algorithms, with the optimal algorithm as the upper bound of the caching hit rate. We compare the caching hit rates under the three datasets, and find that there exists a large gap between the algorithms and the optimal one as depicted in Fig. 1. We find that LFU is the best algorithm under iQiyi dataset, and PopCaching is the best algorithm under iQiyi Small Cell dataset and Movielens dataset. Moreover, the best algorithm can only reach 52.6%∼75.5% of the optimal solution in cache hit rate. Thus, there exists much room for optimization.

To better understand the reason why the gap exists, we further provide the measurement result on *how good* each decision step is in the whole dataset. Specifically, when the cache replacement is triggered, with all the future information, we can rank the currently cached contents in the order of future arrival interval. The optimal method (Belady's MIN algorithm) is to evict the content with the largest future arrival interval. We provide an illustration example in Fig. 2. The left part shows a caching node with three units of cache size. The

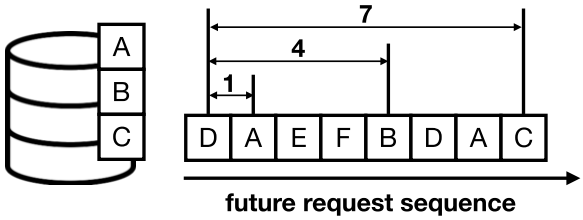
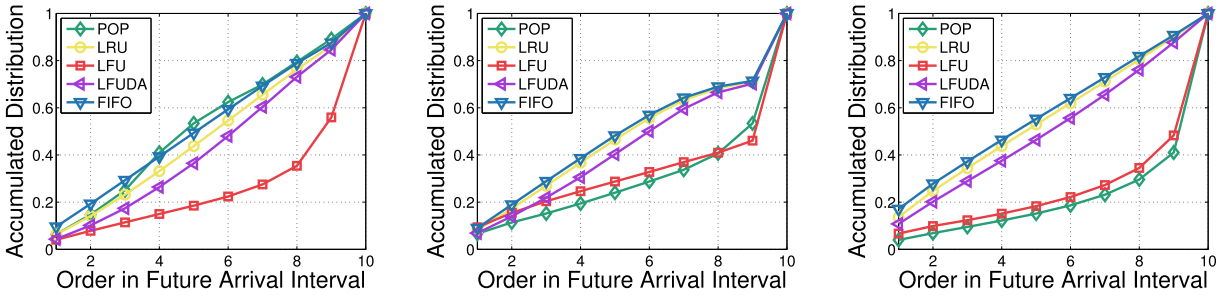


FIGURE 2. Illustration of the optimal policy.

cached contents are A, B, and C. As shown in the right part in Fig. 2, when a new content D arrives, the optimal solution is to evict content C, which has the largest arrival interval 7, and the order of arrival interval is 3. Suppose an algorithm decides to evict A, it makes the worst decision (arrival interval of A is 1, and the order of arrival interval is 1). To measure the performance of the algorithms, we investigate in which order of the future arrival interval an algorithm can achieve. We provide the distribution of the order of an algorithm on three datasets is provided in Fig. 3. We assume that the caching size is 10, so the range of order (i.e., the index in the caching space) on the x-axis is from 1 to 10. Taking order 2 as an example, if there is an evicted operation at index 2, the number of operation at index 2 is added one. Therefore, the optimal method always evicts the item in the last index, i.e. order 10. The observations are illustrated as follows: all the algorithms bear large gap with the optimal solution. PopCaching algorithm can make 45% and 60% optimal decision under iQiyi Small Cell dataset and Movielens dataset. But PopCaching can only make 12% optimal decision under iQiyi dataset. We pay particular attention to Fig. 3(b), where the performances of LFU and PopCaching are quite close. The probability that PopCaching makes an optimal decision is 7% lower than that of LFU, but PopCaching makes fewer worst decisions. PopCaching obtains higher cache hit rate according to Fig. 1. This is because once a mistaken decision is made, it may have cascading effects on the caching hit rate. For example, replacing the most popular content with an unpopular content may cause many cache misses and replacement executions, because the probability of an unpopular content being requested is low, making cache misses happen again. This makes it more important to find an algorithm closer to optimality.

E. INSIGHTS FOR A BETTER CACHING ALGORITHM

The limitation of the rule-based algorithms is that there may be different performances under different request patterns. Data-driven algorithms rely too much on manual feature-engineering repeatedly, as the extracted features working on one dataset may not work on the other. Ideally, we need an algorithm relying on no pre-determined rules or manually extracted features, with powerful representation ability to capture the inherent patterns, and can adjust with timely request pattern. Existing works already showed the superior of deep learning-based design in term of resource management [24], adaptive quality selection [25], viewers' behavior



(a) iQiyi (b) iQiyi Small Cell (c) Movielens

FIGURE 3. Distribution of prediction order in arrival interval on three datasets.

prediction [26], etc. Such designs enable automatic feature extraction and timely pattern capture. We therefore believe that deep learning has the potential to address the above challenges.

III. CONTENT CACHING WITH DEEP LSTM (LSTM-C)

A. DEEP LEARNING-BASED CACHE SYSTEM

We first introduce the content caching model with the novel deep learning technique. Learning-based caching algorithm has the potential to achieve good caching performance, as the learning model can capture the time-variant pattern adaptively by learning online [9]. However, previous learning-based caching algorithms have the following weaknesses: 1) Conventional learning methods rely too much on artificial feature engineering and parameter tuning, causing poor scalability across different scenarios. 2) The representation ability of such models is usually limited, thus cannot well capture the complex and variant pattern in real-world caching systems. The newly emerged deep learning technique has the potential to overcome the above weaknesses. Specifically, deep learning can take the "raw" data as input (requires no preprocessing), and characterize the inherent pattern precisely with the powerful representation ability of the neural network.

We present the architecture of the deep learning-based cache system in Fig. 4. In addition to basic modules, i.e., Request Database, Service Module, Local Cache, and Replace Module, the deep learning-based caching node also includes the Training Trigger, Training Module, and Prediction Module to enable the online deep learning.

When a *User Request* arrives at the cache system, the new request will be stored in the *Request Database* and sent to *Training Trigger* as well as *Service Module* at the same time. The Request Database stores historical requests and feed the request data for training or predicting. The training trigger is designed to make the training and predicting processes asynchronous for the system robustness and lower the computing load. It determines whether to trigger the training process of the deep learning module, and if so, it will instruct the *Training Module* of the deep learning module to perform the training process. The Service Module determines where to fetch the content, i.e., from the local cache or the remote

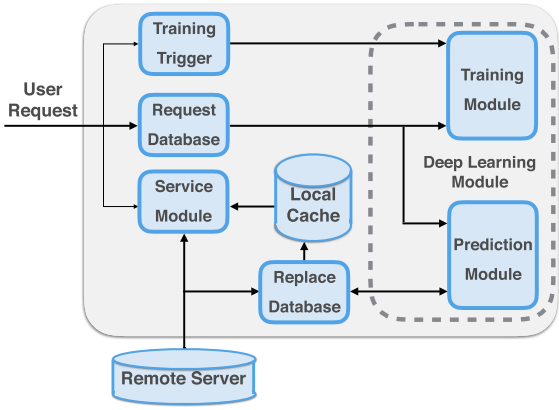


FIGURE 4. Framework of LSTM-C-based cache system.

server. When the content is not locally cached, it will be fetched from the remote server and a replacement order will be made by the *Replace Module*. The replace module sends an order to the *Prediction Module*, and the Prediction Module returns which content to evict from Local Cache. In the end, the replace module executes the content eviction.

The deep learning module requires no feature extraction process, which is a necessary module in the conventional learning-based cache systems [18]. This improves the scalability of the deep learning based caching system, as the feature extraction is artificial and becomes the barrier when applied to another scenario. In addition, the deployment of the deep learning model requires little modification of the cache system. Specifically, we only need to equip the cache system with a light-weighted neural network model consuming affordable storage space and computation resource. We provide the result of the computation cost in Section IV, which is quite low and thus satisfies the online training process.

B. CONVENTIONAL LSTM CELL

Inspired by the recent success of deep LSTM network processing streaming data, e.g., speech recognition [27], [28], we design a novel LSTM network algorithm for content caching. Despite the large number of neural network types in deep learning, we select the LSTM as the network unit as it has the potential to address the challenges for our problem:

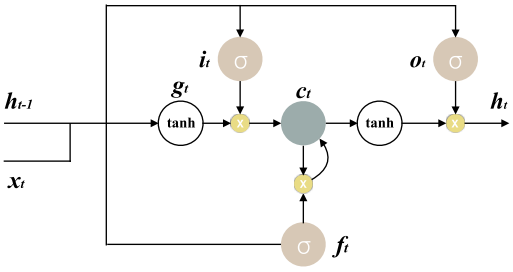


FIGURE 5. LSTM cell architecture.

1) the video content requests form a time sequence naturally, while LSTM is especially good at sequence modeling task. 2) LSTM's sophisticated network structure enable itself with strong representation ability from raw data input, thus requires little data pre-processing. 3) The memory structure inside LSTM can make full use of the historical sequence information when making decisions or predictions. 4) The LSTM network can be updated online to capture the timely popularity of the contents. We provide the description of LSTM as follows.

LSTM is a variant of recurrent neural networks (RNNs) that is specifically designed for sequence labeling of temporal data. In LSTM, the input and output gates incorporate the incoming and outgoing signals to the memory cell, and the forget gate controls whether to forget the previous state of the memory cell. Fig. 5 shows the structure of an LSTM cell. The input gate (i_t), forget gate (f_t), and candidate memory cell state (g_t) at time t are computed by Equations (2) $\sim$ (4), respectively, in which W and U are weight matrices for the input (x_t) at time t and the cell output (h_t) at time $t-1$, b is the bias vector of each unit, and σ as well as *tanh* are the logistic sigmoid and hyperbolic tangent function, respectively.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2}$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3}$$
$$g_t = tanh(W_r x_t + U_r h_{t-1} + b_r) \tag{4}$$

Once these three vectors are computed, the current memory cell's state is updated to a new state (c_t) by incorporating the current memory candidate value (g_t) via the input gate (i_t) and the previous memory cell state (c_{t-1}) via the forget gate (f_t). This step is described in Equation (5):

$$c_t = i_t \odot g_t + f_t \odot c_{t-1} \tag{5}$$

where $\odot$ stands for element-wise multiplication. This process decides whether to forget the previous memory cell state via the forget gate and regulates the candidate of the current memory cell state via the input gate. In Equation (7), the output gate (o_t) is utilized to compute the memory cell output (h_t) of the LSTM memory block at time t, based on the updated cell state (c_t) as in Equation (6):

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{6}$$
$$h_t = o_t \odot tanh(c_t) \tag{7}$$

The cell output (h_t) is used to predict the label of the current training instance.

C. DEEP LSTM NETWORK FOR CACHING

Although a single LSTM cell has sequential architecture by memorizing former inputs, the input features in a given time slot are only processed by a single non-linear layer before contributing the output. However, in deep LSTM networks the input in a given time slot goes through multiple LSTM layers with different parameters in addition to propagation through time. The real-world caching replacement problem is time-variant and complicated, thus requires the strong ability of the prediction model. To this end, we use the deep network architecture as an expressive and scalable way to incorporate the sequential requests into the caching policy.

We utilize the deep LSTM network for the content caching problem, which is built by stacking multiple LSTM layers. Fig. 6 shows the architecture of the proposed network, which has multiple fully-connected LSTM layers (e.g., 2 layers in the example), and one softmax layer which outputs the prediction result. Specifically, the LSTM layers take the historical sequence as input and output a vector $\mathbf{x}$ with dimension C. Then, we output the vector $\mathbf{x}$ to a *softmax* layer to calculate the probability for each content to arrive. The softmax function is an activation function that transfers the input into probabilities that sum to one. In our network, the input to softmax layer is the output of the last LSTM layer in the stacked LSTM layers, which can be considered as the score for each content to arrive and normalized by a softmax function. The softmax function is defined as follows:

$$\text{softmax}_i = \frac{e^{x_i}}{\sum_{i'=1}^{C} e^{x_i'}} \tag{8}$$

The full connection architecture enables the network to fully exploit the inherent correlations among cells, hence represent the complicated content request pattern better. In order to achieve better performance, we explore two hyperparameters in LSTM network: the number of a hidden layer, and the number of cells in each layer, both of which have potential to influence the performance.

In the deep LSTM network, the content request sequence can be naturally considered as the feature to predict which content to evict when a content out of the local cache arrives. Specifically, each content is represented as one-hot encoding. We can train the model by feeding the "raw" request sequence to the deep LSTM model in an online fashion. It gradually learns to make better caching replacement decisions through the online training process, in which the algorithm try to achieve the optimal caching performance on the historical data. For example, when we feed the latest request sequence to the model, it makes caching decision-based on which content will be least popular according to the long-term and short-term memory. By keeping feeding the latest information to our model, it can better characterize the popularity of different contents. By contrast, algorithms using fixed rules or simplified models are unable to optimize their

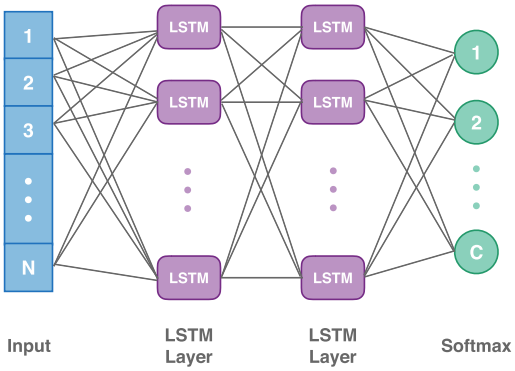


FIGURE 6. Deep LSTM network architecture.

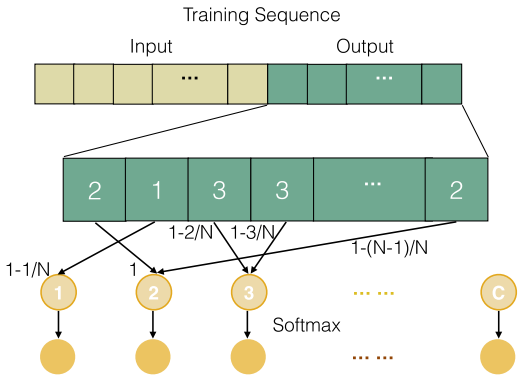


FIGURE 7. Input and output in the training phase.

caching choice based on all available information about the content request sequence. Although a "deep" network is used in our problem, it is quite light-weighted, which converges fast and induces small computation overhead (results are provided in Section IV).

D. APPROXIMATE METHOD FOR MODEL TRAINING

Unlike most sequential learning tasks which predict the next-step instance with the highest probability, whose ground truth can be obtained when the next-step instance arrives, we aim to decide which content to evict from the cache space. Specifically, we want to find the content which is most *unlikely* to arrive among the cached contents. Because we use an approximate method in the training phase, which is not designed for the prediction of next-step arrives, we cannot use F1 and accuracy to show the performance of the proposed framework in the performance evaluation. In this problem, challenges arise in two aspects: On one hand, it is hard to let the neural network output which content should be evicted directly, because this depends on the knowledge of which contents are cached currently. On the other hand, the ground truth for training cannot be obtained in the near future.

In order to meet the first challenge, we define caching priority as the benefit of keeping a content in the cache, and calculate the caching priority of all the contents with LSTM-C. Then we rank the cached contents by the caching priority score and evict the content with the lowest priority score.

We design an approximate method to cope with the second challenge. The optimal strategy is to evict the content which ranks last in order in the future [20]. However, the optimal strategy relies on all the future information, which cannot be obtained in reality. The most commonly used one-hot encoding in prediction tasks is not applicable either, as it cannot reflect the priority of the cached contents. In this way, we design a method that uses a request sequence to approximate the ground truth. Fig. 7 shows the training phase, where $\{c_1, c_2, \ldots, c_M\}$ is the input sequence with length M, and $\{c_{M+1}, c_{M+2}, \ldots, c_{M+N}\}$ is the output sequence of length N used to generate the approximate caching priority. We pay particular attention to the output sequence. Inspired by the optimal strategy of content replacement, we notice that the order of the future contents is important information. We first set the caching priority in the n-th order as $f(n)$, which is monotonously decreasing with n. This is consistent with the fact that a content arriving in the near future should be kept in the local cache. In addition, the number of a content arriving is another important factor, as a content being requested many times should be kept in the local cache. Incorporating the influence of both the order and the number of a content, we compute the overall priority of content c as $W(c)$:

$$W(c) = \sum_{I(c_{M+n}=c)} f(n), \tag{9}$$

where $I(x) = 1$, when x is true. In summary, $W(c)$ is the sum weight of a content in all arrivals.

Note that the neural network training indeed requires looking at the future as the output. But the training is conducted offline, the testing stage does not need future information. Specifically, when the framework is deployed in a real system, it predicts the future using the trained neural network model, which only generates a short delay ($300\mu s$) and does not affect the performance in practice.

Generally speaking, the video request datasets can be divided into two categories. In one category, the popularity of video contents is highly time-variant, and the lifespan of a content is short. For example, video contents like weekly TV shows and TV series burst in views and die down soon. In the other one, the popularity of video contents is stable, and the lifespan of a content is long. For example, most movie contents release long ago, and still attract the fans to watch. In order to characterize the patterns of different video contents, we provide a generalized formulation of $f(n)$ for all possible datasets as:

$$f(n) = 1 - (\frac{n-1}{N})^{\alpha}, \quad \alpha > 0, \tag{10}$$

where α is a parameter that is different among datasets, and N is the output length. In the example depicted in Fig. 7, we set $\alpha = 1$. The priority of content 1 is $1 - \frac{1}{N}$ as it arrives in the second order once. The priority of content 2 is $1 + \frac{1}{N}$

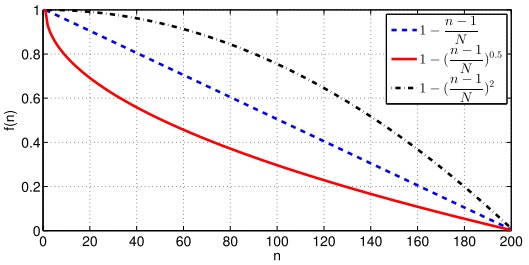


FIGURE 8. Examples of $f(n)$.

as it arrives in the first order ($f(1) = 1$) and the N-th order ($f(N) = \frac{1}{N}$), respectively.

The formulation of $f(n)$ has the following properties: (1) it is monotonously decreasing with n, (2) the descent rate can be adjusted by α. Examples of $f(n)$ under $N = 200$ are depicted in Fig. 8, where a larger α implies that the content frequency is more important for caching priority (e.g., movies), as the values of $f(n)$ are large. A smaller α implies that the content recency is more important for caching priority (e.g., TV shows), as the values of $f(n)$ decrease fast with n. Similarly, the output length N can also reflect the content request patterns. A larger N fits contents with stable request pattern (e.g., movies), because the long-term history corresponds to precise content popularity. A smaller N fits contents with bursty request pattern (e.g., TV shows), because only the short-term history can represent the content popularity. The above insight can be used to determine the parameter setup (i.e., N and α): if the request pattern of most contents in a dataset is bursty, we should set smaller N and α; if the request pattern of most contents is stable, we should set larger N and α.

After deriving the weights of all contents, we employ a *softmax* function with $W(c)$, $c \in C$ as the input and the output probability distribution is denoted as $P(c)$. We train the LSTM-C model in a supervised learning framework, and define the loss function as the cross-entropy error:

$$loss = \sum_{c=1}^{C} P^g(c) \cdot \log(P(c)), \tag{11}$$

where $P^g(c)$ is the approximate priority, and $P(c)$ is the predicted priority of content c. We take the derivative of loss function through back-propagation with respect to all parameters, and update parameters with stochastic gradient descent.

E. ASYNCHRONOUS TRAINING

Considering practical deployment, we implement an asynchronous mechanism between the prediction process and the training process by introducing a training trigger module. In a real content cache system, the number of user requests is very large even in a short-time period. The prediction process is invoked every time when the requested content is not locally cached. However, compared to the prediction process, the training process is more time-consuming and computing-consuming. It may harm the robustness of the system if the training and predicting processes are triggered at the same time. Moreover, frequent training may cause too much computing load to the cache node.

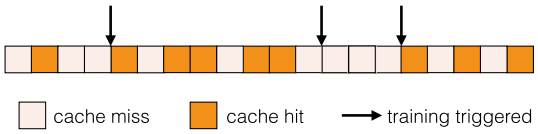


FIGURE 9. Illustration of the training process when $Lag = 3$.

In this way, we employ an *asynchronous* method to decide when to invoke the training process. We define a parameter *Lag* as the training lag threshold. When the number of cache misses reaches *Lag* since last training, a new training process is triggered. Fig. 9 show an example when $Lag = 3$, from which we can see the training frequency is largely lowered. With this mechanism, we decouple the training and the predicting process, and reduce the computing load of the cache node. The benefit of this method is that when the prediction performs good, i.e., few cache misses happening, the prediction model is not updated frequently. When the prediction performance is bad, i.e., many cache misses happening, the prediction is updated frequently until it captures the current request pattern.

F. DYNAMIC CONTENT INVENTORY

The proposed method can be directly applied to the task with static content inventory, thus the size of the inventory is a constant. However, the practical scenario in reality is that the video contents provided by a content provider are usually dynamic. For example, videos out-of-date will be removed from the inventory, and new video contents like newly released talk shows will be added to the inventory. An important issue arises with the dynamic content inventory, as the input length of the model and representation of each bit cannot be changed. In this way, we provide an approach to deal with the issue without much modification to the proposed algorithm.

We calculate the current inventory size, and then set the input size in the model larger than that to reserve void bits for future video contents. As these void bits are not mapped to any contents, they will never be put into the local cache, hence the training process will not be polluted by the void bits. When a content is deleted or out-of-date, the content is removed from the inventory and the corresponding bit is marked as void. As the real-time training goes, the model will rate the content on this bit with lower cache priority. Thus, the void bits can be ranked by the order of the *latest use*, which is the last time it represents a video content that has been deleted currently. The bit with the largest latest use has a relatively low cache priority. Once a new video content arrives (not in the inventory before), it is allocated to the void bit with the largest *latest use* and added to the inventory. This design is to avoid the pollution of the previous deleted

content (the same bit) on the new content. If the new content is popular, the model will capture the real-time pattern, and keep improving the cache priority of the content just as the contents in the inventory. Once the void bits drain, we need to train the model with a larger input size. Since the convergence of the algorithm is fast (the result is provided in Section IV), we can simply use the latest requests to retrain the model.

IV. EXPERIMENT RESULT

We provide a public uniform experiment environment as the platform to compare and evaluate algorithms. We evaluate the performance of LSTM-C on the three datasets with state-of-the-art algorithms introduced in Section II. We further discuss the parameter settings and the training time of LSTM-C.

A. UNIFORM EXPERIMENTAL ENVIRONMENT

Previous work has done experiments of content caching on different real-world datasets or simulated datasets. Considering the heterogeneous environments researchers use in doing such research, the performance of different algorithms is not convenient to compare with each other under the same environment. Moreover, building an environment to test the content caching algorithms is time-consuming. In this way, we aim to provide a uniform experiment platform to evaluate different content caching algorithms on different public datasets conveniently.

Taking advantage of *OpenAI Gym*, which is a toolkit for artificial intelligence research [29], we formalize the content caching problem as a benchmark problem that exposes a common interface. Specifically, we define the state in this problem as follows:

- The cache size.
- The contents that are currently cached.
- The newly requested content.
- The historical requests.

Based on the state, the agent, i.e., the cache system, has to decide the action, i.e., choosing which content to be evicted. After taking the action, the state will transit to a new one, i.e. the requests are served, until a new content which is not cached arrives. This process continues until all the requests in the experiment are served. We provide a uniform experiment environment in [30]. Researchers on this topic can share their results, and compare the performance of algorithms on the uniform experimental environment. To better scale the environment, we allow for multiple datasets, thus researchers can upload and incorporate their own datasets into the environment.

B. PARAMETER SETTINGS IN LSTM-C

Without additional mention, the default parameter settings are as follows: We set the default network layer as 2, and the cell number in each layer as 16. We further set the training lag as 5, and the default input length as 5 for all datasets. With careful tuning, we set the default α as 0.06, 0.1, and 0.125, respectively in the three datasets, and we set the default output length to calculate the label as 100, 200, and 1000, respectively in the three datasets.

C. PERFORMANCE COMPARISON

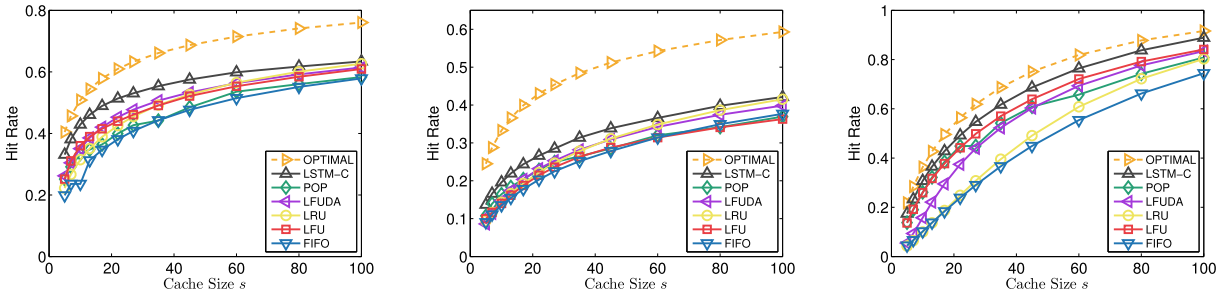
To evaluate the performance of LSTM-C, we compare it with baseline caching algorithms on the three datasets. In each experiment, LSTM-C is trained online aiming to evict the most unpopular contents. According to the previous works [8], [11], [18], we select the five state-of-the-art algorithms introduced in Section II as the comparisons. We also present results of the offline optimal algorithm, which serves as the upper bound of the cache hit rate with complete future information. Fig. 10(a), 10(b), and 10(c) show the cache hit rate of different algorithms with different cache spaces on iQiyi Dataset, iQiyi Small Cell Dataset, and Movielens Dataset. There are three key observations from the result.

First, we find that LSTM-C outperforms the baseline algorithms on cache hit rate in all datasets and cache size settings. In particular, the performance improvement is more significant when the cache size is small. For example, LSTM-C outperforms the best baseline algorithm by 20%∼32% when the cache size is limited in iQiyi Dataset and iQiyi Small Cell Dataset. The closest competing algorithms are LFU under iQiyi Dataset, and Movielens Dataset, and PopCaching under iQiyi Small Cell Dataset, respectively. When the cache space is 10, LSTM-C can outperform baselines by 28%∼ 54%.

Second, we observe that the performance of existing caching algorithms struggle to optimize for different datasets and cache sizes. PopCaching achieves the best performance in *{iQiyi Small Cell Dataset,* $s < 30$*}*, and *{Movielens Dataset,* $s < 22$*}*. LRU achieves the best performance in *{iQiyi Dataset,* $s > 60$*}*, and *{iQiyi Small Cell Dataset,* $s > 50$*}*. LFU achieves the best performance in *{iQiyi Dataset,* $s < 20$*}*, and *{Movielens Dataset,* $s > 22$*}*. LFUDA achieves the best performance in *{iQiyi Dataset,* $20 < s < 60$*}*, and *{iQiyi Small Cell Dataset,* $30 < s < 50$*}*. The reason for the intense competition among the baselines is that these algorithms utilize fixed replacement rules or extract features in advance, while optimization for different datasets requires inherently different strategies. However, LSTM-C is able to automatically learn these strategies and generalize under different scenarios.

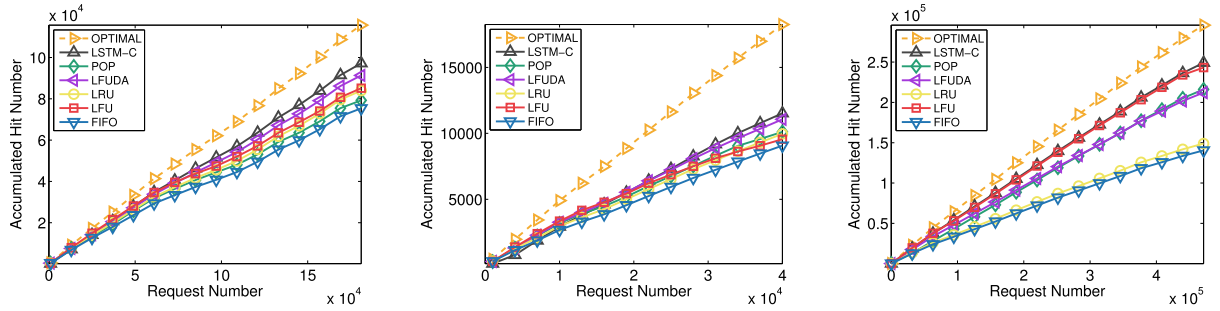
Third, the average hit rate on the iQiyi Small Cell Dataset is significantly lower than on the other two datasets. Specifically, the hit rate when $s = 100$ on the iQiyi Small Cell Dataset is 41%, while the hit rates are over 60% on the other two datasets. Furthermore, the gap between the best algorithm and the optimal solution is also larger. Specifically, the average hit rate of LSTM-C is 61% of the optimal solution on the iQiyi Small Cell Dataset. While the average hit rates are above 80% of the optimal solution on the other datasets. This indicates that edge caching in small regions is more challenging as the group pattern is harder to represent.

With the above observations, we find that the advantages of LSTM-C are manifold: (1) Higher cache hit rate compared to the state-of-the-art method. (2) Better generalization ability



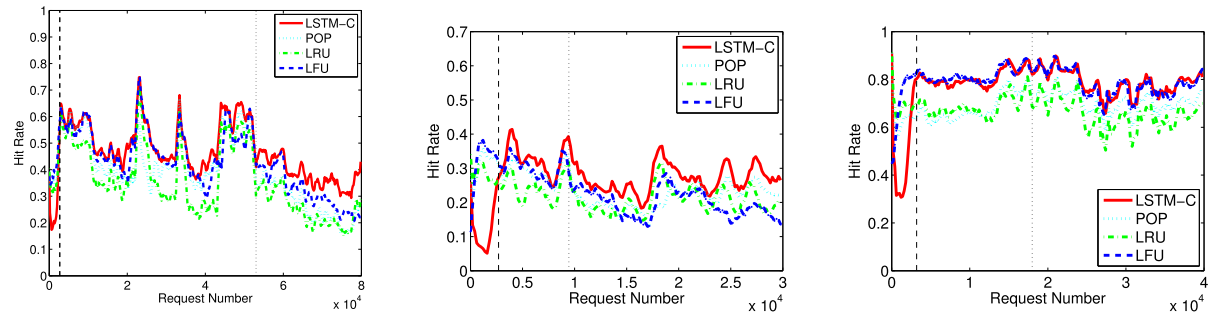
(a) iQiyi (b) iQiyi Small Cell (c) Movielens

FIGURE 10. Cache hit rate on three datasets.



(a) iQiyi (b) iQiyi Small Cell (c) Movielens

FIGURE 11. Accumulated hit number on three datasets.



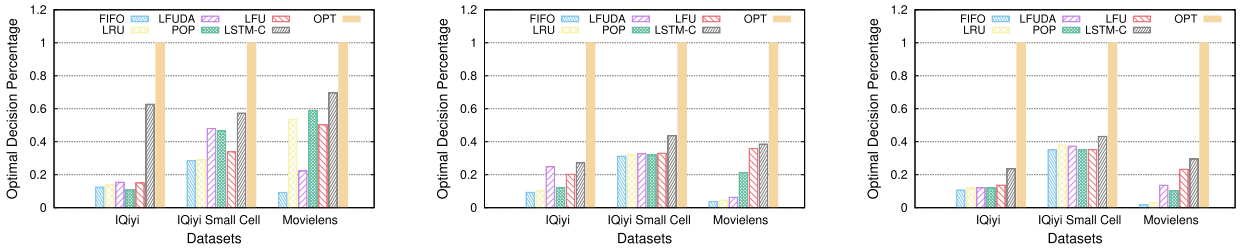
(a) iQiyi (b) iQiyi Small Cell (c) Movielens

FIGURE 12. Convergence on three datasets.

across datasets and cache sizes compared to the state-of-the-art methods. (3) Adaptation to novel scenarios such as edge caching.

We further look into the cache hit number with the content requests in Fig. 11(a), 11(b) ($s = 30$), and 11(c). We notice that in the beginning of the video requests, LSTM-C is the worst among all the algorithms, as LSTM-C is learning-based and needs iteration to achieve relatively good performance. Other algorithms are rule-based and do not need the learning process. After some iteration times, LSTM-C can always achieve the higher cache hit rate compared to other algorithms. Another observation is that the performance of PopCaching in Fig. 11(a) decrease when the request number is large, as the historical request pattern cannot be wiped out. Once the request pattern has changed, the cache hit rate will decrease, causing cache inefficiency.

We investigate the convergence speed of the LSTM-C algorithm by looking into the instant cache hit rate in the request process ($s = 30$). Fig. 12(a), 12(b), and 12(c) show the convergence pattern on the three datasets, where x-axis is the request number, and y-axis is the instant hit rate in the time unit. We pay particular attention to two moments: the time when the instant hit rate of LSTM-C first reaches baselines, and the time when the instant hit rate of LSTM-C first outperforms all baselines. We notice that in different datasets, the first time when LSTM-C reaches baselines are close, which is around 2000 requests (i.e., about 400 training iterations as the training lag is 5). The convergence speed is



(a) Cache size $s = 10$ (b) Cache size $s = 30$ (c) Cache size $s = 60$

FIGURE 13. Probability of optimal decisions on three datasets.

relatively fast. While the time when LSTM-C outperforms all baselines is quite varied, as the long-term patterns are different across datasets. As the hit rate of LSTM-C is low before convergence, we recommend that before applying LSTM-C to online real systems, an offline training process with at least 2000 requests should be performed to obtain the initial model. We also find that when the cache hit rate reaches a stable high level, the loss we defined in optimizing the model approximates zero. This indicates that there exists no underfit or overfit in the model.

Next, we investigate how close the algorithms are with the optimal one. We provide the probability of the algorithms make the optimal decisions in all datasets and different cache sizes in Fig. 13(a), 13(b) and 13(c). We find that in all cases, the LSTM-C algorithm obtains the highest optimal decision probability. If the cache size is larger, it is harder for an algorithm to find the optimal solution, because there exist too many candidates to choose from. However, even when the cache size $s = 60$, LSTM-C has 22%∼42% probability to make an optimal decision, which is much higher than the baseline algorithms. We also notice that the baselines perform differently under different datasets and cache sizes. This also validates that without adaptation, algorithms are hard to satisfy diverse scenarios.

D. PARAMETERS DISCUSSION

Based on the default learning architecture, we sweep a range of neural network hyperparameters to understand the impact that each has on cache hit rate. We choose the iQiyi dataset and set the cache size as $s = 10$. First, using a two fixed hidden layer, we varied the number of cells in each hidden later. Results are presented in Table 1. As shown, performance begins to plateau once the number of cells each exceeds 16, and when the cell number reaches 128, the hit rate starts to decrease.

Next, after fixing the number of cells to 16, we varied the number of hidden layers in the LSTM-C architecture. The result cache hit rate is listed in Table 2. We find that the shallowest network of 1 hidden layer yields the worst performance. Performance remains the plateau when the hidden layer is more than 1. The reason that the performance of the first layer network is poor may be due to the fact that

TABLE 1. Hit rate versus cell number.

Cell Number	Hit Rate
4	0.4293
16	0.4289
32	0.4288
64	0.4290
128	0.4284

TABLE 2. Hit rate versus layer number.

Layer Number	Hit Rate
1	0.4113
2	0.4289
5	0.4288
10	0.4288

the complicated request pattern cannot be well captured with only one layer.

We discuss the parameter selection in the train phase, i.e., how to set the output length for the LSTM-C model, and the cache size is set as $s = 80$. Then, we investigate the output length setup in the three datasets in Table 3. We find that the hit rate increases with the output length at first, and then decreases when the output length reaches a threshold in all datasets. Note that when the output length equals 1, the approximate training method reduces to the commonly used one-hot encoding, i.e., predicting which content will arrive next. We can see that our proposed approximate training method can improve the hit rate by 68%∼75% compared to the one-hot encoding.

Then, we vary α and look into the cache hit rate in Table 4. The relationship between hit rate and α bears similar pattern as with output length, which is a single-peak function. We find it important to set α as it affects the hit rate greatly.

Recall the analysis of request pattern we provide in the approximate method, we also observe the principle to set the parameters N, and α. Movielens is a dataset with all the contents are movies, and the request pattern of each content is relatively stable. While most contents in the iQiyi dataset are TV shows and talk shows, which have burst request pattern.

TABLE 3. Hit rate versus output length.

Output Length	Hit Rate		
	iQiyi	iQiyi Small Cell	Movielens
1	**0.368**	**0.229**	**0.470**
10	0.567	0.340	0.741
100	**0.619**	0.389	0.792
200	0.606	**0.402**	0.797
500	0.602	0.391	0.800
1000	0.592	0.390	**0.809**
1500	0.556	0.375	0.808
5000	0.377	0.216	0.701

TABLE 4. Hit rate versus α.

α	Hit Rate		
	iQiyi	iQiyi Small Cell	Movielens
0.010	0.588	0.356	0.773
0.060	**0.619**	0.396	0.767
0.100	0.608	**0.402**	0.768
0.125	0.598	0.398	**0.809**
0.250	0.601	0.394	0.794
0.500	0.601	0.391	0.779

TABLE 5. Hit rate versus input length.

Input Length	Hit Rate
1	0.4243
3	0.4252
5	0.4291
10	0.4289
20	0.4294
50	0.4284

To this end, the parameters N and α of Movielens should be set with larger values to obtain precise historical information, and those of iQiyi should be set with smaller values to capture the recent pattern. In addition, the user request pattern in iQiyi Small Cell dataset is more static, as the regional users tend to watch specific contents. Thus, the parameters in iQiyi small cell dataset are larger than those in iQiyi dataset. The experiments in Table 3 and Table 4 provide consistent results with the analysis.

Next, after fixing other parameters, we investigate the input length. We find that the hit rate increases with input length, after the input length reaches 5, the maximum of hit rate is achieved. We see that increasing the input length in the LSTM-C model more than 5 does not give performance improvements. Consider that processing time will grow dramatically if we increase the input length, we take 5 as the default input length in the optimal setting.

E. PROCESSING TIME ANALYSIS

To measure the overhead of generating caching algorithms using LSTM-C, we profile LSTM-C's training and predicting processes. The running time measurement was conducted on a laptop with Intel Core i7-6700HQ CPU @ 2.6GHz*4. Note that we train the neural network using CPU as there may be no available GPU on cache nodes. Table 6 shows the predicting time of all algorithms. As we can see in Table 6, the average predicting time for LSTM-C is 300 μs. Although the time consumption of LSTM-C is larger than the baselines, the predicting time is small enough in such a system. In addition, training the algorithm of requests in two weeks (about 7,300,000 requests) online requires approximately 1,460,000 training iterations (training processes every 5 cache misses). The total time for training is 3.5 hours, where each training iteration takes about 8.6 *ms*. To this end, our algorithms can be applied to most human-in-the-loop scenarios, like video watching, web browsing, and picture browsing, where the bottleneck of running speed is not the caching algorithm and the request frequency is relatively low. We also admit that the LSTM-C algorithm cannot be applied to scenarios where the running speed of the caching algorithm is very important, e.g., RAM cache in embedded system [31].

V. RELATED WORK

A. CONTENT CACHING

Content caching has been widely adopted in the Internet. A large number of previous works focus on the network architecture optimization for content caching. Reference [32] designed a low-complexity algorithm for caching in networks with arbitrary topologies. Reference [5] studied the techniques for caching in the future 5G mobile network. Reference [33] introduced the concept of content caching in an information-centric network, taking into account the information about the locations of caches. A delivery path selection approach is also proposed. Currently, most deployed caching algorithms in real systems are still FIFO [7], LFU, LRU, and their variations [8]. Some other works design data-driven cache algorithms. Researchers in [18] proposed a popularity based learning algorithm for cache replacement, and proved the learning regret is sublinear with the request number. Reference [12] captured the regional video preference and used a matrix factorization method to decide the video contents placement. In [13], propagation information of content over social media is utilized to optimize content replication strategies. Reference [11] conducted extensive measurement on a video request dataset, and designed cache algorithm based on the derived insights. Reference [34] conducted measurement on a dataset of TV series requests, and designed a learning algorithm based on the insights that users tend to watch TV series sequentially. Above works use data analysis to improve cache performance, with additional information on content type (TV series, social video), regional information or user preference. These algorithms require specific information and data pre-processing, hence may be

TABLE 6. Comparison of running speed. Results are shown in average time for once content replacement decision.

Algorithms	LSTM-C	POP	LFU	LFUDA	LRU	FIFO
Processing time (μs)	300	23	11	17	8	6

hard to generalize. Our previous work has proposed a deep-learning based solution to understand the request patterns in individual base stations and accordingly make intelligent cache decisions [35]. We consider the previous design as a sub-module in the proposed framework of this paper.

B. EDGE CACHING FOR SMALL REGION

A newly-emerged scenario of content caching is edge caching, which is a key feature in the 5G network [36]–[39]. An example is that contents can be cached in the small cell base stations with storage capacity. Reference [11] conducted data measurement on a real dataset to prove the effectiveness of edge caching, and designed a caching strategy based on the measurement insights. Reference [40] investigated the challenges of caching with a small population, i.e., making timely estimates of content popularity and inferring popular content from a small sample. They designed an age-based threshold policy to estimate the varying popularity to alleviate the limitations. Reference [41] optimized the profit of service providers, content providers, and users with contract theory. Reference [42] designed an approximation algorithm to maximize the content requests served by the small cell. Reference [43] introduced an efficient cooperative caching scheme among small cells. References [19], [44] employed multi-armed bandit method to decide the caching policy in the range of small Base Stations. Small cell caching is different from caching in representative caching scenarios, as the request number is small and regional pattern is important. In order to justify the generalization of caching algorithms, we compare the algorithms on a small cell caching dataset in the experiment section.

C. LSTM FOR SEQUENTIAL LEARNING

As a type of recurrent neural network with a more complex computational unit, LSTM network [16] has shown its superior ability to preserve sequence information over time, and obtained an excellent result on many sequence modeling tasks. Reference [45] utilized the LSTM network to address the ad recommendation problem. LSTM network has also been widely adopted in the natural language processing problems [46], [47]. Reference [48] used a tree-structured LSTM network to improve semantic representation. LSTM has also been applied to speech recognition tasks, and achieved great performance [27], [28], [49]. For our problem, the users' video requests form a sequence by nature, and we employ the LSTM network to model the request sequence in order to make an efficient caching decision. To the best of our knowledge, we are the first to apply the LSTM network to the content caching problem.

VI. CONCLUSION

We propose an edge-assisted intelligent caching replacement algorithm named LSTM-C. LSTM-C requires no data pre-processing, pre-programmed model or additional information to decide the cache replacement. Instead, it learns to make replacement decision by memorizing the former request sequence with the built-in memory cells. We propose a deep network architecture, and design an approximate training method. We set multiple conditions in the experiment, and LSTM-C outperforms state-of-the-art algorithms on all conditions, which validate the effectiveness and generalization ability of our algorithm.

Deploying in practice: In our current experiment, the LSTM-C runs under the OpenAI-Gym environment. This setup offers the advantage over deployment in real-world caching system for research purpose, as it is convenient to compare the performance of different methods and hyper-parameter settings. However, based on the analysis above, LSTM-C can be deployed in a real-world caching system conveniently. First, the LSTM-C approach requires no modification of the caching server. As introduced in Section III, the LSTM-C-enabled caching node only need the prediction module and the training module additionally (the other modules are already provided in the current cache server). The two modules only depend on installation of the python development environment and the tensorflow library. The running of LSTM-C requires no GPU, as we have examined that the processing time with CPU can satisfy the real-world request sequence.

The potential to improve edge caching: The large scale model has a global view of the content requests, compared to the small cell caching that only have limited requests. Thus, the global caching model can converge faster with more requests. It also has a better understanding of the video popularity. While the model in the small cell certainly has the most accurate information about the regional caching. We plan to improve the performance of the small cell caching by incorporating the global model in future work.

ACKNOWLEDGMENT

(Cong Zhang and Haitian Pang contributed equally to this work.)

REFERENCES

[1] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2016—2020 White Paper*, Cisco, San Jose, CA, USA, 2016.

[2] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, 2010.

[3] Accessed: Aug. 1, 2019. [Online]. Available: http://www.iqiyi.com/

[4] Accessed: Dec. 31, 2017. [Online]. Available: https://en.wikipedia.org/wiki/iqiyi

[5] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[6] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[7] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li, "Ripq: Advanced photo caching on flash for Facebook," in *Proc. FAST*, 2015, pp. 373–386.

[8] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of Facebook photo caching," in *Proc. 24th ACM Symp. Oper. Syst. Princ.*, 2013, pp. 167–181.

[9] S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.

[10] J. Wang, "A survey of Web caching schemes for the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 5, pp. 36–46, Oct. 1999.

[11] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, "Understanding performance of edge content caching for mobile video streaming," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 5, pp. 1076–1089, May 2017.

[12] S. Dernbach, N. Taft, J. Kurose, U. Weinsberg, C. Diot, and A. Ashkan, "Cache content-selection policies for streaming video services," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[13] Z. Wang, W. Zhu, X. Chen, L. Sun, J. Liu, M. Chen, P. Cui, and S. Yang, "Propagation-based social-aware multimedia content distribution," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 9, no. 1s, p. 52, 2013.

[14] H. Gao, W. Huang, Y. Duan, X. Yang, and Q. Zou, "Research on cost-driven services composition in an uncertain environment," *J. Internet Technol.*, vol. 20, no. 3, pp. 755–769, 2019.

[15] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "QoS prediction for service recommendation with deep feature learning in edge computing environment," *Mobile Netw. Appl.*, pp. 1–11, Apr. 2019.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating content management techniques for Web proxy caches," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 27, no. 4, pp. 3–11, 2000.

[18] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[19] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE ICC*, Jun. 2014, pp. 1897–1903.

[20] R. L. Mattson, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, no. 2, pp. 78–117, 1970.

[21] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, p. 19, Jan. 2016.

[22] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal proxy cache allocation for efficient streaming media distribution," in *Proc. IEEE INFOCOM*, Jun. 2002, pp. 1726–1735.

[23] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1310–1319.

[24] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM HotNets*, 2016, pp. 50–56.

[25] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. ACM SIGCOMM*, 2017, pp. 197–210.

[26] C.-L. Fan, J. Lee, W.-C. Lo, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "Fixation prediction for 360° video streaming in head-mounted virtual reality," in *Proc. ACM NOSSDAV*, 2017, pp. 67–72.

[27] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE ICASSP*, May 2013, pp. 6645–6649.

[28] A. Graves, N. Jaitly, and A.-R. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Proc. IEEE ASRU*, Dec. 2013, pp. 273–278.

[29] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*. [Online]. Available: https://arxiv.org/abs/1606.01540

[30] Accessed: Dec. 31, 2017. [Online]. Available: https://github.com/unicache/unicache-env

[31] P. Panda, G. Patil, and B. Raveendran, "A survey on replacement strategies in cache memory for embedded systems," in *Proc. IEEE DISCOVER*, Aug. 2016, pp. 12–17.

[32] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[33] V. G. Vassilakis, M. F. Al-Naday, M. J. Reed, B. A. Alzahrani, K. Yang, I. D. Moscholios, and M. D. Logothetis, "A cache-aware routing scheme for information-centric networks," in *Proc. IEEE CSDNSP*, Jul. 2014, pp. 721–726.

[34] W. Hu, Y. Jin, Y. Wen, Z. Wang, and L. Sun, "Toward Wi-Fi AP-assisted content prefetching for an on-demand TV series: A learning-based approach," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 7, pp. 1665–1676, Jul. 2018.

[35] H. Pang, J. Liu, X. Fan, and L. Sun, "Toward smart and cooperative edge caching for 5G networks: A deep learning based approach," in *Proc. IEEE/ACM IWQoS*, Jun. 2018, pp. 1–6.

[36] Y. Yin, Y. Xu, W. Xu, M. Gao, L. Yu, and Y. Pei, "Collaborative service selection via ensemble learning in mixed mobile network environments," *Entropy*, vol. 19, no. 7, p. 358, 2017.

[37] B. Han, S. Wong, C. Mannweiler, M. R. Crippa, and H. D. Schotten, "Context-awareness enhances 5G multi-access edge computing reliability," *IEEE Access*, vol. 7, pp. 21290–21299, 2019.

[38] W. He, Y. Su, X. Xu, Z. Luo, L. Huang, and X. Du, "Cooperative content caching for mobile edge computing with network coding," *IEEE Access*, vol. 7, pp. 67695–67707, 2019.

[39] N. Wang, G. Shen, S. K. Bose, and W. Shao, "Zone-based cooperative content caching and delivery for radio access network with mobile edge computing," *IEEE Access*, vol. 7, pp. 4031–4044, 2019.

[40] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[41] T. Liu, J. Li, F. Shu, M. Tao, W. Chen, and Z. Han, "Design of contract-based trading mechanism for a small-cell caching system," *IEEE Trans. Wireless Commun.*, vol. 16, no. 10, pp. 6602–6617, Oct. 2017.

[42] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3665–3677, Oct. 2014.

[43] Q. Li, W. Shi, X. Ge, and Z. Niu, "Cooperative edge caching in software-defined hyper-cellular networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2596–2605, Nov. 2017.

[44] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, "Learning distributed caching strategies in small cell networks," in *Proc. IEEE ISWCS*, Aug. 2014, pp. 917–921.

[45] D. Kong, F. Wu, S. Tang, and Y. Zhuang, "Ad recommendation for sponsored search engine via composite long-short term memory," in *Proc. ACM MM*, 2016, pp. 416–420.

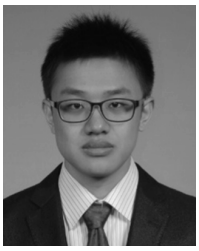
[46] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," in *Proc. AAAI*, 2016, pp. 2741–2749.

[47] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proc. 13th Annu. Conf. Int. Speech Commun. Assoc.*, 2012, pp. 194–197.

[48] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," 2015, *arXiv:1503.00075*. [Online]. Available: https://arxiv.org/abs/1503.00075

[49] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," 2014, *arXiv:1402.1128*. [Online]. Available: https://arxiv.org/abs/1402.1128

CONG ZHANG received the M.Sc. degree from Zhengzhou University, Zhengzhou, China, in 2012, and the Ph.D. degree from Simon Fraser University, Canada, in 2018. He is currently an Associate Researcher with the School of Computer Science, University of Science and Technology of China. His research interests include multimedia communications and cloud/edge computing.



HAITIAN PANG received the B.E. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2014, and the Ph.D. degree from the Department of Computer Science, Tsinghua University. He is currently a Researcher with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His research interests include edgecomputing, deep learning for network, cellular-WiFi networking, video streaming system design, and mobile networking optimizations.

JIANGCHUAN LIU received the B.Eng. degree *(Cum Laude)* from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from the Hong Kong University of Science and Technology, in 2003. He is currently a Full Professor (with University Professorship) with the School of Computing Science, Simon Fraser University, British Columbia, Canada. He is also an NSERC E. W. R. Steacie Memorial Fellow. He is a Steering Committee Member of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He was a co-recipient of the ACM Multimedia Best Paper Award, in 2012, the ACM TOMCCAP Nicolas D. Georganas Best Paper Award, in 2013, and the Test of Time Paper Award of the IEEE INFOCOM, in 2015. He is an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE TRANSACTIONS ON BIG DATA, and the IEEE TRANSACTIONS ON MULTIMEDIA.

SHIZHI TANG is currently pursuing the bachelor's degree with the Department of Computer Science and Technology, Tsinghua University, China. His research interests include the optimization of machine learning from a system approach, including heterogeneous architectures, high performance communications, and compiling technologies.

RUIXIAO ZHANG received the B.E. degree from the Electronic Engineering Department, Tsinghua University, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology. His research interests include the area of content delivery networks, the optimization of multimedia streaming, and reinforcement learning.

DAN WANG received the B.Sc. degree in computer science from Peking University, Beijing, China, the M.Sc. degree in computer science from Case Western Reserve University, Cleveland, OH, USA, and the Ph.D. degree in computer science from Simon Fraser University, Vancouver, BC, Canada. He is currently an Associate Professor with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His current research interest includes the Internet architecture and QoS, smart buildings, and green computing.

LIFENG SUN received the B.S. and Ph.D. degrees in system engineering from the National University of Defense Technology, Changsha, China, in 1995 and 2000, respectively. He joined the Department of Computer Science and Technology (CST), Tsinghua University (THU), Beijing, China, in 2001, where he is currently a Full Professor. His research interests include networked multimedia, video streaming, 3-D/multiview video coding, multimedia content analysis, multimedia cloud computing, and social media. He has authored or coauthored more than 100 high quality articles and one chapter of a book. He was a recipient of the Annual Best Paper Award of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, in 2010, the Best Paper Award of the ACM Multimedia, in 2012, and the Best Student Paper Award of Multimedia Modeling, in 2015, and the IEEE Multimedia Big Data, in 2017.