IntellCache: An Intelligent Web Caching Scheme for Multimedia Contents

Nishat Tasnim Niloy
Institute of Information Technology
University of Dhaka
bsse0723@iit.du.ac.bd

Md. Shariful Islam
Institute of Information Technology
University of Dhaka
shariful@iit.du.ac.bd

***Abstract*—The traditional reactive web caching system is getting less popular day by day due to its inefficiency in handling the overwhelming requests for multimedia content. An intelligent web caching system intends to take optimal cache decisions by predicting future popular contents (FPC) proactively. In recent years, a few approaches have proposed some intelligent caching system where they were concerned about proactive caching. Those works intensified the importance of FPC prediction using the prediction models. However, only FPC prediction may not help to get the optimal solution in every scenario. In this paper, a technique named IntellCache has been proposed that increases the caching efficiency by taking a cache decision i.e. content storing decision before storing the predicted FPC. Different deep learning models such as- multilayer perceptron (MLP), Long short-term memory (LSTM) of Recurrent Neural Network (RNN) and ConvLSTM a combination of LSTM and Convolutional Neural Network (CNN) are compared to identify the most efficient model for FPC. The information on the contents of 18 years from the MovieLens data repository has been mined to evaluate the proposed approach. Results show that this proposed scheme outperforms previous solutions by achieving a higher cache hit ratio and lower average delay and thus, ensures users' satisfaction.**

***Index Terms*—Intelligent Web Caching, Proactive Caching, Deep Learning, Content Popularity, Cache Decision**

I. Introduction

Owing to the abundance of internet facilities, the requests for the multimedia contents are increasing exponentially day by day. Struggling with this expanding internet traffic, the traditional web caching system often fails due to its inefficiency [1]. An intelligent web caching system makes less expensive cache decisions by analyzing the web content continuously. It offers proactive caching by predicting the future popular contents (FPC) based on the popularity dynamics of the users' interest in past usages [2], [3]. While the network is flooded by various kinds of information, the bandwidth usage can be maximized and the response time can be minimized by caching contents proactively [4]. Since, with the advances of data mining techniques, the deep learning models have achieved higher prediction accuracy, those models are used in predicting FPC [5].

Caching reduces the content retrieval time by storing the contents at the nodes and result in less internet traffic, latency, and load from the internet server [6]. When the internet user requests for any content, these contents are generally fetched from nearby proxy servers (PSs) or base stations (BSs) rather than remote data servers. These caching nodes use a web caching scheme to store the contents that are accessed recently or frequently [7], [8]. Some popular caching algorithms are used in traditional web caching schemes - such as in First-in-First-out (FIFO), Least-Recently-Used (LRU) algorithm, the least recently accessed objects are removed as long as there is not enough space for the new objects. In Least-Frequently-Used (LFU) algorithm, the objects with the least number of access are replaced for making sufficient space. The Most-Recently-Used (MRU) algorithm removes the most recently used resource first. Other than these three algorithms, there is another algorithm named Segmented LRU(SLRU). This algorithm considers both the popularity of use and access time [4], [9]. Though these algorithms are very efficient, these are only applicable in reactive web caching.

In recent years, the proactive caching system has become a topic of concern. The responsiveness of the caching nodes helps to increase the customers' satisfaction along with the system efficiency. A proactive approach can help to obtain a better service with backhaul savings up to 22% as well as it helps to achieve a ratio of satisfied users up to 26% [7]. The related studies regarding proactive web caching approaches have used several artificial neural networks for prediction problems named - Multilayer Perceptron (MLP), Recurrent Neural Network (RNN), Long short-term memory (LSTM), Convolutional neural network (CNN) or their combinations [10]–[12]. These deep learning models are used to analyze the users' requests collected from the PSs or BSs or cache node. Based on the result of these deep learning models, the FPC is predicted for the future usages. Through FPC prediction, those approaches intended to maximize the cache hit and reduce the access delay.

It is very difficult to find out a deep learning model to trade-off between prediction accuracy and resource consumption. Both the model configuration and the hyperparameter tuning are responsible to select the best prediction model. Besides, during the storing of FPC, the cache decision is another matter of concern. An efficient caching decision helps to the maximum utilization of the cache node of a PS or BS [13].

An intelligent web caching scheme provides an optimal solution to handle the users' requests by minimizing internet traffic. Hence, they incorporated a proactive mechanism that uses deep learning models. Designing a good prediction model

978-1-7281-9331-1/20/$31.00©2020IEEE

is difficult because it needs to find out the model configuration (the type of layers, depth of networks) hyperparameters tuning, activation functions and so on. Moreover, it requires a very high computation resource to process the big data while the caching scheme needs to be faster [14], [15]. Therefore, the prediction model must trade-offs between prediction accuracy and resource consumption. Besides, the previously proposed intelligent web caching schemes prioritize the FPC prediction and store those based on cache node capacity only. Nevertheless, just storing the FPC may not always provide the optimal solution. When the caching approach does not follow an efficient cache decision during storing FPC, the heavier contents occupy the cache node faster while leaving a good number of lighter candidate contents. In order to maximize the utilization of the cache node, a proper caching decision is essential. IntellCache is a mechanism that provides a solution based on FPC prediction and cache decision. The main contributions are as follows:

- Deep learning model identification by comparing the performances of MLP, LSTM, and ConvLSTM to predict the Future Popularity Contents (FPC).
- A cache decision algorithm that provides an optimal solution to store the predicted FPC based on the cache node size.
- Performance analysis of the proposed solution based on the cache metrics, i.e. cache hit, the average delay.

II. Related Works

The idea of intelligent web caching using deep learning models has been coined very recently. This helps to efficiently handle the usage of bandwidth consumption, server load, and latency. Among those approaches, most of the solutions have used a machine learning model to implement the idea. As for prediction models, Classification and Regression Trees (CART), Multivariate Adaptive Regression Splines (MARS), Random Forest (RF), and TreeNet (TN) have been used to improve the caching system [16]. Support vector machine (SVM) and a decision tree (C4.5) have been incorporated in [17], to improve the conventional Least-Recently-Used (LRU), Greedy-Dual-Size (GDS), and Greedy-Dual-Size-Frequency (GDSF). Naïve Bayes classifiers have been used to predict the probable contents to increase the hit ratio and byte hit ratio in [18]. Along with the deep learning approach, the Blockchain mechanism is combined to provide a more secure approach in the solution proposed in [19]. However, all these approaches are reactive approaches.

Proactive caching has become a popular term in the last decade. The notion of proactive resource allocation has come up in [20] to increase the predictability of user behavior for load balancing. In the proactive caching, predicting FPC gets the top priority. Since the popularity of the contents is very subjective as they depend on various factors and dynamically change very frequently, a deep learning approach is considered to be a better way to obtain the prediction accuracy [5], [21]. Another solution proposed an architecture that provides a proactive caching architecture for 5G wireless networks using machine learning tools for content popularity predictions to process a huge amount of data [22]. In [23] a solution was proposed where stacked autoencoders were used to predict the content's popularity. Some algorithms were proposed to classify the contents based on popularity for mobile edges or multi-access edges to ensure better internet service [11], [12], [24].

Hence, some of the related solutions tried to propose a better web caching solution for reactive approaches using different machine learning models. Other solutions are about predicting FPC using deep learning models to cache the contents proactively [25], [26]. However, to the best of the authors' knowledge, no previous approach offered a solution that predicts FPC using deep learning models and stores those contents using an efficient cache decision.

III. Proposed Methodology

IntellCache is an approach that uses deep learning models to predict FPC proactively and stores the FPC efficiently based on the limited capacity of the caching nodes. The methodology of this approach can be divided into four major parts- (1) Data collection based on users' requests in the BS, (2) Prediction model identification, (3) FPC prediction using trained model in the Cloud Data Centre (CDC) and (4) Cache decision for the predicted FPC in the BS. Figure 1 illustrates the overview of the system model. In the following sub-sections, the details of the four parts are described.

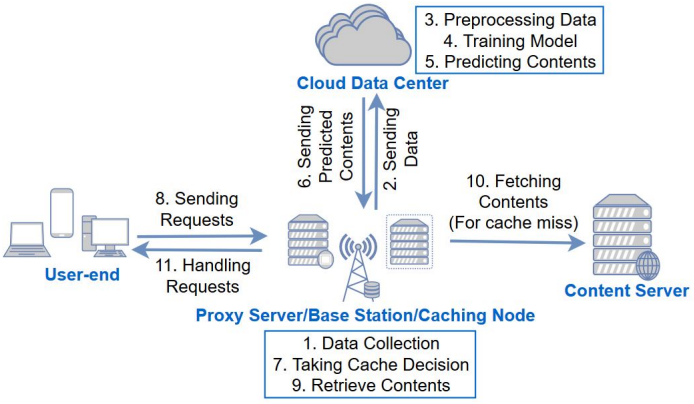


Fig. 1: System Model of the Proposed Approach

A. Data Collection

The end-users send requests for any multimedia content and the PS or BS is responsible to handle those requests. In this scenario, it also collects information about the requests. For any cache miss, it redirects the request to the respective content server and keeps a copy of the request. For every t amount of time, the information about the requested contents is collected, so that, the FPC of the next t amount of time depends on this information. The collected contents can be denoted as, $C = \{C_1, C_2, C_3, ..., C_n\}$, where n is the total number of contents. After every t time, the BS sends the collected data C to the cloud data center for the next steps.

B. Prediction Model Identification

This portion is handled by the CDC. It can be divided into three steps - metadata collection, feature selection, and model training.

1) Metadata Collection: The CDC collects the metadata of the contents from the content server. For any new content or updates, it always communicates with the origin server. This metadata contains information about the content feature.

2) Feature Selection: Features of the contents are extracted from the metadata. The set of features can be denoted as, $F = \{F_1, F_2, F_3, ..., F_n\}$, where n is the total number of features. All the features having non-numerical values are neglected for the simpler analysis. The importance of the rest of the features is measured through the correlation matrix. The higher the score in the matrix, the higher that feature can be useful to the model. Therefore, from F , m no of features are qualified, where $0 < m \leq n$ based on their scores.

3) Model Training: The preprocessed data have been fitted to different deep learning models to find a suitable one. These data are collected from the BS. Previous studies have already established that it is very much difficult to find the best prediction model considering their configurations or hyperparameters [11], [12]. Performance comparisons were made on three types of deep learning models- MLP, LSTM and ConvLSTM. Based on the Mean Square Error (MSE) and Root Mean Square Error (RMSE) the accuracy of the models is compared. The configurations of the models are tuned up to a fixed range to get the optimal result. Afterward, CDC saves the optimal model to predict the FPC.

Algorithm 1 FPC Prediction

Input : Log of cache content C, Feature List F
Output : Future popular content List C'

function PREPAREFUTUTEPOPULARCONTENTLIST
  Retrieve the trained model M
  for each of the F **do**
    if $C.f$ is not in F **then**
      Drop($C.f$)
  for each of the input contents C **do**
    if c has missing value **then**
      remove(c)
  ▷ Predicting request count (rc)
  $C.rc \leftarrow M.predict(C)$
  $C' \leftarrow merge(C.rc, C)$
  ▷ Sorting C' based on $C'.rc$ in descending order
  $C' \leftarrow Sort(C')$

C. FPC Prediction

In Algorithm 1, how the FPC from the trained models is predicted is instructed. After getting the collected data, all the non-numerical and lowly scored features, i.e, title, overview, tagline, production companies and so on, get dropped. Furthermore, the data containing the missing values gets filtered. Then, the saved model has to be retrieved and the preprocessed data needs to be fit to the model. The prediction model provides the predicted request count values to decide the FPC.

Algorithm 2 Content Caching Decision

Input : Future popular content List C', Cache memory size S_{cache}, PWF_b
Output : Content List for caching C''

function STORECACHECONTENT ▷ Defining thresholds
  $avg_s \leftarrow average(C'.size)$
  $avg_{cr} \leftarrow average(C'.rc)$
  $T_s \leftarrow 1.5 * avg_s$
  $T_{rc} \leftarrow 1.5 * avg_{cr}$
  $mem\_limit \leftarrow S_{cache}/2$
  $con\_mem \leftarrow 0$
  ▷ storing contents
  for each of the popular content List C' **do**
    $\alpha \leftarrow 0.025$
    $PWF \leftarrow \alpha \times c.a\_rc + (1 - \alpha) \times PWF_b$
    if $S_{cache} > c.size + con\_mem$ **then**
      if $c.size < T_s$ **then**
        $con\_mem \leftarrow con\_mem + c.size$
        $C'' \leftarrow store(c)$
      else
        if $c.rc + c.PWF > T_{rc}$ and $c.size < mem\_limit$ **then**
          $con\_mem \leftarrow con\_mem + c.size$
          $C'' \leftarrow store(c)$

D. Caching Decision

The cache node mainly takes the decision based on its storage capacity. During contents storing, two criteria of the contents have been considered- comparative size and comparative request count. Both of this threshold is dynamic as these are dependent on the average size of the contents of time t and the caching capacity of BS. Again, the popularity of the contents is directly proportional to the timestamp of the corresponding requests. Hence, following an exponentially weighted moving average (EWMA), a popularity weight factor (PWF) of the contents can be calculated which is used to signify the thresholds. In Equation 1, The calculation of PWF is stated-

$$PWF_t = \alpha \times a\_rc_t + (1 - \alpha) \times PWF_{t-1} \quad (1)$$

Here, PWF_t indicates the estimated request count at time t, a_rc_t is the current request count at time t. α is the tuning parameter and the value is considered as 0.025.

- BS intends to ignore the comparatively heavier contents, i.e., contents that require more space based on size threshold. The comparative size of a content depends on the average size of all the predicted contents. When the size of a content is more than the one and a half of the average content size, BS checks if that content fulfills the second criteria which is explained in the next point. The

size threshold to handle such a situation is explained in Equation 2.

$$T_s = \beta \times (\frac{1}{n} \times \sum_{i=1}^{n}(s_i)) \quad (2)$$

Here, s indicates the size of the contents, n is the total number of the predicted contents and T_s is the size threshold. The value of β is considered as 1.5.

- When content fails to fulfill the first criteria, the request count threshold is considered. This one depends on the comparative request count of that content. If its request count is more than the one and a half of the average request count, BS stores it in spite of its heavy size. However, if its size exceeds half of the storage size of BS, the content is filtered despite its request count. The comparative request count of content depends on the average request count of the total contents. Equation 3 shows how the request count threshold is measured.

$$T_{rc} = \beta \times (\frac{1}{n} \times \sum_{i=1}^{n}(rc_i)) \quad (3)$$

Here, rc indicates the request count of the contents, n is the number of the predicted contents and T_rc is the request count threshold. The value of β is considered as 1.5.

When the request count threshold is considered to check if a content should be stored or not, the corresponding PWF is also taken into account. As the value of the request count is changing with the increasing time, the PWF will be added to the request count before comparing it with the T_{rc}. Algorithm 2 shows how to store the FPC based on the above-mentioned criteria. When each of the criteria is met by a content, BS stores it finally.

IV. Performance Evaluation

The proposed system has been implemented to evaluate the performance. A simulation-based environment has been created to do the experiments. Besides, a very popular dataset is used in the process that has been used in previous works [11], [12]. In the following subsections, the details are described.

A. Environment Setup

Python(3.6) language is used for experimentation. Panda is used for preprocessing the raw data. Tensorflow (2.1) and Keras (2.3) is used for developing deep learning models. The configuration of the PC is- CPU Processor: Intel(R) Core(TM) i7-7700 CPU @3.60GHz, RAM: 16.0 GB, graphics card: NVIDIA GeForce GTX 1070 and operating system: Windows 10 Pro - 64 bits.

B. Dataset Description

The Movielens Dataset [27] has been used for the experiment. The log of all the movies' information, ratings, the timestamp for accessing the content from the year 2000 to 2017 is collected. More than 673K data of the contents have been used to train the prediction model. Those data contain the information more than 4500 movies. Considering the dataset is from a single service provider, the amount of the user requests is smaller for a short time period. Besides, similar works on proactive caching have also worked on this dataset and took yearly data for a single time frame [11], [12].

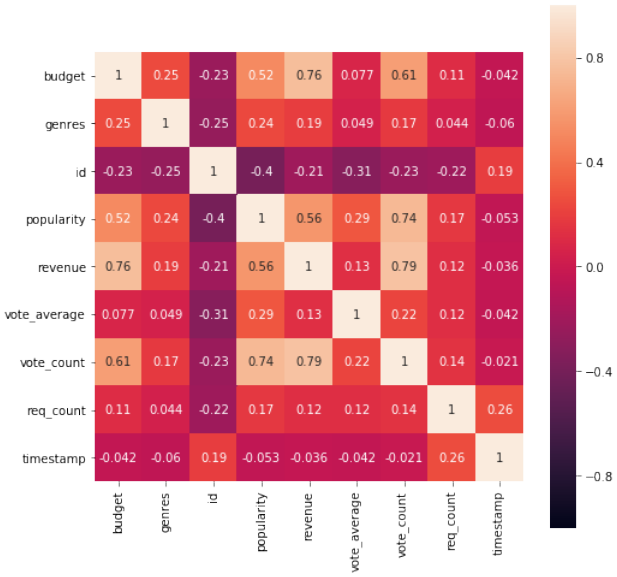


Fig. 2: Heat Map of the Correlation among the Features

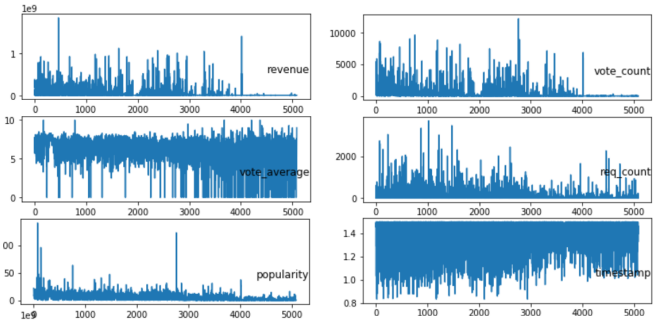
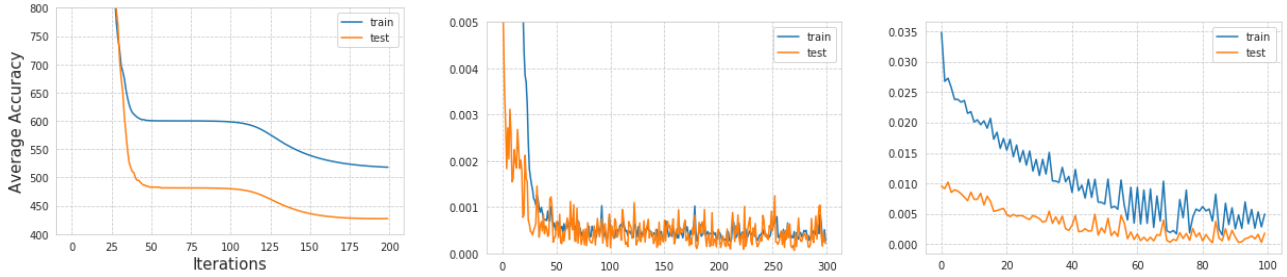


Fig. 3: Distribution of the Features

C. Feature Selection

The non-numerical and missing value containing features have been discarded to avoid the computational complexity. Among 26 features found from the dataset, 8 features are numerical that does not contain any missing values. The distribution of the features is shown in Figure 3. Since the service provider is the same, the video quality should not vary. Therefore, the runtime of a movie is considered as its size. The correlation among the features is depicted in Figure 2. Among all the features, timestamp has the most correlation with the request count. Along with timestamp, popularity, revenue, vote_average, and vote_count is selected for their high correlation to req_count.



(a) MLP Model (b) LSTM Model (c) ConvLSTM Model

Fig. 4: Average Performances of different deep learning models for FPC Prediction

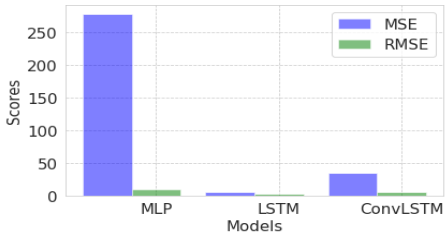


Fig. 5: Average MSE and RMSE of the Deep Learning Models

D. FPC Storing

According to the Section III-A, the FPC for the next t time is predicted based on last t time. Therefore, during the implementation, a duration of one year is considered for time t (also the dataset provides annual data). The data of one whole year is fed to a model to train and know its prediction accuracy for the next year. During the interpretation of the dataset, the total number of rating count is assumed as request count. Since the request count of content is predicted, the model is a regression model. Three types of deep learning with several sets of configurations and hyperparameters has been experimented to find out the optimal model. In Figure 4, the training and validation accuracy of the three models has been compared. Figures 4a indicates the performance accuracy of MLP model, Figure 4b shows LSTM model and 4c shows ConvLSTM model. From the figure, it can be said that the LSTM model results in better performance.

TABLE I: Performance Evaluation of MLPs having Different Configurations

Models	Cache Hit	Average Delay	Computation Time(s)
MLP[4,3]	0.32	0.40	0.76
LSTM[4,2]	0.40	0.36	2.34
ConvLSTM[4,2]	0.38	0.42	2.53

V. Result Analysis

To analyze the performance of the proposed approach, the data collected from the Movie Lens dataset has been considered to test the accuracy. This section discusses the outcome of the trained model based on those collected data.

TABLE II: Performance Evaluation of Cache Decision Algorithm

Before Cache Decision			After Cache Decision		
ID	**Size**	**Request Count**	**ID**	**Size**	**Request Count**
6934	96	59	6934	96	59
1387	43	26	1387	43	26
19	153	23	1690	94	18
			911	62	15
Total	**292**	**108**	**Total**	**295**	**118**

TABLE III: Performance Comparison of IntellCACHE with Different Caching Strategies

Strategy	Cache Hit	Average Delay	Caching Approach
FIFO	0.35	0.75	Reactive
LFU	0.28	1.30	Reactive
LRU	0.32	0.90	Reactive
MFU	0.30	1.25	Reactive
DeepMEC	0.38	0.36	Proactive
IntellCACHE	0.40	0.36	Proactive

A. Performance of the Deep Learning Models

To measure the performance of the models, two metrics are used- cache hit and Average delay. Both of these metrics have been compared to the baseline without the cache scheme. Three types of deep learning models are compared with one another to find out the best prediction results. The more the value of the cache hit, the more it provides better performance. From Table I, it is seen that the LSTM model having configuration[4,2] i.e. 4 cells in the first layer and 2 cells in the second layer, provides the best cache hit than the others. This means that this model can predict the future content list better than the other two models. With the increase of prediction accuracy, the system has a tendency for retrieving fewer contents from the original server. Because the contents are already in the cache node. So, the less the delay amount, the better the system will respond to the requests. Again, Table I indicates that the LSTM model shows an average delay of 0.38, which is less than the other two models.

B. Cache Decision

In Table II, a portion from the output of the experiment shows the differences due to adding caching decisions. In the

left section, the content 19 is of size 153 while its request count is 23. If the size of the cache node is 300, then it can only store the first three contents. But it follows Algorithm 2, the third contents will be discarded and the cache node will still have space to store another two contents. Then it can store a total of four contents instead of three, shown in the right section. Caching more content helps to increase the cache hit along with the average delay of the process.

C. Performance Comparison with Existing Strategies

The traditional reactive methods determine which of the user requests should be cached for future use, where this caching strategy predict future requests based on past behavior and takes cache decision accordingly. Since these two strategies are very different, no direct comparison can be made among those. However, Table III shows the apparent differences in their performances for the cache hit ratio and average delay to have a brief idea. From the table, it is clear that the average delay for the reactive methods is very higher than the proactive methods. The reason is, all the reactive strategies intend to fetch any contents after being triggered by the users which increases its average delay significantly.

VI. Conclusion

This paper proposed a deep learning-based intelligent caching scheme named IntellCache that can proactively cache the web content for the clients' future uses. Based on cache hit and average delay metrics, the performances of three types of deep learning models were compared to find out the most efficient one among them. Two algorithms were proposed to design the caching scheme. The first algorithm explained how to predict the FPC where the other algorithm helped to decide which contents from the predicted FPC list should be stored to get an optimal result.

Future works will aim to implement this idea on other hybrid neural networks to find out the optimal results through comparative analysis. Nevertheless, the hyperparameters and neural network configuration tuning also need to follow a definite way to offer a fixed neural network model.

References

[1] "Service provider mobility solutions," last accessed- 8 Feb. 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/service-provider/mobile-internet/index.html

[2] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2595–2621, 2018.

[3] A. Tatar, M. D. De Amorim, S. Fdida, and P. Antoniadis, "A survey on predicting the popularity of web content," *Journal of Internet Services and Applications*, vol. 5, no. 1, p. 8, 2014.

[4] W. Ali, S. M. Shamsuddin, A. S. Ismail *et al.*, "A survey of web caching and prefetching," *Int. J. Advance. Soft Comput. Appl*, vol. 3, no. 1, pp. 18–44, 2011.

[5] 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.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 1. IEEE, 1999, pp. 126–134.

[7] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[8] N. H. Tonekaboni, L. Ramaswamy, and S. Sachdev, "A mobile and web-based approach for targeted and proactive participatory sensing," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2019, pp. 215–230.

[9] V. A. Kakde, S. K. Mishra, A. Sinhal, and M. S. Mahalle, "Survey of effective web cache algorithm."

[10] F. R. Costa, R. da Rosa Righi, C. A. da Costa, and C. B. Both, "Nuoxus: A proactive caching model to manage multimedia content distribution on fog radio access networks," *Future Generation Computer Systems*, vol. 93, pp. 143–155, 2019.

[11] K. Thar, N. H. Tran, T. Z. Oo, and C. S. Hong, "Deepmec: Mobile edge caching using deep learning," *IEEE Access*, vol. 6, pp. 78 260–78 275, 2018.

[12] K. Thar, T. Z. Oo, Y. K. Tun, K. T. Kim, C. S. Hong *et al.*, "A deep learning model generation framework for virtualized multi-access edge cache management," *IEEE Access*, vol. 7, pp. 62 734–62 749, 2019.

[13] W. Li, C. Wang, D. Li, B. Hu, X. Wang, and J. Ren, "Edge caching for d2d enabled hierarchical wireless networks with deep reinforcement learning," *Wireless Communications and Mobile Computing*, vol. 2019, 2019.

[14] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning based edge caching in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, 2020.

[15] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and D. I. Kim, "Distributed deep learning at the edge: A novel proactive and cooperative caching framework for mobile edge networks," *IEEE Wireless Communications Letters*, vol. 8, no. 4, pp. 1220–1223, 2019.

[16] S. Sulaiman, S. M. Shamsuddin, A. Abraham, and S. Sulaiman, "Intelligent web caching using machine learning methods," *Neural Network World*, vol. 21, no. 5, p. 429, 2011.

[17] W. Ali, S. M. Shamsuddin, and A. S. Ismaail, "Intelligent web proxy caching approaches based on machine learning techniques," *Decision Support Systems*, vol. 53, no. 3, pp. 565–579, 2012.

[18] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "Intelligent naïve bayes-based approaches for web proxy caching," *Knowledge-Based Systems*, vol. 31, pp. 162–175, 2012.

[19] Y. Dai, D. Xu, S. Maharjan, Z. Chen, Q. He, and Y. Zhang, "Blockchain and deep reinforcement learning empowered intelligent 5g beyond," *IEEE Network*, vol. 33, no. 3, pp. 10–17, 2019.

[20] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 28–35, 2018.

[21] E. Baştuğ, M. Bennis, E. Zeydan, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data meets telcos: A proactive caching perspective," *Journal of Communications and Networks*, vol. 17, no. 6, pp. 549–557, 2015.

[22] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–6.

[23] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[24] Q.-V. Pham, F. Fang, V. N. Ha, M. Le, Z. Ding, L. B. Le, and W.-J. Hwang, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *arXiv preprint arXiv:1906.08452*, 2019.

[25] J. Yang, J. Zhang, C. Ma, H. Wang, J. Zhang, and G. Zheng, "Deep learning-based edge caching for multi-cluster heterogeneous networks," *Neural Computing and Applications*, pp. 1–12, 2019.

[26] Q. Fan, J. Li, X. Li, Q. He, S. Fu, and S. Wang, "Pa-cache: Learning-based popularity-aware content caching in edge networks," *arXiv preprint arXiv:2002.08805*, 2020.

[27] R. Banik, "The movies dataset," last accessed- 9 Feb. 2020. [Online]. Available: https://www.kaggle.com/rounakbanik/the-movies-dataset