DeepCache: A Deep Learning Based Framework For Content Caching

Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, Zhi-Li Zhang
University of Minnesota
Minneapolis, Minnesota
{arvind,verma,eman,babai008,zhzhang}@cs.umn.edu

ABSTRACT

In this paper, we present DeepCache a novel Framework for content caching, which can significantly boost cache performance. Our Framework is based on powerful deep recurrent neural network models. It comprises of two main components: i) *Object Characteristics Predictor*, which builds upon deep LSTM Encoder-Decoder model to predict the future characteristics of an object (such as object popularity) – to the best of our knowledge, we are the first to propose LSTM Encoder-Decoder model for content caching; ii) *a caching policy component*, which accounts for predicted information of objects to make smart caching decisions. In our thorough experiments, we show that applying DeepCache Framework to existing cache policies, such as LRU and k-LRU, significantly boosts the number of cache hits.

CCS CONCEPTS

• **Information systems** → *Multimedia information systems*; • **Networks** → *Network services*; • **Computing methodologies** → *Neural networks*;

KEYWORDS

DeepCache; deep learning; machine learning; caching; lstm; seq2seq; smart caching policies; cache hit; video object caches; prefetching; proactive caching; popularity prediction; fake requests;

ACM Reference Format:
Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, Zhi-Li Zhang. 2018. DeepCache: A Deep Learning Based Framework For Content Caching. In *NetAI'18: ACM SIGCOMM 2018 Workshop on Network Meets AI & ML, August 24, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229543.3229555

1 INTRODUCTION

Recent years have witnessed a rapid increase in video streaming services, as a result of the massive content published by content providers, high-speed Internet, and the number of devices connected to the Internet. Thus, content providers have resorted to employ one or more content distribution networks (CDNs) to handle scalability, and improve the quality of experience (QoE) for users. By 2021, 77% of the Internet video traffic is expected to cross CDNs [1]. Hence, adding cache storage space at routers becomes of utmost importance to handle this massive growth, and improve the network performance as well as user's QoE. Consequently, information centric networks (ICNs) (*e.g.*, NDN [11], DONA [12], CONIA [16]) have been developed as an emerging architecture for content delivery. ICN offers new primitives such as in-network caching, in which storage becomes an integral part of the network substrate (*i.e.*, routers have the capability to cache objects on-the-fly, and serve user requests for cached objects). This makes caching algorithms a major aspect in video streaming applications. Without caching, every user request is fetched from the backend/origin server, which increases the network load, as well as user-perceived latency. As a result, user engagement is impacted, which leads to significant revenue loss for content providers.

One of the most difficult decisions is which object to cache and evict, given the limited capacity of the cache network and large number of objects to cache. Caching algorithms can be classified based on which entity controls the caching decision, and the available information to make these decisions. Least Recently Used (LRU), Least Frequently Used (LFU), and their variants are examples of reactive caching, in which individual caches decide which objects to cache purely based on the recent locally observed object access patterns. They are easy to implement and widely used in today's CDNs [18]. On the other hand, static caching is proactive caching, in which centralized controllers have global view of user demands and object access patterns. They decide which objects to cache, and push these objects to cache nodes. Reactive caching reacts faster to changes in object access patterns, but leads to caching non-popular objects, which are evicted before receiving their next request, due to their lack of knowledge about future object popularity. This leads to thrashing problem and wasting cache resources (see (§3) for more details). Proactive caching is the optimal solution only if the object access pattern is stationary. Thus, it cannot cope with sudden changes in object popularity as reactive caching.

Content objects are heterogeneous as they vary in size (*e.g.*, web pages vs. videos), access pattern, and popularity. A study in [3] shows that 70% of objects served by a cache server are requested only once over a period of days. Object access patterns are frequently changing due to the frequent changes in object popularity as shown by the study in [20] using real traces, object popularity changes within each day according to the diurnal pattern, and also over days according to the object's life span. In addition, changes in request routing algorithms due to network/server failures can also cause changes in object access patterns. Due to these frequent changes, the assumption of stationary object access patterns becomes invalid. Thus, caching algorithms cannot rely on the locally

NetAI'18, August 24, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5911-5/18/08…$15.00
https://doi.org/10.1145/3229543.3229555

observed object access patterns for making decisions. On the other hand, manually tuning the caching algorithm for each cache server according to the changes of request access patterns is very expensive and is not scalable.

Thus, if we know ahead of time an estimation for object characteristics, we can utilize such information in the caching mechanism to cope with the predicted changes. The cache performance depends on the prediction accuracy, and how it is being utilized to make decisions. This task has many challenges, (1) the future object characteristics need to be forecasted to be available at the time of making cache decisions; (2) these characteristics change over time, and hence, this forecasting needs to run continuously; (3) finally, the caching mechanism needs to carefully utilize these predicted object characteristics to improve cache performance.

Our goal is to develop a self-adaptive caching mechanism, which automatically learns the changes in request traffic patterns, especially bursty and non-stationary traffic, and predicts future content popularity, then decides which objects to cache and evict accordingly to maximize the cache hit. In recent years, recurrent neural networks (RNN) have become the cornerstone for sequence prediction. RNNs have shown their unchallenged dominance in the area of natural language processing [15], machine language translation [2], speech recognition [7], and image captioning [8]. Many variants of RNN exist in literature, among which Long Short-Term Memory (LSTM) [10], Gated Recurrent Unit (GRU) [5] are the most popular ones for sequence prediction. Thus, it is natural to wonder their ability to predict content popularity where content requests arrive in a form of a sequence.

In this paper, we build the DeepCache Framework, which successfully demonstrates the ability of our LSTM based models to predict the popularity of content objects. The main contributions of our paper are two folds. We recognize the problem of content popularity prediction as a seq2seq modeling problem, and to our knowledge, we are the first to propose LSTM Encoder-Decoder model for content popularity prediction. Secondly, we create a general framework called DeepCache for making end-to-end cache decisions presented in (§3) and (§4). In (§5), we evaluate DeepCache Framework by applying it to existing caching policies like LRU and k-LRU, and show that it *significantly* boosts the number of cache hits. Lastly, we also show the dominant performance of employing LSTM Encoder-Decoder model for content popularity prediction on our datasets. We discuss existing caching mechanisms and how they adapt to user access patterns in (§2). Finally the paper is concluded in (§6).

2 RELATED WORK

Reactive caching such as LRU, LFU, and their variations rely on locally observed access patterns to decide the order of cached objects for replacement. However, they cannot predict future object popularity. Another approach which has attracted a lot of attention is timer-based caching, where each object is associated with a time-to-live (TTL) timer, and is evicted when the timer expires. Due to Che's approximation [4] objects within a cache are viewed independently, and their hit probabilities/rates are analyzed separately. Assuming the knowledge of object inter-arrival process and their popularity distributions, [6] provides optimal timer set up by solving an optimization problem to maximize cache hit rate. However, extracting request patterns for each object is a challenging task in real systems. [3] designed an adaptive algorithm to set up TTL-values for objects to handle heterogeneity, burstiness, and non-stationary nature of real-world content requests workloads. However, they rely on the history of received requests to change the TTL-values regardless of the changes which may happen in the future.

One of the recent attempts in learning objects access patterns is made in [9] for prefetching program counter's memory address in order to avoid on-chip misses. In [9], the authors treated the problem as a sequence classification, and then employed LSTM for the same purpose. In contrast, our DeepCache prediction problem belongs to seq2seq modeling, in which we employ LSTM Encoder-Decoder Model for object popularity prediction. The most closest work that partially focuses on object popularity prediction is Pensieve [13]. However, their content prediction is based on gathering statistics over time regardless of the temporal behavior. We aim to account for the temporal pattern in our prediction approach. Recently reinforcement learning (RL) has also been introduced in [13, 17] for making cache decision policy based on the notion of local and global popularity. Their work complements our DeepCache framework, in the sense that the proposed RL mechanisms can be integrated into our caching policy for making better decisions.

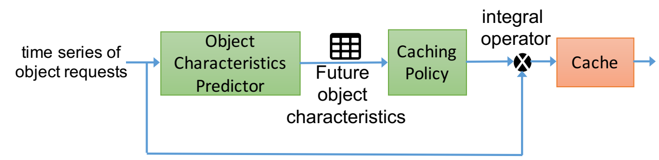


Figure 1: Data Flow in DeepCache

3 OVERVIEW OF DEEPCACHE

In DeepCache framework, we try to leverage state-of-the-art machine learning algorithms to improve cache efficiency. The core idea behind DeepCache is to predict characteristics of objects ahead of time. For instance, in this paper we focus on predicting object popularities. If we know the popular objects ahead of time, we can proactively make decisions based on the cache network objectives. If one wants to increase the cache hit efficiency, they can prefetch objects ahead of time; another goal could be to reduce network costs whereby knowing object popularities in advance can help reduce the problem of *cache thrashing* – by avoiding the eviction of future popular objects (or to evict future unpopular objects from the cache). However, to keep it simple, we drive this paper with a goal to increase the number of cache hits. This way, we are also able to verify the accuracy of DeepCache's object popularity prediction model. Figure 1 illustrates the design of applying DeepCache to a timeseries of object requests.

One of the most important characteristics for any caching system is object popularties. Therefore, in this paper we develop a content popularity prediction model. For every object request, DeepCache uses this model to predict the probabilities of future requests. This

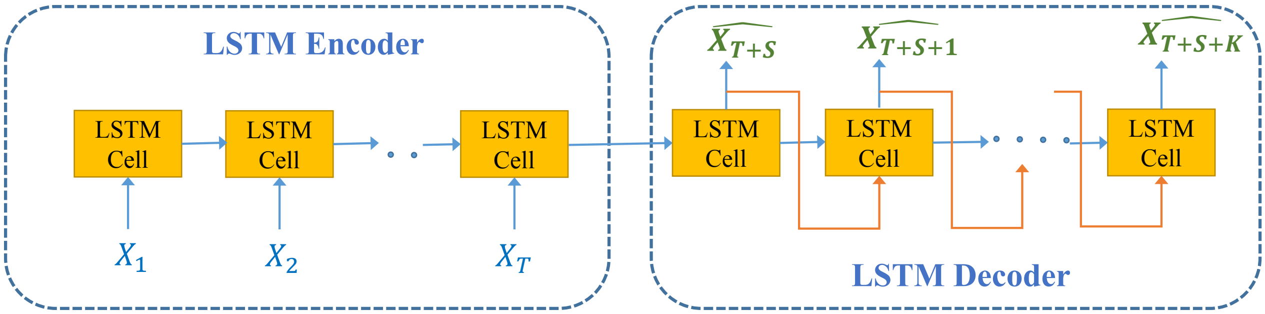


Figure 2: LSTM Encoder-Decoder Model used in DeepCache framework. We have an input sequence of request objects $\{x_1, x_2, ..., x_T\}$ at time T and desired output sequence $\{x_{T+S}, x_{T+S}, ..., x_{T+S+K}\}$ where $S > 0$ represent the shift in time and $K > 0$ is the desired number of outputs.

prediction can be made for multiple timescales. For instance, in our evaluation, for every object request, we predict popularties of all objects for three separate time intervals in future: 1-3 hours, 12-14 hours and 24-26 hours. This information is then used by the *Caching Policy* component to make decisions that control the caching behavior. For example, a Caching Policy could control what objects to cache and evict. To make DeepCache interoperable with traditional caches or caching strategies, we have an *integral* operator. This operator combines the information from the original object request and the output of the Caching Policy. Such an architecture allows us to build novel and "*smart*" caching policies that can leverage the predicted object characteristics to increase cache efficiency. (§5) provides an example where we use DeepCache along with traditional LRU-based caches.

4 DEEPCACHE COMPONENTS

In this work of DeepCache, we aim to bring the paradigm of cache prediction problem under *seq2seq modeling* [19]. Seq2Seq modeling gives much more flexibility in terms of predicting variety of outputs together with possibly varying input/output sequence length.

4.1 Seq2Seq Prediction for Caching

Seq2seq modeling enables us to jointly predict several properties of objects required for making cache decisions including: a) predicting object's popularity over multiple time steps in the future; b) predicting any sequential pattern that might exists among object requests; c) classifying sub-sequences of object requests into pre-defined categories, for instance to identify anomalies such as flash crowd phenomenon (which falls under *sequence classification* problem).

For Seq2Seq modeling, recurrent neural networks (RNN) have undoubtedly shown their dominance especially in the area natural language processing, machine translation and speech recognition. In particular, we focus on RNN of type long short term memory (LSTM) networks [10] for capturing any long-term and short-term dependency among the object requests (or among their popularity). LSTM models are quite popular due to their special design property related to carefully avoiding vanishing and exploding gradient problem when building deep layer neural network models.

For our purpose, we adopt LSTM Encoder-Decoder model as shown in Figure 2 for seq2seq prediction. This model essentially consists of encoder part which encodes the input sequence into a hidden state vector, and decoder part which decodes the output sequence from the hidden state vector.

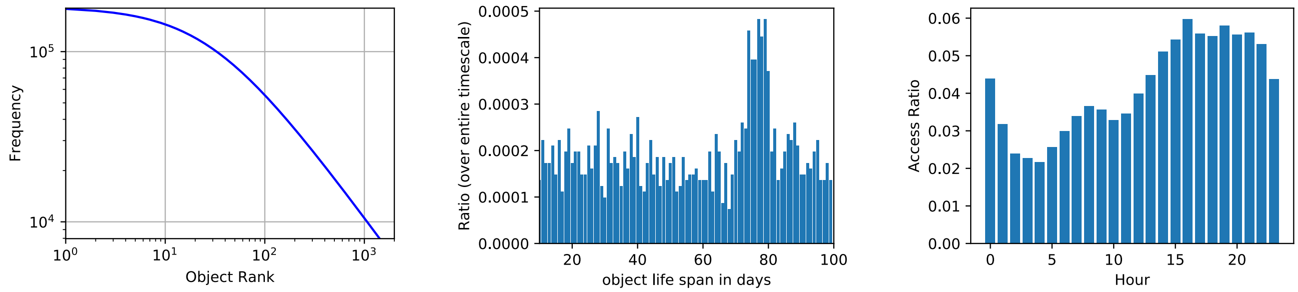
The main challenge we tackle in this paper is designing appropriate input features and predicting plausible desired outputs based on LSTM that are helpful for making cache-relation decisions under our DeepCache framework.

Problem Formulation: In abstract, let $X_t = \{x_1, x_2, ..., x_t\}$ be a sequence of objects requested so far at time t where each $x_t \in \mathbb{R}^d$ represents the input feature vector ($d-$dimensional) corresponding to the object. Let $Y_t = \{y_1, y_2, ..., y_k\}$ be the sequence of k outputs associated with the arrival of object x_t where $y_t \in \mathbb{R}^p$ represents the output feature vector of $p-$dimension. Our goal is to construct meaningful X_t and Y_t that are helpful in making caching decisions. In this paper, we primarily focus on constructing Y_t to be future content popularities based on past X_t popularities.

4.2 Content Popularity Prediction Model

For predicting object's popularity, we construct x_t as the probability vector of all unique objects at time t computed in a pre-defined probability window. Here, the definition of probability window can be time-based or can also be calculated based on a fixed length window of object requests previously seen. In this case, input dimension d is equal to the number of unique objects. Our output Y_t at time t, is also a sequence of k future probabilities, where k represents the number of probabilities to predict – possibly at multiple time steps of nearby and long-term future probabilities. Here again, output dimension p would be equal to the number of unique objects i.e., $p = d$.

For better performance of LSTM, we provide multiple past probabilities (denoted as m number of past probabilities) as input. Each of these probabilities are calculated using a predefined probability window. As a result, our input and output can be seen as a 3D Tensor with dimension (#*samples*, m, d) and (#*samples*, k, d), respectively. We construct the input and output tensor from a given trace of object requests (*i.e.*, a workload) and split the data further



(a) Object Popularity **(b) Histogram of Object Life Span** **(c) Hourly Access Ratio**

Figure 3: Workload Properties of Dataset 2

into training and testing parts. Finally, we train our LSTM Encoder-Decoder model to predict future probabilities of any requested object at time t. One important finding based on our datasets is that LSTM encoder-decoder performs much better if we separately feed the probabilities of each object as a sample data (instead of appending in an input feature vector). This results in an input and output tensor with dimension $(\#samples * d, m, 1)$ and $(\#samples * d, k, 1)$, respectively. The reason it works better is because – in our datasets, time series of object popularities are independent of each other. For cases where popularity of objects are correlated over time, we expect the former mentioned input/output data construction to work better.

4.3 Caching Policy

The *Caching Policy* component gets the characteristics predicted by the Object Characteristics Predictor (see Figure 1). It makes decisions on what to cache or evict. In this work, we consider content popularity as the object characteristic that is being predicted. Therefore, Content Popularity Prediciton Model predicts future object popularity. We design a *simple* yet *smart* caching policy in a way to make DeepCache interoperate with traditional caching strategies such as LRU, LFU, etc. The main idea of this caching policy is that – since we are given the future content popularities from the predictor component, our naive caching policy generates "*fake content requests*" and forwards it to the integral operator. The integral operator in our paper is a simple *merge* operator, which merges a stream of fake request to the original request and then sends it to the cache. Such fake requests would make the traditional cache (*e.g.*, LRU) to prefetch the objects. If the fake requests indeed represent those objects that will be popular in the near future, prefetching them would lead to an increase in the overall number of cache hits. Depending upon the definition of probability window, we can either generate such "fake content requests" periodically over time or for every n^{th} request. Our evaluation considers both approaches. One may argue that this approach would lead to many cache evictions and additions – thus increasing overall network and system load. However, this trade-off is subjective to the cache network objectives, which is out of scope in this paper.

5 EXPERIMENTAL RESULTS

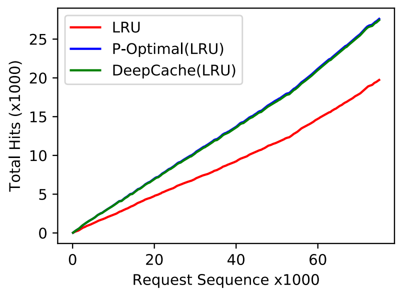
We evaluate our approach using two synthetic datasets with different characteristics. In this section, we explain the data generation process of the synthetic datasets, and show the results of applying DeepCache on them under different settings.

5.1 Synthetic Data Generation

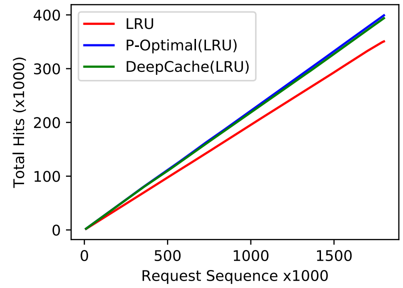
Dataset 1: Dataset 1 has 50 unique objects with more than 80K requests. It includes six intervals of time series which primarily differ in object popularities. Within each interval though, object popularity ranks remain constant. Object popularities are generated using Zipf distributions with $\beta = [0.8, 1, 0.5, 0.7, 1.2, 0.6]$ as the parameters for 50 objects. The popularity rank of objects is generated by a random permutation for each interval. The interval length is approximately a thousand requests.

Dataset 2: In order to consider a more realistic workload, we use MediSyn [20] to create Dataset 2. This dataset has a total of 1,425 unique objects with more than 2 million requests. The workload has static properties as well as temporal properties denoted using `S:` and `T:` prefixes, respectively. To implement the temporal properties of the workload, we assume that each object has a life span, all the object requests follow a diurnal pattern, and the access ratio to an object diminishes each day.

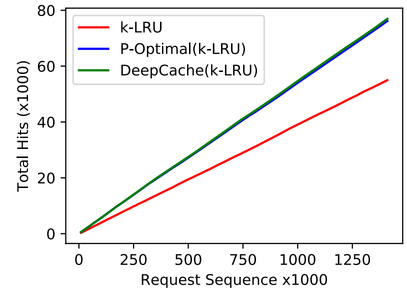
- `S:`**Object Frequency:** We use a *generalized Zipf distribution* to generate object's frequencies. We use $\beta = 0.8$ as popularity parameter along with $M = 175,000$ as the maximum frequency and the scale parameter $k = 30$ for $1,425$ objects. (See Figure 3a for Object Rank v/s frequency).
- `T:`**Object Life Span:** We define object life span as the number of days the object is seen during the whole timeseries of the workload. We use *log-normal distribution* to generate object life spans, 3b. For dynamic generation of life spans, the parameters of log-normal distribution, μ(mean) and σ(standard deviation), are generated by two normal distributions. The parameters are stated in Table 1.
- `T:`**Diurnal Pattern:** The diurnal pattern for each object's request arrival process within a given day is modeled as a *non-homogeneous Poisson process*. Each bin which is an



(a) DeepCache LRU on Dataset 1



(b) DeepCache LRU on Dataset 2



(c) DeepCache k-LRU on Dataset 2

Figure 4: Cache Hit Performance using DeepCache

Normal dist. Parameters	lognormal μ	lognormal σ
μ	3.0935	1.1417
σ	0.9612	0.3067

Table 1: Object Life Span Parameters

hour, has an specified Poisson parameter. The Diurnal ratio values are generated with user specified function of time (see Figure 3c for the hourly ratio of requests per day).

- T: **Object Access Rate:** We use linear and non-linear functions to control the request arrival rates, access rates, for an object during its life span. The number of requests received for an object diminishes each day.

5.2 Experiment Settings

LSTM Encoder-Decoder Model Settings: For our datasets, we use a two-layer depth LSTM Encoder-Decoder model with 128 and 64 as the number of hidden units. All experiments were run on a 2× GPU TITAN V. The loss function is chosen as mean-squared-error (MSE). We ran our experiments for a number of epochs equal to 30, with the batch size set to 10% of the training data. Runtime for all of our experiments is confined within the period of 30−120 minutes.

LSTM Input-Output Data Construction Settings: The sample sequence length is set to be 20, i.e., $m = 20$ for both datasets. For dataset 1, the probability of object o^i is calculated as $N_i/1000$, where N_i represents the number of occurrences of o^i in the window of past 1K objects. While for dataset 2, the probability of o^i is the normalized frequency of that object in an hour. In dataset 1, we aim to predict next $K = 10$ future probabilities (i.e. 10 future time units). For dataset 2, we set $K = 26$, but only used subset of these predicted probabilities. We use these future probabilities to make caching decisions.

Cache Policy Settings: As discussed earlier, we use a naive caching policy to enable DeepCache to interoperate with traditional caches. For every object request o^i_t at time t, we generate a varying number of "fake object requests" (denoted as F_t). For dataset 1, we generate F_t by calculating the top $M = 5$ objects with highest probability at $t + 1$. For dataset 2, we rather consider a varying number of top M objects with the highest probability from multiple time intervals.

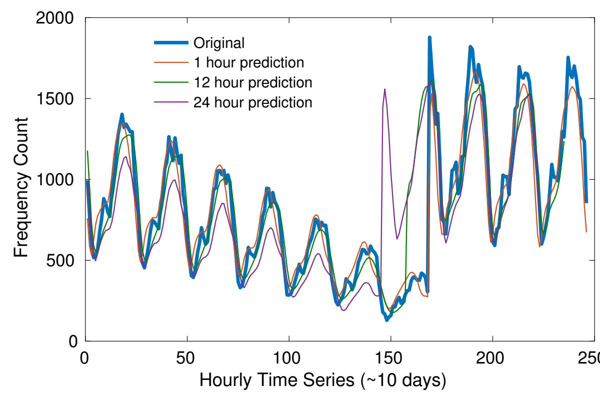


Figure 5: Performance of our LSTM-based Content Popularity Prediction Model of an object. Here, we see LSTM performs well for predicting $i^{th} = \{1, 12, 24\}$ hour ahead of time in comparison with the original values over a time series of ∼10 days.

In other words, our fake set of requests F_t not only considers immediate future, but also considers popular objects in the next 12 hours and 24 hours with a diminishing weight for the number of selected objects from each interval.

Integral Operator: For both datasets, the operator is a simple *merge* operator, where the actual object request is followed by all the fake requests generated by our *Caching Policy*. This helps us to update the state of the cache by prefetching objects based on future object popularity and evict unpopular ones.

Cache: For dataset 1, we set the cache size to 5, while for dataset 2 we set the cache size to 150.

	MSE	MAE
Dataset 1	1.2×10^{-6}	4.8×10^{-3}
Dataset 2	3.8×10^{-6}	8.3×10^{-3}

Table 2: Prediction Accuracy.

5.3 Results

LSTM Encoder-Decoder Prediction Accuracy: Table 2 shows the mean-squared-error (MSE) and mean-absolute-error (MAE) of our prediction accuracy on Datasets 1 and 2, both ranging from 0 to

1 in our case. These low error rates show the strong performance of our LSTM model for object popularity prediction. To give a sense of our predictions, Figure 5 shows the ability of LSTM predicting the next i^{th} hourly count for object requests. As evident from Figure 5, LSTM performs quite well in tracking the original time series over multiple future time steps.

Cache-Hit Efficiency: Figure 4 shows the result of applying a simple form of DeepCache Framework on both Datasets 1 and 2. For instance, in Figure 4a, we compare traditional LRU with DeepCache, and without DeepCache. P-Optimal shows the performance of DeepCache with 100% accuracy in content popularity prediction. For Dataset2 which represents a more realistic workload with large number of object catalog and cache size, we evaluate DeepCache using both LRU (see Figure 4b) and k-LRU (see Figure 4c). In k-LRU, the object has to traverse $K - 1$ virtual caches before it is inserted in the physical cache [14]. For all experiments, we found DeepCache significantly outperforms simple LRU. Surprisingly, in Figure 4c, we observe that DeepCache with k-LRU has slightly higher cache-hit than P-Optimal. We hypothesize this is due to LSTM's smooth probability prediction behavior. In case of P-Optimal, probabilities are frequently changing, which makes caching policy less stable compared to DeepCache. As a result, slightly more cache hits are observed for DeepCache over longer period of time.

6 CONCLUSION

In this paper we proposed DeepCache Framework, a paradigm to use state-of-the-art machine learning tools to the problem of content caching. In that, using such a framework, we proposed how to reason about the cache prediction problem under seq2seq modeling. We successfully show the ability of our LSTM based models to predict the popularity of content objects. To show the efficacy of our approach, we evaluated it using two synthetic datasets under multiple settings out of which one tries to emulate realistic workloads. Results show that enabling DeepCache with existing cache replacement algorithms such as LRU, k-LRU significantly outperforms algorithms without it.

7 ACKNOWLEDGEMENTS

This research was supported in part by US NSF grant CNS-1411636, CNS-1618339 and CNS-1617729, and DTRA grant HDTRA1-14-1-0040.

REFERENCES

[1] 2017. Cisco Visual Networking Index: Forecast and Methodology, 2016-2021. https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Soumya Basu, Aditya Sundarrajan, Javad Ghaderi, Sanjay Shakkottai, and Ramesh Sitaraman. 2017. Adaptive TTL-Based Caching for Content Delivery. *SIGMETRICS Perform. Eval. Rev.* 45, 1, 45–46. https://doi.org/10.1145/3143314.3078560

[4] H. Che, Z. Wang, and Y. Tung. 2001. Analysis and design of hierarchical Web caching systems. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, Vol. 3. 1416–1424 vol.3. https://doi.org/10.1109/INFCOM.2001.916637

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[6] Andrés Ferragut, Ismael Rodriguez, and Fernando Paganini. 2016. Optimizing TTL Caches Under Heavy-Tailed Demands. *SIGMETRICS Perform. Eval. Rev.* 44, 1, 101–112. https://doi.org/10.1145/2964791.2901459

[7] Alex Graves and Navdeep Jaitly. 2014. Towards End-to-end Speech Recognition with Recurrent Neural Networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. JMLR.org, II–1764–II–1772. http://dl.acm.org/citation.cfm?id=3044805.3045089

[8] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. 2015. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623* (2015).

[9] Milad Hashemi et al. 2018. Learning Memory Access Patterns. *arXiv preprint arXiv:1803.02329* (2018).

[10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[11] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '09)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/1658939.1658941

[12] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. 2007. A Data-oriented (and Beyond) Network Architecture. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '07)*. ACM, New York, NY, USA, 181–192. https://doi.org/10.1145/1282380.1282402

[13] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM.

[14] V. Martina, M. Garetto, and E. Leonardi. 2014. A unified approach to the performance analysis of caching systems. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 2040–2048. https://doi.org/10.1109/INFOCOM.2014.6848145

[15] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[16] E. Ramadan, A. Narayanan, and Z. L. Zhang. 2015. CONIA: Content (provider)-oriented, namespace-independent architecture for multimedia information delivery. In *2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. 1–6. https://doi.org/10.1109/ICMEW.2015.7169811

[17] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis. 2018. Optimal and Scalable Caching for 5G Using Reinforcement Learning of Space-Time Popularities. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (Feb 2018), 180–190. https://doi.org/10.1109/JSTSP.2017.2787979

[18] Muhammad Zubair Shafiq, Alex X. Liu, and Amir R. Khakpour. 2014. Revisiting Caching in Content Delivery Networks. *SIGMETRICS Perform. Eval. Rev.* 42, 1, 567–568. https://doi.org/10.1145/2637364.2592021

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks *(NIPS'14)*. MIT Press.

[20] Wenting Tang, Yun Fu, Ludmila Cherkasova, and Amin Vahdat. 2003. Medisyn: A synthetic streaming media service workload generator. In *NOSSDAV*. ACM.