# DeePref: Deep Reinforcement Learning For Video Prefetching In Content Delivery Networks

**Nawras Alkassab**
University of South Carolina
alkassab@email.sc.edu

**Chin-Tser Huang**
University of South Carolina
huangct@cse.sc.edu

**Tania Lorido Botran**
Roblox
tbotran@roblox.com

## Abstract

Content Delivery Networks carry the majority of Internet traffic, and the increasing demand for video content as a major IP traffic across the Internet highlights the importance of caching and prefetching optimization algorithms. Prefetching aims to make data available in the cache before the requester places its request to reduce access time and improve Quality of Experience on the user side. Prefetching is well investigated in operating systems, compiler instructions, in-memory cache, local storage systems, high-speed networks, and cloud systems. Traditional prefetching techniques are well adapted to a particular access pattern, but fail to adapt to sudden variations or randomization in workloads. This paper explores the use of reinforcement learning to tackle the changes in user access patterns and automatically adapt over time. To this end, we propose, DEEPREF, a Deep Reinforcement Learning agent for online video content prefetching in Content Delivery Networks. DEEPREF is a prefetcher implemented on edge networks and is agnostic to hardware design, operating systems, and applications. Our results show that DeePref DRQN, using a real-world dataset, achieves a 17% increase in prefetching accuracy and a 28% increase in prefetching coverage on average compared to baseline approaches that use video content popularity as a building block to statically or dynamically make prefetching decisions. We also study possible transfer learning of statistical models from one edge network into another, where unseen user requests from unknown distribution are observed. In terms of transfer learning, the increase in prefetching accuracy and prefetching coverage are [30%, 10%], respectively.

## 1 Introduction and Motivation

Recently, Multi-access Edge Computing (MEC) enabled caching and prefetching techniques to be implemented at edge networks, enhancing Quality of Experience (QoE) at the user-side while benefiting cloud server at the core layer in CDNs. Video streaming applications can utilize MEC environments to prefetch video segments to end users at edge networks, allowing multi-users that are, possibly, close geographically to benefit from the prefetched segments. SPACE [2] is a prefetching and caching framework that utilizes MEC to prefetch video segments to edge networks in cellular networks (i.e., 5G). Behravesh *et al.* [4] proposed a segment prefetching algorithm based on bit rate prediction of client segment requests using a near-optimal machine learning approach. MVP [9] is a video segment prefetching technique implemented at edge networks for 4K Videos On Demand (VoD) applications. MVP achieves seamless playback in realistic LTE-A network infrastructure.

To this end, we hypothesize that prefetching an entire video content with high quality has its advantages over segment prefetching in CDNs where the bottleneck is the upstream network between edge networks and cloud networks. First, prefetching video content as a bulk saves resources in the upstream network, however, segment prefetching is still subject to fluctuations in the upstream network since a continuous Transmission Control Protocol (TCP) session needs to be maintained throughout watching the video content. Second, Round Trip Time (RTT) of edge-cloud network is

Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans, LA, USA.

much larger than access-edge network due to the propagation delay, therefore, prefetching an entire video content eliminates any propagation delay of future requests. Third, QoE is guaranteed in bulk prefetching, and it is only subject to Adaptive Bit Rate (ABR) algorithms of the end user network (i.e., access networks).

In this work, we utilize both Deep Q-networks (DQN) [18] and Deep Recurrent Q-Networks (DRQN) [18] designs as building blocks to tackle the prefetching problem by modeling video content prefetching as POMDP. We propose, DEEPREF, a prefetcher which is implemented on edge networks and takes only the request ID to decide what to prefetch, hence, DEEPREF is agnostic to hardware design, operating system, and application. We also study possible transfer learning of RL models from one edge network into another. We discuss the results for transfer learning in Sec.3.2.

Our results show that DEEPREF DRQN, using a real-world dataset, achieves 17% increase in prefetching accuracy and 28% increase in prefetching coverage on average compared to baseline approaches that use video content popularity as a building block to statically or dynamically make prefetching decisions. In terms of transfer learning, the increase of prefetching accuracy and prefetching coverage are [30%, 10%], respectively. We discuss the experimental design and results in Sec. 3.

## 2 System Architecture

We design, DeePref, such that a prefetching decision is made at edge networks every time a user request is received from access networks. Consequently, the prefetching requests are synchronous to user requests. Our results show that DeePref is more moderate compared to baselines, as discussed in Sec.3.2. Moreover, we assume that the cloud network periodically updates the edge network with newly added video content to accommodate for prefetching new/unseen items.

### 2.1 Reward Design

In DEEPREF, we simplify the reward design as much as possible to entail all aspects of prefetching in numerical values ranging between $[-2, 2]$ based on normalized latency, $[0, 1]$ where the agent receives one of the following rewards each time step, as described below:

- 2 if there is a cache hit and the agent decided not to prefetch. This indicates that the item already resides at edge and had been prefetched before.
- $2 - l_i$ if there is a cache hit and the agent decided to prefetch an item $i$ with latency $l_i$.
- $-1$ if there is a cache miss and the agent decided not to prefetch.
- $-1 - l_i$ if there is a cache miss and the agent decided to prefetch an item $i$ with latency $l_i$.

DQN architecture does not capture the inter-sample dependencies across user requests since the input of DQN is a concatenation of user requests mimicking request history, which only captures spatial dependencies across samples. Therefore, we utilize Deep Recurrent Q-Network (DRQN) which uses Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) [11] and Gated Recurrent Unit (GRU) [6] to counter the partial observability of the received user requests and to account for inter-sample dependencies across user requests. Further, we append to the one hot encoded user request a binary indicator vector representing items at edge network. Afterward, the input is fed to a linear body as an input layer of size 512 followed by LSTM which is connected to an output layer where each neuron on the output layer represents the $Q(s, a)$ of each possible action $a$ being in state $s$. DEEPREF DRQN architecture is shown in Fig. **??**. Moreover, DRQNS suffer from slowness during training when dealing with large sequences (i.e., large time-steps); therefore, we used Truncated Back Propagation Through Time (TBPTT) with $k_1 = k_2 = 300$ where $k_1$ is the number of forward-pass time-steps between LSTM updates, and $k_2$ is the number of time-steps to which we apply Back Propagation Through Time (BPTT).

## 3 Experiment Design

### 3.1 Dataset

We used the benchmark dataset MovieLens-100K (ml-100k) [10] which contains users' information such as age, gender, occupation, and zip code. We extract the longitude and latitude of each zip code

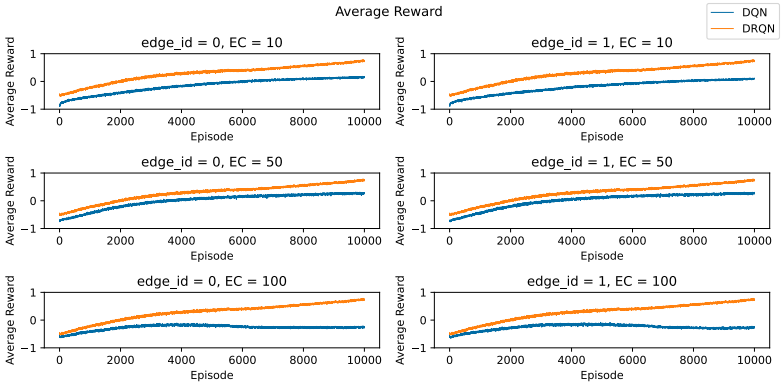


Figure 1: Average episodic reward during training phase

from the US Census dataset and used k-means clustering to categorize user requests geographically into clusters, where each cluster represents an edge network. We used the elbow method with silhouette analysis to determine the number of clusters. We chose $k = 3$ as it has one of the lowest silhouette scores.

We calculate the latency of each content to be prefetched from the cloud network based on the content size. We assign content size randomly, and to calculate the $throughput = \frac{8 \times CWND}{RTT}$ (RFC 6349 [8]), We assume Round Trip Time (RTT) to be 100 ms and the receiver TCP window size (CWND) of 65536 B.

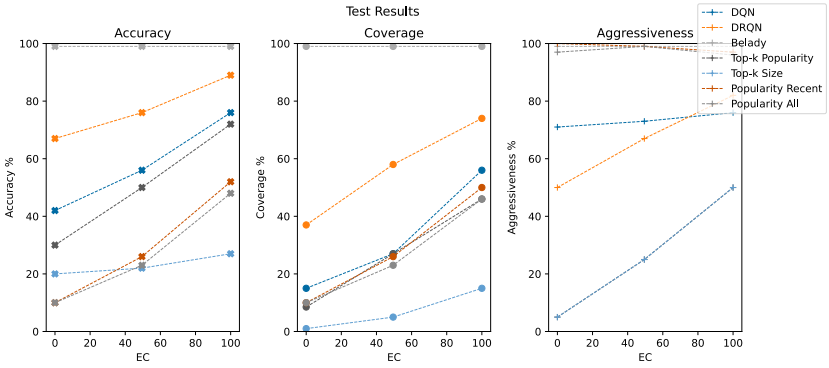


Figure 2: Test results for edge ID = 1

### 3.2 Results

We trained two agents $(ID = 0, ID = 1)$ using DEEPREF DQN and DEEPREF DRQN architectures and compared our testing and transferring results to the following baseline prefetching algorithms:

- ***Belady-Prefetch***: Prefetch the item that will requested next in case of a cache miss. Belady eviction algorithm [5] will be used to evict the item in which its occurrence happens the furthest. This approach will serve as an oracle in our experiments.
- ***Top-k Popularity***: Prefetch Top-k popular video content that is known to the cloud network.
- ***Top-k Size***: The edge network will prefetch Top-k largest video content that is known to the cloud network.
- ***Popularity Recent***: Using on-edge prefetching, the agent (i.e., edge network) will prefetch the most popular content using a sliding time window of 24 hours.
- ***Popularity All***: Using on-edge prefetching, the agent (i.e., edge network) will prefetch the most popular item that has been seen so far since the beginning of the experiment.

| Edge Capacity | EC = 10 | | | | EC = 50 | | | | EC = 100 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Metric | Acc. | Cov. | Timeliness | Aggr. | Acc. | Cov. | Timeliness | Aggr. | Acc. | Cov. | Timeliness | Aggr. |
| Belady-Prefetch | 0.99 | 0.99 | $0.03 \pm 0.19$ | 0.99 | 0.99 | 0.99 | $20.39 \pm 39.54$ | 0.99 | 0.99 | 0.99 | $44.21 \pm 60.79$ | 0.99 |
| Top-k Popularity | 0.3 | 0.085 | $127.1 \pm 61.74$ | 0.05 | 0.5 | 0.27 | $108.64 \pm 47.24$ | 0.25 | 0.72 | 0.46 | $83.83 \pm 24.97$ | 0.5 |
| Top-k Size | 0.2 | 0.01 | $183.25 \pm 35.55$ | 0.05 | 0.22 | 0.05 | $175.35 \pm 38.98$ | 0.25 | 0.27 | 0.13 | $155.43 \pm 43.07$ | 0.5 |
| Popularity Recent | 0.1 | 0.1 | $8.28 \pm 3.25$ | 1.0 | 0.26 | 0.26 | $28.18 \pm 19.3$ | 0.99 | 0.52 | 0.5 | $33.16 \pm 23.77$ | 0.97 |
| Popularity All | 0.1 | 0.1 | $8.08 \pm 3.39$ | 0.97 | 0.23 | 0.23 | $31.59 \pm 18.95$ | 0.99 | 0.48 | 0.46 | $35.86 \pm 27.48$ | 0.96 |
| **DeePref DQN** | 0.42 | 0.15 | $98.0 \pm 56.21$ | 0.71 | 0.56 | 0.27 | $80.11 \pm 47.37$ | 0.73 | 0.76 | 0.56 | $66.81 \pm 38.88$ | 0.76 |
| **DeePref DRQN** | 0.67 | 0.37 | $69 \pm 54.56$ | 0.5 | 0.76 | 0.58 | $83.3 \pm 50.9$ | 0.67 | 0.89 | 0.74 | $70.64 \pm 38.37$ | 0.82 |

Table 1: Testing Results on Edge ID = 1

| Edge Capacity | EC = 10 | | | | EC = 50 | | | | EC = 100 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Metric | Acc. | Cov. | Timeliness | Aggr. | Acc. | Cov. | Timeliness | Aggr. | Acc. | Cov. | Timeliness | Aggr. |
| Belady-Prefetch | 0.99 | 0.99 | $0.04 \pm 0.21$ | 0.99 | 0.99 | 0.99 | $0.17 \pm 0.37$ | 0.89 | 0.99 | 0.99 | $0.27 \pm 0.44$ | 0.99 |
| Top-k Popularity | 0.23 | 0.05 | $150.2 \pm 50.95$ | 0.05 | 0.36 | 0.2 | $131.76 \pm 41.67$ | 0.25 | 0.42 | 0.34 | $108.99 \pm 32.86$ | 0.5 |
| Top-k Size | 0.0 | 0.0 | $200 \pm 0.0$ | 0.05 | 0.28 | 0.07 | $160.53 \pm 53.99$ | 0.25 | 0.29 | 0.15 | $150.19 \pm 43.96$ | 0.5 |
| Popularity Recent | 0.12 | 0.12 | $7.48 \pm 3.39$ | 0.95 | 0.23 | 0.25 | $29.52 \pm 19.53$ | 0.99 | 0.43 | 0.51 | $34.55 \pm 27.46$ | 0.93 |
| Popularity All | 0.08 | 0.07 | $8.67 \pm 3.92$ | 0.85 | 0.21 | 0.22 | $31.41 \pm 19.0$ | 0.99 | 0.41 | 0.48 | $36.31 \pm 28.87$ | 0.93 |
| **DeePref DQN** | 0.32 | 0.2 | $102.92 \pm 66.88$ | 0.23 | 0.42 | 0.37 | $84.54 \pm 50.04$ | 0.44 | 0.65 | 0.46 | $70.27 \pm 40.36$ | 0.32 |
| **DeePref DRQN** | 0.56 | 0.34 | $85.32 \pm 49.59$ | 0.25 | 0.66 | 0.48 | $74.55 \pm 47.15$ | 0.42 | 0.72 | 0.58 | $72 \pm 50.36$ | 0.23 |

Table 2: Transfer results of edge ID = 1 to edge ID = 2

As shown in Fig. 1, DEEPREF DRQN model clearly outperforms DEEPREF DQN model during the training phase in terms of average episodic reward. We design the replay memory of DEEPREF DRQN using both LSTMs and GRUs where both show comparative results. Moreover, the replay memory is random instead of sequential since user requests are coming from heterogeneous networks that are in nature represent a non i.i.d distribution.

As shown in Table 1 and depicted in Fig. 2, the testing results show that both DEEPREF DQN and DEEPREF DRQN architectures outperform traditional prefetching schemes such as *Top-k Popularity* and *Popularity Recent* in terms of prefetching accuracy and prefetching coverage. DEEPREF DRQN outperforms baseline approaches with an increase of at least [17%, 28%] in terms of prefetching accuracy and coverage, respectively. As presented in Table 1, our results show that more aggressive algorithms tend to have better prefetching accuracy and coverage since some prefetching algorithms such as *Top-k Size* may decide not to prefetch at all, missing $100\%$ of the chances of observing a prefetch hit. Furthermore, Fig.2 shows that the increase in the Edge Capacity (EC) increases both prefetching accuracy and coverage.

When transferring DEEPREF DRQN model from the trained edge (e.g., $EdgeID = 1$) to an entirely new edge network (e.g., $EdgeID = 2$) which represents a different statistical distribution, our results show similar improvements compared to baseline approaches. As shown in Table 2, DEEPREF DRQN improves both prefetching accuracy and prefetching coverage compared to baseline approaches with an increase of [$30\%$,$10\%$], respectively.

## 4 Concluding Remarks

We proposed DEEPREF, a prefetcher for video prefetching in CDNs. DEEPREF is a Deep Reinforcement Learning agent that utilizes the network latency in terms of punishments and rewards to make prefetching decisions online. DEEPREF is a prefetcher that is implemented at edge networks and is agnostic to hardware design, operating system, and application. Our results show that DEEPREF DRQN outperforms baseline approaches in terms of prefetching accuracy and prefetching coverage. Furhter, DEEPREF is an auto-aggressive prefetcher in the sense that it adapts to user requests online without any changes in the model architecture or system design. Our results also show that aggressive approaches tend to have better prefetching accuracy and coverage.

## Acknowledgement

We thank the Research Computing team at the University of South Carolina for their efforts in making High-Performance Computing (HPC) clusters available for us to run our experiments.

## References


[1] Vijay K Adhikari, Yang Guo, Fang Hao, Volker Hilt, Zhi-Li Zhang, Matteo Varvello, and Moritz Steiner. Measurement study of netflix, hulu, and a tale of three cdns. *IEEE/ACM Transactions on Networking*, 23(6):1984–1997, 2014.

[2] Jesús Aguilar-Armijo, Christian Timmerer, and Hermann Hellwagner. Space: Segment prefetching and caching at the edge for adaptive video streaming. *IEEE Access*, 11:21783–21798, 2023.

[3] Nawras Alkassab, Chin-Tser Huang, Yu Chen, Baek-Young Choi, and Sejun Song. Benefits and schemes of prefetching from cloud to fog networks. In *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, pages 1–5. IEEE, 2017.

[4] Rasoul Behravesh, Akhila Rao, Daniel F Perez-Ramirez, Davit Harutyunyan, Roberto Riggio, and Magnus Boman. Machine learning at the mobile edge: The case of dynamic adaptive streaming over http (dash). *IEEE Transactions on Network and Service Management*, 19(4):4779–4793, 2022.

[5] Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 2018.

[8] Barry Constantine, Gilles Forget, Ruediger Geib, and Reinhard Schrage. Framework for tcp throughput testing. Technical report, 2011.

[9] Chang Ge, Ning Wang, Gerry Foster, and Mick Wilson. Toward qoe-assured 4k video-on-demand delivery through mobile edge virtualization with adaptive prefetching. *IEEE Transactions on Multimedia*, 19(10):2222–2237, 2017.

[10] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

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

[12] Wen Hu, Jiahui Huang, Zhi Wang, Peng Wang, Kun Yi, Yonggang Wen, Kaiyan Chu, and Lifeng Sun. Musa: Wi-fi ap-assisted video prefetching via tensor learning. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2017.

[13] Andreas M Kaplan and Michael Haenlein. Users of the world, unite! the challenges and opportunities of social media. *Business horizons*, 53(1):59–68, 2010.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[15] Ke Liang, Jia Hao, Roger Zimmermann, and David KY Yau. Integrated prefetching and caching for adaptive video streaming over http: an online approach. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 142–152. ACM, 2015.

[16] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data: the management revolution. *Harvard business review*, 90(10):60–68, 2012.

[17] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[19] George Pallis and Athena Vakali. Insight and perspectives for content delivery networks. *Communications of the ACM*, 49(1):101–106, 2006.

[20] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[21] Stuart Russel, Peter Norvig, et al. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2013.

[22] Eht E Sham and Deo Prakash Vidyarthi. Intelligent admission control manager for fog-integrated cloud: A hybrid machine learning approach. *Concurrency and Computation: Practice and Experience*, page e6687, 2021.

[23] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[24] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[25] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[26] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12:1057–1063, 1999.

[27] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[28] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[29] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[30] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[31] Hao Wu, Youlong Luo, and Chunlin Li. Optimization of heat-based cache replacement in edge computing system. *The Journal of Supercomputing*, 77:2268–2301, 2021.

# A Appendix

## A.1 Background

In recent years, we have witnessed an exponential growth in Internet traffic. Mainly, this is due to the explosion of Big Data [16], the increased usage of social media [13], the emergence of cloud computing [17] and edge computing [23]. This rapid growth has accelerated the implementation and design of CDNs. Cisco [7] had predicted that around 64% of the entire Internet traffic will cross CDNs in 2020, up from 45% in 2015. Video streaming service providers such as Netflix [1] and Hulu [2] rely heavily on CDNs to distribute their video content to end users [1]. For example, Akamai Technologies [3] is the market leader (80% of the overall CDN market) in providing content delivery services where it owns more than 12,000 servers over 1,000 networks in 62 countries [19]. Furthermore, There exists an increased trend in the popularity of video content as a major internet traffic. According to Cisco [7], global Internet video traffic (business and consumer, combined) will grow 4-fold from 2015 to 2020, a compound annual growth rate of 31%. Internet video traffic will be 79% of all Internet traffic in 2020, up from 63% in 2015. In addition, Internet video traffic will reach 127.8 Exabytes per month in 2020, up from 33.7 Exabytes per month in 2015, and Ultra HD and HD

[1] https://www.netflix.com

[2] https://www.hulu.com

[3] https://www.akamai.com

video content will be 15.7% of Internet video traffic in 2020, up from 2.3% in 2015. This recent trend of video popularity highlights the importance of video content prefetching from cloud networks to fog networks in CDNs. Further, prefetching can be implemented in various nodes from end users to cloud servers in CDNs, as illustrated in Fig.3. Recently, Hu *et al.* [12] proposed a Reinforcement Learning (RL) approach to prefetch successive episodes of video series to access points that are shipped with large storage capacity and close to end users, the assumption is that the end user is more likely to watch more than one episode without skipping. IPAC [15] integrates prefetching with caching to allow the video player at the end user side to generate prefetch requests for successive video segments with the same bit-rates as the current request. IPAC requires a modification of the underlying cache eviction algorithm, and it does not consider multi-user prefetching (inter-correlation among user requests). Further, Alkassab *et al.* [3] addressed the advantages and the disadvantages of video content prefetching across different layers (core, distribution, and access) in CDNs and discussed the utilization of out-of-band bandwidth for prefetching in fog networks. Wu *et al.* [31] proposed a prefetching scheme based on Bayesian Networks and Markov chains to prefetch video files and designed a cache eviction algorithm which utilizes the content heat and re-access probability such that the cached content with the least re-access weight is evicted. On the other hand, Sham *et al.* [22] proposed an admission control system to place prefetching requests based on CPU, memory and other resource parameters in hybrid edge-cloud networks. Such design of prefetching algorithms require system-level and application-level information that may not be available at the prefetcher side, limiting the applicability/implementation of such algorithms.

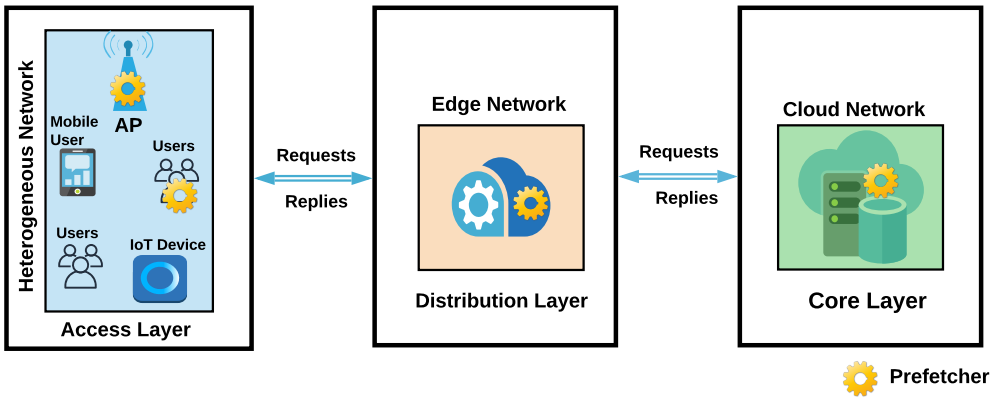


Figure 3: Content Delivery Networks Architecture with possible prefetching implemented in different nodes across different layers.

### A.2 Large State And Action Space

**Definition A.1.** Markov Decision Process or MDP is defined as a tuple $M := (S, A, P, R, \rho_0, \gamma)$. Where $S$ is the state space, $A$ is the action space, and $P$ is the transition function from state $s$ to state $s^{'}$ after taking an action $a$, that is $(s \xrightarrow{a} s^{'})$. The reward function $R : S \times A \times S \rightarrow \mathcal{R}$ is a random variable representing the reward after transitioning from one state to another and observing a reward $R_t = R(S_t, A_t, S_{t+1})$, $\rho_0$ is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor that trades off the instantaneous and future rewards.

In its most abstract setting, RL aims to find an optimal policy for an agent sequentially interacting with an environment over a sequence of time-steps, $t = 1, 2, 3, ...,$. The agent executes actions and receives observations and rewards, and the aim is to maximize the future cumulative reward [21]. Value-based approaches such as Q-learning [29] and policy-based ones (policy-gradient) such as REINFORCE [30] constitute classification approaches to solve RL problems [24]. Value-based approaches present many advantages such as seamless off-policy learning and better sampling efficiency compared to policy-gradient methods, however, they are prone to learning instability when mixed with function approximation [26]. Although policy convergence in value-based methods is not well-studied, the emergence of deep learning [14] has empowered the recent success of many value-based methods such as Deep Q-networks (DQN) [18] and DRQN [18].

The goal of a single agent *Reinforcement Learning* (RL) is to solve the underlying MDP by finding a policy $\pi : S \rightarrow \Delta(A)$, a mapping from the state space $S$ to a distribution over the action space $A$, so that $a_t \sim \pi(\cdot|s_t)$ and maximizing the return $G$. The *return* is the discounted accumulated reward, that is, $G_t = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$. We can define a *value function* $V(s)$ Eq. (1) or *action-value function* $Q(s, a)$ Eq. (2) under policy $\pi$ as follows:

$$V(s) := E[G_t \mid S_t = s], \tag{1}$$

$$Q(s, a) := E[G_t \mid S_t = s, A_t = a], \tag{2}$$

In Reinforcement Learning (RL), we usually use methods of *Dynamic Programming (DP)* [20], to maximize the return $G_t$ by calculating a *value function* $V(s)$ or the *action-value function* $Q(s, a)$ when using policy $\pi$. The computations carried in this procedure to approximate an optimal policy $\pi^*$ is called *policy improvement* and *policy evaluation* [25] where we typically use *value-based methods* or *policy-based methods* such as *Temporal Difference* (TD) learning algorithms [27, 28, 25] which have convergence guarantees under some conditions. For example, Q-learning [29] is an *off-policy* TD learning algorithm that may converge to an optimal policy $\pi^*$ by performing the following update rule

$$Q_{t+1}(s, a) \leftarrow (1 - \alpha)Q_t(s, a) + \alpha[r + \gamma \max_{a^{'}} Q(s^{'}, a^{'})], \tag{3}$$

where $\alpha$ is the learning rate and $r$ is the reward received upon taking action $a$ from state $s$. We usually derive a new policy $\pi^{'}$ which is greedy with respect to $Q^{\pi}(s, a)$, that is, $\pi^{'} \in argmax_a Q^{\pi}(s, a)$. The new policy $pi^{'}$ is guaranteed to be at least as good as policy $\pi$. Q-learning convergence and performance guarantees leading to an optimal policy $\pi^*$ is studied in [29, 24].

If we decide to prefetch one video content naively from the cloud network to an edge network, the problem of prefetching can be thought of as selecting one of the available video content among all content residing on the cloud network. Thus, we can model the prefetching problem as a Markov Decision Process (MDP) [20] as in Definition, A.1, which is a tuple $(S, A, P, R, \gamma)$. Where $S$ is the state space, $A$ is the action space, $P$ is the transition function, $R$ is the reward function, and $\gamma$ is the discount factor $\gamma \in [0, 1]$. Consequently, we can construct the underlying MDP as follows.

- ***State Space:*** Let $S = \{s_1, s_2, ..., s_t, ..., s_T\}$ denote the state space. Such that the current state at time step $t$ can be represented as: $s_t = \{m_1, m_2, \ldots, m_i, m_M\} \in \mathbb{R}^{C_e}$ where $m_i \in M$, $M$ is the total number of video content residing at the cloud network, and $C_e$ is the storage capacity at the edge node $e$.

  Intuitively, the state dimensions/space can be expressed through the following equation:

$$\mid S \mid = \binom{M}{C_e}, \tag{4}$$

- ***Action Space:*** Let $A = \{a_1, a_2, ..., a_t, ..., a_T\}$ denote the action space. Where $a_t$ is the prefetching action to be taken by the cloud device at time step $t$. The cloud device can make the following possible prefetching decisions:

  1. ***Prefetch one video content:*** The edge node will choose to prefetch one video content among the available $M$ video content located at the cloud.
  2. ***No prefetching action required:*** The edge node $E$ can decide not to prefetch at all. Consequently, there will be no eviction of items taken at the edge node $E$.

  Consequently, the action space can be expressed as follows.

$$\mid A \mid = \binom{M}{1} + 1, \tag{5}$$

- ***Transition Function:*** At time step $t + 1$, the system state transitions to another state after taking an action where the transition probability can be given by the following equation:

$$P(s_{t+1} \mid s_t, a_t) = P({m^{\mathrm{e}}}_{t+1} \mid {m_t}^{\mathrm{e}}, a_t), \tag{6}$$

Thus, the system's state/action space can be expressed as follows:

$$\mid S \mid \cdot \mid A \mid = (\binom{M}{1} + 1) \cdot \binom{M}{C_e}, \tag{7}$$

Where $M$ is the total number of video content residing at the cloud network, and $C_e$ is the storage capacity at edge network, such that $M \gg C_e$. The state space (e.g., how many possible combinations of item that can reside at edge network is given by $\mid S \mid = \binom{M}{C_e}$, and the action space is $\mid A \mid = \binom{M}{1} + 1$ (e.g., selecting one video content to prefetch or decide not to prefetch at all).

Naturally, the prefetching problem suffers from large state space and action space and deriving an optimal prefetching policy is tricky since the action space is non-stationary and items are continuously added to the cloud network (e.g., continuous action space). Moreover, an optimal prefetching policy differs from an optimal caching policy such that in prefetching the item selection is not constrained to the set of items that had been previously seen (e.g., cached) by edge networks, rather, an optimal prefetching policy needs to *consider/predict* any item to be requested despite how many times it had been requested before. Unlike caching, where an optimal policy can be derived by selecting one of the items that had been previously cached before, which reduces the action space and problem dimensionality.

### A.3 LSTM Equations

We list LSTM equations below for completeness:

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i), \tag{8}$$

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f), \tag{9}$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o), \tag{10}$$

$$\hat{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c), \tag{11}$$

$$c_t = f_t * c_{t-1} + i_t * \hat{c}_t, \tag{12}$$

$$h_t = o_t * tanh(c_t), \tag{13}$$

Where $x_t$ is the input, $h_{t-1}$ is the hidden state at time-step $t-1$, $i_t$ is the input gate, $f_t$ is the forget gate, and $o_t$ is the output gate. The weights for the input gate $i_t$, forget gate $f_t$, and the output gate $o_t$ are noted as $w_i, w_f, w_o$, respectively. Similarly, the biases for these gates are noted as $b_i, b_f, b_o$, respectively. Where $c_t$ is the cell state at time-step $t$, $\hat{c}_t$ is the candidate cell state at time-step $t$, and $\sigma$ is the sigmoid activation function.

### A.4 Evaluation Metrics

We use the following evaluation metrics to compare DeePref DQN and DRQN architectures with baselines:

- ***Prefetching Accuracy (Precision Ratio)***: Prefetching Accuracy describes how precise the prefetching decisions are in terms of sent prefetches. Prefetching accuracy is also referred to as the precision ratio of prefetching, which is defined below:

$$Accuracy\ (Precision) = \frac{\#hits}{\#hits + \#misses} = \frac{\#hits}{\#user\ requests} \tag{14}$$

- ***Prefetching Coverage (Recall Ratio)***: Prefetching Coverage describes the usefulness of sent prefetches. Prefetching coverage is also referred to as recall ratio, which is defined in the following equation:

$$Coverage\ (Recall) = \frac{\#used\ prefetches}{\#sent\ prefetches} \tag{15}$$

- ***Prefetching Aggressiveness***: Prefetching Aggressiveness describes how many prefetches had been sent throughout the entire experiment. This metric is used to compare prefetching algorithms when they are achieving similar prefetching accuracy and coverage and is defined below:

$$Aggressiveness = \frac{\#sent\ prefetches}{\#hits + \#misses} = \frac{\#sent\ prefetches}{\#user\ requests} \tag{16}$$

- ***Prefetching Timeliness***: Timeliness measures how long the item had been residing at edge storage before it got evicted. If the item received a cache hit, its timeliness timer is refreshed. This metric in correlation with prefetching aggressiveness is important since we are decoupling prefetching algorithms from eviction algorithms, as stated previously.