Deep Learning Approach for Efficient Mobile Edge Computing

Thesis submitted to the
Indian Institute of Technology Guwahati
for the award of the degree

of

Doctor of Philosophy

in

Computer Science and Engineering

Submitted by

Anirban Lekharu

Under the guidance of

Prof. Arijit Sur

and

Dr. Moumita Patra

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

August 4, 2023

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Prof. Arijit Sur
and
Dr. Moumita Patra
Department of Computer Science and Engineering
IIT Guwahati
Email : arijit@iitg.ac.in and moumita.patra@iitg.ac.in

Certificate

This is to certify that this thesis entitled, **"Deep Learning Approach for Efficient Mobile Edge Computing"**, being submitted by **Anirban Lekharu**, to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a bonafide work carried out by him under my supervision and guidance. The thesis, in my opinion, is worthy of consideration for award of the degree of Doctor of Philosophy in accordance with the regulation of the institute. To the best of my knowledge, it has not been submitted elsewhere for the award of the degree.

....
Prof. Arijit Sur
Professor

....
Dr. Moumita Patra
Assistant Professor

Date: August 4, 2023
Place: Guwahati

Declaration

I certify that:

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor is not in a position to check for any possible instance of plagiarism within this submitted work.

Date: August 4, 2023
Place: Guwahati

(Anirban Lekharu)

Dedicated to

My Family & Friends

For their unconditional love, blessings and constant inspiration

Acknowledgements

The completion of this dissertation was made possible by a great many individuals. I owe a debt of gratitude to everyone who supported me in preparing this doctoral thesis in its current form.

I want to start by sincerely thanking my supervisors, Prof. Arijit Sur and Dr. Moumita Patra, for their unwavering support, inspiration, endless patience, and valuable guidance throughout this journey. I sincerely appreciate them pointing out my errors and keeping me focused on my PhD work. Prof. Sur and Dr. Patra not only paved the basis for my advancement as a research scientist but also significantly altered my personality, aptitude, and nature. I was privileged to have such mentors who allowed me the independence to explore on my own while also providing me with the guidance to correct my course when my steps faltered.

I also want to extend my sincere thanks to the members of my doctoral committee, Dr. Vijaya Saradhi, Dr. Pinaki Mitra, Prof. Partha Sarathi Mandal, and Dr. Palash Ghosh, for their insightful remarks and recommendations that helped me to increase the calibre and clarity of my work. I appreciate the anonymous reviewers who provided constructive criticism of my research work in numerous forums since it allowed me to improve the quality of my work.

I wish to express my gratitude to Prof. Jatindra Kumar Deka, the head of the Department of Computer Science and Engineering at IITG, for permitting me to use the facilities and resources while pursuing my PhD. I appreciate the technical and administrative assistance provided by the Department of Computer Science and Engineering faculties and staff for the successful completion of my research project. I also like to thank the Academic Affairs office employees for their assistance in handling my grant requests and application submissions.

I want to convey my appreciation to the Ministry of Human Resource Development, Government of India, for the financial assistance provided throughout my years of doctoral study; without it, this research would not have been possible. Additionally, I would like to thank the IITG Welfare Board and the Department of Computer Science and Engineering for providing me with travel grants so I could present my research.

I am grateful to my friends Alakesh Kalita, Hemanta Baruah, Debanjan, Suraj, Yashdeep, Hemraj, Menaxi, Nidhi, and Vanshali for their encouragement and support in helping me overcome any challenges I have encountered at work or elsewhere. Many conversations and the exchange of ideas have advanced our research. I would to mention a few seniors of my Lab Brijesh Singh, Sibaji Gaj, Satish Kumar, Nilkanta Sahu, Prasen Sharma, Shuvendu Rana who guided and helped me as their own young brother during every phase of my PhD. I also want to express my gratitude to other junior members of my Lab for their unwavering assistance and support, in particular, Alik Pramanik, Sonal Kumar, Sandipan Sarma, Akshay, and Avinash. My time at IIT Guwahati was made memorable by you. I would like to mention few undergraduate Lab members Mouli, Annanya, Pranav and Saket for their unconditional love and support.

Last but not least, I want to express my gratitude to the Almighty God and my family, including my wife Panchali, parents, brother, and in-laws, for their constant support, love, warmth, and profound inspiration over the years. My wife, Panchali, deserves special thanks for her unwavering faith in me, which has made the voyage extremely joyful, secure, and comfortable. I want to express my gratitude to my parents for being a consistent source of support and care. I am at a loss for words to thank them adequately.

Abstract

Mobile data traffic has increased enormously in recent years with an increase in mobile and smart devices. Global mobile data traffic is set to increase manifold in the coming years. With the rise in mobile data traffic and heterogeneous mobile devices, substantial improvement has been achieved in wireless media technology in terms of providing a varied range of multimedia services. These multimedia services are often resource-hungry and require high-speed data and low-latency transmission. High-speed networks like the Fifth Generation (5G) networks help in faster data delivery resulting in less congestion at the backhaul links and higher transmission capacity. Integrating Mobile Edge Computing (MEC) into the cellular architecture provides advantages like intelligent and efficient context-aware caching, video adaptations for content delivery, live transcoding, lower energy consumption and many more. With MEC in the picture, the internet and its multimedia content are brought closer to the end-users. A cellular network can now be described as a three-layer architecture consisting of the Core Network, MEC server within the purview of a given Base Station (BS) and end-users/edge devices.

With the exponential rise in mobile video traffic and dynamic request patterns, maintaining a decent Quality of Experience (QoE) for end-users is challenging for content service providers. MEC provides an opportunity for caching the most requested content closer to the end-users, thereby reducing the overall traffic cost and access delay. Therefore, the primary objective of such a caching strategy is to increase the cache hit rate at the edge server, aiming to improve the end-users' overall QoE. In recent scientific literature, it has been observed that various heuristic and Machine Learning-based caching strategies at the MEC server, have been presented. However, most of the existing caching techniques are not adaptive enough to handle diverse and complicated requests across temporal and geographical dimensions Intuitively, the above problem can be formulated as a multi-objective optimization problem. To solve such a hard problem, learning-based solutions have recently gained popularity, especially using Deep Learning (DL) techniques. Keeping the massive success of DL techniques in mind, in this dissertation, Deep Reinforcement Learning (DRL) is used to design an efficient and robust caching mechanism at the MEC server. The five major contributory works presented in this thesis can be summarized as follows.

It has been observed that most of the existing edge caching methods needs to consider the category or the genre of video content being watched by the end-users. The first contribution of this thesis considers the user profile, which frequently changes in different time slots of the day. For this, a DL-based content-aware caching model (called DCache) has been proposed, which is deployed at the MEC server. This model considers different users viewing profiles at different time slots of a day to predict the popularity of a video content (movie, for our

example) for efficient network caching. To do this end, a two-step model architecture has been proposed to predict genre-based movie popularity index. The first step predicts the most prevalent movie genre at different time slots of the day. And the next step, predicts the movies' future views in different time slots of the day.

The users' preference for viewing a particular video (multiple bitrate representations) for a wide range of users might vary depending on the dynamic network conditions. For example, a specific user with an HD device with higher network bandwidth may demand higher resolutions or HD videos. But the same is not valid for users with low network bandwidth, causing significant delay due to high network congestion and thus the degradation of users' QoE. Under such a scenario, Adaptive Bit-Rate (ABR) streaming have been employed as a solution in the content delivery networks to improve users' overall QoE. To this end, in the second contributory chapter, a Reinforcement Learning (RL)-based ABR caching mechanism at the MEC server within the purview of a single BS has been proposed. This work introduces a novel joint optimization framework using RL called *ABRCache*, which improves the overall QoE for a video streaming session and reduces the traffic load on the backhaul links and the overall access delay simultaneously. In addition, *ABRCache* handles the variations in bandwidth pertaining to different mobility models with the inclusion of a Long Short Term Memory (LSTM) module.

However, most of the caching models based on a single dedicated BS are not efficient enough to handle the diverse and complex nature of video request patterns from heterogeneous end-users. To further improve the caching performance, the third contributory chapter presents a collaborative caching called *ColabCache*, where MECs within a given cluster collaborate to serve the requested content. Existing collaborative caching strategies use transcoding at the MEC server, which is computationally expensive. In *ColabCache*, a novel Deep Reinforcement Learning (DRL) using Asynchronous Advantage Actor Critic Network (A3C) network for caching at the edge server has been proposed. *ColabCache* introduces a novel cache *admission* and *eviction* policy based on the calculated Priority Score of video segments concerning all MEC servers in the cluster.

The final contributory chapter is divided into two parts. The first part of the final chapter focuses on caching the video content in a federated manner. Most learning-based caching models are generally trained in a centralized way, which over-consumes the network resources during training and transmission of video requests. Therefore, a Federated Learning (FL)-based caching framework called *FedCache* has been proposed. In *FedCache*, user data is not centrally collected; instead, the model is trained on the individual data of each user, and the central server aggregates the updates from each user. The second part of this chapter focuses on users' viewing experience for a video streaming session. QoE management using a fixed set of rules may not always guarantee optimal bandwidth utilization, video quality enhancement and accurate buffer estimation, especially in the face of severely varying and

often unpredictable bandwidth fluctuations. To handle these issues across a wide range of varying network conditions and QoE parameters, a Deep Neural Network (DNN)-based model is proposed that selects the appropriate video bitrates to maximize the overall QoE of a user for a video streaming session. Finally, the thesis is concluded by summarizing the significant contributions and proposing some relevant future research directions.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

χ_t	Average Reward
δ	Access Delay
γ	Delay in the backhaul link
$\hat{\phi}_{c_i}^{t+1}$	Predicted Movie Views
$\hat{O}_{t+1}^{S_i}$	Predicted slot-wise genre vector for s_i slot
$\mathcal{D}$	Set of Neihboring MEC server
$\mathcal{J}$	Jitter
$\mathcal{L}$	Predicted List of Movies
$\mathcal{M}$	Set of MEC Servers in a Cluster
$\mathcal{P}_{hit}$	Probability of Cache Hit Rate
$\mathcal{V}$	Set of Movies
$\overrightarrow{G}_{avg}$	Slot-Wise Genre Vector
$\overrightarrow{M}$	Movie Genre vector
τ_t	Throughput
ϱ	Backhaul Traffic
ζ	Current Buffer Size
b	Available Bitrates
F_{c_i}	Content Popularity Score

K	Cache Size of MEC Server
K_m	Cache Size of m^{th} MEC server in a collaborative environment
k_{ij}^b	Size of Video Content c_{ij}^b
M	Total number of MEC Servers
n	Total number of videos
$p, \hat{p}$	Actual and Predicted Values
$R(t)$	Re-buffering time
r	Number of Arriving Requests
U	Total Number of Users
v_{ij}^b	Video quality of j^{th} segment of i^{th} video
w_{ij}^{bm}	Cache decison variable for Cluster Miss
x_{ij}^{bm}	Cache decison variable for neighbor hit
y_{ij}^b	Binary Cache Decision Variable
z_{c_i}	Total number of views for movie c_i

Acronyms

1DCNN 1D Convolutional Neural Network.

3GPP 3rd Generation Partnership Project.

5G Fifth Generation.

A3C Asynchronous Advantage Actor Critic Network.

ABR Adaptive Bit-Rate.

API Application Programming Interfaces.

BPTT Back Propagation Through Time.

BS Base Station.

CAGR Compound Annual Growth Rate.

CAM Caching Module.

CCS Core Content Server.

CDM Cache Decision Module.

CDN Content Delivery Network.

CFS Customer-Facing Service.

CHR Cache Hit Rate.

CM Cache Memory.

CNN Convolutional Neural Network.

CRM Content Request Module.

CS	Core Server.
DASH	Dynamic Adaptive Streaming over HTTP.
DL	Deep Learning.
DNN	Deep Neural Network.
DQN	Deep Q-Network.
DRL	Deep Reinforcement Learning.
ETSI	European Telecommunications Standards Institute.
FC	Fully Connected.
FIFO	First-In-First-Out.
FL	Federated Learning.
HD	High Definition.
ILP	Integer Linear Programming.
ISG	Industry Specification Group.
IT	Information Technology.
LFU	Least Frequently Used.
LRU	Least Recently Used.
LSTM	Long Short Term Memory.
MAE	Mean Absolute Error.
MANO	Managment and Orchestration.
MEC	Mobile Edge Computing.
ML	Machine Learning.
MPD	Media Presentation Description.
MSE	Mean Squared Error.

OSS	Operations Support System.
OTT	Over-the-Top.
PSNR	Peak Signal-to-Noise Ratio.
QoE	Quality of Experience.
QoS	Quality of Service.
RAN	Radio Access Network.
RESNET	Residual Neural Network.
RL	Reinforcement Learning.
RMSE	Root Mean Square Error.
RNIS	Radio Network Information System.
RNN	Recurrent Neural Network.
UE	User Equipment.
UPF	User Plane Function.

"A person who never made a mistake never tried anything new."

~Albert Einstein

1 Introduction

In recent times, driven by the pervasive growth in the popularity of mobile devices along with the drastic advancement of network infrastructure and wireless bandwidth, there has been an exponential rise in mobile data traffic. According to a survey done in [8], mobile data traffic increased sevenfold globally between 2017 and 2022. It grew at a Compound Annual Growth Rate (CAGR) of 46 % between 2017 and 2022, reaching 77.5 exabytes per month by 2022. The survey also stated that by 2023, the number of mobile devices worldwide is expected to grow up to 13.1 billion. Video traffic currently accounts for 66% of the total mobile data traffic, which is projected to grow up to 77% by 2026 [9]. This rapid rise in video traffic is due to the proliferation of a wide range of multimedia services and applications such as Over-the-Top (OTT) video streaming services (Netflix, Amazon Prime, YouTube, etc.).

Along with the growth of mobile devices, there are significant advancements in wireless communications for mobile networks. The development of wireless technologies presents a wide range of multimedia services and applications that requires high-speed data rates and low-latency transmissions. In order to meet the traffic needs of diverse and resource-hungry multimedia services, the Fifth Generation (5G) networks help provide faster data delivery, resulting in higher transmission capacity and lesser traffic congestion at the backhaul links. This futuristic high-speed network has to ensure higher bandwidth capacity, lower latency and lower energy consumption [10]. The core and mobile edge networks are the two most essential components of the 5G architecture [11].

Mobile Edge Computing (MEC) [2], [7], [12], [13] strengthens the mobile networks by incorporating cloud computing capacities within the Radio Access Network (RAN). MEC was developed by European Telecommunications Standards Institute (ETSI) [14]. With the integration of MEC between the Core Server (CS) and the RAN, the 5G architecture can provide low latency, proximity, high bandwidth, real-time radio network information and location awareness [15]. With the deployment of the MEC servers, the internet, with its multimedia content, is brought closer to the end-users. The use case of such an approach can be listed as [16]: 1) *Firstly, the Content and Context Aware Caching* at the MEC servers helps in reducing the backhaul traffic, thus improving the latency for a video streaming Session and 2) *Secondly, in Video Adaptations* the MEC servers could capture the real-time network conditions from the various end-users through the Radio Network Information System (RNIS) that can aid in delivering the appropriate bitrate of a video segment during the entire video streaming session. Thus, the MEC servers, which are hosted closer to the end-users, could intelligently store the content based on the context and popularity of the data.

The definition of MEC as provided by ETSI is as follows:

ETSI defined MEC as:

Mobile Edge Computing provides an IT service environment and cloud computing capabilities at the edge of the mobile network, within the RAN and in close proximity to mobile subscribers.

MEC acts as a bridge between the CS and end-user, enabling the network services to be closer to the end-user. MEC which is integrated at the Base Station (BS) or nearer to the BS, improves computations and reduces the backhaul usage, access delay, and system failure [17] [18]. Figure 1.1 illustrates an essential representation of the cellular architecture with MEC integrated within the BS.

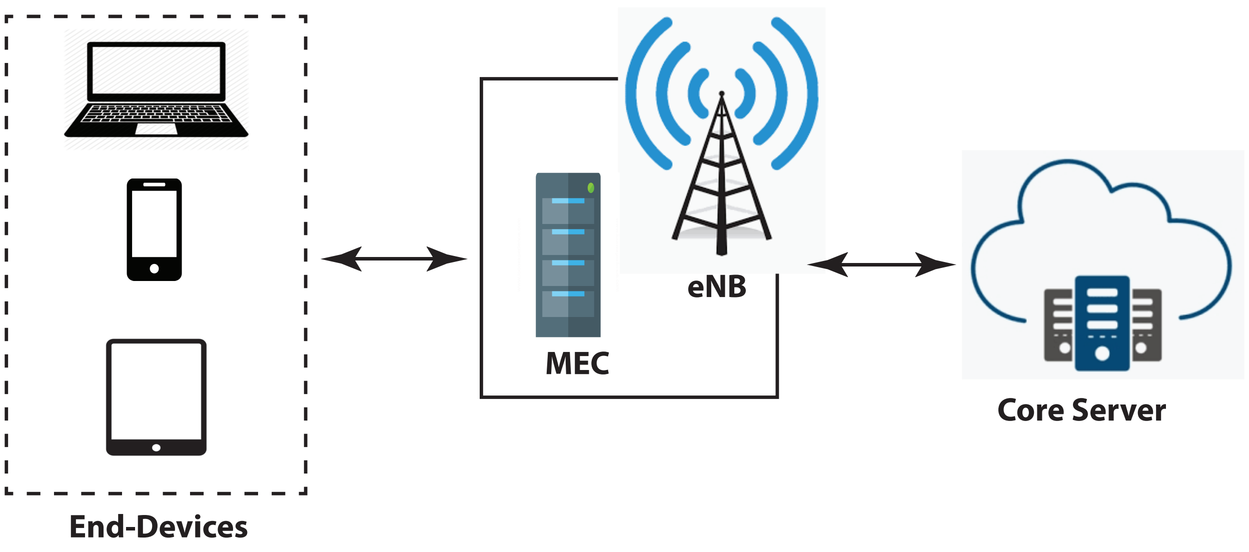


Figure 1.1: *Cellular Architecture With MEC Integrated*

As depicted in Figure 1.1, the BS is connected through the Ethernet/IP to the CS, supporting a high data transfer rate. This high-speed link is referred to as the backhaul link in this dissertation. The end-users / User Equipment (UE) are wireless connected to the BS. With the introduction of MEC into the existing cellular architecture that is placed between the CS and mobile devices, a new three-layer architecture is defined. This three-layer hierarchy comprises the CS, MEC and UE. MEC aids in bringing various network services closer to the end-users that help improve the overall experience of an end-user [19]. The MEC servers, along with storage capacity, also provide highly intelligent CPU/GPU

MEC server	Manufacturer	Storage (GB)	CPU	GPU
Jetson TX2	NVIDIA	32	Quad ARM A57	NVIDIA Pascal
HyperFlex Edge	Cisco	128	Intel Xeon	/
Edge Cloud	Intel	16 to 64	Intel Xeon	/
Power S822LC	IBM	32 to 1024	Power 8	NVIDIA Tesla P100

Table 1.1: *Real-world MEC servers [7]*

computation services. Consider the example of a cache replacement strategy using MEC servers. MEC could collect real-time information about the user preferences and the network conditions for a given channel and make intelligent decisions by predicting which segments to be replaced in the cache to maximize the overall cache hit ratio, thereby reducing the network traffic in the backhaul link [17] [18].

Table 1.1 presents a configuration listing of the MEC servers with respect to CPU, GPU and storage capabilities. From the table it can be seen that the storage capacities of the MEC servers is between 16 GB and 1024 GB. Such huge capacities of MEC servers could be utilized to cache popular content which are expected to increase manifold in the coming future. MEC servers not only provides ample space for caching but also has the ability to provide deep learning computations for making intelligent decisions. For example, Nvidia Jetson TX2 is a high-end GPU server which is specially designed for delivering artificial intelligence services in the edge network [7].

Figure 1.2 illustrates the Network architecture for video content delivery using the MEC. In order to provide lower latency or lower access delay for accessing content, MECs are situated in close proximity to the BS. Whenever a user requests content, it is first searched in the MEC server; if found, a cache hit occurs for the given content, and the UE can directly fetch it. Otherwise, when there is a miss for the requested content, the content is retrieved from the CS. Under such a scenario, the access delay increases and also, if there are a lot of misses, then the backhaul usage increase, which may lead to congestion between the CS and MEC.

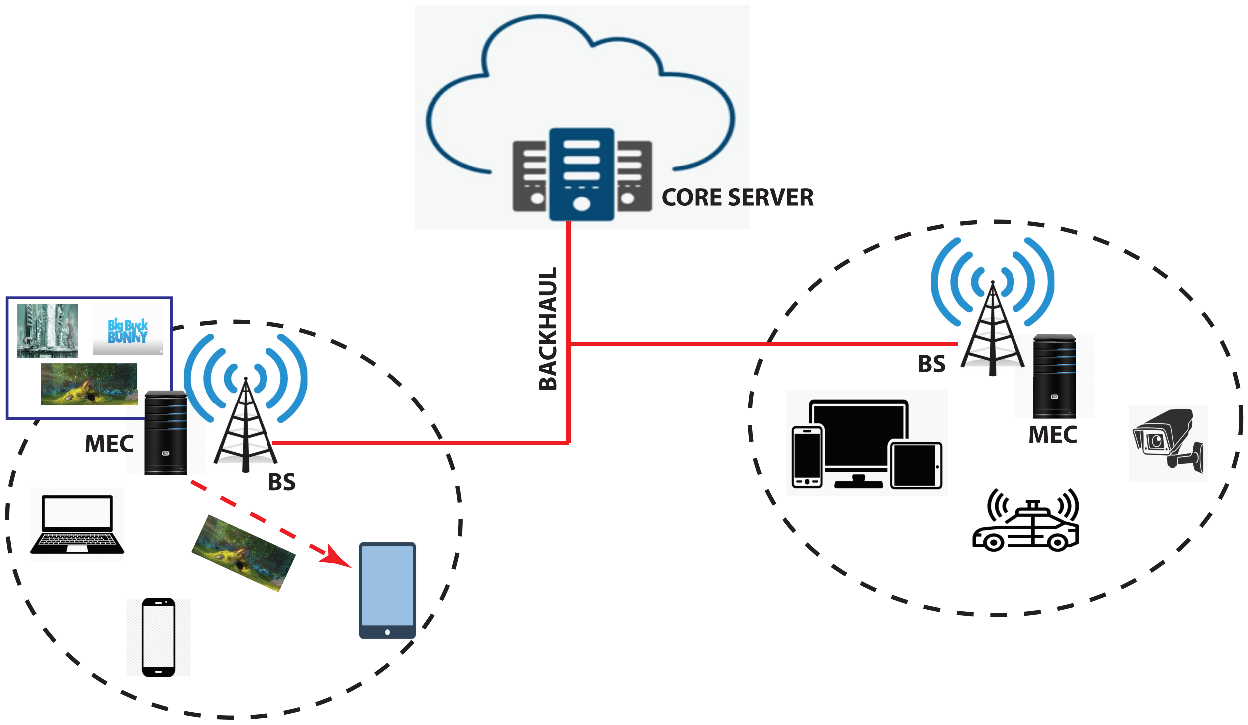


Figure 1.2: *Network Architecture for Video Content Delivery using MEC*

1.1 Characteristics of MEC

MEC can be characterized by the following criteria [20]:

- *Proximity:* MEC being placed at close proximity to the BSs that is closer to the UEs, adds the advantage of processing big data for serving resource-hungry services.
- *Lower Latency:* With MEC services being deployed closer to the UEs, time taken to access a given content directly from the MEC server is less, as network route from the CS is avoided.
- *Location:* As MEC servers receive the network and channel conditions from the UEs, it helps to get the location of the edge devices which help in providing better services.
- *Network Context Information:* The MEC servers receive the real-time network data from RNIS. This real-time information helps in predicting the network bandwidth and estimating the congestion in the radio cell, which helps in making intelligent decision for caching and content delivery based on video adaptations.

With MEC in the picture, content and service providers are left with a challenging task to deliver a satisfactory Quality of Experience (QoE) to the end-users during peak traffic hours with the continued exponential growth of mobile data traffic. The primary goal in achieving this is to design a caching mechanism which is effective for the edge server that caches the most popular or often requested content. An efficient caching mechanism ensures that very few requests are forwarded to the core network like the internet, and instead, as many requests are serviced from the edge server. Therefore, the overall traffic at the backhaul link reduces, along with the access delay in retrieving the requested content. Numerous heuristic and Machine Learning (ML)-based caching strategies have been introduced in recent times that aim to improve the overall cache hit rate at the MEC server. Additionally, improving the end-users overall QoE, or viewing experience, is the other goal.

1.2 Motivation of the Research Work

Despite the rise in cellular infrastructure and wireless data transmission speeds, delivering a satisfactory QoE to various latency-sensitive applications remains a challenging task for cutting-edge cellular networks. Providing an adequate user experience remains an open research problem at the edge network with the exponential growth of mobile video traffic. Due to a high degree of temporal variability in mobile video traffic, the traffic congestion between the CS and BS (the backhaul link) increases during peak traffic hours. The network is under-utilized during off-peak hours [21]. Optimal utilization of the MEC servers storage could be achieved by efficiently caching the most popular video content. An efficient caching mechanism at the network edge improves the overall Cache Hit Rate (CHR), which results in the reduction of traffic congestion in the backhaul link along with access latency.

Over the years, there have been many developments in heuristic-based caching [13] [22]. Even though these algorithms are simple and compact, many still try to find the patterns in data and use various mathematical frameworks and analytical models to achieve better cache efficiency. Traditional caching strategies such as Least Least Recently Used (LRU) [23],First-In-First-Out (FIFO) and Least Frequently Used (LFU) [24] are unable to provide an efficient caching strategy as the number and dynamicity of the request grows. However, with the advent of big data and ML-based techniques [25–27] to understand data and its features, caching at the edge server is expected to provide an optimal solution. Meanwhile, the idea of video caching is currently being expanded from video-level to segment-level (consisting of multiple bitrate) [16] [28] and [29]. The users' preference for viewing a particular video (multiple bitrate representations) for a wide range of users might vary depending on the dynamically changing network conditions. For example, a specific user with an High Definition (HD) device with high-speed network conditions may demand higher resolutions or HD videos. But the same is not valid for users with low network bandwidth, as the delay incurred will be too large, resulting in the degradation of users' QoE. Under such a scenario, Adaptive Bit-Rate (ABR) streaming solutions like Dynamic Adaptive Streaming over HTTP (DASH) [30] have been employed in the content delivery networks to improve users' overall QoE. In DASH, a particular video is split into segments of various time durations (for example, 2, 4, 6, and 8 seconds durations [3]). Every video segment (of particular time duration) consists of multiple bitrates/representations (for example, 240p, 360p, 480p, 720p, 1080p, etc.). Due to heterogeneous user demands, it is challenging to calculate and predict a video segment's most popular (frequently requested) bitrate in advance. Proactively predicting the segments and storing those popular segments in the MEC server's cache reduces the BS network load.

It has been observed that most of the previous caching schemes based on a single BS such as [25,31,32] are not adaptive enough to handle such diverse and complicated requests across temporal and geographical dimensions. Thus, to further improve the caching performance at the edge node, *collaborative caching* [33–35] has been introduced recently. In collaborative caching, MEC servers collaborate amongst themselves through the high bandwidth fronthaul links to serve the requested videos. In collaborative caching, decisions must be made considering factors such as past request patterns and current network conditions. However, a collaborative caching mechanism faces several challenges. *Firstly*, in contrast to non-collaborative caching, a collaborative caching strategy also has to decide the MEC server at which the content should be cached. Thus, another degree of complexity is added to the caching decision. *Secondly*, for DASH videos, caching multiple representations will increase the overhead with respect to the storage capacity of an MEC server. *Finally*, designing an efficient caching mechanism that efficiently utilizes the available computational and storage resources to improve the overall caching efficiency at the edge nodes.

Most of the existing caching schemes, expandability becomes a concern for the architectures when number of clients and the information they generate grows. Furthermore, these existing caching systems are built for overly regulated scenarios where users must share their personal information to an edge server, which relies massively on the central server. Such learning-based caching models are generally trained in a centralized way, which over-consumes the network resources during training and transmission of the video requests. Therefore, to further improve the overall system performance, decentralized caching through Federated Learning (FL) [36–38] has been explored more recently. FL-based caching at the edge server efficiently offloads the computation task from the CS to the end-users. In FL, a model is trained at the user's end instead of the edge server, where the user data is not needed to be transferred to the servers. Moreover, along with designing an efficient caching methodology for storing the most popular content at the edge server, content delivery is another important use case of MEC [39] [40]. ABR streaming, for example DASH is a very well-known adaptive streaming solution, where videos are broken down into segments and bitrates are selected based on the network conditions. The primary challenge for service providers in an ABR streaming is to deliver a satisfactory QoE to the end-users.

Therefore, both content and service providers are left with a challenging task to deliver a satisfactory QoE to the end-users during peak traffic hours with the continuous exponential growth of mobile data traffic. Thus, it can be said that designing an efficient caching strategy in such a dynamic scenario is a challenging task. Thus, the primary objective of this dissertation improve the overall caching efficiency at the MEC and provide a satisfactory QoE to the end-users.

Thesis Goal:

Improving the caching efficiency at MEC using deep learning approaches and optimizing various parameters related to caching (such as CHR, backhaul traffic and acess delay) and QoE (such as video quality, re-buffering, video quality switching).

1.3 Thesis Objectives

Motivated by the above observation, the main objectives of this dissertation are as follows:

- *Content Aware Caching based on the Users Viewing Profile:* Proposing a deep learning model for content-aware caching at the MEC server based on the users viewing profiles at different time-slots of the day.
- *QoE-Aware Adaptive BitRate Caching:* Developing a joint optimization framework using Deep Reinforcement Learning (DRL) that improves the overall QoE of the end-users by focusing and giving equal weightage to both the ABR and caching mechanism at the MEC.
- *Collaborative Video Caching in a Clustered Edge Network:* Proposing a DRL based Collaborative Caching strategy in clustered edge networks, where BSs are clustered based on geographical locations.
- *Decentralized Caching Mechanism using Federated Learning:* Designing a decentralized-based caching mechanism using FL, where data from users is not centrally collected; rather, the model will be trained on the individual data of each user.
- *Prediction Model for Content Delivery in Adaptive Video Streaming:* Finally, the content delivery for a video streaming session is investigated, and a Deep Neural Network-based model is proposed that chooses the proper video bitrates to maximize the user's overall QoE.

1.4 Thesis Contributions

To improve the caching efficiency at MEC, we propose various deep learning based caching mechanism pertaining to the availability of MEC storage capacity, variations of number of users along with dynamic network condition. In th next subsection, we briefly present each of the proposed deep learning-based caching strategies along with content delivery.

1.4.1 Content Aware Caching based on the Users Viewing Profile (DCache)

In our first contributory chapter, we propose a deep learning-based content-aware caching called *DCache* at the MEC server. The proposed learning-based caching strategy focuses on the users' viewing profiles at different times of the day. A time-slot-based hypothetical popularity index has been introduced as a *"genre vector"* to fulfill the heterogeneous end-users (users with various viewing preferences like action, comedy, drama, etc.) demands at different time slots of the day. For simplicity of the design, it has been restricted initially for the Movie database. In this work, *"genre vector"* essentially signifies what genre is prevalent at what time of the day. The ultimate goal is to find a time-slot (of the day) based popularity index of the videos (Movie for our case). The MEC caches are then updated so that a higher number of movie requests can be met. In addition, a novel two-step model has been proposed to predict the popularity of movies. The proposed *DCache* model uses a stream of video requests from various end-users as input data for caching the most popular video content at the MEC server. For experimental purposes, we have used the *MovieLens* [41] [42] dataset in our proposed work. The first step of the model is based on Long Short Term Memory (LSTM) and it predicts the most prevalent movie genre at different time slots of the day. The second step of the proposed model predicts the movies' future views in different time slots of the day using a combination of LSTM and Deep Neural Network (DNN). The efficiency of the proposed *DCache* model, has been evaluated against CHR, *backhaul traffic* and *access delay*. The *CHR* results, along with the other evaluation metrics show that *DCache* significantly outperforms the performance of existing caching policies.

1.4.2 QoE-Aware Adaptive BitRate Caching (ABRCache)

In the second contributory chapter, a QoE-aware ABR caching mechanism named *ABR-Cache* at the MEC server using DRL is proposed. In this work, a Reinforcement Learning (RL)-based joint optimization framework is proposed to improve the overall QoE of the end-users by efficient and appropriate utilization of both the ABR and caching mechanism at the MEC server. The DRL-based framework uses three types of modules namely ABR, *Planner* and *Evictor* to optimally select the most appropriate and popular bitrate based on segment-level popularity and caches it accordingly. In addition, *ABRCache* handles the variations in bandwidth pertaining to different mobility models. The proposed model, combined with LSTM, can determine and extract patterns from the time-series data analysis of variable bandwidth input data.

The proposed model is trained at the MEC server which consists of the ABR module and the *Cache* manager. The cache manager consists of the *Planner* and *Evictor* module, which decides whether to cache a particular video segment or not? The experiments are performed based on the standard DASH [43] video dataset and *network bandwidth* [44] logs. In the simulation setup, the request from the users for DASH video segments follows *Zipf* distribution. *ABRCache* outperformed existing strategies both heuristics and ML-based. As the cache size increases, the overall reward for the proposed *ABRCache* also increases. Therefore, the overall performance of the proposed model is expected to increase with an increase of the cache size in the MEC server.

1.4.3 Collaborative Video Caching in a Clustered Edge Network (ColabCache)

It is observed that caching strategies based on single BS are not robust and adaptive enough to handle diverse and growing mobile video data traffic. Hence, in the third contributory chapter, a DRL-based Collaborative Caching strategy *(ColabCache)* in clustered edge networks is proposed. The BSs are clustered based on geographical locations. Collaborating MEC servers in the clustered edge network caches the video content based on their segment level popularity. A novel *cache admission* and *eviction* policy is proposed, unlike the previous works, which primarily use simple eviction policies such as LRU and LFU. *ColabCache* collaborates amongst the clustered MEC servers to make caching decisions based on the calculated *Priority Score* of video segments with respect to all MEC servers in the cluster. In addition, *ColabCache* is independent of the size of media library at the CS. The proposed model is independent of the total number of videos in the media library, and hence the performance is optimal even when the media library is continuously increasing.

For DASH video streams storing all the representations is not efficient. Hence, only the popular/frequently requested representation of the video segments are cached at the MEC servers. For collaboration, MEC servers are grouped into clusters based on their geographical proximity. Each MEC server collaborates with the other MEC servers in its cluster, referred to as its *Neighbours* in the proposed work. Each user requests and receives video contents from the BS closest to them (with respect to signal strength), referred to as the user's *Home* BS. For experimental purpose, *EUA-Dataset* [45] is used which contains the geographical locations of MEC servers. Along with this, the *Iflix* [46] movie streaming dataset is also used. The *actor* network of the DRL framework takes several input features related to the popularity of content along with various network parameters to generate policies π_θ for deciding the target BS to store the requested video content. The performance of proposed *ColabCache* model have been evaluated against CHR, *backhaul traffic* and *access delay* ,and compared with some of the existing state-of-the-art ML and heuristic based approaches. Substantial improvement is observed for the proposed model.

1.4.4 A Decentralized Caching Mechanism using Federated Learning (FedCache)

In most of the existing caching schemes, expandability becomes a concern for the architectures when number of clients and the information they generate grows. Furthermore, these existing caching systems are built for overly regulated scenarios where users must share their personal information to an edge server, which relies massively on the central server. Therefore, in the first part of the final contributory chapter, a hierarchical Federated Reinforcement Learning-based Content Caching *FedCache* strategy is presented to address these issues. For the first time, a Asynchronous Advantage Actor Critic Network (A3C) DRL network has been trained in a Federated way for making caching decisions at the edge server. *FedCache* offers a scalable solution for training diverse request patterns by transferring the training process to the UEs instead of centrally at the edge server. The proposed model uses *Iflix* [46] dataset for evaluating caching scenarios.

The proposed architecture of DRL-based *FedCache* consists of a novel Caching Module (CAM) using A3C network. *FedCache* primary objective is increasing the CHR. Experimental results reveal that, as the cache size of MEC server increases from 64 GB to 128 GB, the CHR increases by almost 21%. Therefore, more popular video segments could be stored as the cache size increases, resulting in higher CHR. *FedCache* performance increases by almost 5% and 12% for cache size of 64 GB and 256 GB respectively when compared with the next best strategy *AviC* [25].

1.4.5 Prediction Model for Content Delivery in Adaptive Video Streaming (LASH)

In the second part of the final contributory chapter, DL-based content prediction model for adaptive video streaming called *LASH*. QoE management using a fixed set of rules may not always guarantee optimal bandwidth utilization, video quality enhancement and accurate buffer estimation, especially in the face of continuously varying and often unpredictable bandwidth fluctuations. To handle these issues across a wide range of varying network conditions and QoE parameters, ML strategies are being used in recent times. However, it has been observed from existing literature that both heuristics and ML-based approaches fail to satisfy the three important QoE verticals *(perceived video quality, buffering time and video quality switches)* simultaneously. Hence, in this work a LSTM-DNN based on DRL has been devised which is trained with a large set of input parameters. These parameters essentially model the dynamic control rules for handling varying network conditions and end user demands. Thus, a more efficient QoE management model can be achieved. The proposed DRL architecture starts learning the control policy for adaptive algorithm and gradually keeps on improving the reward signal measured in terms of QoE. The proposed model maximizes the overall QoE by satisfying the three QoE verticals such as overall video quality, re-buffering and video quality switches simultaneously. The model is trained over the standard HSDPA dataset [47].

1.4.6 Summary of Contributions

In this dissertation, deep learning based approaches for content management along with content delivery is presented. A chapter-wise summary of contributions is narrated as follows.

- Chapter 3 contributions is summarized as:
  - A time-slot-based hypothetical popularity index has been introduced as a *"genre vector"* to fulfill the heterogeneous end-users (users with various viewing preferences like action, comedy, drama, etc.) demands at different time slots of the day.
  - A novel two-step model has been proposed to predict the popularity of movies. The first step predicts the most prevalent movie genre. And in the second step, the proposed model predicts the movies' future views in different time slots of the day.

  - The next significant contribution is proposing a cache storing and replacement strategy learned using the *"genre vector"*.

- Chapter 4 contributions is summarized as:
  - To improve the overall QoE for a video streaming session and to reduce the traffic load on the backhaul links, we introduce a novel joint optimization framework using DRL.
  - The variations in bandwidth pertaining to different mobility models is handled by the proposed QoE-aware caching model.
  - A novel hierarchical architecture comprising of ABR, *Planner* and *Evictor* module is proposed to maximize the overall QoE.

- Chapter 5 contributions is summarized as:
  - A novel DRL-based collaborative caching mechanism using A3C for clustered MEC server is presented.
  - A collaborative caching framework is proposed that collaborates amongst the clustered MEC servers to make caching decisions based on the calculated Priority Score of video segments.
  - The collaborative caching model is independent of the total number of videos in the media library, and hence the performance is optimal even when the media library is continuously increasing.

- Chapter 5 contributions is summarized as:
  - A FL-based caching model is proposed that offers a scalable solution for training diverse request patterns by transferring the training process to the UEs instead of centrally at the MEC server.
  - The Federated caching model measures segment-level popularity and trains the A3C network to learn which and how many segments to be evicted from the local cache, to cache a new segment.

– Propose a content prediction model for ABR streaming that predicts the optimal bitrate of video segments.

– The proposed content prediction model maximizes the overall QoE by optimizing various parameters simultaneously.

1.5 Organization of the Thesis

The overall organization of the thesis is outlined as follows:

Chapter 1: Introduction

This Chapter introduces Mobile Edge Computing and content caching and delivery, address the motivation of this dissertation considering the research gaps in the recent literature. A brief chapter-wise contributions are presented, followed by summary of contributions and thesis organization.

Chapter 2: Background and Literature Survey

This Chapter provides detailed background on Mobile Edge Computing followed by the discussion on state-of-the-art works related to caching and content delivery using both heuristic and deep learning approaches. The chapter is concluded with a summary of research gaps of the existing literature.

Chapter 3: Content Aware Caching based on the Users Viewing Profile

In first contributory Chapter, a two-step deep learning model for content-aware caching at the MEC server based on the users viewing profile at different periods of the day is proposed.

Chapter 4: QoE-Aware Adaptive Bit- Rate Caching

The next contributory Chapter introduces a QoE-Aware caching mechanism, which jointly optimizes both ABR and caching parameters simultaneously. The proposed model uses DRL to solve the above problem improving the overall QoE for a video streaming session along with traffic load on the backhaul links.

Chapter 5: Collaborative Video Caching in Clustered Edge Network

The third contributory Chapter presents a novel caching mechanism to further improve the caching performance, where multiple BSs collaborate amongst themselves. A novel cache admission and eviction policy based on the priority of video segments is proposed using DRL.

Chapter 6: Federated Caching and Prediction Model for Content Delivery

In the fourth and final contributory Chapter, a decentralized based caching mechanism using Federated Learning is proposed. Further, a deep learning based content prediction model for adaptive video streaming is presented to improve the overall QoE of a video streaming session.

Chapter 7: Conclusions and Future Perspectives

The thesis is concluded in this Chapter, along with some possible future research scopes .

"The way to get started is to quit talking and begin doing."

~Walt Disney

2 Background and Literature Survey

This chapter initially gives a brief description of MEC deployment. As a significant part of this thesis concerns ABR; therefore, a system-level framework and an example scenario of adaptive video streaming is presented. After that, a thorough analysis of the works that considered caching at the MEC server is presented. In addition, comprehensive details about the dataset that was utilised to perform caching in this dissertation are presented.

2.1 Mobile Edge Computing

Mobile Edge Computing (MEC) is recognised as a critical technology to introduce application-oriented capabilities into the core of a carrier's network and to explore a wide range of new use cases, particularly those requiring low latency. MEC is an essential component in the 5G architecture that extends cloud computing capacity to the edge of cellular networks closer to the UEs, resulting in ultra-low latency, near proximity, context awareness, and high throughput. ETSI Industry Specification Group (ISG) (Industry Specification Group) is considered the home of technical standards for MEC and has published various specifications. For instance, the Managment and Orchestration (MANO) of MEC applications [48, 49], the Application Enablement Application Programming Interfaces (API) [50], the Service APIs [51], and the User Equipment (UE) application API [52] are examples of specifications. The service APIs provide the ability for applications to access underlying network data and capabilities, whereas MANO and application enablement functions help to enable service environments in edge data centres. Applications can use these standardised APIs to obtain contextual information and real-time awareness of their local surroundings, which is one of the core value-adding capabilities of the MEC specification. Next, a brief description of the high-level functional entities of MEC framework is presented followed by the MEC reference architecture.

2.1.0.1 MEC Reference Architecture

Figure 2.1 illustrates the high-level functional entities involved in the MEC framework. The entities are divided into three different categories, namely, *network-level, host-level,* and *system-level* entities. The MEC host and the accompanying MEC host-level management entity comprise the MEC host level. The MEC platform, the MEC apps, and the virtualization infrastructure constitute the remainder of the MEC host.

The 3rd Generation Partnership Project (3GPP) cellular network, the local networks, and the external networks constitute linked external entities of the network level. Local area networks, cellular networks, and external networks such as the Internet are all connected through this layer. The MEC system-level management is on top of everything and, by definition, has a broad view of the entire MEC system. The MEC system comprises the MEC hosts and MEC management required to operate MEC applications inside an operator network or a portion of the operator network.

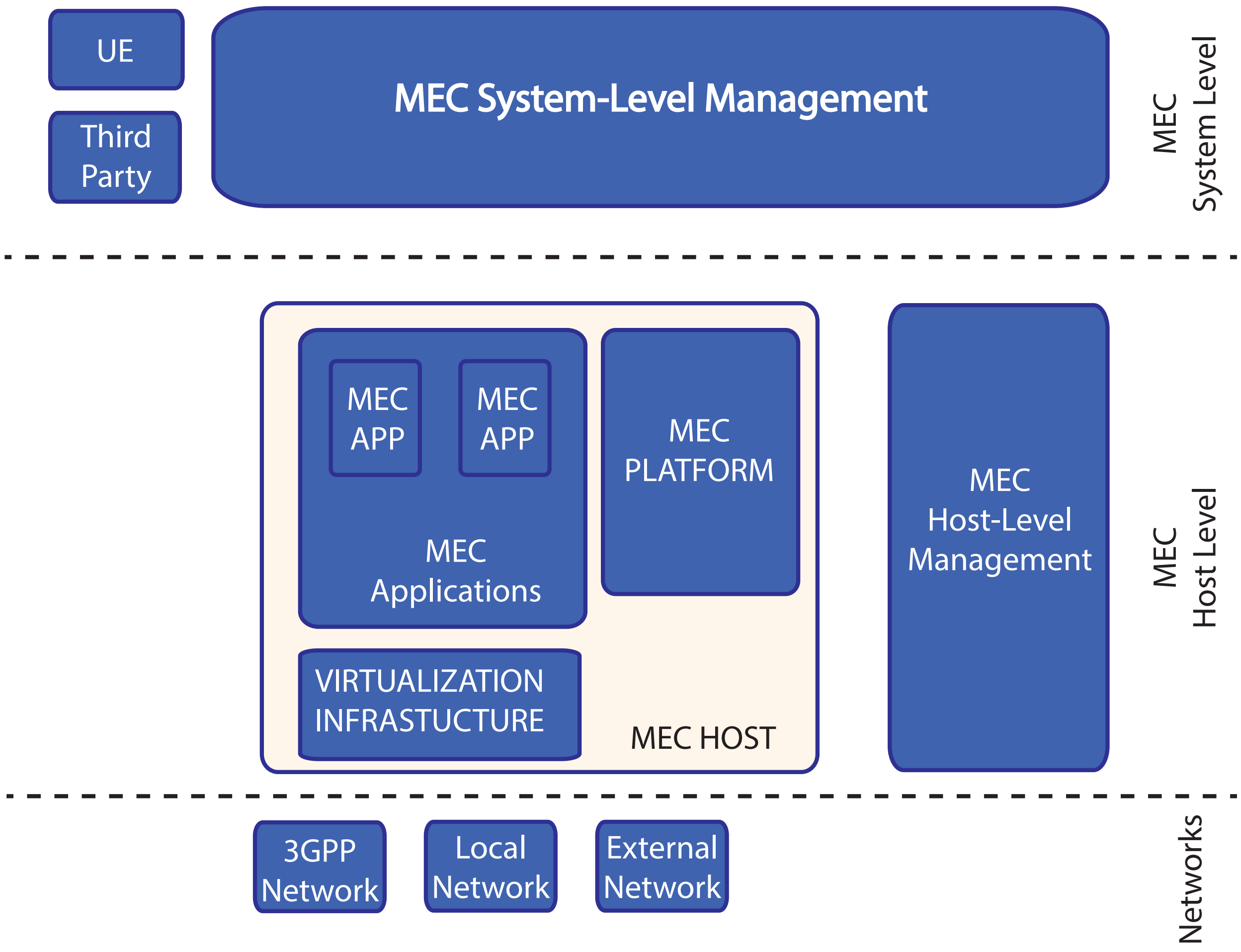


Figure 2.1: *MEC Framework [1]*

As illustrated in Figure 2.2, the MEC reference architecture specifies the functional entities in more depth. It shows how they relate to one another, providing a more detailed understanding of MEC systems. The MEC reference architecture can be partitioned into Host and System level.

- MEC Host Level: The virtualization infrastructure and MEC platform together make up the MEC host, which offers processing, storage, and network resources for MEC applications. The virtualized infrastructure has a data plane that carries out the forwarding instructions sent to the MEC platform and directs traffic between the networks, services, and applications. The MEC platform's fundamental baseline functionalities are required to direct traffic between applications, services, and networks and receives the traffic forwarding rules from the MEC platform manager, MEC applications, and MEC services. The Mp3 reference point helps the MEC platform communicate with other MEC platform. The Mp1 reference point provides service registration, service discovery, and communication support for services. The data plane of the virtualized infrastructure receives instructions from the MEC platform via the Mp2 reference point on how to route traffic across applications, networks, services, etc. Managing the virtualized resources for the MEC applications falls under the purview of the VIM (Virtualized Infrastructure Manager). Allocating and releasing virtualized compute, storage, and network resources offered by the virtualization infrastructure constitutes management activities.

- MEC System Level: The MEC system's primary functionality is served by the MEC orchestrator, which has access to all of the resources and capabilities of the MEC network. Additionally, by instructing virtualized infrastructure managers on how to handle the apps, the orchestrator prepares the instantiation processes. The Operations Support System (OSS) is the highest-level management system that can help the MEC system get the MEC apps running in the desired place on the network. The Customer-Facing Service (CFS) portal and the clients in the UE send requests to the OSS to start and stop the MEC applications UE. The MEC orchestrator receives the requests that the OSS approves for further processing. The CFS serves as a point of entry for third parties and can be used to oversee the ordering, choosing, and provisioning of MEC applications. The MEC-related clients and apps use the user application lifecycle management proxy (user app LCM proxy) function to make service requests for the applications' on-boarding, instantiation, and termination.

2.1.0.2 Deployment Scenarios

A variety of solutions are available for the physical deployment of MEC servers depending on different operational, performance, or security-related requirements, which are mentioned in the White Paper published by ETSI [2].

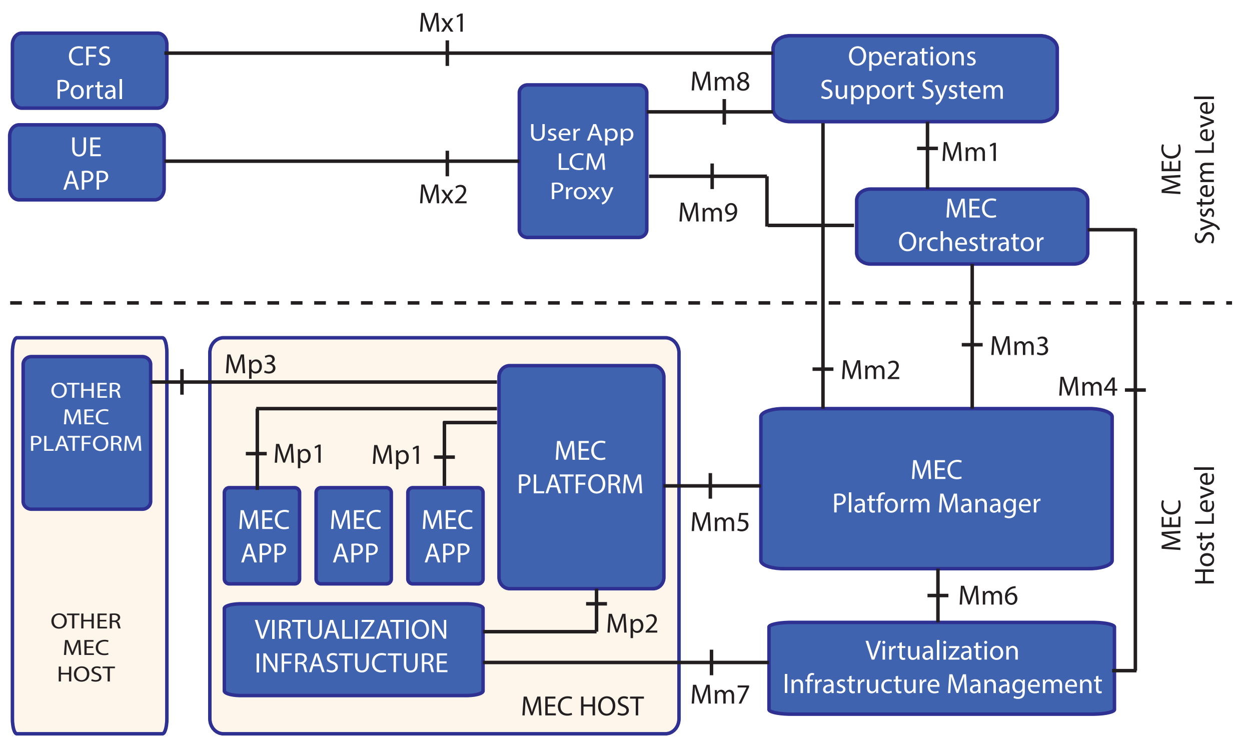


Figure 2.2: *MEC Reference Architecture [1]*

1. MEC and the User Plane Function (UPF) collocated with the BS. (Used in Our Proposed Work)
2. MEC collocated with a transmission node, possibly with a local UPF.
3. MEC and the local UPF collocated with a network aggregation point.
4. MEC collocated with the Core Network functions (i.e. in the same data centre)

As illustrated in Figure 2.3, the options for the physical deployment of MEC demonstrate how MEC can be adaptably implemented in various locations, from close to the BS to the core Data Network. The UPF is deployed and utilised to direct traffic towards the network and the targeted MEC applications in all deployments. In this dissertation, we assumed the first scenario throughout our proposed experimental setup, where MEC and the BS are collocated to simulate realistic network scenarios.

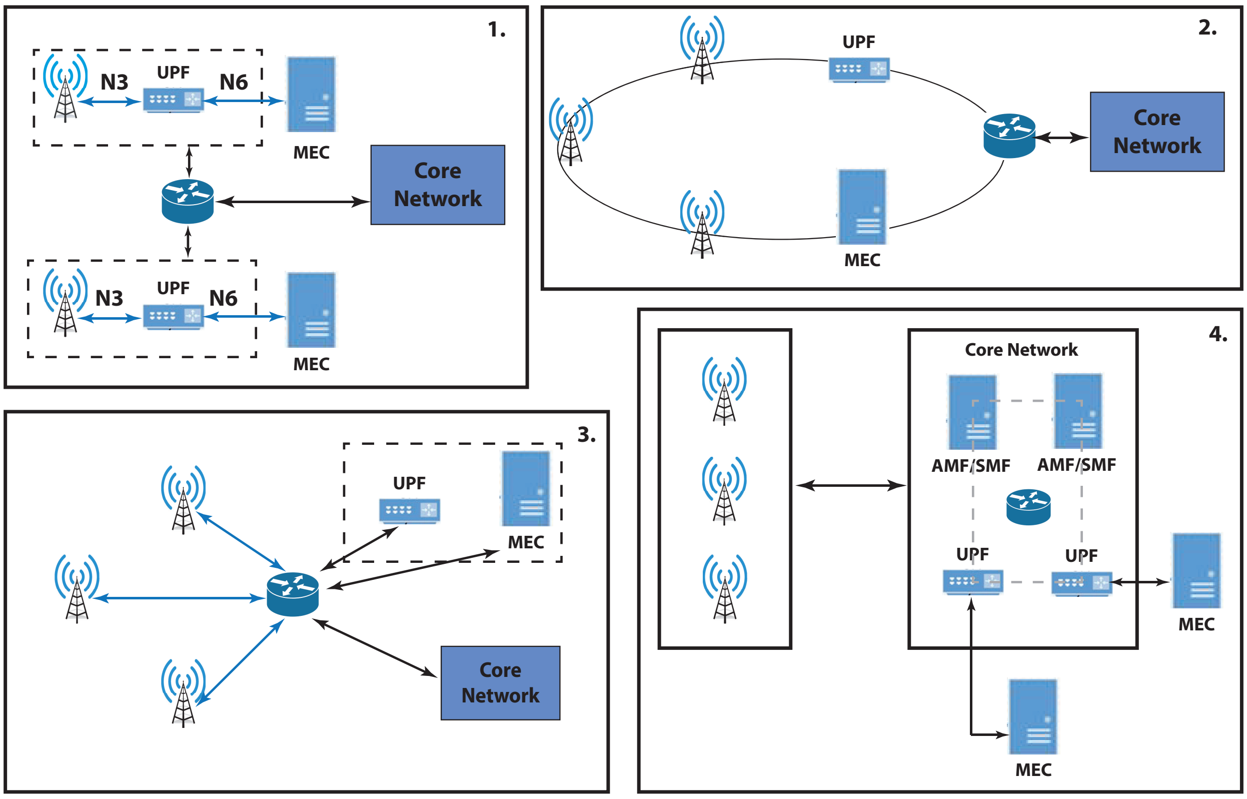


Figure 2.3: *MEC Deployment Scenarios [2]*

2.2 Adaptive Video Streaming

In this dissertation, we considered Dynamic Adaptive Streaming over HTTP (DASH) [43] as the streaming solution for evaluating the proposed content caching and delivery mechanism. DASH is codec independent and works with various standard video codecs such as MPEG, AVC, SVC, H.264, and other codes. The DASH dataset available at [53] breaks down videos into various segment duration (1, 2, 4, 6, 10, 15) seconds. Every video segment (of a particular duration) consists of multiple bitrates/representations (240p, 480p, 720p, 1080p, etc.). We can generate segments of various duration using the DASH Encoder [54]. Wowza's integrated CDN, Akamai, Microsoft Azure, and Amazon CloudFront are some of the CDNs for live streaming that offers DASH support which is cost-effective and scalable.

Figure 2.4 illustrates the working principle of DASH [3]. The video content stored in a DASH server is broken down into various segments of multiple representations. The Media Presentation Description (MPD) contains information about the available video representations in the DASH server. UE parses through the MPD and fetches the appropriate video

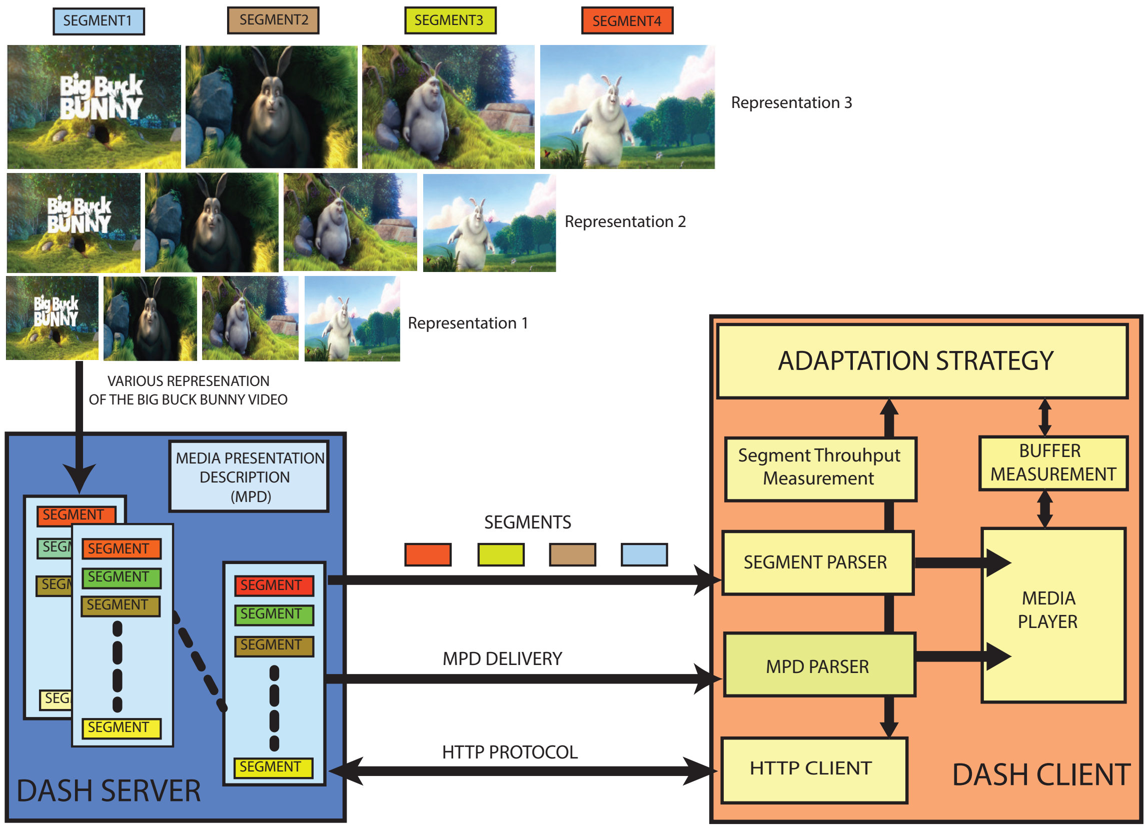


Figure 2.4: *Working Principle of DASH [3]*

representation based on available network conditions. Such a dynamic adaptation control strategy enables the end-users to experience an enhanced QoE for a video streaming session. In the contributory chapters of this dissertation, such as Chapter 4, 5, and 6, we considered the DASH videos for content caching and delivery at the MEC server.

The bitrates corresponding to the video resolution considered in this dissertation are 240p $\rightarrow$ 300kbps, 480p $\rightarrow$ 1200kbps, and 720p $\rightarrow$ 1850kbps. This is consistent with the YouTube video requirement [55]. The videos are in DASH format, where every video is broken down into segments of various seconds. Such types of consistency regarding video resolutions were used for the experimental evaluation of the proposed works, which could be changed to any number and type while re-training the proposed models.

2.3 Literature Survey

This section presents a literature survey of various caching strategies at the MEC server. We categorized the caching techniques into two types: 1) Caching within a Single BS and 2) Caching amongst Multiple BS. Both these types of caching focus on heuristic and ML-based approaches.

2.3.1 Edge Caching Within Single BS

2.3.1.1 Heuristic Based Approaches

Most of the existing caching strategies used simple heuristic-based mechanisms such as Least Recently Used (LRU) and Least Frequently Used (LFU), and their variants [23], and [24] for caching popular content in the MEC server. However, LRU and LFU caching strategies are not robust enough to handle diverse video requests, which have increased drastically at present. Therefore in recent years, various caching strategies that focus on the context of video have been studied extensively. In order to ease and reduce the costly transmissions from the BSs to UEs by controlling the backhaul network congestion, some of the most popular contents need to be cached at the femto base stations and the UEs [56]. The authors in [57] proposed a heuristic-based approach to select the most popular video segments for caching. A joint optimization strategy based on video caching and real-time transcoding is formulated. Chang Ge et al. [13] presented a video caching strategy, which aims at improving the QoE. The proposed scheme performs a two-stage caching mechanism where videos are sorted based on their content-level popularity. Each content is then sorted based on the segment-level popularity. Segments with the least popularity are deleted, with at least one representation for every segment being present in the cache. In [58] Suoheng Li et al. proposed *PoPCache*, a caching algorithm based on the popularity of content. The popularity of the requested content is measured. If its popularity score is more than the existing content in the local cache, *PoPCache* replaces the content with the least popular score in the cache. Otherwise, no content is evicted from the current cache. Berger et al. [31], proposed *AdaptSize* an adaptive, size-aware caching policy for the Content Delivery Network (CDN) servers Hot Object Cache (HOC). *AdaptSize* is based on the Markov cache model, which adapts to the varying request sequences from heterogeneous end-users. A dynamic size threshold is used by *AdaptSize*'s admission control to determine whether or not to accept video segments into the cache. It assigns a low probability of admission to video chunks which are large. High bitrate chunks are frequently prevented from accessing the cache by *AdaptSize*'s admission control, which has a negative impact on performance. Chenglin Li et al. [29] presented a QoE-driven cache placement policy for adaptive streaming. The strategy considers videos' rate-distortion (R-D) characteristics and the coordination among edge servers using Integer Linear Programming (ILP) to optimally place the various bitrates of multiple videos in the cache.

2.3.1.2 Machine Learning Based Approaches

The heuristics or the fixed-based rule for content caching could hardly adapt to the huge volumes of streaming data along with network parameters to study the feature patterns of the data collected at the MEC server. To handle the big data, numerous deep learning models [59] are being introduced in the recent times for improving the computations capacity and also to extract more accurate and precise feature patterns from the data collected. [60] mentioned that the three *V's* model portray the characteristics of big data namely, *volume, variety and velocity.* Here *volume*, represents the huge amount of data, *variety*, signifies the variations of different data types and *velocity*, refers to the rate at which data is being streamed. Therefore, in the near future it is expected that DL models along with the big data framework plays an important role in shaping and designing intelligent network architecture that learns complicated and diverse network information and finally improve the overall working principle of the network. In [61], Arvind et al. proposed *DeepCache*, which uses *Object Characteristics Predictor* to predict the future popularity of content. The caching policy of the *DeepCache* is based on LRU. In *RLCache* [32], a cache admission policy for content caching is proposed at the CDN using Deep Learning. *RLCache* uses a large set of features like object size, recency, and frequency of access to calculate and predict the object popularity based on traces like video, images, and webpages collected from *Akamai's* CDN server [62]. Zahaib et al. [25], proposed *AViC* a caching policy based on prediction of video request and the presence of highly unpopular chunks called *singletons*. *AViC* avoids caching the video contents, which are singletons, so that the cache is not filled with the least popular chunks. Again, the authors in [63] proposed a DRL-based framework with the *Wolpertinger* architecture for caching at the BS. Without prior knowledge of content popularity, the proposed framework maximizes the long-term cache hit rate. Sadeghi et al. [64] proposed a scalable DRL-based framework using hierarchical caching in CDN. The work uses a hyper-Deep Q-Network (DQN) model to learn the parameters for training an optimal cache replacement policy.

2.3.2 Edge Caching Amongst Multiple Base Station

Most of the above-mentioned caching mechanisms use a single MEC server within a given BS for making caching decisions. The continuous increase of mobile video traffic of heterogeneous end-users has led to high variations in video request patterns across temporal and geographical dimensions. The diverse and dynamic nature of the request patterns has revealed that dedicated caching models based on a single MEC need to be more robust for such dynamic and complicated context-aware content. Thus, to further improve the caching performance at the edge node, caching amongst multiple BS, called collaborative caching, has been explored recently. In collaborative caching, MEC servers collaborate amongst themselves through the high bandwidth links to serve the requested videos.

2.3.2.1 Heuristic Based Approaches

The authors of [65] suggested a cooperative caching method in an effort to reduce the overall cost that the content producers would have to bear. They solved the said optimization problem using ILP. The proposed model did not consider the popularities of content for making caching decisions. Again in [66], the authors proposed a cooperative hierarchical caching strategy called *CHC*. *CHC* caches the content both at BBU (*Baseband Unit*) and RRHs (*Remote Radio Head*). Such caching policies ensures minimizing the latency/capacity for an edge based caching model. Tran et al. [33] proposed a collaborative-based caching mechanism for adaptive streaming. They formulated a caching policy at the MEC server using ILP. The collaborative caching mechanism in this work deploys two primary mechanisms, one is for caching and the other for processing (transcoding). In [67], Baccour et al. proposed a heuristic-based collaborative caching mechanism at the MEC server. The authors presented a collaborative caching and transcoding mechanism at the MEC server, which proactively caches the content based on the popularity.

2.3.2.2 Machine Learning Based Approaches

However, heuristics or the fixed-rule-based caching mechanism for a cooperative caching policy is not very much efficient. As the number of collaborating MEC increases, the overall framework becomes more complex and will not be robust enough to handle such big data that varies across heterogeneous users. With heuristic-based approaches solving multi-objective optimization problems becomes challenging. Therefore, Guo et al. [68] proposed a collaborative caching mechanism for adaptive bitrate streaming using transcoding at the edge of the network. Another collaborative-based caching for vehicular edge network is presented by Guanhua Qiao et al. [69]. The authors proposed a Deep Deterministic Policy Gradient (DDPG) method for jointly optimizing the content placement and delivery in vehicular edge networks. Fangxin Wang et al. [34] proposed multi-agent DL-based caching mechanism using RL called *MacoCache*. *MacoCache* improves the caching performance by minimizing the viewing latency and the traffic cost at the same time. [35] proposed another DL-based collaborative caching strategy using DQN at the MEC. The authors formulated a joint optimization problem for content caching and transcoding at the edge server. The above discussed caching mechanism focused on a centralized collaborative training approach. In this dissertation, we also explore decentralized caching mechanism called Federated Learning (FL). FL [36–38] has been introduced recently to improve the caching performance at the edge server and efficiently offloads the computation task from the central server to the end-users. In FL, a model is trained at the user's end instead of the edge server, where the user data is not needed to be transferred to the servers. Zhengxin Yu et al. [36] proposed a FL-based caching strategy called *FPCC*, which carries out training in a decentralized way using stacked auto-encoder. *FPCC* primary objective is to increase the caching efficiency and reduce the users request response time. In [37] Wang et. al proposed a Federated DRL-based caching mechanism called *FADE* for a decentralized cooperative based caching mechanism. In this work, the BSs train in a federated way for making a caching decision. Zhengxin Yu et al. [70] proposed a Mobility-aware Proactive edge Caching scheme based on FL, called *(MPCF)*. *MPCF* employs a Context-aware Adversarial AutoEncoder to predict the highly dynamic content popularity and multiple vehicles collaboratively learn a global model.

2.3.3 Improving the QoE for Content Delivery

With MEC in the picture, both content and service providers are left with a challenging task to deliver a satisfactory QoE to the end-users during peak traffic hours with the continuous exponential growth of mobile data traffic. Therefore, in this dissertation along with video caching strategies we also explored optimal video delivery for adaptive video streaming using DRL. Literature reveals that research based on adaptive video streaming have progressed and primarily grouped into two classes of strategies namely, heuristics [71–74] and ML-based [6,75,76] approaches to maintain uninterrupted high quality video viewing experience. ABR strategies like [73] and [77] are based on bandwidth measurement. C Liu et al. [77] proposed a rate adaptation strategy for adaptive video streaming called *RAHS* that captures the variations in the bandwidth and fetches the most relevant bitrate of the video segment. The ABR strategy in *RAHS*, being over aggressive may encounter frequent bitrate switches when fluctuation within the network parameters is high. ABR strategies that rely only on bandwidth measurement, without any prior information about the client buffer status, may result in buffer underflow as there may not be sufficient playback buffer to handle the next video segment. In [71] Huang et al. mentioned that the bitrate of a video segment fetched by the client is a function of current buffer status. However, such buffer-based approaches [71] and [78] fails to adequately solve the buffer underflow problem that solely depends on the buffer occupancy of the client and is unaware of the bandwidth fluctuations. In order to rule out the possibilities of buffer outages and frequent switching of video quality arising out of ABR strategies mentioned, works like [74] and [79] considers both the buffer occupancy and the bandwidth of the network. The ABR strategy in [74] is a combination of both buffer and bandwidth. It fetches a higher video quality whenever both the current play-out buffer and the available network bandwidth is adequate for upgradation. Similarly, when both the buffer occupancy and the instantaneous bandwidth is not sufficient enough, it selects the next lower quality video.

ML-based strategies, provide an edge over the heuristics-based approaches as it can handle a large number of input data simultaneously for the often unpredictable varied network conditions of heterogeneous end-users. Different system parameters describing frequently changing network characteristics and diverse requirement for heterogeneous end users are considered as input feature of the neural network for designing the ABR model. Lekharu et al. [80] proposed a LSTM based model for forecasting the video bitrate, buffer and switching independently. The major problem with this model is that it fails to optimize theQoE verticals simultaneously while modeling an ABR strategy to improve the QoE of a video streaming session. To handle this issue, *Pensieve* [6] is proposed, that models the ABR algorithm by a DRL model. *Pensieve*'s gain in performance with respect to the overall QoE reward obtained, comes from minimizing the re-buffering penalty.

2.3.4 Limitations of Existing Works

From, the above related literature survey we observed the following issues:

- It has been observed that most of the previous work fails to consider the category or the *"genre"* of video content being watched by the end-users. In addition, it is very common that user profile frequently changes at different times of the day.

- Most of the caching strategies discussed above failed to optimize the traffic generated at the backhaul links while improving the overall QoE. Existing works either focus on the ABR part or the caching part. Such standalone strategies may not constantly improve the overall experience of the end-users.

- In previous caching schemes, caching under a single BS is not adaptive enough to handle diverse and complicated requests across temporal and geographical dimensions. However, it also has to decide the MEC server at which the content should be cached in a collaborating environment. Thus, another degree of complexity is added to the caching decision. Also, while considering ABR content for caching, multiple representations will increase the overhead concerning the storage capacity of a MEC server.

- It is also observed that most of the collaborative caching mechanisms discussed above deploy transcoding at the MEC server, which is computationally costly. Transcoding a video from a higher to a lower bitrate at the MEC server requires significant time, and computational resources might rapidly deplete the available resources on the edge servers.

- Almost all the previous policies for caching deploy standard algorithms like LRU, LFU, and its variants for content eviction. It is also observed that relatively less attention is paid to policies evicting the least popular content.

- In most existing caching systems, the centralized server collects user information and local data for training a caching model. Hence, as the number of content requests increases, the diversity in request patterns also increases, thus making the model less scalable and adaptive to the ever-changing dynamic scenarios. Such learning-based caching models are generally trained in a centralized way, which overconsumes the network resources during training and transmission of the video requests.

- Finally, in content delivery scenario for a video streaming session, it has been observed that both heuristics and ML-based approaches fail to satisfy the three important

QoE verticals (perceived video quality, buffering time and video quality switches) simultaneously.

2.4 Datasets

The proposed models in this dissertation have been evaluated against various standard datasets related to content popularity, network bandwidth logs, etc. The details of these datasets are mentioned as follows:

- **MovieLens:** The *MovieLens* [41] [42] dataset consists of $26,000,000$ ratings done by $1,62,541$ users between January 09, 1995 and November 21, 2019 for $62,423$ movies. Each chosen user had given ratings for at least 20 movies. Information related to the users such as age, sex and occupation are present in the *MovieLens* dataset. Along with the user's data, the dataset also contains information related to a movie such as Movie-ID, genre, year, day of the year, month and day.

- **Iflix:** *Iflix* [46] movie streaming dataset consists of information related to users profile, viewing time, information related to movie content (such as show_time, genre, video_id). The dataset also identifies psychographic and demographic tags about some *Iflix* users.

- **4G Bandwidth Traces:** The 4G bandwidth logs consists of bandwidth traces for users traveling in different transportation. Jeroen et al. [44] collected the network throughput logs in the city of Ghent, Belgium, for different types of transportation such as foot, bicycle, bus, tram, train, and car.

- **EUA:** The *EUA*-Dataset [45] contains the geographical locations of the MEC servers. The *EUA*-Dataset maintains a set of *edge servers* and *user locations*. The edge servers dataset consists of information such as $\langle SiteID - Latitude - Longitude - Name - State - Licensing - AreaID - PostCode\rangle$.

2.5 Summary

This chapter presents background concepts on MEC framework and its physical deployment along with a brief introduction on DASH. In addition, a short description of the datasets based on content popularity and network bandwidth is discussed, which is used in performing experiments related to the contribution of this dissertation. Finally, related works based on content caching and content delivery is presented. The literature is primarily grouped into two main categories: heuristic-based and ML-based. In the related literature, we first discussed about simple heuristics-based caching strategies within a single BS. But with the increase in diverse video requests along with the volume of content being generated is unable to provide an efficient caching model that could study the feature pattern of the data. For handling such big data, various DL-based models have been introduced recently, which can extract more accurate and meaningful patterns from the data. However, the exponential growth of mobile video data has led to high variations across temporal and geographical dimensions. Dedicated caching models based on single MEC is not robust enough for handling such dynamic context-aware content. Therefore, collaborative caching strategies amongst multiple BS have been explored more recently. In collaborative caching, multiple BS collaborate in making caching decisions. Brief literature on FL is presented along with collaborative caching strategy. In FL, the caching model is trained on the users' local data instead of the central server. Thus offloading the computation task from the central server to the end-users. Finally, we present literature on content delivery to the end-users, which focuses on improving the overall QoE for a video streaming session.

With this background and related literature, this thesis's first contribution will be discussed in the next chapter, where the first caching strategy focuses on an efficient video content-aware cache replacement strategy based on the video content popularity. The popularity is measured in various time slots of the day through a deep learning-based model.

"The best way to predict your future is to create it."

~Abraham Lincoln

3 Content Aware Caching based on the Users Viewing Profile

In this chapter, we discuss the first contribution to improve the caching efficiency at the MEC server by considering the users viewing profiles at different time slots of the day. For this, we propose a novel two-step model to predict the popularity of movies. Movies are grouped based on different *"Genre"*, and the popularity of every movie is calculated based on the number of views recorded for the movie. A DL-based model for caching at the MEC servers based on the content popularity at different time slots of a day is proposed. The proposed model has been extensively evaluated using the standard *MovieLens* [41] dataset.

3.1 Introduction

MEC can provide low-latency, high bandwidth, real-time radio network information, and location awareness for boosting the performance of modern resource-hungry services [15]. Consider the example of a cache replacement strategy using MEC servers, where the RNIS provided by the MEC can collect real-time information about the user preferences and the network conditions within a given BS. The MEC servers can decide in advance on an optimal cache replacement policy to maximize the overall cache hit rate, thereby reducing the backhaul usage of the network. The idea behind content-aware caching is to cache the most requested video content. To achieve this objective, two different types of strategies based on caching exist, namely, *Proactive Caching* [81] and *Reactive Caching* [82]. In *Proactive Caching*, the caching strategy at the MEC servers or the edge nodes predicts the popularity of video content. It caches the most frequently requested content even before the requests from the clients have arrived. Whereas, in *Reactive Caching*, the decision whether to cache a given video content is taken only when the request for the given content arrives.

As mentioned in the literature survey of Chapter 2, the costly transmissions from the BSs to UEs could be eased up and reduced by caching the most popular content at the MEC server and minimizing backhaul network congestion. Suoheng Li et al. [58] introduced a cache replacement strategy based on content popularity called Popularity-Driven Content Caching *(PopCaching)*. *PopCaching* is modelled to learn the popularity of content in the near future and decides whether to cache a given content. Xing Chen et al. [57] proposed a heuristic-based approach to select the most popular video segments for caching. The authors considered three critical components for measuring the video request probability, namely, *1) popularity of a video, 2) preference of the users, and 3) features of various video representations.* A joint optimization strategy based on video caching and real-time transcoding is formulated to solve the problem. However, the traditional heuristic or fixed-rule-based strategies are not efficient and robust enough to handle large volumes of data generated at the MEC. Therefore, ML-based approaches have been explored more recently to improve the computational capacity and extract more accurate and precise feature patterns from the collected data. Kyi Thar et al. [83] proposed a DL-based model called *DeepMEC* to predict the popularity of the videos based on the number of requests received. The authors in [61] proposed *DeepCache*, a DL framework for content caching. *DeepCache* uses LSTM to predict object characteristics such as content popularity and then applies *DeepCache* with LRU as an eviction algorithm for content caching. In [32], the authors proposed *RLCache*, a robust learning-based strategy of cache admission for content delivery. *RLCache* uses object size, recency, and frequency of access to train the model with LRU as the eviction algorithm.

It has been observed that most of the existing caching strategy fail to consider the category or genre of videos/movies being watched by the end-users. Besides, it is very common that user profile frequently changes in different time slots of the day. For example, news and devotional programs may be more frequent choices in the morning, while sports is frequently requested/chosen in the daytime. Hence, the variations in users' viewing profiles through a time-based genre analysis are considered to improve the popularity prediction model. This work presents a DL-based model for content-aware caching based on the users' viewing profiles at different time periods of the day.

The significant contributions of this chapter is summarized as follows:

- A time-slot-based hypothetical popularity index has been introduced as a *genre vector* to fulfill the heterogeneous end-users' (users with various viewing preferences like action, comedy, drama, etc.) demands at different time slots of the day. For simplicity of the design, it has been restricted initially for the Movie database. In this work, "*genre vector*" essentially signifies what genre is prevalent at what time of the day. The ultimate goal is to find a time-slot (of the day) based popularity index of the videos (Movie in our case). The MEC caches are then updated so that a higher number of movie requests can be met.

- A novel two-step model has been proposed to predict the popularity of movies. The first step of the model predicts the most prevalent movie genre at different time slots of the day. And in the second step, the proposed model predicts the movies' future views in different time slots of the day.

- The next significant contribution is proposing a cache storing and replacement strategy learned using the "*genre vector*" defined earlier.

The proposed model outperforms three existing state-of-the-art caching strategies (*DeepMEC* [83], *DeepCache* [61] and *RLCache* [32]) along with three baseline works (*LRU*, *LFU* and *FIFO*) in terms of access delay, backhaul usage, and the cache hit rate .

The rest of the chapter is organized as follows. In Section 3.2 illustrates the system overview of the proposed work. Section 3.3 describes the proposed model for content popularity-based caching at MEC servers. Section 3.4 presents a comparative set of experimental results along with the dataset for the proposed model against recent literature. Finally, the chapter is concluded in Section 3.5.

3.2 System Overview

In the conventional Cellular RAN setting, various end-users are connected with BS, which provides them the requested video content from the Core Content Server (CCS). The CCS is connected with BS through Backhaul Link. The MEC servers are co-located with every BS in a cellular RAN. MECs can run various ML-based algorithms to help the BS efficiently serve multiple requests from heterogeneous end-user devices. For example, Nvidia Jetson TX2 is a high-end GPU server specially designed to deliver artificial intelligence services in the edge network [2]. In recent times, MEC servers deployed at the edge of RAN have used efficient caching mechanisms to store popular video content to improve the overall cache hit rate, reducing network congestion in the backhaul links. Figure 3.1 depicts a hypothetical example of such a MEC setting. The MEC consists of three essential modules, namely, Content Request Module (CRM), Cache Decision Module (CDM) ,and Cache Memory (CM). The CRM module receives video requests from heterogeneous end-users within a given BS. CRM then forwards the requests to the local cache, and the request log is sent to the CDM. These request logs are then used for training a popularity-based caching model. If the

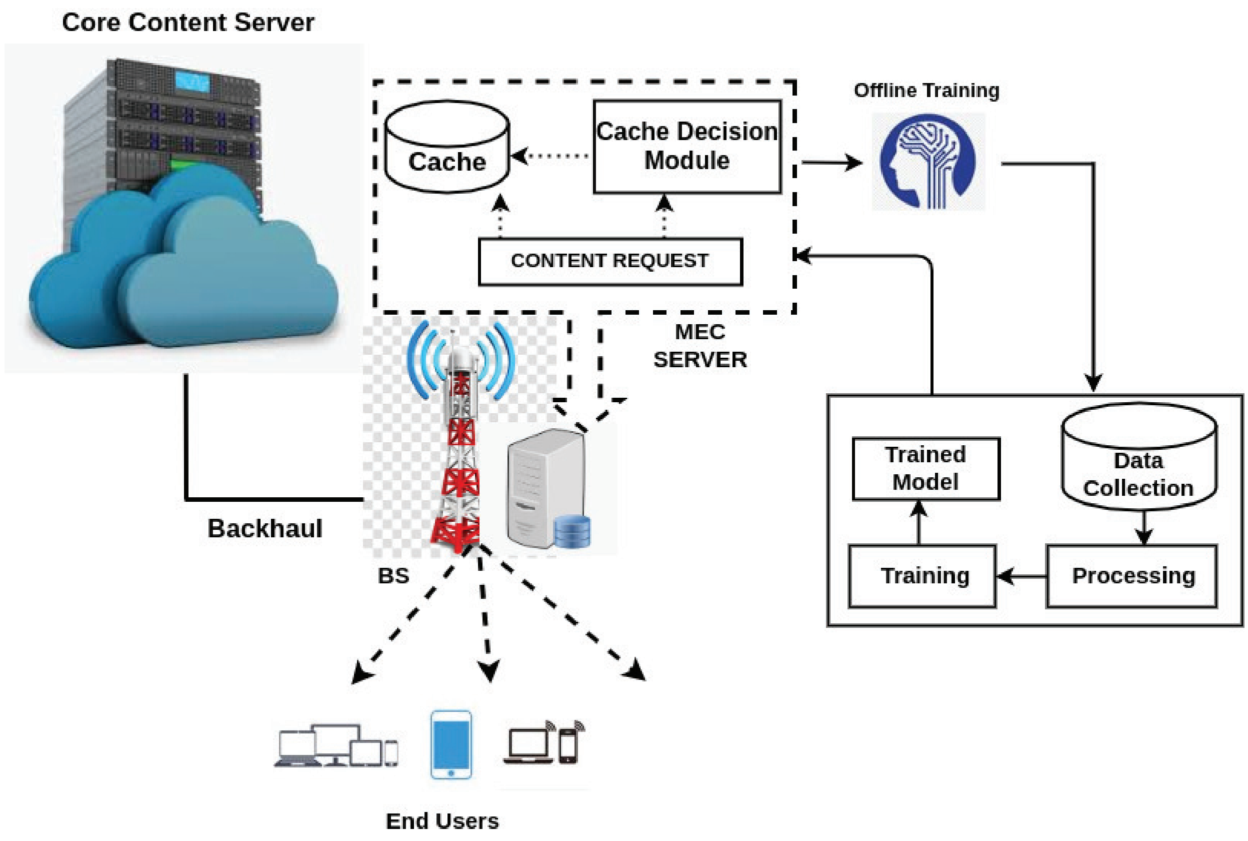


Figure 3.1: *Hypothetical Scenario of Content Aware Caching in MEC*

requested content is present in the MECs cache, the MEC server immediately sends the desired content to the end-users. Otherwise, the content is fetched from the CCS and returned to the respective user. Now, intuitively, the caching mechanism with respect to the overall cache hit rate can be improved by storing the most popular video content in a given time frame. The popularity of the video content can be predicted from previous (historical) data sequences and present popularity information. In general, the proposed deep architecture has been trained in an offline manner. The learned parameters (pre-trained coefficients) are updated for the MEC server's testing model in a regular interval since a part of the input data is online in this case. The frequency of such updates depends on the average changing frequency of the video content (Movies, in our case). The changing of video frequency refers to the fact that the requested videos change as time progresses, and new videos are added to the CCS in a particular time slot of the day. A DL-based popularity prediction scheme is devised to improve the caching mechanism in this work.

The details of data preparation and model training are discussed in detail in Section 3.3. MEC stores the content in its local cache according to the list of the most popular content predicted by the proposed caching model.

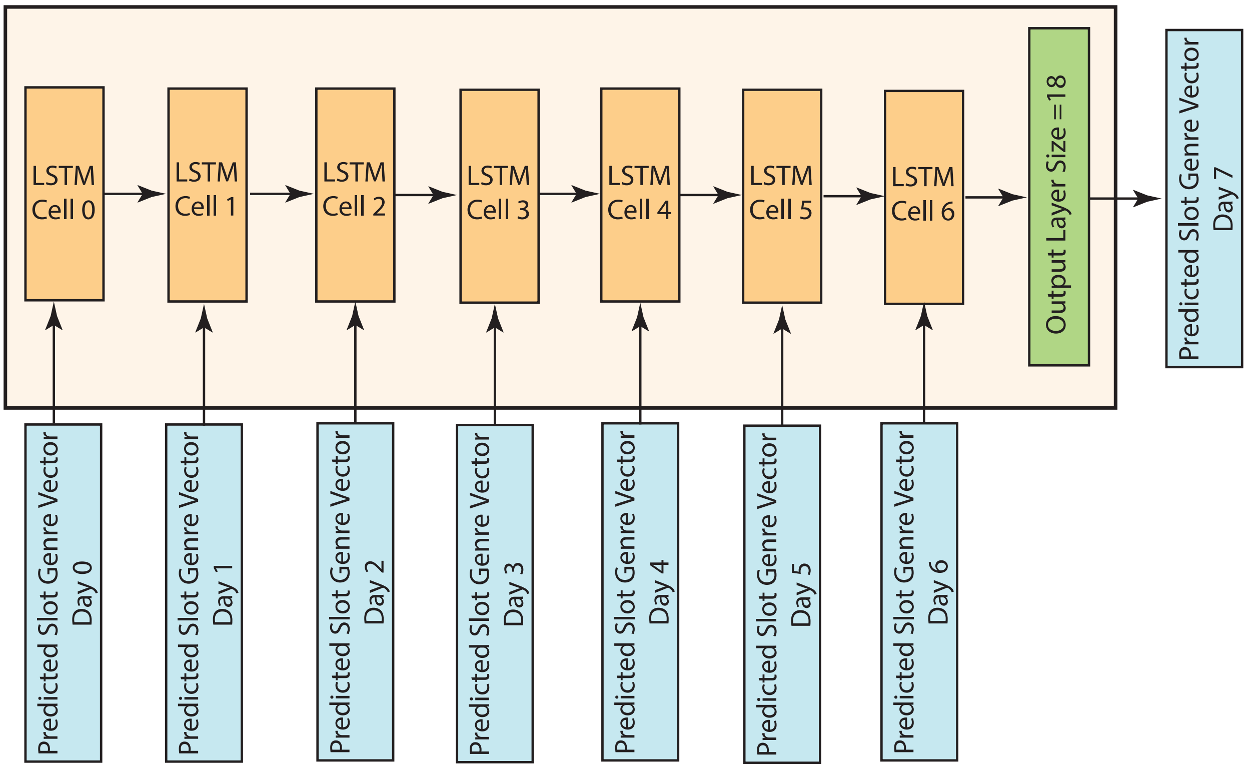

LSTM Cell 0
LSTM Cell 1
LSTM Cell 2
LSTM Cell 3
LSTM Cell 4
LSTM Cell 5
LSTM Cell 6
Output Layer Size =18
Predicted Slot Genre Vector Day 7
Predicted Slot Genre Vector Day 0
Predicted Slot Genre Vector Day 1
Predicted Slot Genre Vector Day 2
Predicted Slot Genre Vector Day 3
Predicted Slot Genre Vector Day 4
Predicted Slot Genre Vector Day 5
Predicted Slot Genre Vector Day 6

Figure 3.2: *Overview of LSTM for Genre Prediction*

3.2.1 Long Short Term Memory for Popularity Prediction

Recurrent Deep Neural Network (DNN) like LSTM [84] [85] are used to model sequence learning problems efficiently. LSTM could learn the temporal dependency from a sequential data through Back Propagation Through Time (BPTT) algorithm. In our work, LSTM considers the slot-wise genre vector of the last seven days to predict the most prevalent movie genre in a particular time slot of the day. The working principle of LSTM is illustrated in Figure 3.2, which considers the genre vector $\overrightarrow{SGV}_{ij}$ at different time slot j of the last i^{th} days. The LSTM model learns an approximate function given by the Equation (3.1) and predicts the most prevalent movie genre at a particular time of the day.

$$\overrightarrow{SGV}_{ij} = f(\overrightarrow{SGV}_{(i-1)j}, \overrightarrow{SGV}_{(i-2)j}, ..., \overrightarrow{SGV}_{(i-7)j}) \tag{3.1}$$

In this work, we model the popularity of the content (in our experiment, it is Movie). Now the popularity of the "movie" may change over time. Intuitively, popularity can be defined locally (in a present time window, for example, in the current month or week) or temporally (considering over a long time, for instance, in the last five years, etc.). Again, a movie with a particular genre may be prevalent in a specific time window of day (for example, "family drama" may be more demanding in the evening, whereas "Horror" movies may be preferred late at night, etc.). Observing such spatio-temporal characteristics of the popularity (of a movie), LSTM could be used to model the popularity, considering its efficacy in modelling the sequential data.

3.3 Proposed Model

The proposed *DCache* model uses a stream of video requests from various end-users as input data for caching the most popular video content in the MEC server. Efficient caching models could be deployed in MEC servers for providing services to different OTT streaming services like Netflix, Amazon Prime Video, YouTube, etc. For experimental purposes, we have used the *MovieLens* [41] [86] dataset in our proposed work.

- **Input Stream:** In our proposed work *DCache*, a stream of requests (for a real scenario) is pre-processed to be used as input data. In other words, the stream of requests are converted to a dataset having mainly four parameters, namely, *a) UserID:* A specific ID that is always assigned to a user who requests the movie, *b) Movie-Id:* A unique ID for every movie/video hosted by the OTT services, *c) Timestamp:* The time when a given movie is rated ,and *d) Tag:* Tags denotes ratings/number of views/comments given for a particular movie/video. The *MovieLens* dataset consists of parameters mentioned above.

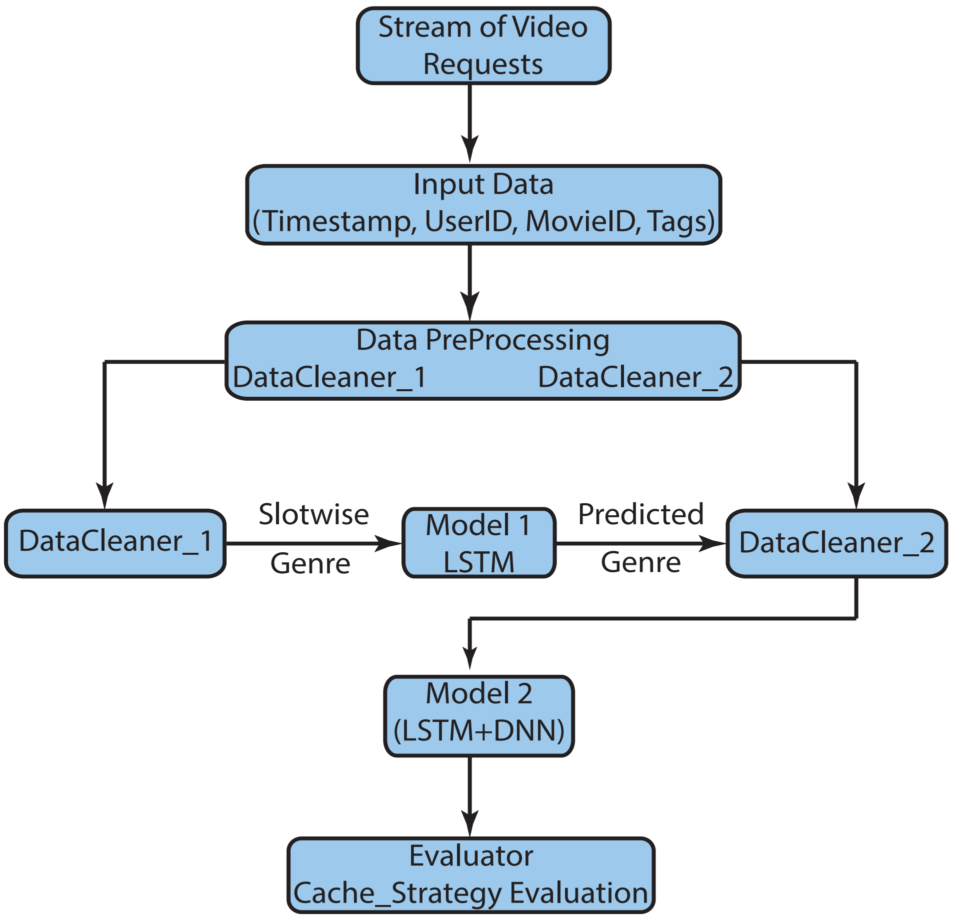

Stream of Video Requests
Input Data (Timestamp, UserID, MovieID, Tags)
Data PreProcessing DataCleaner_1 DataCleaner_2
DataCleaner_1
Slotwise Genre
Model 1 LSTM
Predicted Genre
DataCleaner_2
Model 2 (LSTM+DNN)
Evaluator Cache_Strategy Evaluation

Figure 3.3: *Overview of Proposed DCache Model for Content-Aware Caching*

- **Data Pre-processing Module:** As illustrated in Figure 3.3, the data pre-processing module consists of two sub-modules, namely:

    1. *DataCleaner_1* considers features of *MovieLens* dataset like *timestamp, MovieID, userId, ratings* and the *genres* of a movie and creates a movie genre vector $\overrightarrow{M}$.

    2. *DataCleaner_2* module combines the output of $Model\_1$ with various input features from the *MovieLens* dataset like previous views of a given movie, timestamp, MovieID, and the movie genre vector. This processed data is then fed into $Model\_2$.

- **Slot-wise Genre Prediction Module** ($Model\_1$)**:** $Model\_1$ is the first step of the proposed two-step model for predicting the movies' popularity, taking input features prepared by *DataCleaner_1*. It predicts the most prevalent genre of content at six different time slots of the day using LSTM architecture.

- **Total Request Count Module** ($Model\_2$)**:** $Model\_2$ predicts the total request count for a movie using a combined neural network comprising LSTM and DNN. It takes the slot-wise predicted genre vector and the previous views of the movie for the last seven days processed by the *DataCleaner_2* module as input.
- **Evaluator Module:** The caching decision strategy collects the final list $\mathcal{L}$ of predicted views from the $Model\_2$. The movies are then cached in the MEC server in decreasing order of their popularity until the cache is full.

The efficacy of the proposed model is measured using the cache hit rate, backhaul usage, and the access time taken to retrieve a given movie.

3.3.1 Problem Formulation

In our proposed work, content refers to a movie from the *MovieLens* dataset. Let us assume that the number of movie requests arriving at a designated BS is given by $\mathcal{R} = \{r_1, r_2, ...r_k\}$ where r_k is the number of arriving requests. $\mathcal{V} = \{c_1, c_2, ...c_n\}$, be the sets of movies which is being requested. Whenever a request r_{c_i} for a movie c_i arrives at the BS at time t, the MEC server within the BS first checks whether the incoming request r_{c_i} from a user is present in its local cache or not. If the movie c_i is present in the MEC server's local cache, it is sent to the user without delay. Otherwise, the requested movie is fetched from the CCS with a delay of γ. k_{c_i} for $c_i \in \mathcal{V}$ is the size of each movie c_i and the capacity of each BS is denoted by K. Therefore, the size of all the movies k_{c_i} to be cached in the MEC server cannot be more than the total cache size K, which is denoted by the Equation (3.2):

$$\sum_{i=1}^{n} k_{c_i} \leq K \tag{3.2}$$

A cache decision variable $y_{c_i}^t$ decides whether or not to cache a given movie of size k_{c_i} at time t at the MEC server of size K as mentioned in Equation (3.3) for a set of movies $\mathcal{V}$.

$$\sum_{i=1}^{n} y_{c_i}^{t} k_{c_i} \leq K \tag{3.3}$$

$y_{c_i}^{t} \in \{0, 1\}$, where $y_{c_i}^{t}$ is 0 when the movie is not cached and 1 when cached. Access delay for retrieving the requested movie at time $(t + 1)$ is given by Equation (3.4) as mentioned in [83].

$$\delta_{t+1} = \sum_{i=1}^{n} \Big((1 - y_{c_i}^{t}) \gamma * k_{c_i} \Big) \hat{\phi}_{c_i}^{t+1} \tag{3.4}$$

where γ is the latency for retrieving the movie of size k_{c_i} from CCS and $\hat{\phi}_{c_i}^{t+1}$ is predicted number of request for the movie c_i in future time $(t+1)$. The access time could be efficiently minimized by maximizing the probability of cache hit at the MEC servers. In our proposed work, we formulate the optimization problem as maximizing the cache hit rate for the movies at the MEC server defined by Equation (3.5):

$$\begin{aligned} &Maximize \quad \mathcal{P}_{hit} \\ &\mathcal{P}_{hit} = \frac{\sum_{t \in T} \sum_{i=1}^{n} y_{c_i}^{t} r_{c_i}^{t}}{\sum_{t \in T} \sum_{i=1}^{n} r_{c_i}^{t}} \\ subject \quad to \quad &\sum_{i=1}^{n} y_{c_i}^{t} k_{c_i} \leq K \end{aligned} \tag{3.5}$$

To find an optimal solution to our formulated optimization problem, we need to efficiently predict the expected request count $\hat{\phi}_{c_i}^{t+1}$, for the movie set $\mathcal{V}$. Due to the large solution space, it is not feasible to exhaustively search for an optimal solution. Therefore in this work, we employed a combination of two DL models LSTM and DNN, to search for the solution space in order to accurately determine the values of $\hat{\phi}_{c_i}^{t+1}$. In the following subsection, we discuss *DCache*, our proposed model for predicting expected request counts and caching strategy.

3.3.2 Architecture of Proposed DCache Model

3.3.2.1 Data Pre_processing

In the first module of our proposed model, we process the data from the *MovieLens* dataset and divide it into slots and genres, which is then fed into the slot-wise genre prediction model. Content popularity is defined as the ratio of the number of requests received for a movie to the total number of requests obtained for a given period. In this work, we have assumed that a viewer always rates the movie when he/she has watched the movie. With this assumption, the content popularity is calculated by the number of times a movie is rated, divided by the total number of ratings for all the movies. In other words, to calculate the content popularity for a given time slot, we measure the number of requests received for a movie c_i in a slot s_0 to the total number of requests N_{s_0} received for the given slot. Let us consider z_{c_i} be the total number of views for a movie c_i in a slot s_0. Then the content popularity F_{c_i} is given by Equation (3.6):

$$F_{c_i} = \frac{z_{c_i}}{\sum_{i=1}^{N_{s_0}} z_{c_i}} \tag{3.6}$$

For the data preparation, we first group every movie into a genre represented by a movie genre vector $\overrightarrow{M}$. A total of 18 classes of the genre are present, some of which are represented by $\overrightarrow{M}$ as shown in (3.7)

$$\overrightarrow{M} = <' Action',' Adventure',..'War',' Western' > \tag{3.7}$$

A movie can be marked with multiple genres. To represent the movie with the movie genre vector $\overrightarrow{M}$, 1 indicates that the movie belongs to a particular genre; else, it is 0. The profile of various users viewing different movies and shows changes at different times of the day. We consider the time "t" in days in a slot-wise manner. We divided the entire day into different slots, as one would not expect a particular movie to be viewed throughout the day. Therefore, we consider the changes in the users viewing patterns. We divided each day into six slots of four hours each i.e.

$$\mathcal{S} =< s_0, s_1, s_2, s_3, s_4, s_5 > \tag{3.8}$$

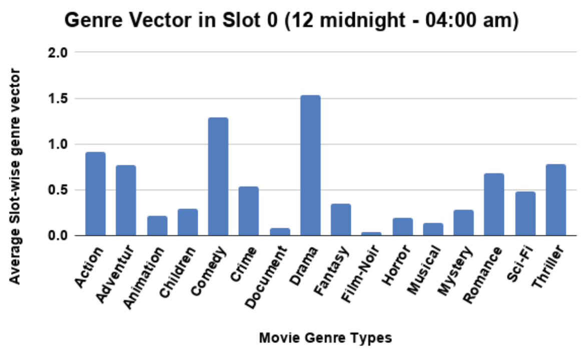
Every slot in $\mathcal{S}$ consists of movie genre vectors of all those movies viewed in the given slot. To calculate which genre is prevalent in a given slot we measure the weighted average slot-wise genre vector $\overrightarrow{G}^{s_0}_{avg}$ over all movie request z_c of a slot s_0 represented by (3.9)

$$\overrightarrow{G}^{s_0}_{avg} = \frac{\sum\limits_{i=1}^{N_{s_0}} \overrightarrow{M}^{s_0}_{c_i} z_{c_i}}{\sum\limits_{i=1}^{N_{s_0}} z_{c_i}} \tag{3.9}$$

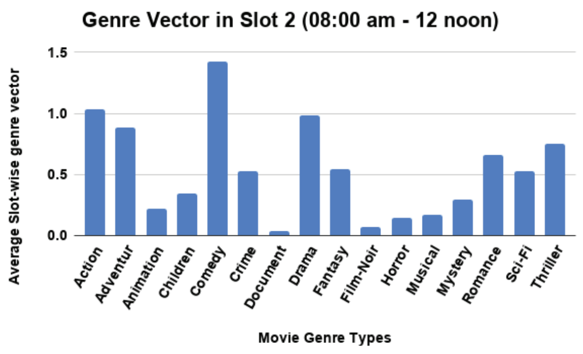
where, $c \in \{1, 2, ...n_{s_0}\}$ represents the total number of movies viewed in slot s_0 and z_{c_i} is the number of views recorded for movie c_i. $\overrightarrow{G}^{s_0}_{avg}$ consists of averaged weighted views of all the movies in the slot s_0 and the genre having the highest value is the most prevalent genre in that slot for a given day. Figure 3.4 presents an analysis of the slot-wise genre vector at different time slots of the day. The analysis represents the genre of a movie that is most viewed in a particular slot of the day. For example, Figure 3.4(a) shows that in the first time slot (from 12 midnight to 4 am), the most watched category of movie is *Drama*. Similarly, in other time slots of the day Figure 3.4(b) (from 8 am to 12 noon) and Figure 3.4(c) (from 4 pm to 8 pm), the most watched category of movie is *Comedy* and *Action* respectively. Therefore, it is evident from the analysis that user profile changes in different time slots of the day.

3.3.2.2 Slot-wise Genre prediction model using LSTM

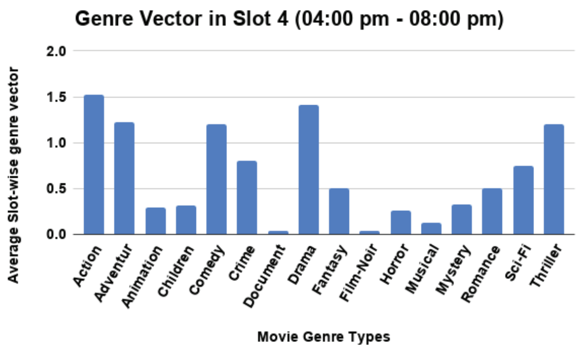
As illustrated by Figure 3.2, LSTM can efficiently handle the long-term dependencies in time series problems. Therefore, in our proposed work LSTM is used in $Model\_1$ to predict the slot-wise Genre vector $\hat{O}^{s_d}_{t+1}$ of future slots. We consider the window size to be seven, i.e., genre vector of the last seven days.



(a) *Genre Vector from 12 midnight to 4 am*



(b) *Genre Vector from 8 am to 12 noon*



(c) *Genre Vector from 4 pm to 8 pm*

Figure 3.4: *Statistical Analysis of Slot-wise Breakdown of Genre Vector at Different Time Slots of a Day*

$$D =< D_0, D_1, D_2, D_3, D_4, D_5, D_6 > \tag{3.10}$$

Figure 3.5 depicts the working principle of $Model\_1$. The input feature X_t of $Model\_1$ denoted by Equation (3.11) consists of the average weighted genre vector $\overrightarrow{G}^{s_d}_{avg}$ of the six slots for the last seven days.

$$\begin{aligned} Model\_1(X_t) &= D_t \overrightarrow{G}^{s_d}_{avg} \\ for \quad & d \in \{0, 1, 2, 3, 4, 5\} \end{aligned} \tag{3.11}$$

The input feature X_t is then fed to a LSTM model to predict the slot-wise genre vectors of future slots $\hat{O}^{s_d}_{t+1}$. For instance, for predicting the genre vectors of day D7's slot s_4 whose timing is from 4 PM to 8 PM, the LSTM model uses slot s_4's genre vectors for the last

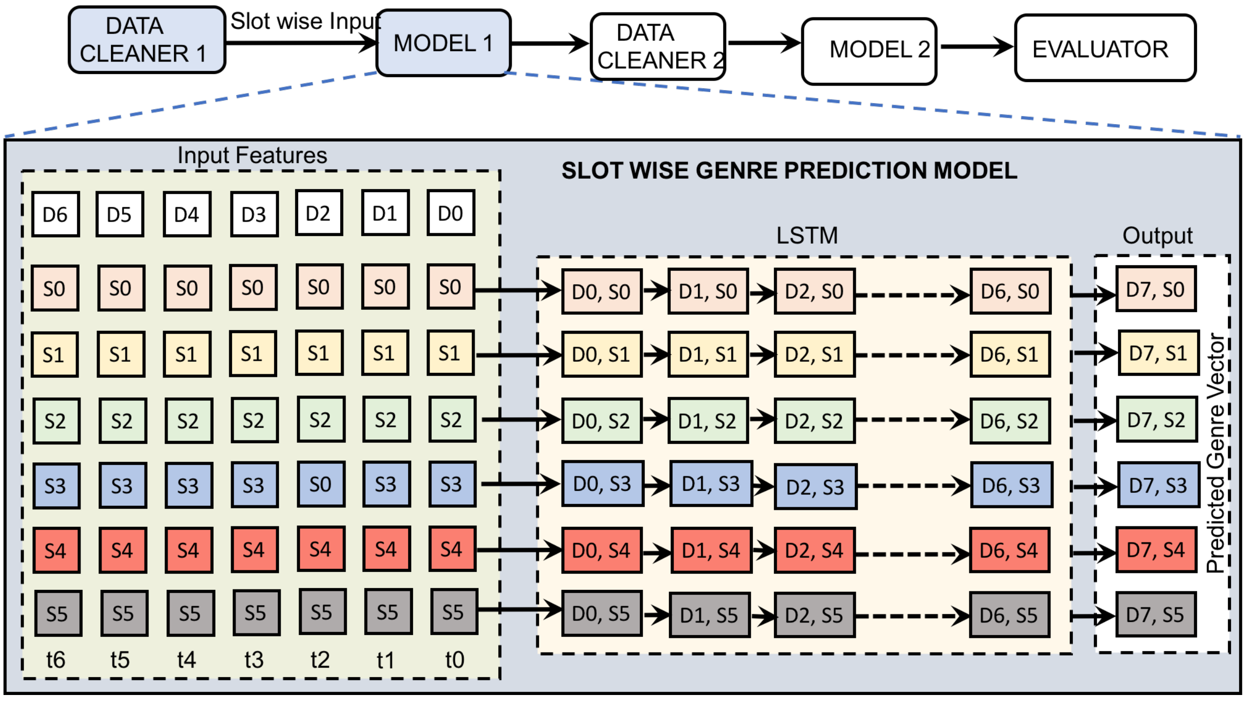

DATA CLEANER 1
Slot wise Input
MODEL 1
DATA CLEANER 2
MODEL 2
EVALUATOR
SLOT WISE GENRE PREDICTION MODEL
Input Features
D6 D5 D4 D3 D2 D1 D0
S0 S1 S2 S3 S4 S5
t6 t5 t4 t3 t2 t1 t0
LSTM
D0, S0 D1, S0 D2, S0 D6, S0
D0, S1 D1, S1 D2, S1 D6, S1
D0, S2 D1, S2 D2, S2 D6, S2
D0, S3 D1, S3 D2, S3 D6, S3
D0, S4 D1, S4 D2, S4 D6, S4
D0, S5 D1, S5 D2, S5 D6, S5
Output
D7, S0 D7, S1 D7, S2 D7, S3 D7, S4 D7, S5
Predicted Genre Vector

Figure 3.5: *Proposed Architecture for Slot Wise Genre Prediction Model*

seven days from $D0$ to $D6$. Loss function used in $Model\_1$ is Mean Squared Error (MSE) which is the most common loss function used in regression problems. MSE is the squared difference between the actual value p and predicted value $\hat{p}$ denoted by Equation (3.12):

$$H(p, \hat{p}) = \frac{1}{N}\sum_{i=1}^{N}(p_i - \hat{p}_i)^2 \tag{3.12}$$

The model takes as input the average weighed genre vector of the six different slots of a day for the last seven days and predicts the seventh day's genre vector. The predicted genre vector indicates the most prevalent genre in the given time slot of a day. For the slot-wise genre prediction model using LSTM a window of size 7 is kept, with 18 input features which are the genre vector of all the movies in the *MovieLens* dataset. The output $\hat{O}^{s_d}_{t+1}$ along with input features from the *MovieLens* dataset is fed to the next model *Total Request count* prediction model ($Model\_2$) which is discussed in the following subsections.

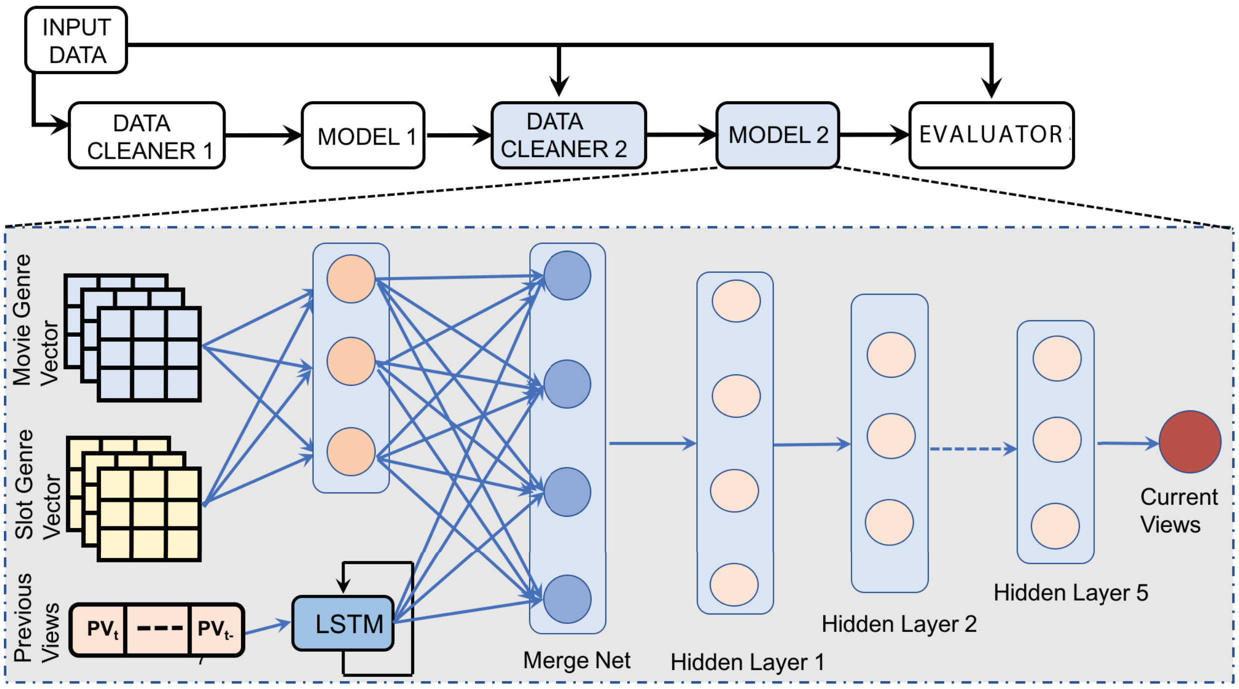


Figure 3.6: *Proposed Architecture for Total Request Count Prediction Model*

3.3.2.3 Total Request Count prediction model using LSTM+DNN

In this module, we train $Model\_2$ (m_2^*) a combination of LSTM and DNN for predicting the number of requests or view counts $\hat{\phi}_{c_i}^{t+1}$ which will be received for each movie in a given slot. To train model m_2^*, two different types of input are used: sequential and instantaneous. For the sequential input, we consider previous views $\overrightarrow{\mathcal{PV}}_{idj}$ input feature for i^{th} video in dj^{th} slot, where d is the number of slots and j is the number of days. The previous views vector $\overrightarrow{\mathcal{PV}}$ for 0^{th} slot and i^{th} movie can be represented as (3.13):

$$\overrightarrow{\mathcal{PV}}_{i0j} =< \mathcal{PV}_{i0(j-1)}, \mathcal{PV}_{i0(j-2)}, ...\mathcal{PV}_{i0(j-7)} > \tag{3.13}$$

We consider a window size of the last seven days for the previous views input features. As illustrated in Figure 3.6, the $\overrightarrow{\mathcal{PV}}$ being sequential is fed into the LSTM network. The instantaneous inputs like the movie genre vector $\overrightarrow{M}$ and predicted slot-wise genre vectors $\hat{O}_{t+1}^{s_d}$ are fed into a DNN. The trained model m_2^* takes the input as shown in (3.14):

$$Model\_2(X_t) =< \overrightarrow{\mathcal{PV}}_{idj}, \overrightarrow{M}, \hat{O}^{s_d}_{t+1} > \tag{3.14}$$

The outputs obtained from these layers are combined using a merge net. Five more hidden layers are used between the merge net and the output layer. A different loss function *Huber Loss* and *Adam* optimizer is used for training model m_2^*. *Huber Loss* is mostly used in regression problems, an absolute error that becomes quadratic when the error is small. *Huber Loss* combines the good properties of the MSE and Mean Absolute Error (MAE). Huber loss approaches MSE when $e \sim 0$ and MAE when $e \sim \infty$ (for larger numbers.)

$$H(p, \hat{p}) = \begin{cases} \frac{1}{2}(p-\hat{p})^2, & \text{if } |p-\hat{p}| \leq e. \\ \delta|p-\hat{p}| - \frac{1}{2}e^2, & \text{otherwise.} \end{cases} \tag{3.15}$$

$Model\_2(m_2^*)$ takes as input two types of input features, namely 1) sequential input features, consisting of the last seven days of views of a given movie and 2) instantaneous data consisting of slot-wise genre vector predicted from the previous model and the movie genre vector. Therefore, a total of 43 input features are fed into the second model. The *Evaluator* module uses the predicted views list to measure the proposed model's efficacy by calculating the cache hit rate, access delay, and backhaul usage. Finally, model m_2^* predicts the views of movies in the future stored in a list $\mathcal{L}$. The entire process for predicting movies' view count is described in Algorithm 1.

3.3.2.4 Caching Decision and the Evaluator Module

The overall cache decision strategy is described in Algorithm 2. Model m_2^* predicts the final list of predicted views. The list consists of slot-wise predicted views of movies which is listed as:

$$\mathcal{L} = \{l_{s_0}, l_{s_1}, l_{s_2}, l_{s_3}, l_{s_4}, l_{s_5}\}$$

Algorithm 1: *DCache* for Content Popularity Prediction

```
Input  : Slot-wise genre vector of movies $\overrightarrow{G}^{s}_{avg}$, Movie Vector $\overrightarrow{M}$, Views in the last
         Seven days $\overrightarrow{\mathcal{PV}}_{idj}$
Output: Predicted request counts $\hat{\phi}^{t+1}_{c_i}$
1  Request Arrival $\mathcal{R} = \{r^{ci}_1, r^{ci}_2, .......r^{ci}_q\}$ ;
2  ▷ Train $Model\_1$ ($m^*_1$) (LSTM) predicting the slot-wise Genre Vector;
3  $m^*_1$.train();
4  for $slots \quad s \leftarrow 0$ to 5 do
5      for $data\_m_1 \leftarrow \overrightarrow{G}^{s}_{avg}$ do
6          optimizer ← Adam;
7          $\hat{p} = m^*_1(data\_m_1)$;
8          loss = MSE($\hat{p}, p$);
9          loss.backward()                    ▷ Back Propagation ;
10         optimizer.step()                  ▷ Updates the parameters;
11     end
12 end
13 ▷ Train $Model\_2$ ($m^*_2$) (LSTM+DNN) predicting views of movies;
14 $m^*_2$.train() ;
15 for $data\_m_2 \leftarrow \hat{O}^{s}_{t+1}, \overrightarrow{M}, \overrightarrow{\mathcal{PV}}_{idj}$ do
16     optimizer ← Adam;
17     $\hat{p} = m^*_2(data\_m_2)$;
18     loss = SmoothL1($\hat{p}, p$);
19     loss.backward()                        ▷ Back Propagation ;
20     optimizer.step()                      ▷ Updates the parameters;
21 end
22 Finally store the predicted request counts $\hat{\phi}^{t+1}_{c_i}$ in a list $\mathcal{L}$
```

The movies in the slot-wise list l_{s_j} are then sorted in descending order of content popularity so that 20% of movies with the highest popularity index is considered. According to *Pareto-principle*, 20% of the most popular movie receives 80% of the requests. Equation 3.16 indicates that from the predicted list $\mathcal{L}$ of every slot, we consider only the top 20%.

$$\mathcal{C}^{s_j}_{c} = 0.2 * l_{s_j} \quad for \quad j \in \{0, 1, 2, 3, 4, 5\} \tag{3.16}$$

Along with the *Pareto-principle*, the caching strategy also ensures that the movies cached in the MEC server do not exceed the storage size K. The caching strategy defined in Algorithm 2 stores the sorted predicted list of movies in the MEC server in a slot-wise manner. The predicted list of movies $\mathcal{C}^{s_0}_{c}$ for slot 0 (s_0) represents the predicted movies in

Algorithm 2: *DCache* for Caching Strategy

```
▷ Caching Strategy ;
$\mathcal{L} = \{l_{s_0}, l_{s_1}, l_{s_2}, l_{s_3}, l_{s_4}, l_{s_5}\}$ slot-wise predicted list of movies;
for $j = 0, 1, 2, 3, 4, 5$ do
    $\mathcal{C}_{c_i}^{s_j} = 0.2 * l_{s_j}$        ▷ $\mathcal{C}_{c_i}^{s_j}$ consists of predicted list in sorted order;
    while $\mathcal{C}_{c_i}^{s_j}$ do
        if $\sum_{i=1}^{n} k_{c_i}^{l_{s_j}} \leq K$ then
            $Cache \longleftarrow Cache + \mathcal{C}_{c_i}^{s_j}$;
        end
    end
    while $r_{c_i}^{s_j}$ do
        if ($c_i$ present in $\mathcal{C}_{c_i}^{s_j}$) then
            MEC Cache $\longrightarrow c_i$;
            Calculate $\mathcal{P}_{hit}$,   $P_{backhaul}$,   $Delay$
        end
        else
            Main Content Server $\longrightarrow c_i$;
            Calculate $\mathcal{P}_{hit}$,   $P_{backhaul}$,   $Delay$
        end
    end
    $Cache \longleftarrow Cache$ - $\mathcal{C}_{c_i}^{s_j}$        ▷ Delete Cache of slot $s_j$ ;
end
```

the first slot of the day (12 midnight - 4 am). Initially, the movies listed by $\mathcal{C}_c^{s_0}$ are cached in the MEC server one by one until the cache space is full. Next, we consider the predicted list of movies $\mathcal{C}_c^{s_1}$ from the next slot 1 (s_1) for caching. All the movies from the previous slot 0 (s_0) are deleted before caching the new movies. Caching is performed similarly for the subsequent slots by deleting the previous slot's movies and caching the present slot's movies until the cache is full. The strategy ensures that the movies viewed frequently in a given time slot of the day are stored in the cache, and the least popular videos are not considered for caching. If a user requests $r_{c_i}^{s_j}$ for a movie c_i is present in the predicted list $\mathcal{C}_{c_i}^{s_j}$, the movie is directly fetched from the MEC cache. Otherwise, the requested movie c_i, if not present in $\mathcal{C}_{c_i}^{s_j}$, is fetched from the main CCS.

3.4 Experiments and Results

The efficiency of the proposed DL model *DCache*, has been evaluated by performing an extensive set of experiments based on the standard *MovieLens* dataset [41] [42] and compared with caching strategies like *LRU* [87], *LFU* [88], *FIFO* [89], *DeepMEC* [83], *DeepCache* [61] and *RLCache* [32]. To evaluate the performance of the proposed model for content popularity, we used the *MovieLens* dataset for training, testing, and validating. In our proposed work, we used the *movies* and the *ratings* file from the *MovieLens* dataset in order to prepare the data in slot-wise genre format. The *movies* file contains the data in the following format:

$$< movieID, title, genres >$$

whereas data in the ratings file has the following format:

$$< userId, movieId, tag, timestamp >$$

Data from both these files are used to generate the slot-wise genre vector $\overrightarrow{G}_c^s$ for $c \in \{1, 2, 3, ..n\}$ and $s \in \{0, 1, 2, 3, 4, 5\}$, containing a total of 18 input features being fed to $Model\_1$. Data before 2017 is used for training, and post 2017 is used for testing. We need the movie popularity information that is not present in the *MovieLens* dataset for our proposed work. With the lack of a publicly available dataset, we made certain assumptions while calculating the popularity of a given movie. We assumed that a viewer always rates the movie when he/she has requested the movie. Similar assumptions are made by the authors in [36], [90], and [91] which stated that, "*Whenever a user gives rating for a movie, we set the movie as requested by the users*". With this assumption, the movie's popularity is calculated by the number of times a movie is rated, divided by the total number of ratings for all the movies.

3.4.1 Implementation and Experimental Setup

The proposed *DCache* architecture for content-aware caching is implemented using PyTorch at the DGX workstation, which consists of a Tesla V100 GPU card with 5120 Cuda Cores and a GPU memory of 32 GB. We also tested our models in a CPU server with 128 GB CPU memory. For a fair comparison, a similar experimental setup has been used. The *RLCache* [32] and *DeepCache* [61] model is simulated and tested in MEC environment identical to the proposed *DCache* model for various hyper-parameters settings.

3.4.1.1 Evaluation Metrics

To evaluate the efficacy and correctness of the proposed *DCache* model, the following metrics were used.

- Cache hit rate $\mathcal{P}_{hit}$ denoted by Equation (3.17) measures the probability that the requested movie is present in the local cache of the MEC server. If the requested movie c_i^t is present in the cache it is a hit. Else it is considered to be a miss.

$$\mathcal{P}_{hit} = \frac{\sum\limits_{t \in T} \sum\limits_{i=1}^{n} y_{c_i}^t r_{c_i}^t}{\sum\limits_{t \in T} \sum\limits_{i=1}^{n} r_{c_i}^t} \tag{3.17}$$

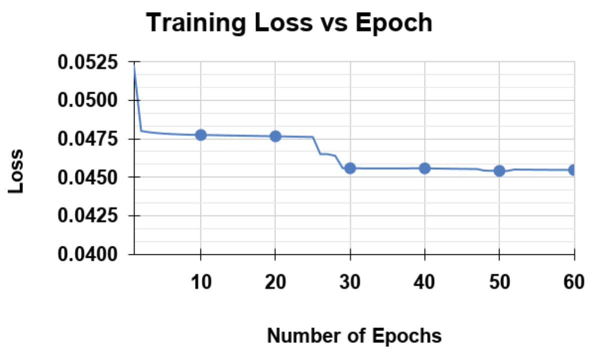
- Backhaul usage calculates the network traffic in the backhaul links between the CCS and the MEC server. Equation (3.18) measures congestion or the usage in the backhaul links.

$$P_{backhaul} = \sum_{t \in T} \sum_{i=1}^{n} k_{c_i}(1 - y_{c_i}^t) r_{c_i}^t \tag{3.18}$$

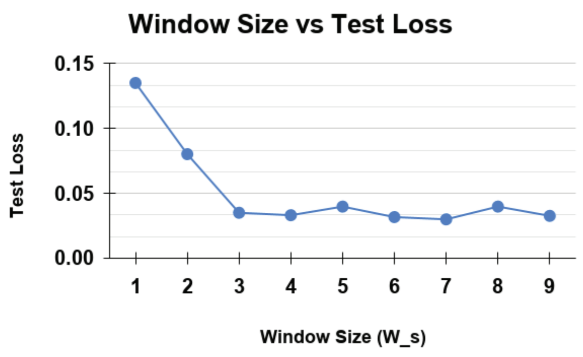
  where k_{c_i} is the size of the movie c_i, $r_{c_i}^t$ is the number of request for the movie c_i at time t.

- Access delay measures the latency or the time taken in retrieving a requested movie from the MEC server. Equation (3.4) is used to measure the overall access delay.

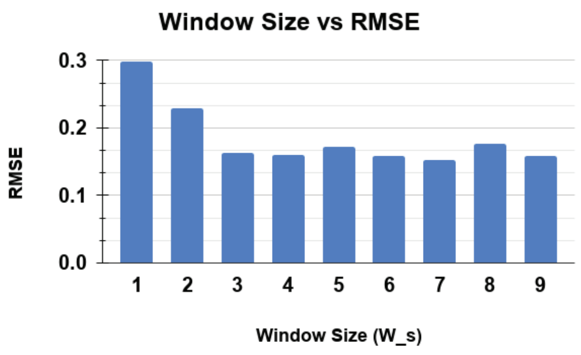
3.4.1.2 Evaluation of Slot-wise Genre Prediction Model



(a) *Train Loss Vs Epoch*



(b) *Window Size vs Test Loss*



(c) *Window Size vs RMSE*

Figure 3.7: *Performance of Slot-wise Genre Prediction Model* m_1^*

In this subsection, we discuss the performance of $Model\_1(m_1^*)$ for predicting the most prevalent genre in a particular time slot of the day. To measure the performance of m_1^* using LSTM, evaluation metrics such as Root Mean Square Error (RMSE) is used. RMSE is a measure to calculate the difference between the observed and predicted values. Along with RMSE, we also calculate the train and test loss during the training of Model m_1^* with varying window sizes. Figure 3.7 illustrates the performance of the slot-wise genre prediction model (m_1^*). Figure 3.7(a) presents the training loss obtained, which reduces at every epoch. We vary the window size W_s to determine how further we need to look back in the past few days to predict the most prevalent genre. The window size W_s is altered from 1 to 9. Figure 3.7(b) and 3.7(c) represents the test loss and RMSE respectively against the varying window size. Window size $W_s = 7$ obtained the best minimum RMSE value of 0.1523. Therefore, we employed $W_s = 7$ in our proposed model *DCache* for m_1^*. We also compare m_1^* with

Table 3.1: *Comparison of RMSE for Model 1*

Model	Window size ($W_s = 3$)		Window size ($W_s = 7$)		Window size ($W_s = 9$)	
	Test Loss	RMSE	Test Loss	RMSE	Test Loss	RMSE
RNN (m_1^*)	0.1121	0.28595	0.03151	0.1599	0.0310	0.15688
LSTM (m_1^*)	0.0350	0.16339	0.03000	0.1523	0.03169	0.15710

another sequential model using Recurrent Neural Network (RNN). Table 3.1 shows that m_1^* using LSTM provided a better RMSE value than the model using RNN with a minimum RMSE score of 0.1523 for a window size of $W_s = 7$.

3.4.1.3 Evaluation of Request Count Prediction Model

The Request Count Prediction Model $Model\_2$ (m_2^*) predicts the list of views for all the movies in various time slots of the day. The proposed model (LSTM+DNN) was compared with DL models such as DNN, 1D Convolutional Neural Network (1DCNN), and Residual Neural Network (RESNET). The efficacy of the proposed model was compared against the cache hit rate. Table 3.2 presents the cache hit rate for the DL models. The proposed model with LSTM and DNN outperforms the other deep models substantially and by a margin of almost 23%. The proposed model considers sequential and instantaneous data separately, and with the use of LSTM, it can learn and memorize the features in a much enhanced manner. The simple DNN model and the RESNET model with five layers could not capture the sequential data even though various layers were added to the model. Even after convolving through the features, 1DCNN could not efficiently learn the data patterns. Therefore, the proposed LSTM model captures the historical relationship among the various features of the data instead of the only last input data, and with its various gates, surpasses the other models in terms of the cache hit rate.

3.4.1.4 Cache Hit Rate

The evaluation of the proposed *DCache* model is performed on the final predicted list $\mathcal{L} = \{l_{s0}, l_{s1}, l_{s2}, l_{s3}, l_{s4}, l_{s5}\}$ of the predicted request count for the movies. The predicted list $\mathcal{L}$ consists of the request in descending order of content popularity in a slot-wise manner.

Table 3.2: *Cache Hit Rate Against MEC Cache Size (GB)*

Models	Cache Hit Against MEC Cache Size (GB)					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
DNN	15.9	18.78	21.59	24.33	26.98	29.6
1DCNN	16.16	19.09	21.9	24.65	27.29	29.85
RESNET	15.94	18.85	21.71	24.47	27.13	29.76
Proposed (LSTM+DNN)	20.38	23.52	26.72	30.07	33.41	36.82

Table 3.3 presents a comparative study of the cache hit rate of the proposed *DCache* model against the caching strategies namely, *LRU*, *LFU*, *FIFO*, *DeepMEC* [83], *DeepCache* [61] and *RLCache* [32]. *DCache* outperforms *LRU*, *LFU* and *FIFO* in case of cache hit rate by almost 21% for 700 GB cache size. *DCache* cache hit rate increases by almost 16% when compared with *DeepMEC* [83] and by 12% when compared with *RLCache* [32]. The cache hit rate further increases significantly when the cache size of the MEC is increased. In our experimental setup we consider a maximum of 1 TB cache size, even though MECs have higher storage and computational capacities as mentioned earlier. The proposed *DCache* also provides a better cache hit rate for a lower cache capacity like 500 GB, which was not true for other DL models as evident from Table 3.2.

Table 3.3: *Cache Hit Rate Against MEC Cache Size (GB). The best results are shown in Red and the second best in Blue*

Models	Cache Hit Against MEC Cache Size (GB)					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
LRU	19.553	20.96	21.87	22.46	22.86	23.12
LFU	20.003	21.26	22.09	22.62	22.93	23.15
FIFO	18.82	20.34	21.39	22.11	22.6	22.93
DeepMEC [83]	15.23	18.56	20.89	24.01	28.11	30.4
DeepCache [61]	16.16	17.15	17.8	18.2	18.48	19.1
RLCache [32]	17.01	20.37	23.65	26.74	29.69	32.44
Proposed *DCache*	20.38	23.52	26.72	30.07	33.41	36.82

3.4.1.5 Backhaul Usage

The cache hit rate at the MEC server directly affects backhaul usage or the network traffic at the backhaul links. Whenever a given movie's request is not present in the MEC cache, the

Table 3.4: *Computation of Slot-wise Backhaul Usage (TB)*

Slots	Slot 0	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
Backhaul Usage (TB)	301.64	235.68	170.6	160.19	182.422	240.94

Table 3.5: *Backhaul Usage (TB) Against MEC Cache Size (GB). The best results are shown in Red and the second best in Blue*

Models	Backhaul Usage (TB) Against MEC Cache Size					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
LRU	1691.5	1661.08	1642.59	1630.19	1621.91	1616.45
LFU	1682.03	1655.58	1638.14	1627.23	1620.32	1615.86
FIFO	1706.76	1674.76	1652.83	1637.72	1627.39	1620.69
DeepMEC [83]	1764.90	1723.99	1661.09	1576.87	1512.03	1448.85
DeepCache [61]	1710.35	1759.76	1747.24	1738.42	1732.46	1720.8
RLCache [32]	1690.2	1660.15	1610.22	1465	1416	1361.01
Proposed *DCache*	1673.96	1607.94	1540.82	1471.68	1399.98	1328.39

request is forwarded to the CCS to fetch the requested movie, increasing backhaul traffic. Whereas the requested movie, if found in the MEC cache, is directly sent to the client. Such actions by the MEC server reduces any additional traffic in the backhaul links. Therefore, the better the cache hit rate, the lesser the network congestion at the backhaul links. Table 3.4 represents the backhaul traffic generated for every slot in a day. The backhaul traffic generated in *Slot 0* from midnight 12 am to 4 am in the morning is maximum while it is relatively lower in *Slot 2* (8 am to 12 noon) and *Slot 3* (12 noon to 4 pm). Table 3.5 illustrates that the proposed *DCache* model has much lower and improved backhaul usage compared to *LRU*, *LFU* and *FIFO*. The backhaul traffic reduces by almost 18% compared with heuristic-based caching strategies that further reduce with the increase of cache size. With *DCache* the backhaul usage also reduces by 12% when compared with *DeepMEC*. Proposed *DCache* also outperforms *DeepCache* and *RLCache* with respect to backhaul usage. For measuring every movie's size, we considered that many online streaming apps like Amazon Prime Video play movies at a data consumption rate of 0.46GB/hour for the best quality video. So, for a 2-hour movie, it would consume almost 1 GB. Hence, in our work, we consider the movie size to be 1 GB.

Table 3.6: *Access Delay (sec) Against MEC Cache Size (GB). The best results are shown in Red and the second best in Blue*

Models	Access Delay (sec) Against MEC Cache Size					
	500 GB	600 GB	700 GB	800 GB	900 GB	1 TB
LRU	17.32	17.016	16.82	16.69	16.60	16.55
LFU	17.22	16.95	16.77	16.66	16.59	16.54
FIFO	17.47	17.14	16.92	16.77	16.66	16.59
DeepMEC [83]	18.29	17.65	17.00	16.14	15.48	14.83
DeepCache [61]	18.04	17.83	17.7	17.62	17.55	17.41
RLCache [32]	17.86	17.04	16.43	15.78	15.13	14.54
Proposed *DCache*	17.14	16.46	15.77	15.07	14.33	13.6

3.4.1.6 Access Delay

Whenever the cache hit rate increases, the average delay for retrieving a given movie also decreases. As a result, the system performs better with reduced access delay and improves the Quality of Service (QoS) provided to the end-users. Nowadays, most streaming solutions use HTTP-based adaptive streaming mechanisms with a slightly higher latency range between 15-45 seconds. In our experimental setup, we considered a delay of 0.01 ms in retrieving a movie. Table 3.6 presents the access delay in seconds while satisfying various users requesting in retrieving the contents. The proposed *DCache* mechanism outperforms the caching strategies *LRU*, *LFU* and *FIFO* and provides a much lower access delay by almost 18%. The access delay for *DCache* in retrieving a given content is better than *DeepMEC*, *DeepCache* and *RLCache*. The access delay further reduces when an increase of MEC cache size.

3.4.1.7 Training Cost

Table 3.7 lists out a comparative study for the training cost (in minutes) between the various DL models. Although the simple DNN model with four hidden layers takes less time than the proposed *DCache* model, *DCache* outperforms the performance of DNN by almost 20%. Table 3.7 also illustrates that the training cost of the proposed *DCache* is far less than the other Deep models 1DCNN and RESNET.

Table 3.7: *Computation of Training Cost (Minutes) for Various Deep Learning Models*

Schemes	80 K Iteration		160 K Iteration		240 K Iteration		320 K Iteration	
	Time (min)	Train Loss	Time (min)	Train Loss	Time (min)	Train Loss	Time (min)	Train Loss
DNN	17.4	0.86	40.03	0.841	53.75	0.834	74.45	0.793
1DCNN	46.24	0.56	92.66	0.549	137.42	0.544	182.8	0.515
RESNET	53.16	0.40	107.01	0.35	160.48	0.331	217.12	0.29
Proposed *DCache*	21.84	0.061	47.82	0.031	74.02	0.021	100.84	0.016

3.5 Summary

The first contributory chapter proposes a DL-based content-aware caching model *DCache*. Network operators can perform video caching efficiently by integrating MEC into the cellular architecture. The cache present in the MEC server immediately replies with the requested movie to the end-users if present in the cache. Else it fetches the movie from the CCS. In doing so, the overall delay and network traffic at the backhaul link decrease with increased cache hit rate. In this work, to achieve this goal, we proposed *DCache* for performing caching at the MEC server based on the content popularity where each day is divided into slots of prevalent genres of movies. We consider various input features obtained from the standard *MovieLens* dataset to predict the expected request counts in the future. The movies' expected request counts are used for caching the movies, with the most requested movie stored first into the MEC cache server in a slot-wise manner. *DCache* outperforms standard caching mechanisms like *LRU*, *LFU*, *FIFO*, *DeepMEC*, *DeepCache* and *RLCache* with respect to cache hit rate, backhaul usage and access delay with a constraint on the cache size.

It has been observed while designing caching strategy, users' viewing experience or QoE is not taken into consideration. This results in the degradation of users' QoE as the delay incurred is too significant for users with low network bandwidth. Therefore, the next contributory chapter proposes an ABR-based caching mechanism at the MEC server. The proposed work focuses on the joint optimization of QoE and caching performance.

"We should not give up and we should not allow the problem to defeat us."

~A P J Abdul Kalam

4 QoE-Aware Adaptive BitRate Caching

In this chapter, we present the second contribution that jointly optimizes the QoE and caching performance. To maintain a decent QoE for end-users, we must consider diverse parameters like content popularity, network conditions, etc. In this work, we propose a QoE-aware ABR caching mechanism at the MEC server using RL. The proposed model predicts content popularity while maintaining the end-user's preferred video quality. An efficient caching mechanism is devised in the MEC server to provide a decent QoE among the end-users. The primary goal of our RL-based framework is to increase the cache hit rate and reduce the backhaul traffic while maintaining a satisfactory QoE. The efficacy of the proposed model is evaluated against *4G* [44] network traces and *Zipf* [92] [93] popularity distribution.

4.1 Introduction

MEC services have emerged as a crucial means of easing the heavy traffic load placed on network operators by the enormous volumes of mobile video traffic generated frequently by heterogeneous devices with varying levels of processing power. Media (image, text, audio, video, etc.) content placed on edge caches helps the network respond faster to the varying requests from UEs. Optimal utilization of the MEC servers storage could be achieved by efficiently caching the most popular video content, as the network is under-utilized during off-peak hours [21]. An efficient caching mechanism at the network edge improves the overall cache hit rate, which results in the reduction of traffic congestion in the backhaul link along with access latency. Let us consider a hypothetical example of video caching at the MEC server, where a user with high-speed network bandwidth requests higher resolutions or HD videos. But the same is not valid for users with low network bandwidth, as the delay incurred will be too large, resulting in the degradation of users' QoE with frequent re-buffering events. Under such a scenario, ABR streaming solutions like DASH [30], have been employed in the content delivery networks to improve users' overall QoE [3], [72], and [94]. The standard DASH dataset mentioned by Lederer et al. [43] and available at [53] breaks down a video into various segment duration (1, 2, 4, 6, 10, 15) seconds. Every video segment (of a particular time duration) consists of multiple bitrates/representations (for example, 240p, 360p, 480p, 720p, 1080p, etc.). Due to heterogeneous user demands, it is challenging to calculate and predict the most popular (most requested) bitrate of a video segment in advance. Proactively predicting the popularity of segments and storing those popular segments into the MEC server's cache reduces the network load on the BS.

In [58] Suoheng Li et al. proposed *PoPCache*, a caching algorithm based on the popularity of content. The popularity of the requested content is measured. If the popularity score is more than the existing content in the local cache, *PoPCache* replaces the content with the least popular score in the cache. Otherwise, no content is evicted from the current cache. However, simple caching solutions, such as *RLCache* [32] and *PoPCache* [58] not including QoE optimization in the caching mechanism, harm the overall QoE of the user.

Therefore, more recently ABR-based caching solutions have been introduced to improve the overall QoE along with caching performance. Hence, Chang Ge et al [13] presented a video caching strategy, which aims at improving the QoE. The proposed scheme performs a two-stage caching mechanism where videos are sorted based on their content-level popularity. Each content is then sorted based on the segment-level popularity. Segments with the least popularity are deleted, with at least one representation for every segment being present in the cache. Zahaib et al. [25] proposed *AViC* a caching algorithm for Adaptive Bitrate video. *AViC* avoids caching the highly unpopular video chunks called singletons, so that the cache is not filled with the least popular chunks. Again, Berger et al. [31] proposed *AdaptSize* an adaptive, size-aware caching policy for the CDN servers. It assigns a low probability of admission to video chunks which are large.

The literature review in chapter 2 revealed that while the overall QoE was improved, most caching methods did not successfully optimize the traffic generated at the backhaul links. Almost all the previous frameworks for caching deploy standard algorithms like *LRU*, *LFU*, and its variants for content eviction. It is also observed that relatively less attention is paid to policies evicting the least popular content. Most of the existing work either focuses on the ABR part [71], [95] or the caching part [32], [58]. Such standalone strategies may not constantly improve the overall experience of the end-users. Considering such scenarios, in this chapter, we propose a joint optimization framework using RL that enhances the overall QoE of the end-users by focusing and giving equal weightage to both the ABR and caching mechanism at the MEC. The main contributions of this chapter are summarized below:

- To improve the overall QoE for a video streaming session and to reduce the traffic load on the backhaul links, we introduce a novel joint optimization framework using RL called *ABRCache*. The proposed RL-based framework uses three types of modules, namely *ABR*, *Planner*, and *Evictor* to optimally select the most appropriate and popular bitrate based on segment-level popularity and caches it accordingly.

- In addition, *ABRCache* handles the variations in bandwidth pertaining to different mobility models using the LSTM module. Combined with LSTM, the proposed model can determine and extract patterns from the variable bandwidth sequential input data.

- A novel hierarchical architecture comprising the three modules (*ABR*, *Planner*, and *Evictor*) is presented. We propose a linear reward function that maximizes the users' overall QoE by improving the caching performance at the MEC server and simultaneously reducing the network load on the backhaul links.

- The *Evictor* module uses a novel approach to evict the least popular segment from the local cache by calculating the segments' short, medium, and long-term popularity. The proposed *ABRCache* model selects the least popular segment and removes it from the cache.

The performance of the proposed *ABRCache* model is evaluated against both caching and QoE metrics. The QoE is estimated by determining the *perceived video quality* and *switching effects*. Whereas caching policy was evaluated using metrics such as *cache hit rate*, *backhaul traffic*, and *access delay*. Extensive sets of experiments were performed, and the proposed model *ABRCache* outperformed both heuristics and ML-based state-of-the-art models.

The rest of the chapter is organized as follows: In Section 4.2, an overview of caching scenario with adaptive streaming at the MEC server is illustrated. The proposed model using RL for QoE-aware adaptive bitrate caching is presented in Section 4.3. The experimental setup, along with comparative results, is described in Section 4.4. Finally, Section 4.5 provides a summary of this chapter with directions toward future work.

4.2 System Overview

The MEC server is collocated with a BS [2] to simulate realistic network scenarios. ETSI has also defined the role of MEC's capability for traffic steering. Using the Mp2 reference point to configure the data plane, the MEC platform manages traffic steering in a generic MEC architecture. Traffic is routed among applications, networks, services, etc., with the Mp2 reference point between the MEC and the data plane. The BS and MEC servers serve

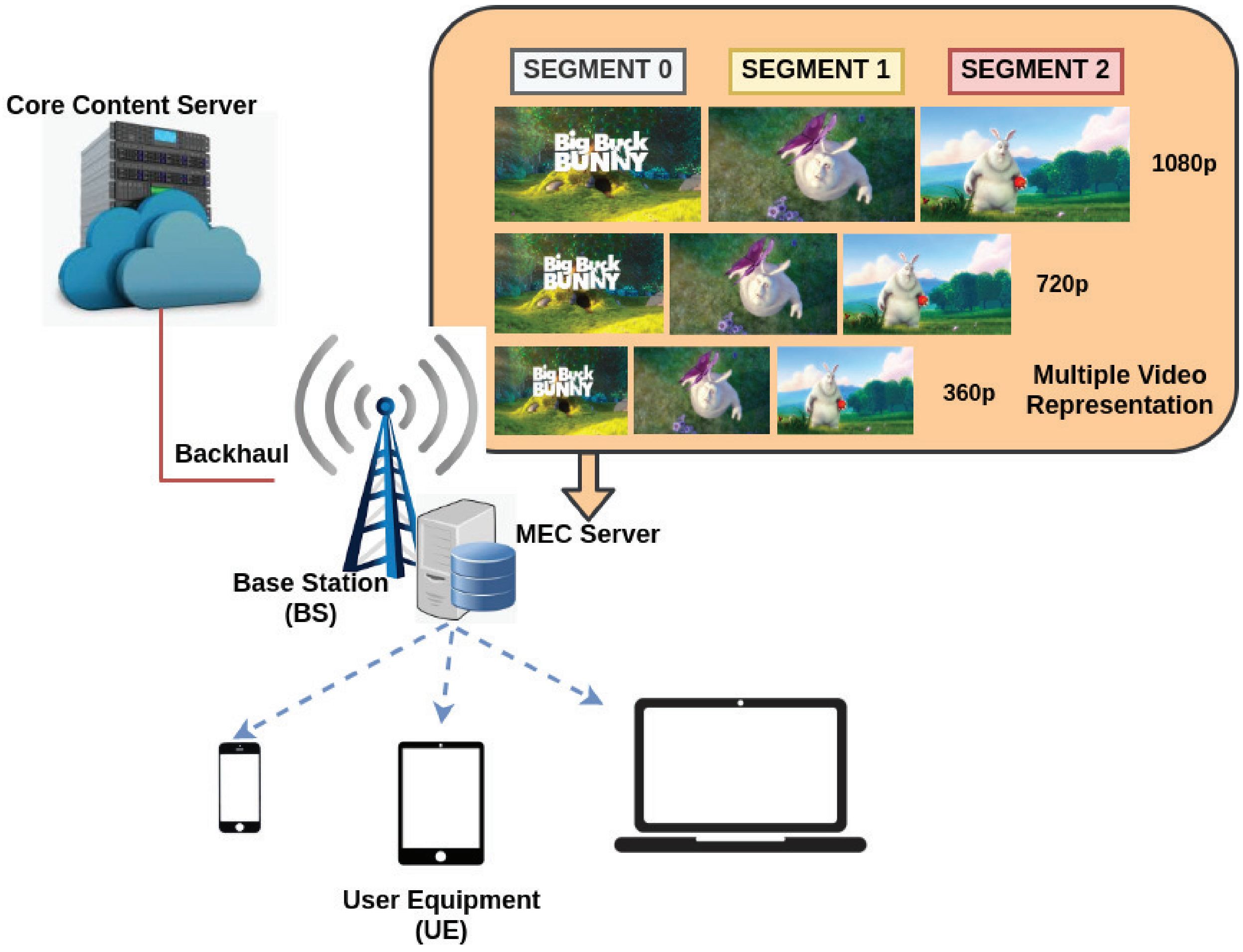


Figure 4.1: *An Overview of DASH Video Caching in MEC For Multiple Bitrates*

requests from various heterogeneous end-users having a wide range of processing capabilities. The CCS resembles the Origin Server as mentioned in [33] and [34], where different media contents are hosted. The BS is connected to CCS through the backhaul links. The MEC server stores the video content in the DASH format. In our proposed work, we have used DASH as the streaming solution. A given video is broken down into segments of various duration like 1, 2, 4, 6, 10, and 15 seconds in the MEC server. For example, for a 5-minute video, the video is broken down into 75 segments of 4 seconds each. Now, every segment consists of multiple bitrates representations. We have considered three types of video resolutions in our proposed work, namely 240p, 480p, and 720p, where pixel (p) is the measured video resolution. The MEC server stores the most popular content based on its popularity level to improve the overall system performance. Consider an example of content caching at the MEC server, where a user requests a video with a particular resolution (say 480p). The ABR module of the proposed *ABRCache* decides on the optimal bitrate supported by the network bandwidth of the requested user. Suppose the selected optimal bitrate of the video segment is present in the local cache of the MEC server *(Cache Hit)*. In that case, the request is directly sent to the respective user. Otherwise, in case of a *Cache Miss*, the request is directed to the CCS through the backhaul links. The primary objective of a caching mechanism is to forward a minimum number of requests (minimizing cache misses) to the CCS, thereby reducing the traffic congestion at the backhaul links. The details of the proposed *ABRCache* model are discussed in detail in section 4.3.

4.3 Proposed Model

The proposed model uses *4G* bandwidth logs and DASH video dataset. In our proposed work, we considered the *4G* bandwidth logs, which consist of bandwidth traces for users travelling in different transportation. The data in [44] belongs to different types of UE with varying mobility (for example, train, bus, tram, car, bicycle, foot). The data collected pertains to different BSs where the UEs move during the simulation. However, data training happens at a single server where all these data belonging to the different BSs are collected. Hence, concepts like hand-offs are handled at the BSs, and we are only concerned with the data received at the server's end. The video segments' request follows the most popular *Zipf* or the power-law distribution. Traffic models generated by *Zipf* distribution have been applied to various caching models such as [33] and [93, 96, 97].

4.3.1 Problem Formulation

Video cached in MEC server follows the DASH format, where every video is broken into equal-sized segments. Let us assume the duration of every segment is denoted by $p + 2$ seconds where $p = \{0, 2, 4, 6\}$. These segments of equal-sized duration's are encoded into three types of video resolutions (240p, 480p and 720p). Let us assume $\mathcal{V}$ be the set of videos cached in the MEC server, where $\mathcal{V} = \{c_1, c_2 ... c_n\}$ and n is the total number of videos. For a video c_i of Z_{c_i} duration, let there be j_{c_i} number of total segments where j_{c_i} is calculated as Equation (4.1)

$$j_{c_i} = \frac{Z_{c_i}}{(p+2)} \tag{4.1}$$

Now, every segment consists of three different bitrates $b = \{b1, b2, b3\}$. The size (the space a particular video segment of a given bitrate will occupy in the cache) of a video segment with different bitrates is different. The total averaged QoE reward for our proposed ABR-Caching mechanism primarily depends on five factors, namely, *video quality, switching factor, cache hit rate, access delay, and backhaul traffic.*

4.3.1.1 Video Quality

Let us assume that at time t, r_t number of requests arrives at the MEC server. v_{ij}^b denotes video quality perceived by an user for the j^{th} video segment of i^{th} video having bitrate b. We denote $Q(v_{ij}^b)$ as a QoE metrics, where $Q(v_{ij}^b)$ maps that bitrate b to the quality perceived by a user *MPC* [98] and *PENSIEVE* [6]. Therefore, our objective is to maximize the overall video quality denoted by Equation (4.2) for all the r requests within a given time T.

$$\begin{aligned} Q(v_{ij}^b) = {} & v_{ij}^b(1) + v_{ij}^b(2) + \dots\dots\dots + v_{ij}^b(r_t) \\ & = \sum_{v_{ij} \in \mathcal{V}} \sum_{d=1}^{r_t} v_{ij}^b(d) \end{aligned} \tag{4.2}$$

4.3.1.2 Buffering Time

To provide a stutter-free viewing experience to the end-users, we need to minimize the re-buffering time $R(t)$ for all r requests. The re-buffering event at time stage t is given by Equation (4.3)

$$R(t) = max(0,\ (\delta - T(t))) \tag{4.3}$$

where, δ denotes the access delay, and $T(t)$ is the buffered video playback duration at time t of the end-user.

4.3.1.3 Access Delay

Access delay δ is the time taken to retrieve a video segment c_{ij}^{b} from the MEC server. Let us assume y_{ij}^{b} be a binary decision variable which denotes that the requested j^{th} video segment of i^{th} video with bitrate b is present in the local cache of MEC. y_{ij}^{b} is 1 when the requested video is present in the cache, else it is 0 when not present. When the requested video is not in the cache, it is retrieved from the CCS with a latency of γ in the backhaul links between the MEC server and CCS. Equation (4.4) represents the access delay.

$$\delta = \sum_{d=1}^{r_t} (1 - y_{ij}^{b}(d))\gamma \tag{4.4}$$

4.3.1.4 Switching Factor

The frequent switching of the bitrates b between the consecutive video segments j and $j+1$ degrades the users viewing experience. The difference between the bitrates of the consecutive segments for the entire video denotes the total switching that occurred during a video streaming session. The switching factor should be minimized, which is given by Equation (4.5)

$$\sum_{d=1}^{r_t} |v_{i(j+1)}^{b}(d) - v_{ij}^{b}(d)| \tag{4.5}$$

4.3.1.5 Backhaul Traffic

The backhaul traffic indicates the network traffic generated in the backhaul links between the MEC server and the CCS. An efficient caching mechanism at the MEC caches the most popular or the most frequently requested video. In doing so, most of the requests will be fetched from the MEC instead of the CCS through the backhaul links, thereby reducing the congestion at those links. Backhaul traffic is defined in terms of load/volume as mentioned in [33]. Therefore, backhaul traffic is measured in terms of GB for the proposed *ABRCache* model. Let us assume that k_{ij}^{b} be the size of video segment c_{ij}^{b} with bitrate b for the j^{th}

segment. The backhaul traffic is measured by Equation (4.6):

$$\varrho = \sum_{d=1}^{r_t} (1 - y_{ij}^b(d))k_{ij}^b \tag{4.6}$$

Therefore, our proposed RL-based model *ABRCache* minimizes the backhaul traffic and access delay by maximizing the cache hit rate at the MEC server. Along with this objective, our proposed model improves the overall QoE of a user by maximizing the perceived video quality and minimizing the switching penalty. A linear reward function is proposed jointly for enhancing the overall performance of the ABR and caching mechanism.

An RL-based model is defined by a state space x_t, an action space a_t and the state-action pair (x_t, a_t). The state-action pair is defined by Equation (4.7)

$$\pi : \pi(x_t, a_t) \to [0, 1] \tag{4.7}$$

where, $\pi(x_t, a_t)$ is the probability that for the input state x_t, action a_t is taken. In our proposed work, there are many (x_t, a_t) pairs. As a result, it isn't easy to design an efficient model. We employed A3C (Actor-Critic) [99] network, an RL-based method for training our proposed model. A3C consists of two networks, namely, *actor* and *critic* network. The actor network defines the state-action policy $\pi(x_t, a_t)$, whereas the critic network rewards the policy defined by the actor network. These two networks train the proposed RL-based *ABRCache* model to maximize the overall QoE reward χ by predicting the expected overall reward $\hat{\chi_t}$ by estimating the gradient. The Advantage function $A^{\pi_\theta}(x, a)$ of the policy gradient method [100] defines the state policies $\pi_\theta(x_t, a_t)$ by the actor network in case of an A3C network. As described in [99], the advantage function for a given experience when there is a transition from state x_t to x_{t+1} is given by Equation (4.8)

$$A(x_t, a_t) = \chi_t + \mu V^{\pi}\theta(x_{t+1}; \theta_v) - V^{\pi}\theta(x_t; \theta_v) \tag{4.8}$$

The three modules, the *ABR*, *Planner* and *Evictor* modules simultaneously train the proposed RL-based *ABRCache* model to improve the overall reward χ_t concerning both ABR

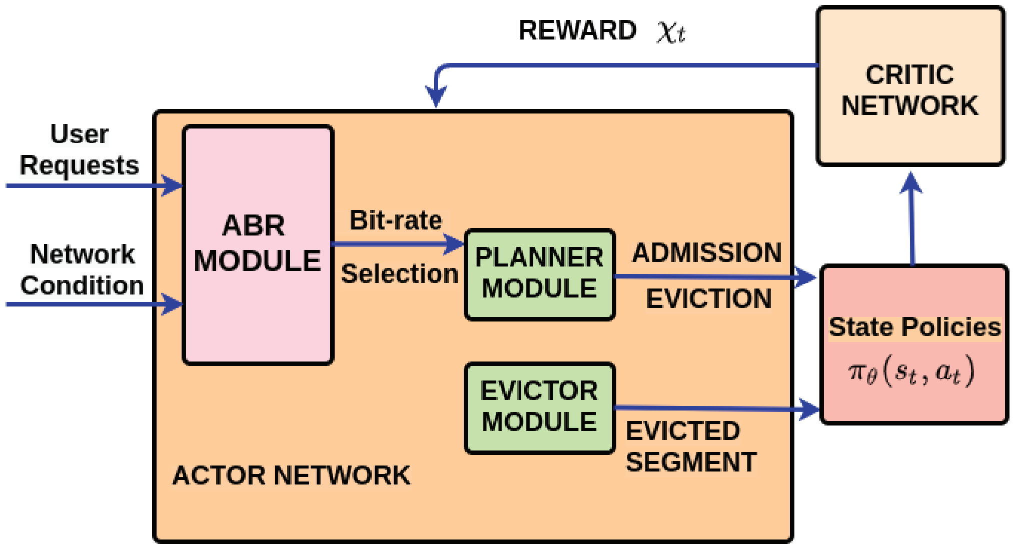


Figure 4.2: *RL-based Model for QoE-Aware Content Caching at the MEC Server*

and Caching parameters. The reward function is defined by Equation 4.9.

$$\chi_t \;=\; \varphi_1 \cdot v^b_{ij}(t) - \varphi_2 \cdot R(t) - \varphi_3 \cdot |v^b_{i(j+1)}(t) - v^b_{ij}(t)| - \varphi_4 \cdot \delta(t) - \varphi_5 \cdot \varrho(t) \quad (4.9)$$

A block diagram of A3C network describing the proposed *ABRCache* model is presented in Figure 4.2. For measuring the importance of future rewards, μ is the discount factor. The critic network parameters are represented by θ_v. $V^{\pi\theta}(\cdot;\theta_v)$ is the estimate of $v^{\pi\theta}(\cdot)$, where $v^{\pi\theta}(\cdot)$ represents the expected reward obtained by following the policy π_θ. The critic network learns an estimate of $v^{\pi_\theta}(s)$ from the observed rewards. The first three terms are related to ABR strategy when the requests for a particular video arrives. The objective of the ABR module is to maximize the overall viewing experience considering the network conditions and the client's buffer occupancy. The following two terms are related to the caching mechanism at the MEC server handled by *Planner* and *Evictor* module. These two modules simultaneously maximize the cache hit rate and decrease the service load on the BS's backhaul links during peak hours.

Algorithm 3: Training in ABRCache Model

```
1  Let $\lambda$ be the total number of requests $r_1, r_2, .., r_\lambda$ ;
2  $b = b1, b2, b3 =$ Number of video quality level ;
3  for $r_\lambda, \lambda = 1, 2...d$ do
4      Input: $x_t^{ABR} = (\tau_t, n_t, \zeta_t, r_t, n_{r_t}^{b1}, n_{r_t}^{b2}, n_{r_t}^{b3})$
5      Output: bitrate $v_{ij}^b$ of video $c_i$ to be sent to the user
6      if $(seg_t$ in MEC$)$ Cache then
7          $seg_t \rightarrow$ User (U)
8          Calculate reward $\chi_t$
9      end
10     else
11         CCS $\rightarrow seg_t$
12         Cache_Decision()
13     end
14 end
```

4.3.2 Architecture of Proposed *ABRCache*

Figure 4.2 represents the block diagram of the proposed RL-based *ABRCache* A3C network. The proposed model is trained at the MEC server, which consists of the ABR module and the *Cache* manager. The cache manager consists of the *Planner* and *Evictor* module, which decides whether to cache a particular video segment. The workflow of the training procedure for the proposed RL-based *ABRCache* model is described in Algorithm 3 and Algorithm 4. Algorithm 3 presents a detailed description of how the *ABR module* performs the training. The *ABR module* takes user requests and network conditions as input. *ABR module* then decides the bitrate b for the requested segment. To decide whether to cache the missing segment into the MEC server's local cache, $Cache\_Decision()$ is defined in Algorithm 4. The $Cache\_Decision()$ module presented in Algorithm 4 states that the *Planner* decides on the Admission/Eviction of seg_t. The requested segment seg_t will be delivered to the user and won't be cached if the *Planner* module decides to *"Don't Admit."* In the event of caching seg_t, the Planner needs to evict some segments to be replaced by the new incoming segment. The MEC server's available cache space is considered for caching the missing segment seg_t. If space is available, the requested missing segment seg_t is simply cached. The least popular segments are evicted in the event of the non-availability of cache space.

Under such a scenario, the *Evictor module* has popularity information of all the segments. The segments with the least popularity value will be evicted until space is available to admit the more popular segments. The critic network evaluates the state policy $\pi_\theta(s_t, a_t)$ defined by the actor-network by measuring the reward χ_t. The actor and the critic network maximizes the overall reward as defined in Equation (4.9).

Algorithm 4: Cache_Decison() Module for ABRCache

```
Function Cache_Decison():
    Input: $x_t^{Planner} = (\tau_t, a_t(ABR), n_{rt}^{b1}, n_{rt}^{b2}, n_{rt}^{b3}, avg_p^{b1}, avg_p^{b2}, avg_p^{b3})$
    Output: $O_{Planner}$: Decides on **Admission / Eviction** of $seg_t$
    $k_{ij}^b$ → Size of $seg_t$
    if $(O_{Planner} == DON'TADMIT)$ then
        Calculate reward $\chi_t$.
    else
        if $(s_{ij}^b \leq K)$ then
            $seg_t \rightarrow$ MEC Cache
        else
            while $(K < s_{ij}^b)$ do
                Evict()
            $O_{Planner} \rightarrow$ Evict()
Function Evict():
    for $i$ in range(1, $w$) do
        for $j$ in range(1, $z$) do
            $x_t^{Evictor_i} = (avg_{sh}^b, avg_{md}^b, avg_{lg}^b)$
            $O_{Evictor_i}$: Denotes the segment that needs to be removed of type $i$.
    return $O_{Evictor_i}$ ;
Calculate reward $\chi_t$.
Function ABRCacheTrain():
    $ActorNetwork \rightarrow \pi_\theta(x_t, a_t)$
    $\pi_\theta(x_t, a_t) \rightarrow Critic\ Network$
    $A(s_t, a_t) \rightarrow$ Experience calculated as in (4.8) when there is a transition from $s_t$ to $s_{t+1}$
    $CriticNetwork \rightarrow \chi_t$
    $Maximize\ \chi_t$
    return
```

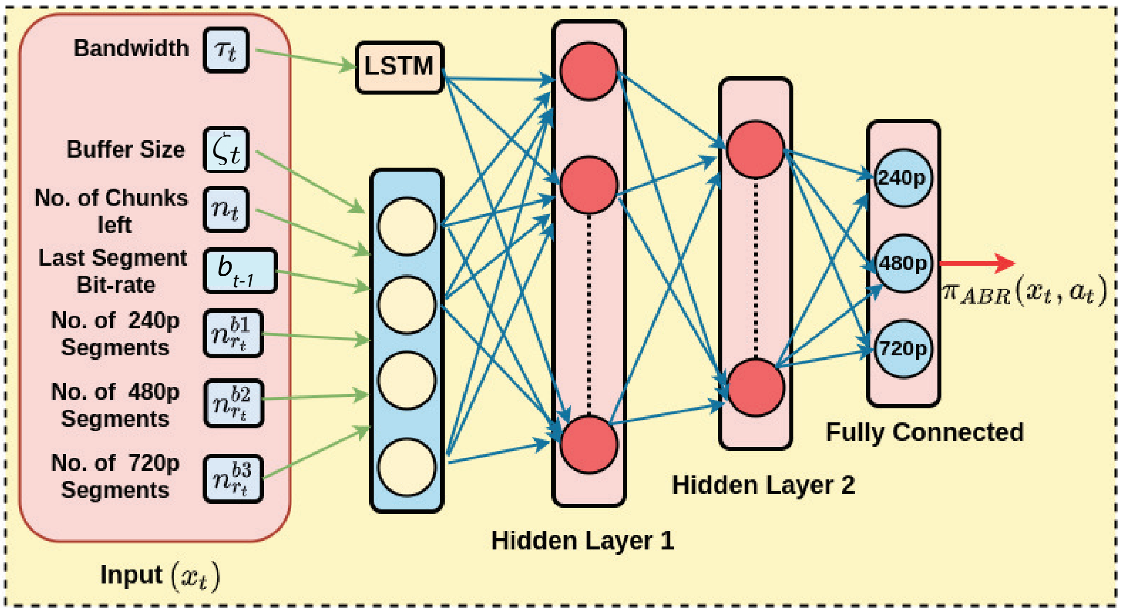


Figure 4.3: *Proposed Architecture of RL-based ABR Module*

4.3.2.1 ABR Module

As illustrated in Figure 4.2, the ABR module receives various video requests from the users and decides the bitrate of the video segment. The most appropriate bitrate of the video segment is then delivered to the end-users based on the network condition. As part of proposed *ABRCache* model, the ABR module takes input as x_t where $x_t = (\tau_t, n_t, \zeta_t, b_{(j-1)}, n_{r_t}^{b1}, n_{r_t}^{b2}, n_{r_t}^{b3})$, where τ_t is the past throughput, n_t is the number of chunks left, ζ_t is the current buffer size, $b_{(j-1)}$ is the last segment bitrate, $n_{r_t}^{b1}$ is number of segments of bitrate $b1$ in cache, $n_{r_t}^{b2}$ is number of segments of bitrate $b2$ in cache and $n_{r_t}^{b3}$ is number of segments of bitrate $b3$ in cache. The ABR module decides the appropriate bitrate for the requested video segment considering the user demand and present network condition. ABR module is a neuronal model comprising of a LSTM and a DNN network, as depicted in Figure 4.3. The bandwidth (τ_t) is considered time series data and is fed to the first hidden layer through a LSTM module. LSTM learns the previous patterns of the varying bandwidth data and helps model an efficient caching strategy. Other input parameters are directly fed to another hidden layer. The outputs from these layers are combined using a hidden layer with 64 neurons. The first hidden layer's output is fed to another hidden layer

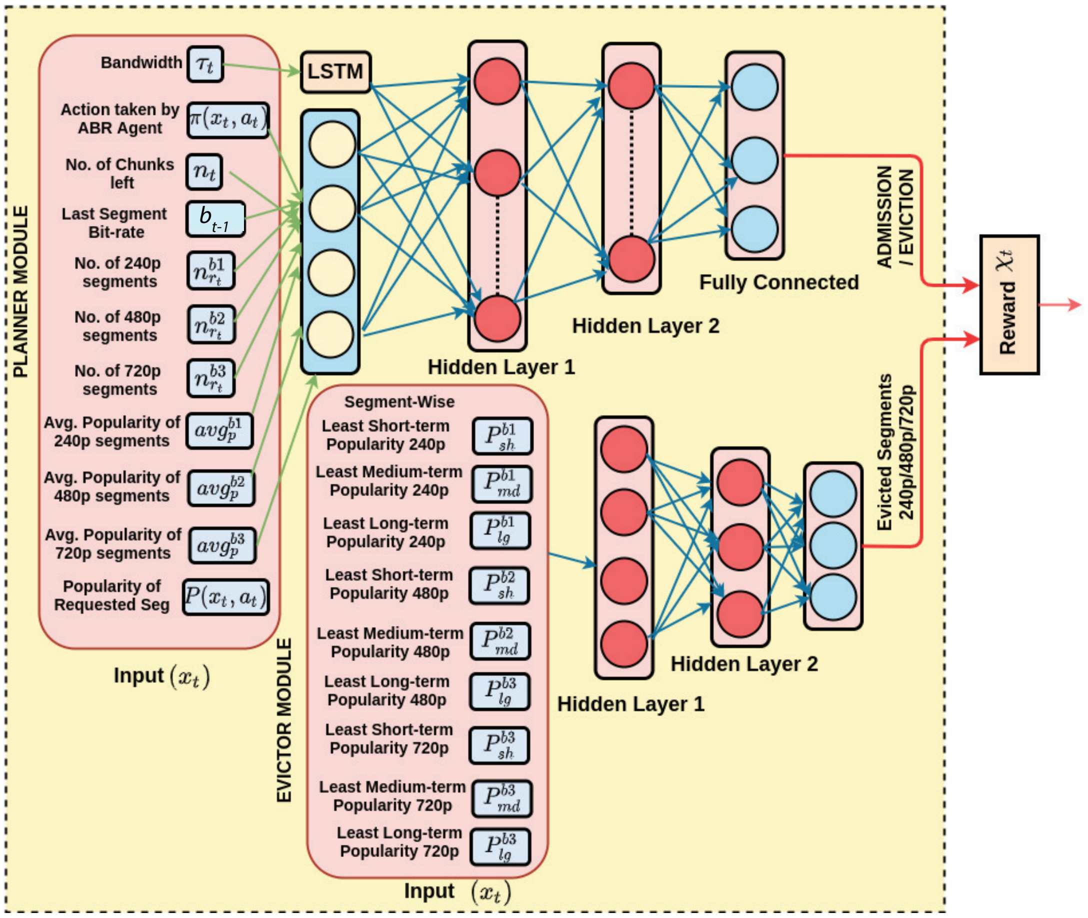

PLANNER MODULE
Bandwidth τ_t
Action taken by ABR Agent $\pi(x_t, a_t)$
No. of Chunks left n_t
Last Segment Bit-rate b_{t-1}
No. of 240p segments $n_{r_t}^{b1}$
No. of 480p segments $n_{r_t}^{b2}$
No. of 720p segments $n_{r_t}^{b3}$
Avg. Popularity of 240p segments avg_p^{b1}
Avg. Popularity of 480p segments avg_p^{b2}
Avg. Popularity of 720p segments avg_p^{b3}
Popularity of Requested Seg $P(x_t, a_t)$
Input (x_t)
LSTM
Hidden Layer 1
Hidden Layer 2
Fully Connected
ADMISSION / EVICTION
Reward χ_t
Evicted Segments 240p/480p/720p
EVICTOR MODULE
Segment-Wise
Least Short-term Popularity 240p P_{sh}^{b1}
Least Medium-term Popularity 240p P_{md}^{b1}
Least Long-term Popularity 240p P_{lg}^{b1}
Least Short-term Popularity 480p P_{sh}^{b2}
Least Medium-term Popularity 480p P_{md}^{b2}
Least Long-term Popularity 480p P_{lg}^{b3}
Least Short-term Popularity 720p P_{sh}^{b3}
Least Medium-term Popularity 720p P_{md}^{b3}
Least Long-term Popularity 720p P_{lg}^{b3}
Input (x_t)
Hidden Layer 1
Hidden Layer 2

Figure 4.4: *Proposed Architecture of RL-based Cache Manager*

with 32 neurons connected to the final fully connected output layer. The output layer gives the final action of selecting an appropriate bitrate corresponding to the resolutions 240p or 480p, or 720p. The predicted bitrate becomes one of the inputs to the *Cache Manager*.

4.3.2.2 Cache Manager

The *Cache Manager* determines whether or not the selected segment of a particular bitrate is present in its cache. If the segment is available, it is instantly retrieved from the MEC server's local cache and transmitted back to the user. Otherwise, the *Cache Manager* fetches the segment from the CCS and delivers it to the requesting user. Now, the *Cache Manager* decides whether to store the missing segment into the MEC cache (called *Cache_Decison()*) or not. In the event of caching the requested missing segment, it decides which segments to be evicted. We need a control policy to make our caching decisions. An RL-based agent handles this control policy. The *Cache Manager* has two main working modules, namely *Planner* and *Evictor* module as depicted in Figure 4.4.

The *Planner module* decides on the *Admission / Eviction* of the requested video segment. If it decides not to admit the incoming segment, no segment is added to the cache, so eviction is not required. The *Planner module* receives the bitrate of the video segment decided by the ABR module. It maintains bandwidth information of the user for the past t time steps and has some overview information regarding the cache. The *Planner module* takes input $x_t = (\tau_t, a_t(ABR), n_{rt}^{b1}, n_{rt}^{b2}, n_{rt}^{b3}, avg_p^{b1}, avg_p^{b2}, avg_p^{b3}, P(x_t, a_t))$ and outputs an action array of the form (ψ_1, ψ_2, ψ_3), where ψ_i signifies the number of segments to be removed by the i^{th} evictor. avg_p^{b1}, avg_p^{b2} and avg_p^{b3} is the average popularity of bitrate $b1, b2, b3$ respectively. $P(x_t, a_t)$ is the popularity value of the requested segment. To measure the popularity of a given segment, we calculate the *short-term popularity* (how many times a segment is requested in last 100 time steps), the *medium-term popularity* (how many times a segment is requested in last 1000 time steps) and the *long-term popularity* (how many times a segment is requested in last 10000 time steps). The architecture of the *Planner module* is shown in Figure 4.4.

As illustrated in Figure 4.4, the *Evictor* module finds the least short-term popularity (P_{sh}^{b1}) segment, medium-term popularity (P_{md}^{b1}) segment and long-term popularity (P_{lg}^{b1}) segment for all the cached segment of $b1$ bitrates. Similarly, for the other cached segments of bitrates, $b2$ and $b3$, the least short, medium and long-term popularity segment is searched. z in Algorithm 4 denotes the number of segments available for a given bitrate b. The popularity scores of these segments are fed as input to a hidden layer of 64 neurons, which is further connected to another hidden layer of 32 neurons. The final output layer is a fully connected layer that predicts which segment (240p or 480p, or 720p) to be removed from the local cache of the MEC server. Finally, the cache manager calculates the proposed ABR-based caching mechanism's overall reward χ_t.

4.4 Experiments and Results

To evaluate the efficacy of the proposed RL-based *ABRCache* model, an extensive set of experiments are performed with respect to evaluation metrics such as *QoE rreward*, *video quality*, *smoothing effect*, *cache hit rate*, *backhaul traffic*, and *access delay*. The proposed *ABRCache* model was trained and tested using the DGX Workstation with GPU Tesla V100 card having 32 GB GPU memory. The proposed model was also tested in a high-end CPU driven server consisting of 128 GB CPU memory. The performance of the proposed *ABRCache* model was compared against various state-of-the-art caching strategies such as *QoECache* [13], *RLCache* [32], *AViC* [25], *PoPCache* [58], *AdaptSize* [31], *Rate Based (RB)* [95], *Buffer Based (BB)* [71], *LRU*, and *LFU*. The comparative strategies were judiciously selected and grouped into various categories based on the following criterion:

- **ABR-based Caching Strategies:** *QoECache* [13], *AViC* [25], and *AdaptSize* [31] are ABR-based caching strategies considered for comparing with the proposed *ABR-Cache* model.

- **Baseline Strategies:** We compare the proposed *ABRCache* with baseline strategies such as *LRU* and *LFU*. *LFU* is one of the most widely deployed caching algorithms. Since our proposed work is a ABR-based caching strategy. Therefore, for a fair comparison, we combined *LRU* and *LFU* with an adaptive streaming mechanism such as *Buffer Based (BB)* [71]. *LRU* and *LFU* combined with *BB* is considered as a baseline for the evaluation of our proposed work similar to [33].

- **Simple Caching and ABR Strategies:** To further measure the efficacy of the proposed work, we carried out another set of extensive experiments. Here, we compared *ABRCache* with simple caching strategies such as *RLCache* [32], and *PoPCache* [58] and then with only ABR strategies namely *Rate Based (RB)* [95] and *Buffer Based (BB)* [24].

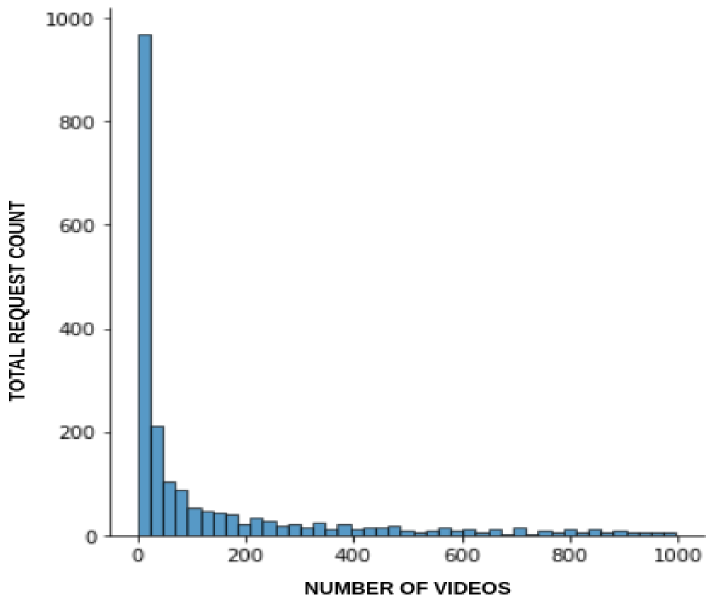
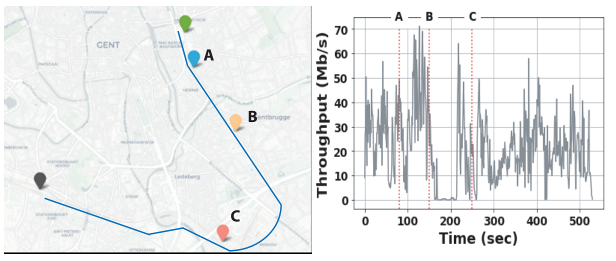


Figure 4.5: *Pattern of Video Request following Zipf Distribution*

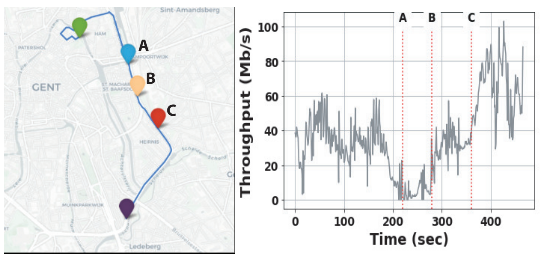
4.4.1 Dataset

The experiments are performed based on the standard DASH [43] video dataset and *4G* bandwidth [44] logs. Jeroen et al. [44] collected the network throughput logs in the city of Ghent, Belgium, for different types of transportation such as foot, bicycle, bus, tram, train, and car. Figure 4.6 illustrates the routes taken by a car and train, respectively, along with the measured bandwidth. It is observed that throughput falls drastically from point A to B in Figure 4.6(a) and B to C in Figure 4.6(b) due to tunnels, large buildings, and various other obstructions. The transportation type significantly impacts the available bandwidth (car throughput ranged from 0 to 100 Mb/s, whereas it ranged from 0 to 70 Mb/s in train). Prominent buildings on the right side of the selected route in Figure 4.6(a) hinder the connection from point A to B. On reaching B, better coverage is achieved, improving the throughput significantly.

In our proposed work, three types of bitrates are considered. The bitrates corresponding to the video resolution which are considered in our work *ABRCache* is: $240p \rightarrow 300kbps$, $480p \rightarrow 1200kbps$, and $720p \rightarrow 1850kbps$. This is consistent with the YouTube video requirement [55]. Video segment durations of *2 and 4 seconds* are considered. In our simulation setup, request from the users follows *Zipf* distribution. *Zipf* request typically follows a popularity distribution called the *Pareto Principle (80-20 rule)*, where 80% of content requests are for the 20% most popular content. In our proposed work, we model the distribution similarly. In our work, as depicted in Figure 4.5, we assigned around 80% of the request for the 20% most popular content. Whereas, for the remaining 80% of the least popular content, we assign around 20% of the request. We assume that the delay (γ) is 1 ms between the CCS and the MEC within a given BS and the cache size to be 2 GB and 4 GB.



(a) *Selected Route in a Car and Measured Bandwidth over Time*



(b) *Selected Route in a Train and Measured Bandwidth over Time*

Figure 4.6: *A Car and a Train Traveling from North to South in Belgium with the Measured Throughput*

Table 4.1: *Comparison of Total QoE Reward Against various Cache Size and Segment Duration's. The best results are shown in Red and the second best in Blue*

Schemes		*Proposed ABRCache*	QoECache [13]	RLCache [32]	AViC [25]	PoPCache [58]	AdaptSize [31]	RB [95]	BB [71]	LRU	LFU	RL_DQN
2 GB	2 sec	35.08	32.54	27.30	23.01	24.06	24.90	31.62	31.80	33.03	34.25	31.19
	4 sec	34.47	33.82	27.04	29.01	22.93	30.27	28.09	30.60	29.84	32.45	32.14
4 GB	2 sec	37.44	34.56	34.11	23.52	25.49	28.8	35.28	34.13	35.92	36.59	33.57
	4 sec	38.26	34.94	32.26	28.9	25.13	33.94	32.12	34.22	32.78	35.69	34.85

4.4.2 Comparison Against Overall QoE Reward

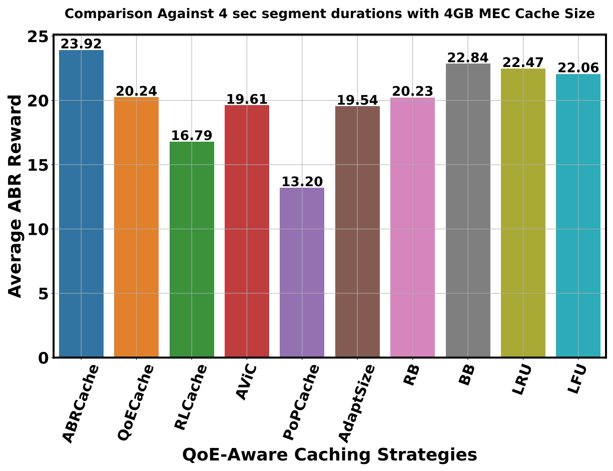
Table 4.1 presents a comparative study of the total averaged QoE Reward χ_t as mentioned in Equation 4.9. The performance of *ABRCache* is compared against various segment durations (2 and 4 seconds) with MEC cache sizes of 2 GB and 4 GB. *ABRCache* outperformed existing strategies like *QoECache* [13], *RLCache* [32], *AViC* [25], *PoPCache* [58], *AdaptSize* [31], *Rate Based (RB)* [95], *Buffer Based (BB)* [71], *LRU*, and *LFU* for different combinations of video segment durations and MEC cache sizes. Table 4.1 illustrates that as the cache size increases, the overall reward for the proposed *ABRCache* also increases. As a result, an increase in the cache size in the MEC server is anticipated to improve the proposed model's overall performance.

4.4.3 Comparison Against ABR Metrics

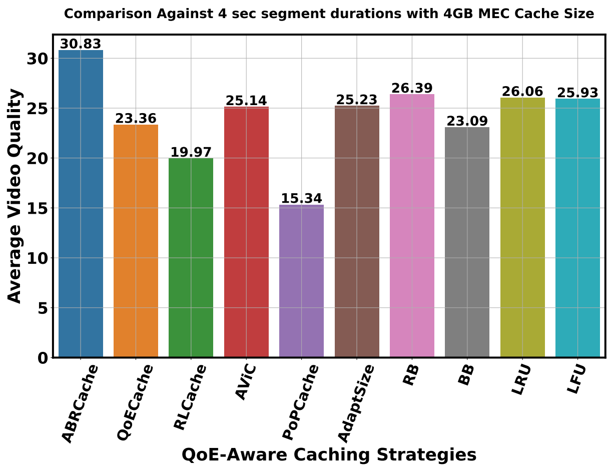
The proposed *ABRCache* model, along with enhancing the overall QoE reward χ_t, also improves the individual ABR and caching metrics simultaneously. The performance of the proposed *ABRCache* model against various metrics such as average *ABR reward*, perceived *video quality*, and *smoothing* effect is depicted in Figure 4.7. The *ABR reward* is calculated by considering the first three terms of the QoE reward χ_t of Equation (4.9). Figure 4.7(a), 4.7(b) and 4.7(c) represents the ABR metrics concerning 4 GB cache size with a segment duration of 4 seconds each. The overall perceived video quality and the smoothing/switching penalty are calculated as Equation (4.2) and (4.5), respectively. As illustrated in Figure 4.7(a), *ABRCache* outperforms the ABR reward against the other competing strategies. The proposed model significantly surpasses the other strategies with respect to overall video quality as depicted in Figure 4.7(b) for 4-second duration. Smoothing penalty is one of the other ABR metrics which measures the number of times the video quality changes for a video streaming session. Figure 4.7(c) shows that the proposed 4-second segment duration model outperforms various strategies. Therefore, it is evident from the pictorial illustration that the proposed scheme, along with improving the overall QoE score, also improves the other ABR metrics.

4.4.4 Comparison Against Caching Metrics

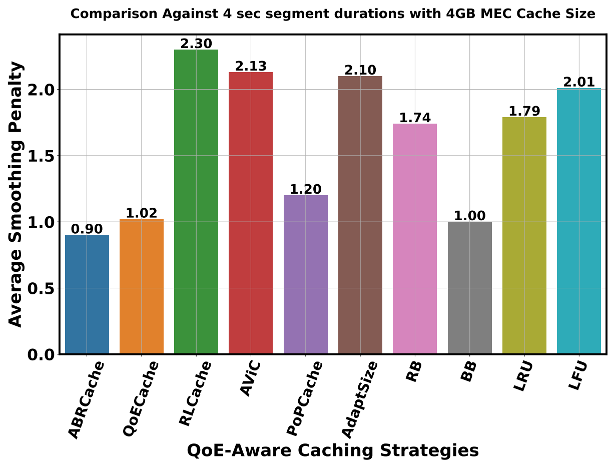
The caching performance of the proposed *ABRCache* model is presented in Figure 4.8 for 4-second duration. The proposed model significantly outperforms existing strategies concerning cache hit rate, access delay, and backhaul traffic for various combinations of MEC cache sizes and video segment durations. Figure 4.8(a) represents the performance of the proposed model against cache hit rate for 4-second duration. It is observed that *ABRCache*



(a) *ABR Reward*



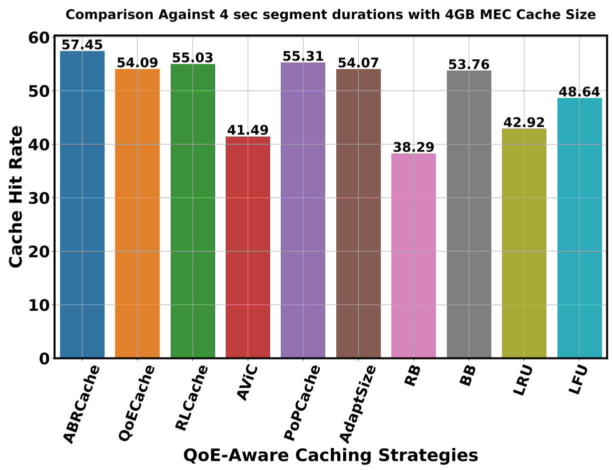
(b) *Video Quality*



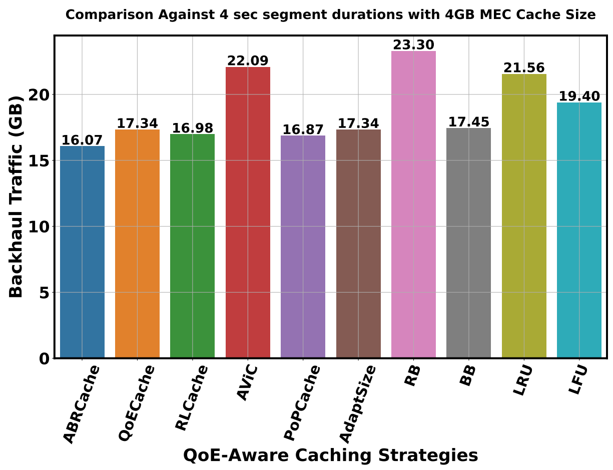
(c) *Smoothing Penalty*

Figure 4.7: *Performance of the Proposed ABRCache Against ABR Reward, Video Quality, and Smoothing Penalty for 4-second Segment with 4 GB Cache Size*

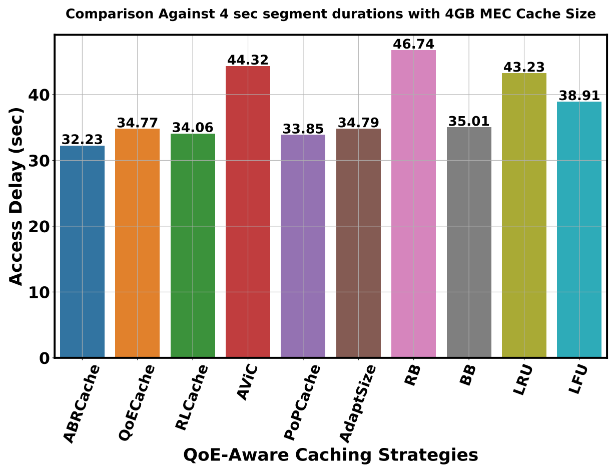
model presents a better cache hit rate concerning both heuristics and ML-based model along with standard baseline models such as *LRU* and *LFU*. The comparative results for access delay, which is measured as Equation (4.4), is presented in Figure 4.8(c). The overall delay in retrieving the video segments is better than other comparative strategies. Also, the total traffic generated at the backhaul links, which increases as the number of misses in the MEC server increases, is improved for the proposed *ABRCache* model in comparison to other caching models. Figure 4.8(b) illustrates the backhaul traffic of the proposed model along with other competitive strategies. With its multiple actor-learners, the proposed RL-based model can interact with the environment efficiently and gather experiences to provide an optimal solution such that the viewing experience and the caching efficiency are optimized simultaneously.



(a) *Cache Hit Rate*



(b) *Backhaul Traffic*



(c) *Access Delay*

Figure 4.8: *Performance of the Proposed ABRCache Against Cache Hit Rate, Backhaul Traffic, and Access Delay for 4-second Segment with 4 GB Cache Size*

4.4.5 Ablation Study

We compare the proposed A3C-based *ABRCache model* with a DQN-based model, which can be considered a baseline model for RL. DQN is one of the first breakthroughs that was successfully applied in deep learning to RL. Using the Q-function, DQN learns the value function, and the policy followed is simply taking the action that maximizes the Q-value at each step. Whereas our proposed model, which uses A3C network, takes all the advantages of policy and value-based models by exploring and exploiting the solution space efficiently. The proposed *ABRCache* model is compared with *DQN*, and the comparative results for the total QoE reward is presented in Table 4.1. Table 4.1 represents the performance of *ABRCache* with *RL_DQN* against various segment durations (2 and 4 seconds) with MEC cache sizes of 2 GB and 4 GB. The proposed model outperforms *RL_DQN* significantly. Table 4.2 presents a comparative study of the proposed *ABRCache* against various cache sizes and segment duration. The performance of *ABRCache* improves with an increase in the cache size from 2 to 4 GB. As the cache size of the MEC server increases, more popular video segments could be stored, resulting in fewer cache misses at the MEC. In this comparative study, it can be seen that caching strategies *RLCache* and *PoPCache* outperforms ABR strategies *RB* and *BB* only with respect to caching metrics such as cache hit rate, backhaul traffic and access delay. Whereas ABR strategies *RB* and *BB* outperform *RLCache* and *PoPCache* against video quality and smoothing penalty. This is because the primary objective of simple Caching and ABR strategies is only improving cache hit rate and QoE, respectively. Also, the cache hit rate of approaches such as *RLCache* and *PopCache* is very close to the proposed model.

However, their QoE values, such as video quality and smoothing penalty, are inefficient compared to the proposed *ABRCache.* The *total average QoE* reward, as mentioned in Equation (4.9), is a combined metric related to caching and QoE parameters. Table 4.1 presents a comparative study of the total averaged QoE reward. It can be seen that strategies which perform fairly only with caching (*RLCache* and *PopCache*) and only with ABR strategies (*RB* and *BB*) failed to perform optimally when compared with *ABRCache* concerning the total averaged QoE reward. Therefore, the caching framework *ABRCache* is a superior strategy, ensuring a better cache hit rate and an enhanced viewing experience. With a higher cache hit rate, the traffic at the backhaul links and access delay in retrieving the requested video segment improves substantially.

In another set of ablation study for our proposed work, we swept a range of Neural Network parameters to understand better each configuration's impact on the overall performance of *ABRCache* measured by the QoE reward. We varied the number of hidden layers for each module (ABR, *Planner*, *Evictor*), and for each dense layer, we varied the number

Table 4.2: *Comparison of ABRCache Against Various Cache Sizes and Segment Duration. The Best results are shown in Red and second best results in Blue*

Cache Size	Segment Duration	Evaluation Metrics	*Proposed ABRCache*	QoECache [13]	RLCache [32]	AViC [25]	PoPCache [58]	AdaptSize [31]	RB [95]	BB [24]	LRU	LFU
2 GB	2 SEC	CHR	47.43	45.98	44.73	44.02	45.66	44.65	25	40.75	30.06	36.97
		Backhaul (GB)	20.2	22.76	23.83	24.68	22.08	24.58	28.45	25.93	27.11	23.73
		Access Delay (sec)	39.29	40.37	41.30	41.84	40.62	41.38	56.02	43.54	53.02	47.112
		ABR Reward	27.27	21.21	17.96	15.41	15.54	16.19	25.78	21.96	25.9	26.99
		Video Quality	31.03	23.03	21.6	17.5	16.79	18.1	29.33	23.58	30.24	30.17
		Smoothing Penalty	1.63	1.64	1.72	1.81	1.98	1.72	1.84	1.67	1.81	1.71
	4 SEC	CHR	47.2	44.9	45.24	43.56	45.42	43.07	27.18	41.75	31.48	39.56
		Backhaul (GB)	20.24	22.5	24.1	25.01	22.8	24.88	24.34	25.86	23.09	22.65
		Access Delay (sec)	39.47	41.18	40.93	44.5	40.81	42.55	54.43	43.54	51.21	45.17
		ABR Reward	23.09	20.06	16.57	19.26	18.86	19.17	22.13	20.01	22.64	22.2
		Video Quality	31.19	23.26	19.9	20.3	23.26	25.73	26.03	23.15	26.22	25.85
		Smoothing Penalty	1.64	1.66	2.5	2	2.89	2.13	1.88	1.69	1.79	1.82
4 GB	2 SEC	CHR	57.67	53.58	56.11	47.42	52.96	55.98	36.62	53.77	41.04	46.49
		Backhaul (GB)	15.98	17.53	16.67	19.86	17.76	16.62	37.87	17.46	22.27	20.21
		Access Delay (sec)	32.06	35.16	33.44	39.83	35.63	33.34	48.01	35.02	44.66	40.53
		ABR Reward	24.54	21.48	18.66	13.17	13.54	13.48	22.86	21.03	23.46	23.73
		Video Quality	30.84	23.43	21.4	15.5	15.34	15.86	27.04	22.83	29.53	28.08
		Smoothing Penalty	1.08	0.9	1.2	1.3	1.31	1.41	1.69	0.95	1.79	1.85
	4 SEC	CHR	57.45	54.09	55.03	41.49	55.31	54.07	38.29	53.76	42.92	48.64
		Backhaul (GB)	16.07	17.34	16.98	22.09	16.87	17.34	23.3	17.45	21.56	19.4
		Access Delay (sec)	32.23	34.77	34.06	44.32	33.85	34.79	46.74	35.01	43.23	38.91
		ABR Reward	23.92	20.24	16.79	19.61	13.2	19.54	20.23	22.84	22.47	22.06
		Video Quality	30.83	23.36	19.97	25.14	15.34	25.23	26.39	23.09	26.06	25.93
		Smoothing Penalty	0.9	1.02	2.3	2.13	1.2	2.1	1.74	1	1.79	2.01

Table 4.3: *Comparison of ABRCache Against Different Number of Dense Layers and Neurons*

Sl. No	Number of Hidden Layers and Neurons/Layers			Training Time (Mins)	Average QoE Reward
	ABR Module	Planner Module	Evictor Module		
Config 1 (Proposed)	Hidden -> 2 Layers 64, 32 Neurons	Hidden -> 2 Layers 32, 32 Neurons	Hidden -> 2 Layers 64, 32 Neurons	144.15	28.01
Config 2	Hidden -> 2 Layers 32, 16 Neurons	Hidden -> 2 Layers 16, 16 Neurons	Hidden -> 2 Layers 32, 16 Neurons	142.27	21.45
Config 3	Hidden -> 2 Layers 128, 64 Neurons	Hidden -> 2 Layers 64, 64 Neurons	Hidden -> 2 Layers 128, 64 Neurons	149.73	23.15
Config 4	Hidden → 2 Layers 64, 64 Neurons	Hidden -> 2 Layers 64, 64 Neurons	Hidden -> 2 Layers 64, 64 Neurons	143.42	25.01
Config 5	Hidden -> 4 Layers 64, 64, 64, 64 Neurons	Hidden -> 4 Layers 64, 64, 64, 64 Neurons	Hidden -> 4 Layers 64, 64, 64, 64 Neurons	205	21.29

of neurons. Table 4.3 presents the results obtained for various configurations. As shown, performance of the proposed configuration *Config-1* gives better results concerning *average QoE reward*, where QoE is a linear combination of parameters concerning both caching and ABR metrics. In this ablation study, results against two hidden layers and a deeper network *(4 layers)* are obtained. Firstly, we fixed the number of hidden layers for all three modules to two. For *Config-1 to 4*, fixing the number of hidden layers and varying the number of neurons per hidden layer, we obtained the overall reward, keeping track of the training time *(100K iterations)* simultaneously. The results for these configurations for a 2 GB cache size and 2-second segment duration are presented in Table 4.3. It is observed that in the configurations of the proposed model, we obtained a better reward for almost a similar training time for all four configurations. However, when we make the model deeper (*4 hidden layers* for all three modules), the training time increases substantially with decreased QoE reward. This is due to the fact that as the number of layers increases, the model overfits for which the model cannot generalize and fits too closely to the training data. As the model complexity becomes higher, it learns the noise along with the training data. Therefore, after tuning the model for various configurations, it is seen that the proposed model presents a more optimal solution for *Config-1*.

4.5 Summary

In the second contributory chapter, we proposes an RL-based QoE-aware adaptive bitrate caching model *ABRCache* at the MEC server. The introduction of MEC at the edge of the network, usually within BS, reduces the load of the network operators and makes it easy to provide different services (like video on demand, intelligent browsing, etc.) to the heterogeneous end-users. For an efficient deployment of MEC, caching popular (frequently requested) video content in the MEC server (within BS) is a requirement. To improve the overall QoE for a user, the proposed model *ABRCache* maximizes the total QoE reward, a function consisting of ABR and caching metrics. Thus, the proposed model maximizes the perceived video quality and cache hit rate while minimizing the smoothing/flickering effect, backhaul congestion between the CCS and BS, and access delay simultaneously. The proposed *ABRCache* model consists of three modules, namely *ABR, Planner,* and *Evictor*. The RL agent trains the overall *ABRCache* model to increase the overall QoE reward. A comprehensive set of experiments demonstrate that the proposed *ABRCache* model outperforms existing works like *QoECache* [13], *RLCache* [32], *AViC* [25], *PoPCache* [58], *AdaptSize* [31], *Rate Based (RB)* [95], *Buffer Based (BB)* [71], *LRU*, and *LFU*.

It has been observed that *ABRCache* is performing well but can be more efficient if it considers multiple BSs rather than a single BS. Intuitively, a collaboration among multiple BSs may improve the performance by handling heterogeneous user requests more efficiently to improve cache hits and reduce the backhaul congestion. Keep this observation in mind, in the next contributory chapter, we plan to extend the proposed model to multiple BS having edge computing facilities. The BS equipped with MEC will collaboratively work amongst themselves to decide upon caching the most popular segments, thus offloading the computation task to multiple BS.

"The biggest risk is not taking any risk... In a world that's changing really quickly, the only strategy that is guaranteed to fail is not taking risks."

~ Mark Zuckerberg

5 Collaborative Video Caching in Clustered Edge Network

In this chapter, we extend the concept of caching within single BS to caching amongst multiple BSs, where we propose an RL-based collaborative caching mechanism where the edge servers cooperate to serve the requested content from the end-users. Specifically, this research aims to improve the overall cache hit rate at the MEC, where the edge servers are clustered based on their geographic locations. The main challenge of the underlying scenario is to meet heterogeneous user demands from multiple collaborative BSs in an ever-changing network environment. The said task is modelled as a multi-objective optimization problem and that has been solved using an RL framework by maximizing the cache hit rate while simultaneously reducing backhaul congestion, access delay, and re-buffering time. In addition, a novel cache admission and eviction policy is defined by calculating the priority score of video segments in the clustered MEC mesh network.

5.1 Introduction

The diverse request patterns across various edge servers, along with the advent of popular streaming solutions such as DASH [30], caching the most requested bitrate of a video segment is challenging task. Most of the previous caching schemes such as [25, 31, 32] are not adaptive enough to handle such diverse and complicated requests across temporal and geographical dimensions. Thus, to further improve the caching performance at the edge node, *Collaborative Caching* [33–35] has been introduced recently. In collaborative caching, MEC servers collaborate amongst themselves through the high bandwidth fronthaul links to serve the requested videos. Whenever a requested content is not present in a MEC server's cache, the MEC server will first contact the collaborating MECs for the requested video instead of directly forwarding the request to the CDN. Tuyen X. Tran et al. [33] proposed a collaborative caching mechanism for adaptive streaming. They formulated a caching policy at the MEC server using Integer Linear Programming *(ILP)*. In this work [33], the collaborative caching mechanism deploys two primary mechanisms, one is for caching and the other for processing (transcoding). In collaborative caching, caching decisions must be made considering various factors such as past request patterns and current network conditions. However, a collaborative caching mechanism faces several challenges. *Firstly*, in contrast to non-collaborative caching, a collaborative caching strategy also has to decide the MEC server at which the content should be cached. Thus, another degree of complexity is added to the caching decision. *Secondly*, for DASH videos, caching multiple representations will increase the overhead with respect to the storage capacity of the MEC server. *Finally*, designing an efficient caching mechanism that efficiently utilizes the available computational and storage resources to improve the overall caching efficiency at the edge nodes is challenging.

In the literature survey presented in chapter 2, it is observed that most of the existing caching strategies [32], [83], [89] are not efficient enough to handle the diverse and complex nature of video request patterns. Collaborative caching strategies such as [33], [67], [68], and [69] use transcoding at the MEC server which is computationally expensive. Transcoding a video from a higher to a lower bitrate at the MEC server requires significant time and computational overhead that might rapidly deplete the available resources on the edge servers. It is also observed that the eviction policy deployed by most of the above discussed literature employs either LRU or LFU for evicting the contents. Intuitively, the caching strategy in a BS under a collaborative environment can be modeled as a dynamic control problem. In this chapter, we propose a DRL-based Collaborative Caching strategy called, *ColabCache* in clustered edge networks. In this collaborative caching environment, BSs are clustered based on geographical location. Since MEC servers are installed in each BS, we can assume this is a cluster of MEC servers. These collaborating MEC servers in the clustered edge network cache the video content based on their segment-level popularity. In our problem statement, we consider four types of video resolutions (240p, 360p, 480p, and 720p), where the videos are in DASH format [43]. The objective of *ColabCache* is to formulate an efficient caching mechanism at the MEC server that maximizes the overall cache hit rate by placing the most popular videos in the collaborating MEC servers. The increased cache hit rate at the MEC server reduces the overall congestion at the backhaul links and access time in retrieving the video content. As the backhaul congestion decreases, re-buffering events also reduce substantially for a video streaming session improving the viewing experience of the end users. In order to achieve the objective mentioned above, we design the reward function of the proposed RL framework to solve the said optimization problem. The reward function is based on different optimization parameters like cache hit rate, backhaul traffic congestion, access delay, re-buffering events, and perceived video quality for the proposed RL-based framework. To evaluate the efficacy of the proposed framework, two different types of datasets are considered, real and simulated. In the simulated dataset, the video request is generated using *Zipf* distribution. We considered the *Iflix* movie streaming dataset [46] for the real-world data.

To fulfill the above-mentioned objectives, the major contributions achieved in this chapter, are as follows:

- We present a novel DRL-based framework called *ColabCache* using A3C network for caching at the edge server. It handles the diversities arising from both temporal and geographical dimensions in the caching environment by exploring and exploiting the solution space with its multiple actor learners.

- In addition, we propose a novel RL-based *cache admission* and *eviction* policy, unlike the previous works, which primarily use simple eviction policies such as LRU and LFU. *ColabCache* collaborates amongst the MEC servers within a cluster to make caching decisions based on the calculated *Priority Score* of video segments with respect to all MEC servers in the cluster.

- *ColabCache* unlike most of the previous works, is independent of the size of the media library at the Core Server. The proposed model is independent of the total number of videos in the media library, and hence the performance is optimal even when the media library is continuously increasing.

The performance of the proposed *ColabCache* is evaluated against various metrics such as cache hit rate *(CHR)*, backhaul traffic, access delay, video quality and re-buffering events. Extensive experiments over the *Iflix* and *Zipf* dataset reveal that *ColabCache* outperforms the existing state-of-the-art strategies substantially.

The remainder of this chapter is organized as follows. In Section 5.2, we present the overall system model and problem formulation for collaborative caching in clustered edge networks. We describe the proposed model using RL in Section 5.3. Section 5.4 presents the experimental results considering both the simulated and real-time datasets. The chapter is finally summarized in Section 5.5.

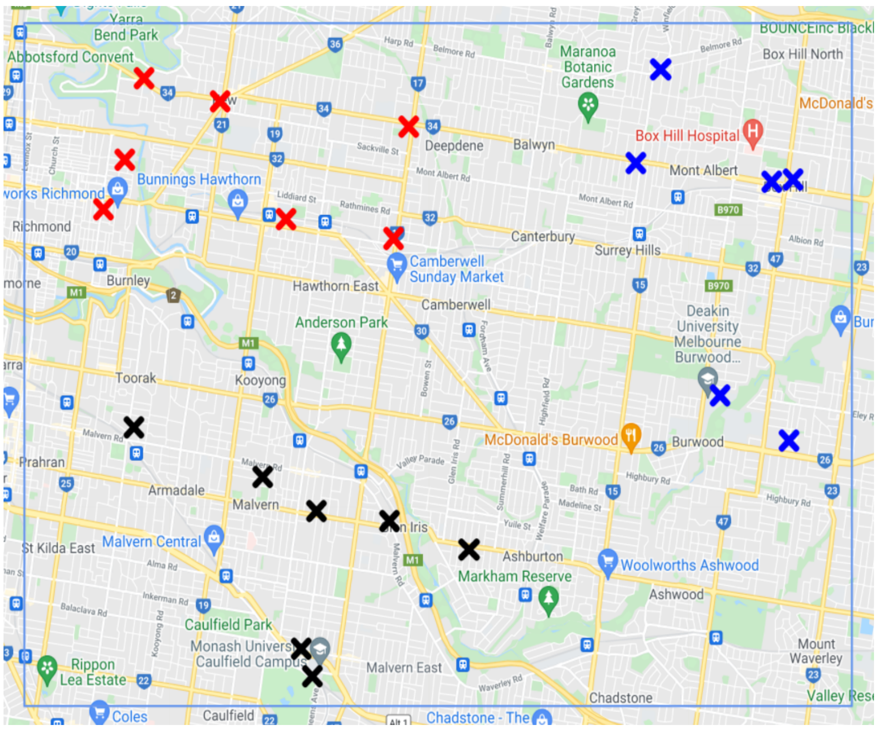


Figure 5.1: *A Representation of Clustered BS within the Target Region*

5.2 System Overview

In this section, we present an overview of the proposed cluster-based collaborative-caching model for the 5G MEC network. In our proposed work, 20 Edge Servers have been randomly sampled from the *EUA* dataset [45] within the target 120 km^2 region. The *EUA* dataset contains the geographical locations of the MEC servers. The *EUA* dataset maintains a set of *edge servers* and *user locations*. The edge servers dataset consists of information such as $\langle SiteID - Latitude - Longitude - Name - State - Licensing - AreaID - PostCode \rangle$. The MEC servers are first grouped into clusters of M-sized mesh networks. The 20 BSs are clustered according to Algorithm 5 into 3 clusters. A representation of clustered BSs within the target region is shown in Figure 5.1. Red, blue and black colors mark the 3 different clusters. Algorithm 5 is used to cluster the BSs based on the longitudinal and latitudinal coordinates for an M-clustered MEC network. The clustering algorithm is similar to the bottom-up *Agglomerative clustering* algorithm used for *Hierarchical clustering* in unsupervised learning. Initially, each BS is a separate cluster. In each iteration, we select 2 clusters such that (i) the size of the merged cluster does not exceed the maximum cluster size M_o and (ii) the distance between any 2 clusters does not exceed the threshold distance d_o. The distance between clusters is the Single-Link distance between them. The single-Link distance between 2 clusters is the distance between the closest pair of BSs (one in each cluster). Furthermore, the threshold d_o is fixed according to the average coverage range of a BS.

Algorithm 5: Clustering Algorithm

```
1 Let $N$ be the set of all BSs within the target region
2 Initialize each BS as a separate cluster
3 Calculate pair-wise distances between all BSs
4 while $(\exists g_1, g_2 \mid g_1.size + g_2.size \leq M_o)$ & $(Dist(g_1, g_2) < d_o)$ do
5     Let $G_1$, $G_2$ be the clusters with minimum distance satisfying the above condition
6     Merge clusters $G_1$ & $G_2$ to form a new cluster $G$
7     Compute the distance between $G$ and all other clusters
```

In a conventional edge caching scenario, a single MEC server is co-located within a BS. The MEC server is connected to the CCS through the backhaul link. Whenever an end-user requests a video segment, the requested video segment is first searched in the local cache of the MEC server. If it is found in the MEC server, a cache hit occurs. Otherwise, the request is forwarded to the CCS through the backhaul link to serve the requested content. With a sharp increase in mobile video traffic, every year [9] and [34], the load on the backhaul links has increased substantially which increases the access delay faced by the end-users. The need to deliver a stutter-free viewing experience to the end-users is challenging for network operators during peak traffic hours. Therefore, to reduce the access delay and the traffic at the backhaul links, the MEC servers serve the video requests collaboratively. Figure 5.2 illustrates a hypothetical scenario for collaborative caching where the MEC servers are clustered based on their geographical locations. Furthermore, in DASH, various representations (say, 240p, 360p, 480p, and 720p) of the same video segment are present and every video is broken down into p second durations (say, 2 and 4 sec). Therefore, for DASH video streams storing all the representations is not efficient. Hence, only the popular/frequently requested representation of the video segments is cached at the MEC servers. For collaboration, we have grouped the MEC servers into clusters based on their geographical proximity. Each MEC server collaborates with the other MEC servers in its cluster, referred to as its *Neighbours* in our proposed work. Each user requests and receives video content from the BS closest to them (with respect to signal strength), referred to as the user's *Home BS*.

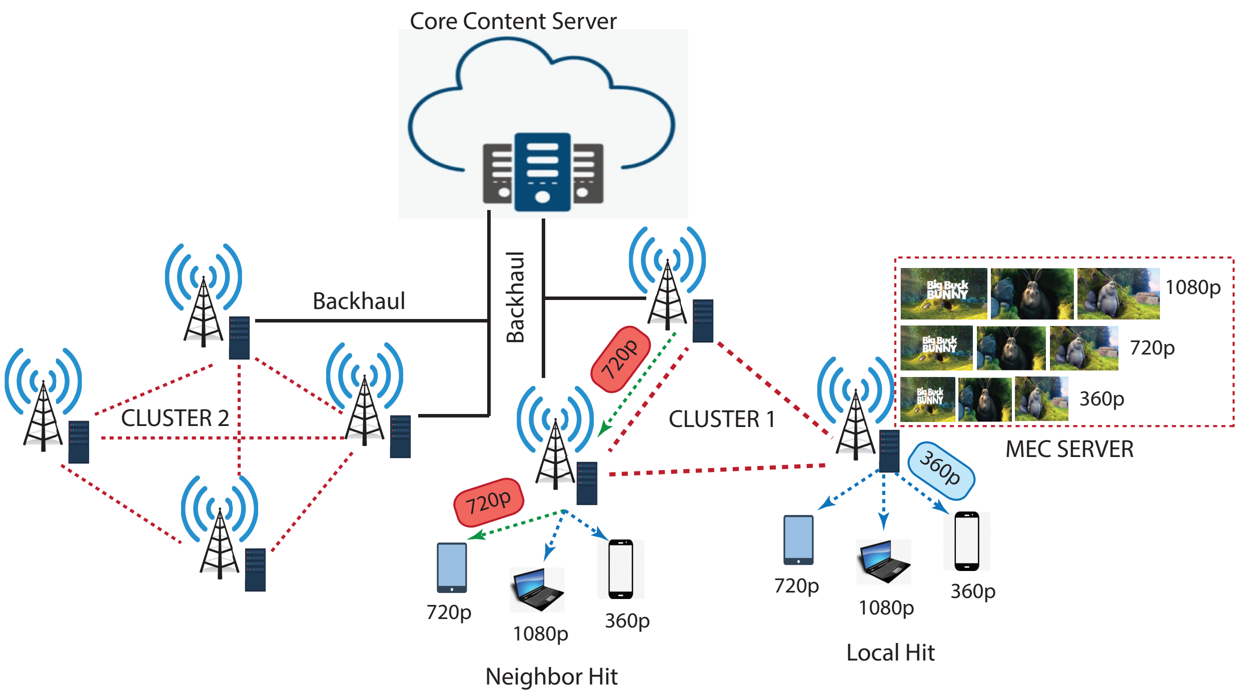


Figure 5.2: *A Hypothetical Scenario for Collaborative Caching using Clustering of MEC Servers for DASH Videos*

As illustrated in Figure 5.2, the possible events which can occur when a user requests a video in a collaborative caching scenario can be mentioned as:

1. **Local Hit:** A local hit occurs when the requested representation (say, 360p) of a video segment is present in the user's *Home BS*'s cache.

2. **Neighbour Hit:** A neighbour hit occurs when the requested representation of a video segment is not present in the *Home BS*'s cache, but it is present in a neighbouring MEC. The request is forwarded to the neighbouring MEC, which replies with the requested representation.

3. **Cluster Miss:** In the event of a miss in both the local and neighbour cache of the MEC server, the requested video segment is fetched from the CS.

ColabCache an RL-based collaborative caching strategy aims to maximize the overall cache hit rate at the edge server, which reduces the backhaul traffic and access delay. Along with caching efficiency, *ColabCache* improves the overall viewing experience with improved video quality and reduced re-buffering events. The A3C network of *ColabCache* considers the above problem as a multi-objective optimization problem. The performance of the proposed model is measured by χ_t defined by Equation (5.12). The model measures the popularity of the video segment both locally (in the MEC server) and globally (amongst all the MEC servers in the cluster network). The popularity of the requested video segments is considered for caching the content based on the availability of space in the MEC server's local cache.

5.3 Proposed Model

In this section, we present the problem formulation of the proposed *ColabCache*, followed by the architecture and training of the proposed model.

5.3.1 Problem Formulation

Let us assume that there are M MEC servers denoted by $\mathcal{M} = \{1, 2, ..., M\}$ within a given cluster. The MEC servers are clustered according to Algorithm 5. The MEC servers are equipped with a local cache having a cache memory size (capacity) K_m for the m^{th} MEC server for $m \in \mathcal{M}$. The MEC servers within a given cluster are all interconnected to each other via a fronthaul mesh network. Let us denote $\mathcal{V}$ to be the set of all videos. Since we are considering DASH video streams, each video is broken down into segments of p seconds duration where $p \in \{2, 4\}$. In this work, all further analysis can be done with respect to video segments. Let us consider c_{ij} ($c_{ij} \in \mathcal{V}$) be the requested j^{th} segment of i^{th} video. We assume that each segment c_{ij} has L representations, where $L = \{240p, 360p, 480p, 720p\}$. We denote content c_{ij} in representation b ($b \in L$) by c^b_{ij}. Unless explicitly specified, the problem is formulated with respect to a time step t.

Figure 5.2 depicts the events of *local hit*, *neighbour hit* and *cluster miss*. For a request arriving at home MEC server ($m \in \mathcal{M}$), binary decision variables $\{x_{ij}^{bm}, y_{ij}^{bmz}, w_{ij}^{bm}\} \in \{0, 1\}$ are introduced, which can be described as follows:

1. $y_{ij}^{bm} = 1$ if c_{ij}^{b} is present in the local cache of MEC server m, i.e. *local hit*; and $y_{ij}^{bm} = 0$ otherwise in case of a miss.

2. $x_{ij}^{bmz} = 1$ if c_{ij}^{b} is not present in the local MEC m, and is fetched from the neighbouring MEC server z, i.e. *neighbour hit*, where $z \in \mathcal{M}$, $z \neq m$; and $x_{ij}^{bmz} = 0$ otherwise.

3. $w_{ij}^{bm} = 1$ if c_{ij}^{b} is not present at any MEC server in the cluster i.e. *cluster miss*, it is fetched from the CCS; and $w_{ij}^{bm} = 0$ otherwise.

The total size of the video segments to be cached in a MEC server m must not exceed the total capacity K_m which is denoted as Equation (5.1), where k_{ij}^{b} is the size of c_{ij}^{b}:

$$\sum_{c_{ij} \in \mathcal{V}} \sum_{b \in L} y_{ij}^{bm} \cdot k_{ij}^{b} \leq K_m \tag{5.1}$$

To cache a given video segment c_{ij}^{b} into the local cache of the MEC server m we first compute the ratio ϕ_{ij}^{bm} for the video segment. ϕ_{ij}^{bm} for a segment c_{ij}^{b} is the ratio of the number of requests r_{ij}^{bm} received by MEC server m for c_{ij}^{b} to the total number of requests received for all segments in a given time frame denoted by Equation (5.2):

$$\phi_{ij}^{bm} = \frac{r_{ij}^{bm}}{\sum\limits_{ij \in \mathcal{V}} \sum\limits_{b \in L} r_{ij}^{bm}} \tag{5.2}$$

The Priority Score for content c_{ij}^{b} in a collaborative environment is represented by P_{ij}^{bm}, denoted by Equation (5.3).

$$P_{ij}^{bm} = \phi_{ij}^{bm} + \sum_{z \in \mathcal{D}_{ij}^{bm}} \left(\phi_{ij}^{bz} \cdot \frac{b(m, z)}{\beta} \right) \tag{5.3}$$

where, $\mathcal{D}_{ij}^{bm}$ is the set of neighbouring MEC servers which are collaborating with m for segment c_{ij}^{b}. $b(m,z)$ is the link bandwidth between m and z. β is a scaling factor. The Priority Score P_{ij}^{bm} represents the value of caching the content c_{ij}^{b} at MEC server m. Caching some content at a particular MEC server (say m) does not only affect the caching decisions of m but also of the neighbouring MEC servers which are collaborating with m for segment c_{ij}^{b}. Therefore, the ϕ_{ij}^{bz} values of collaborating MEC servers also contribute to the Priority Score P_{ij}^{bm}. The contributions of neighbours are scaled down with weights proportional to the bandwidth between the MEC server m and the neighbours. The intuition behind these weights is that larger the bandwidth between two MEC servers, the stronger would be the collaboration between them and hence more would be the Priority Score.

The Cache Hit Rate *(CHR)* is the number of local and neighbour hits in a given cluster to the total number of requests received in a given time frame. The number of requests of c_{ij}^{b} in a given time frame is denoted by r_{ij}^{bm}. The *CHR* is represented by Equation (5.4):

$$CHR = \frac{\sum\limits_{ij\in\mathcal{V}}\sum\limits_{b\in L}(y_{ij}^{bm}r_{ij}^{bm} + x_{ij}^{bmz}r_{ij}^{bm})}{\sum\limits_{ij\in\mathcal{V}}\sum\limits_{b\in L} r_{ij}^{bm}} \tag{5.4}$$

The Peak Signal-to-Noise Ratio (PSNR) is a measure of the quality of video content. According to [101], PSNR of a video segment encoded in bitrate b can be approximated as Equation (5.5), where β depends on the amount motion or stillness of the video segment.

$$PSNR = \alpha \log_{10}(b) \tag{5.5}$$

Jitter, $\mathcal{J}$, is a measure of the variation of the latency in fetching consecutive video segments. In a network with high congestion, *jitter* would be high which hurts the user experience. Let t_0 to t_n be the times when a user, u, is requesting consecutive video segments from the BS. For user u, *jitter* can be calculated as Equation 5.6

$$\mathcal{J} = \sum_{i=1}^{n} (\delta(t_i) - \delta(t_{i-1})) \tag{5.6}$$

The video re-buffering time $R(t)$ at time stage t is given by Equation (5.7):

$$R(t) = max(0,\ (\delta - T(t))) \tag{5.7}$$

where, δ is the *Access Delay* between the time the request for content c_{ij}^{b} is received by the *home* BS and the time when c_{ij}^{b} is delivered to the end-user. $T(t)$ is the buffered video time playback at time t at the end-user. The *Access Delay* δ is calculated as Equation (5.8):

$$\delta = (1 - y_{ij}^{bm}) \cdot \Big(\sum_{\substack{z \in \mathcal{M} \\ z \neq m}} y_{ij}^{bmz} \cdot \gamma_{(m,z)} + w_{ij}^{bm} \cdot \gamma_{(m,back)} \Big) \tag{5.8}$$

where,

$$\gamma_{(m,back)} = \frac{k_{ij}^{b}}{\tau(m, back)} + \Delta_{(m,back)} \tag{5.9}$$

and,

$$\gamma_{(m,z)} = \frac{k_{ij}^{b}}{\tau(m, z)} + \Delta_{(m,z)} \tag{5.10}$$

where, $\Delta_{(m,back)}$ is the traffic delay in the backhaul link from BS m, $\Delta_{(m,z)}$ is the traffic delay between the local BS (m) and neighbour BS (z), and $\tau(m, back)$ is the backhaul bandwidth from BS m. $\gamma_{(m,z)}$ is the delay in the edge link when the content is fetched from the neighbour BS in case of a miss in the local BS represented by Equation (5.10). In case of miss in both the local and neighbour BS the content is fetched from the CCS with a backhaul delay of $\gamma_{(m,back)}$ given by Equation (5.9).

To measure the network congestion in the backhaul links between the MEC and the CCS for a given time frame, Equation (5.11) represents the *backhaul traffic* ϱ. The backhaul traffic is generated as a result of cache miss in the cluster-based collaborative MEC network.

$$\varrho = \sum_{i \in \mathcal{C}} \sum_{b \in L} w_{ij}^{bm} \cdot r_{ij}^{bm} \cdot k_{ij}^{b} \tag{5.11}$$

Therefore, whenever there is a cluster miss ($w_{ij}^{bm} = 1$), the missed video segment is fetched from the CCS which adds to the network traffic in the backhaul link. The congestion in the backhaul link rises as the frequency of misses increases in the cluster-based collaborative network. On the other hand, if there is a hit in the collaborative network (i.e. $w_{ij}^{bm} = 0$), no additional traffic weight is added to the backhaul link, thereby not contributing to the network congestion in the backbone.

In our proposed work *ColabCache*, a reward function χ_t is defined which is governed by the metrics defined earlier *(CHR, access delay, backhaul traffic, and re-buffering events)* along with *perceived video quality ($q(t)$) and jitter ($\mathcal{J}(t)$).* χ_t is represented by Equation (5.12):

$$\chi_t = \varphi_1 \cdot q(t) - \varphi_2 \cdot (\mathcal{J}(t)) - \varphi_3 \cdot \varrho(t) + \varphi_4 \cdot CHR(t) - \varphi_5 \cdot \delta(t) - \varphi_6 \cdot R(t) \tag{5.12}$$

To design a cluster-based collaborative caching strategy in the MEC network, we formulate an optimization problem and solve it using A3C [99] networks of RL. The proposed optimization formulation maximizes the overall reward χ_t by maximizing the CHR, and video quality and minimizing the backhaul traffic, access delay, jitter and re-buffering events simultaneously. Therefore, given the availability of resources such as the total capacity of MEC servers m, K_m , and the event of availability of desired video segments $\mathcal{C} \triangleq \{y_{ij}^{bm}, x_{ij}^{bmz}\}$ in $\mathcal{M}$, the objective is to maximize χ_t for all time t in solving the problem $\mathcal{U}^t$ such that:

$$\mathcal{U}^t : \max_{\mathcal{C}} \sum_{t \in T} \sum_{v \in \mathcal{V}} \chi_t, \tag{5.13a}$$

subject to,

$$\sum_{ij \in \mathcal{V}} \sum_{b \in L} y_{ij}^{bm} \cdot k_{ij}^{b} \leq K_m \ , \forall m \in \mathcal{M} \tag{5.13b}$$

$$y_{ij}^{bm} + \sum_{\substack{z \in \mathcal{M} \\ z \neq m}} x_{ij}^{bmz} + w_{ij}^{bm} = 1 \ , \forall m \in \mathcal{M} \tag{5.13c}$$

The constraint (5.13b) expresses the fact that the total size of content cached at a MEC server m cannot exceed its total capacity K_m. Constraint (5.13c) ensures that for each content request, exactly one of the three possible events occurs (i.e. local hit, neighbour hit or cluster miss).

5.3.2 Proposed Actor-Critic (A3C) model of ColabCache

In this section, we describe the proposed *ColabCache* that uses the state-of-the-art actor-critic A3C [99] network. A3C involves the training of two networks: *Actor and Critic* network.

5.3.2.1 Architecture of ColabCache

Figure 5.3 presents a diagrammatic overview of the proposed actor-network of *ColabCache*. The *actor* network of the RL framework takes several input features related to the popularity of content along with various network parameters to generate policies π_θ for deciding the target BS to store the requested video content. The learning agents of *ColabCache* actor network takes state inputs $d_t = (\overrightarrow{F(t)}, \overrightarrow{FQ(t)}, \mathcal{X}, \tau_{(back)}, \Delta_{(back)}, \tau_{(edge)}, \Delta_{(edge)}, K', \delta, R(t), k_{ij}^{b}, \mathcal{L})$ to its neural network at time t when video segment c_{ij}^{b} is requested. $\overrightarrow{F(t)} = \{\overrightarrow{F_m(t)} \mid m \in \mathcal{M}\}$, where $\overrightarrow{F_m(t)}$ is a sequential vector of the number of requests received for segment c_{ij}^{b} at BS m in the last t time slots; $\overrightarrow{FQ(t)} = \{\overrightarrow{FQ_m(t)} \mid m \in \mathcal{M}\}$, where $\overrightarrow{FQ_m(t)}$ is a sequential vector of the number of times quality b was requested at MEC server m in the last t time

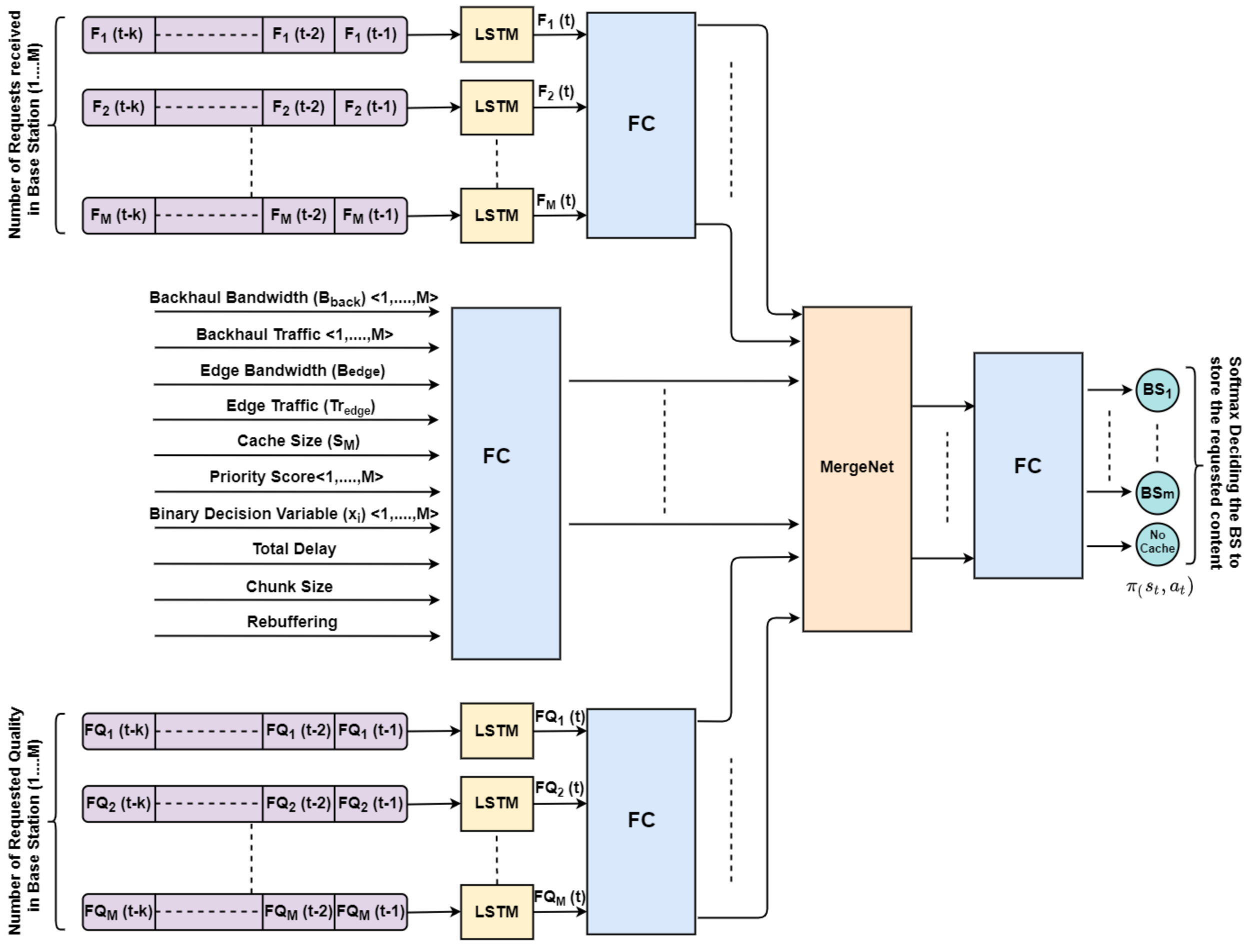


Figure 5.3: *Proposed Architecture of Actor Network for RL-based A3C Network*

slots. $\tau_{(back)} = \{\tau_{(m,back)} \mid m \in \mathcal{M}\}$ and $\Delta_{(back)} = \{\Delta_{(m,back)} \mid m \in \mathcal{M}\}$ are the backhaul bandwidth and backhaul traffic, respectively, between the MEC servers and the CCS ; $\tau_{(edge)} = \{\tau_{(m,z)} \mid m, z \in \mathcal{M}, m \neq z\}$ and $\Delta_{(edge)} = \{\Delta_{(m,z)} \mid m, z \in \mathcal{M}, m \neq z\}$ are the edge bandwidth and edge traffic respectively between each pair of MEC servers in the cluster; $K' = \{K'_m \mid m \in \mathcal{M}\}$, where K'_m denotes the amount of free space in BS m's cache; δ is the access delay; k^l_{ij} is the chunk size for content c_{ij} with b^{th} representation; and $\mathcal{L} = \{LPS^m \mid m \in \mathcal{M}\}$, where LPS^m denotes the Least Popularity Score in BS m's cache. Vector of cache decision variables $\mathcal{X} = \{y^{bm}_{ij} \mid m \in \mathcal{M}\}$ is also fed into the Fully Connected (FC) layer. This input gives control of redundancy of video content c^b_{ij} in the cluster to *ColabCache* which will learn the best redundancy level to reach optimal performance based on the request pattern. Thus, it might reduce the redundancy when the request diversity increases and vice versa. In our proposed *ColabCache* architecture, $\overrightarrow{F_m(t)}$ and $\overrightarrow{FQ_m(t)}$ are sequentially fed to the LSTM layer as shown in Figure 5.3 which is then connected to a FC layer. The other remaining input parameters of d_t is fed to another FC layer. The output from the above FC layers is combined using a *MergeNet* which is then fed to another FC layer. The final layer outputs the BS (MEC server) to store the requested content. The softmax layer consists of $m + 1$ possible outputs (m for the number of BS in the cluster and one *No_Cache*). Here, m is a hyper-parameter that can be changed for different scenarios. We set m as seven for our experimental scenario in this work.

For *No_Cache* output, the content would not be cached into any BS as the popularity score of the requested video segment is significantly less and will be served directly to the end-users. The *critic* network neural architecture is similar to that of the *actor* network, with the difference being that it takes an additional input, π_θ along with other input parameters as explained in the *actor* network. The *critic* network gives the action policy's expected future reward, V_η, as output. *Adam* optimizer is used for training *ColabCache.*

5.3.2.2 Caching Algorithm

The caching policy for the proposed *ColabCache* is governed by the Priority Score (P_{ij}^{bm}) of video segments with respect to BSs in the cluster given by the Equation (5.3). Algorithm 6 describes the caching policy for the proposed model. Let us assume that a user (U) requests a video segment c_{ij}^b from its home BS (m). If c_{ij}^b is present in the cache of m, it is directly serviced to the user (U). Otherwise, the requested content c_{ij}^b is searched in the cache of neighbour MEC (z). If present, it is sent to U. If c_{ij}^b is not present in any BS in the cluster, it is fetched from the CCS. When a local cache miss occurs, the decision of whether to cache and which BS to cache the content is taken by the proposed *ColabCache. ColabCache* takes features mentioned in Section 5.3.2.1 as input and outputs the caching decision. The output (say, o) can either be (i) *No_Cache* denoting that the content will not be cached in the cluster or (ii) the Target BS in the cluster to cache the requested content. In the case of (i), the content is not cached and directly streamed to the user. But in the case of (ii) P_{ij}^{bo} is measured, and if MEC server o has space, then the content is stored in o. If there is no space available in o, then the content having the least popularity score is evicted to make space for the requested content c_{ij}^{b}. The proposed model ensures that an optimal *CHR* is obtained for the MEC servers, reducing the backhaul traffic, and access delay and improving the video quality with reduced re-buffering events.

Algorithm 6: *ColabCache* Caching Policy

```
1  Let $c_{ij}^{b}$ is the requested content at the timestamp $t$ ;
2  if ($c_{ij}^{b}$ in Local_MEC (m)) then
3      $m \xrightarrow{c_{ij}^{b}}$ User (U) ;
4  else if ($c_{ij}^{b}$ not in Local_MEC (m)) then
5      for ($z \in \mathcal{M}$) do
6          if ($c_{ij}^{b}$ in Neighbour_MEC (z)) then
7              $z \xrightarrow{c_{ij}^{b}} m$
8              AdmissionModule($c_{ij}^{b}$)
9              $m \xrightarrow{c_{ij}^{b}}$ User (U)
10 else
11     CDN $\xrightarrow{c_{ij}^{b}} m$
12     AdmissionModule($c_{ij}^{b}$)
13     $m \xrightarrow{c_{ij}^{b}}$ User (U)
14 Function AdmissionModule($c_{ij}^{b}$):
15     $o$ = ColabCache.output
16     if $o$ = No_Cache then
17         return $c_{ij}^{b}$                                  ▷ Do not Cache $c_{ij}^{b}$
18     else
19         Admit($o$, $c_{ij}^{b}$)
20         return $c_{ij}^{b}$
21 Function Admit($o$, $c_{ij}^{b}$):
22     if $K_o' \geq k_{ij}^{b}$ then
23         $c_{ij}^{b} \rightarrow$ o.cache
24     else
25         $LPS^o$ = Least Priority Score in o.cache
26         while $P_{ij}^{bo} > LPS^o$ & $K_o' < k_{ij}^{b}$ do
27             Evict($LPS^o$) ▷ Delete content from the cache with the least popular score
```

5.3.2.3 Training of ColabCache

In the network architecture of *ColabCache*, each MEC server consists of a DRL Agent. These DRL Agents act as Worker Agents in the RL-based A3C network. In each cluster we choose that MEC server as the master agent which has the least average distance from all other servers in the cluster as the Master Agent. During the simulation, worker agents at the MEC servers interact with the environment by making caching decisions and receiving rewards. They then calculate the value function and policy loss, used to calculate the individual gradient at each MEC server. The worker agents then periodically update the master agent's network with the gradient values, thus updating the weights in the master network. The worker agents then reset their network weights to the master agent.

5.3.2.4 Choice of Algorithm

The main idea behind using the A3C network is that the actor network can interact very efficiently with the environment (for example, exploration vs exploitation and sample efficiency). The critic network with the value function learns from the experience very effectively (for example, estimating reward signals). The significant advantage of A3C network is that with its multiple actor-learners, it can interact with the environment, gather experience, and then push their gradient updates to a central target network asynchronously. Our simulation results also show that the use of A3C network in our proposed work helps to perform better when compared with the RL-based model that uses DQN called *DRL-CCT* [35]. The detailed experimental results are given in the next section.

5.3.2.5 Computational Complexity

In prior DL-based works, the input and output sizes of the learning models are $\Omega(C + N)$, where C is the number of collaborating BSs and N is the total number of videos in the media library. In the proposed A3C architecture of *ColabCache*, it can be seen that the input size of both the actor and critic networks (i.e. the size of d_t) is $O(C^2)$ where C is the cluster size and that the output size of either network is $C + 1 = O(C)$. *ColabCache* is independent of N; therefore, *ColabCache's* performance will hold even when the media library grows compared to previous literature because smaller input and output sizes lead to shorter running times. Thus, the proposed work is more practical in the real world, which is independent of the continuously expanding media library (N).

5.4 Experiments and Results

An extensive set of experiments were performed to evaluate the efficacy of the proposed *ColabCache* model against various evaluation metrics such as the *CHR, backhaul traffic, access delay, perceived video quality, jitter* and *re-buffering* events. The performance of the proposed model has been compared with state-of-the-art *Heuristic-based* caching strategies such as *PoPCache* [58], *QoECache* [13], *Learning-based* work like *AViC* [25], and also *Collaborative-based* caching such as *CoCache* [33] ,and *DRL-CCT* [35] along with baseline caching strategies *FIFO, LRU,* and *LFU*.

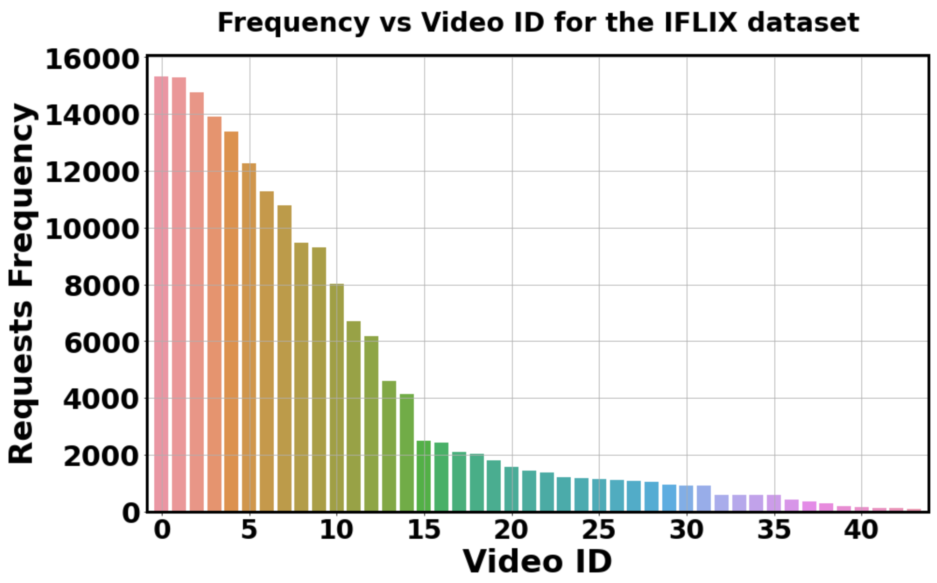


Figure 5.4: *Video Request Frequency for the Iflix dataset*

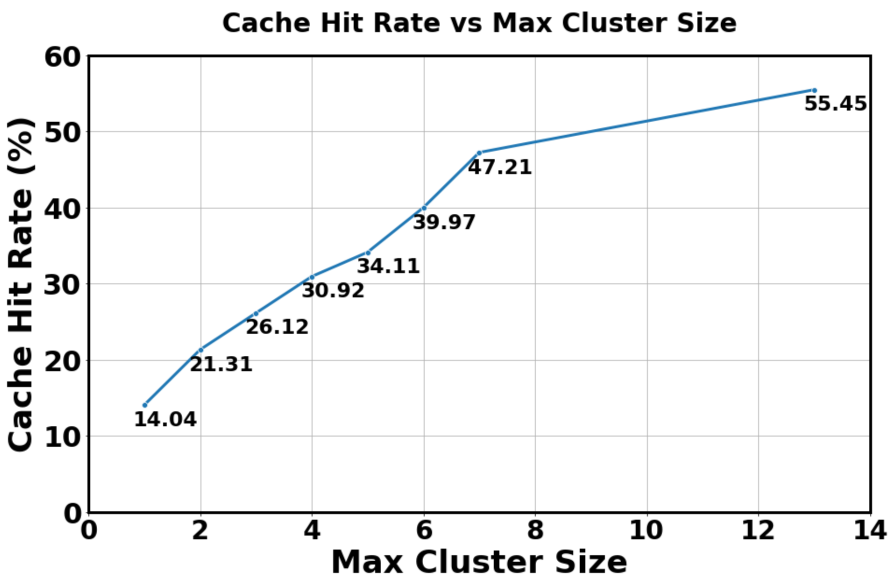


Figure 5.5: *CHR Achieved by ColabCache with respect to the Cluster Size for 2-sec Segment Duration, 256 MB Cache Size on Iflix Dataset*

5.4.1 Dataset

The proposed model has been trained extensively using both the standard *Iflix* movie streaming dataset [46] and the randomly generated dataset (*Zipf* distribution for video requests). The randomly generated dataset using *Zipf* will be referred to as *Zipf* dataset in our experimental evaluation. It follows realistic traffic models using the *Zipf* distribution which has also been supported in works such as [96] and [97]. With the lack of publicly available real-world traces of user requests, we have synthetically generated the request traces from these datasets. Traces from both the datasets consists of *timestamp, UserID, VideoID, chunkID,* and *bitrate* for a given request. Along with that, these traces assume two other important criteria to follow realistic data traffic models, namely, *watch time* (the watch time for the video watching sessions is chosen according to the typical Youtube User Engagement Graph) [102] and *session start time* (the Session start times are chosen such that the requests follow the real-world network traffic pattern). For the *Zipf* dataset, *VideoID* is chosen according to the *Zipf* distribution with $\alpha = 1.3$ and Figure 5.4 represents the video request pattern for the *Iflix* dataset. In our proposed model, DASH video streams are used. We considered segment duration of 2 and 4 seconds along with four types of video resolutions (240p, 360p, 480p and 720p). Cache sizes of 256 MB and 512 MB were considered for the MEC servers. In our proposed work *ColabCache*, we assumed that hand-off scenarios do not occur.

Figure 5.5 represents the cache hit rate achieved by *ColabCache* vs cluster size. Figure 5.5 illustrates that as we move from cluster size 6 to 7, there is an increment of 7% *CHR* with an increase in 5 minutes of simulation time. The *CHR* improves only by 8% as the cluster size further increases from 7 to 13; however, the simulation time rapidly increases by 85%. As the size of the cluster increases, the rate at which *CHR* increases slows down, and the *CHR* start to flatten. But, the simulation time, which is proportional to the operational cost of *ColabCache*, keeps increasing linearly with an increase in cluster size. Therefore, we fixed the maximum cluster size, M_0, to be seven in our proposed model.

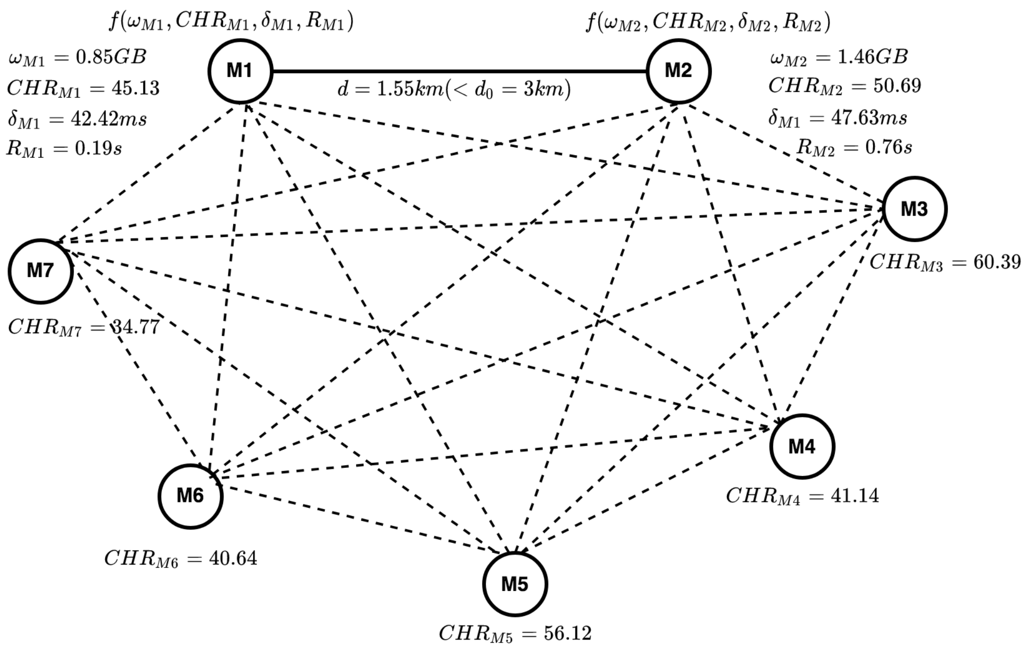


Figure 5.6: *Graphical Representation of ColabCache in a Cluster of 7 MEC Servers and their Corresponding Observed Values over a Time Period.*

Figure 5.6 illustrates a graphical representation of the proposed *ColabCache* in a clustered network consisting of seven MEC servers. The observed values concerning the individual metrics related to χ_t over a time period is presented by $f(\varrho, CHR, \delta, R)$.

5.4.2 Evaluation of ColabCache for various Segment Duration

Table 5.1: *Comparison of ColabCache Against Different Segment Sizes. The best results are shown in Red and the second best in Blue*

Random Video Request Generated Using Zipf Distribution										
Segment Duration	Evaluation Metrics	*Proposed ColabCache*	DRL-CCT [35]	CoCache [33]	FIFO	LRU	LFU	PopCache [58]	QoECache [13]	AViC [25]
2 sec	CHR	56.24	50.1	47.75	43.49	44.22	46.79	44.95	46.3	47.33
	Backhaul Traffic (GB)	13.78	15.72	16.2	17.6	17.41	16.77	17.08	16.71	16.31
	Access Delay (ms)	38.18	42.51	43.88	46.83	46.06	44.05	46.3	45.35	43.37
	Re-buffering (sec)	4.29	4.7	4.17	4.63	4.61	4.58	4.7	4.59	4.57
	PSNR (dB)	47.92	45.82	45.33	44.84	44.78	44.48	44.0	45.61	44.23
	Jitter (ms)	18.09	20.93	20.32	22.35	22.4	21.43	21.3	22.07	19.76
4 sec	CHR	56.32	49.99	47.88	43.51	44.21	47.03	45.17	46.16	46.91
	Backhaul Traffic (GB)	13.81	15.77	16.18	17.63	17.47	16.75	17.07	16.78	16.47
	Access Delay (ms)	76.05	84.63	87.31	93.35	92.01	87.61	91.97	90.59	86.84
	Re-buffering (sec)	7.99	8.94	8.55	8.74	8.98	8.78	9.19	10.01	9.26
	PSNR (dB)	47.97	46.09	45.41	45.01	44.96	44.74	44.36	45.74	44.48
	Jitter (ms)	32.78	37.57	36.93	41.58	40.61	38.48	39.2	41.6	35.42
Iflix Dataset										
Segment Duration	Evaluation Metrics	*Proposed ColabCache*	DRL-CCT [35]	CoCache [33]	FIFO	LRU	LFU	PopCache [58]	QoECache [13]	AViC [25]
2 sec	CHR	47.21	38.17	35.69	31.31	31.59	33.96	31.31	36.04	31.88
	Backhaul Traffic (GB)	16.05	18.77	19.02	20.35	20.34	19.84	20.31	19.32	20.5
	Access Delay (ms)	44.32	50.6	51.34	54.42	54.41	52.65	54.6	51.65	55.05
	Re-buffering (sec)	4.77	5.57	5.65	5.84	5.78	5.57	5.32	5.59	5.57
	PSNR (dB)	47.42	46.22	45.36	44.53	44.55	45.07	44.32	45.63	44.53
	Jitter (ms)	21.97	24.44	25.11	26.24	26.33	25.74	25.67	25.86	26.44
4 sec	CHR	47.15	37.88	35.78	31.13	31.52	34.13	31.13	36.02	32.23
	Backhaul Traffic (GB)	16.13	18.95	19.17	20.55	20.5	19.95	20.52	19.49	20.68
	Access Delay (ms)	88.95	102.1	103.63	109.96	109.32	105.81	110.16	104.01	111.02
	Re-buffering (sec)	8.97	10.54	10.67	11.4	11.09	11.05	10.28	10.96	11.6
	PSNR (dB)	48.76	46.3	45.62	44.63	44.77	45.28	44.54	45.72	44.78
	Jitter (ms)	43.59	47.68	48.94	51.88	51.3	50.76	49.99	50.01	52.19

Table 5.1 presents a comparative study of *ColabCache* with both heuristic as well as ML-based caching strategies for 2 and 4-sec segment duration with a MEC cache size of 256 MB. The comparison was made against popular and recent state-of-the-art works such as *PoPCache* [58], *QoECache* [13], *AViC* [25], *CoCache* [33], *DRL-CCT* [35], *FIFO, LRU* and *LFU*. It is observed that *ColabCache* outperforms all the caching strategies concerning *CHR*, *backhaul traffic*, *access delay*, *video quality (PSNR)*, *jitter* and *re-buffering* for both

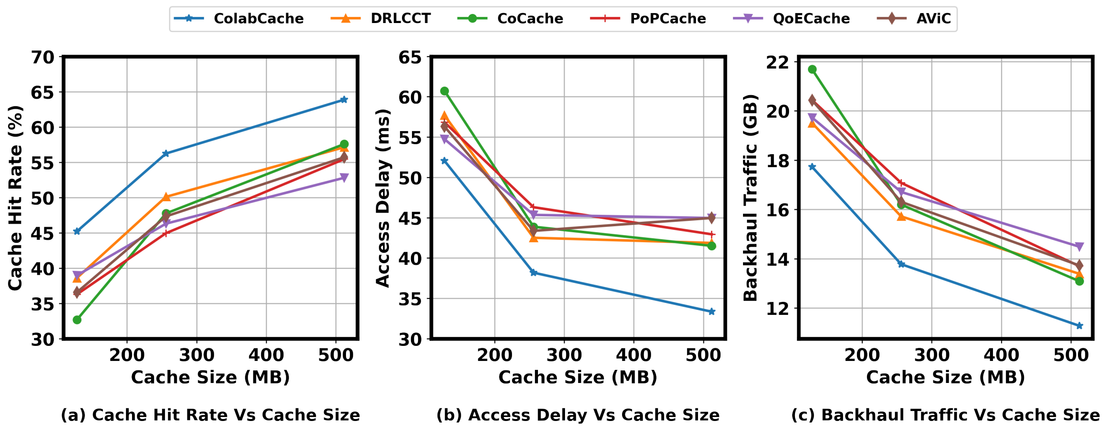
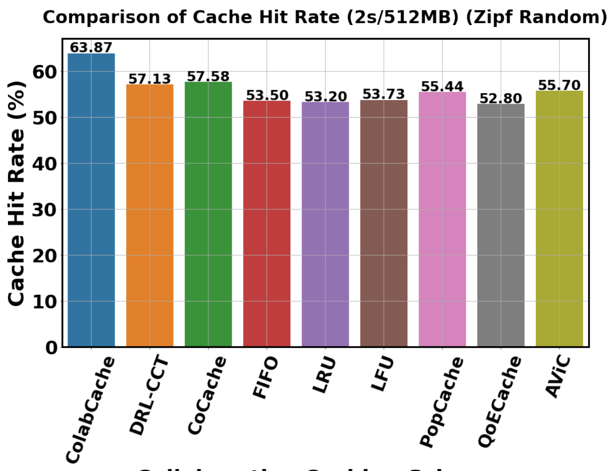


Figure 5.7: *Performance of Proposed ColabCache Against Various MEC Cache Size (MB)*

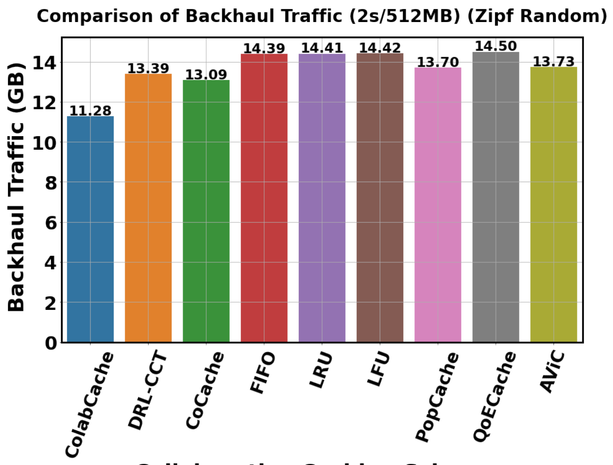
Iflix and *Zipf* dataset. The *CHR* for 2-sec segment duration with a cache size of 256 MB is better by almost 12% for *Zipf* and by 23% for the *Iflix* dataset. The increase in *CHR* has led to a substantial reduction in the *backhaul traffic* and *access delay* by 12% and 15% for the *Zipf* dataset and by almost 14% and 17% for the *Iflix* dataset respectively. The re-buffering events have improved marginally, providing a stutter-free viewing experience with an enhanced video quality of almost 4.6% and the jitter value reducing by almost 14%. The results obtained for the 512 MB cache size with a segment duration of 2 sec for both datasets are depicted in Figure 5.8 and 5.9.

5.4.3 Cache Hit Rate (CHR)

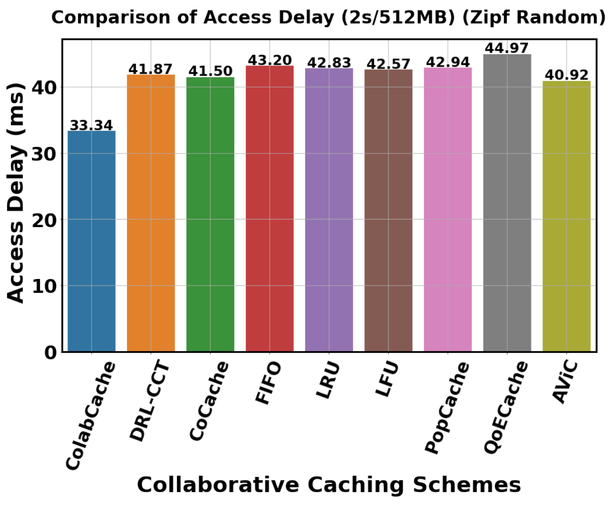
The caching policy defined by Algorithm 2 improves the overall *CHR* as more requests are serviced from the local or the neighbour MEC cache instead of the CCS. As presented in Table 5.1, the *CHR* for *Zipf* dataset increased by almost 12% for 256 MB cache size when compared with the next best-performing strategy *DRL-CCT* [35]. Similarly, for the *Iflix* dataset, there is a rise of 23% *CHR* when compared with *DRL-CCT*. For the 2-sec segment duration and 512 MB cache size, the *CHR* performance for both the dataset is illustrated in Figure 5.8(a) and 5.9(a). Intuitively, *ColabCache* outperforms the previous heuristic-based caching strategies because DL-based strategies can better adapt to the diverse request patterns received by MEC servers. *ColabCache* uses a broader variety of input features



(a) *Cache Hit Rate*



(b) *Backhaul Traffic (GB)*



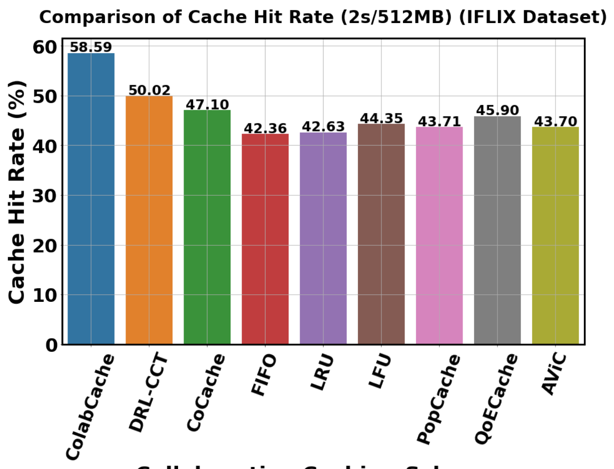
(c) *Access Delay (ms)*

Figure 5.8: *Performance of the Proposed ColabCache against Cache Hit Rate, Backhaul Traffic and Access Delay for Random Zipf Dataset*

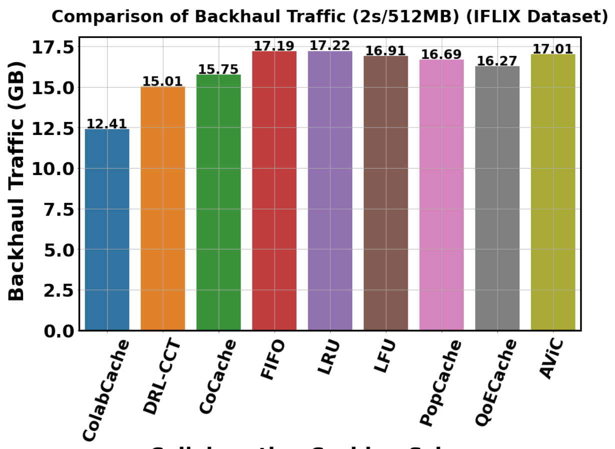
which allows the model to learn a better caching strategy and hence, performs better than the DL-based strategy *DRL-CCT*. Figure 5.7(a) illustrates the fact that as the MEC cache size increases, *CHR* also increases considerably. Cache sizes of 128, 256 and 512 MB are considered for evaluation. It is observed that as the cache size increases, the *CHR* improves substantially, as more video segments could be stored in the local cache of the MEC server.

5.4.4 Backhaul Traffic

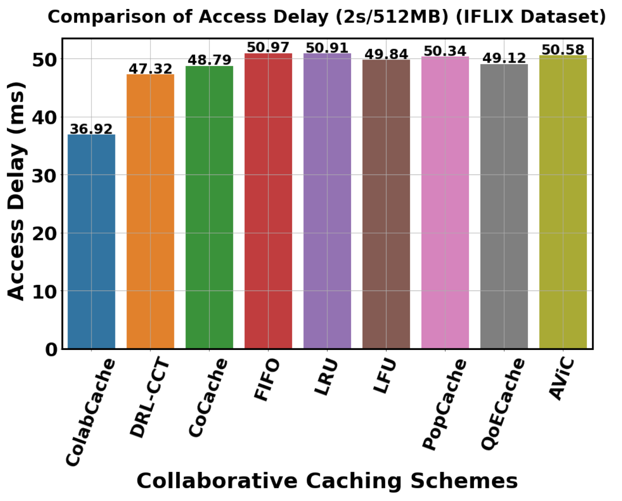
The traffic generated between the MEC server and CCS is referred to as the *backhaul traffic* denoted by Equation (5.11). As *CHR* increases, the video request is serviced more fre-



(a) *Cache Hit Rate*



(b) *Backhaul Traffic (GB)*



(c) *Access Delay (ms)*

Figure 5.9: *Performance of the Proposed ColabCache against Cache Hit Rate, Backhaul Traffic and Access Delay for Random Iflix Dataset*

quently from the local cache of the MEC cluster. Hence, fewer requests are forwarded to the CCS through the backhaul link, and the backbone's congestion reduces considerably. Table 5.1 and Figure 5.8(b) and 5.9(b) justifies the claim. The *backhaul traffic* for *Colab-Cache* improved significantly for both datasets. The backhaul traffic for the 2-sec segment duration with MEC cache size of 256 MB drops by almost 12% for the *Zipf* dataset when compared with *DRL-CCT*. In the case of *Iflix* dataset, the *backhaul traffic* reduces by 14% for 256 MB cache size. 5.8(b) and 5.9(b) represents the comparative illustration of *backhaul traffic* for cache size of 512 MB with 2-sec segment duration. There is a significant drop in the *backhaul traffic* as the cache size increases from 128 MB to 512 MB depicted by Figure

5.7(b) for the proposed *ColabCache*.

5.4.5 Access Delay

Access delay δ given by Equation (5.8) is the total time between the reception of a video request and its delivery to the end-users. In a collaborative caching environment, as M number of MEC servers collaborate to service the requested content, very few requests are forwarded to the CCS in case of a cache miss in all the MEC servers. Therefore, the delay also reduces as *CHR* increases for the proposed model. The delay for 2 and 4-sec segment duration for 256 MB cache size are presented in Table 5.1. Figure 5.8(c) and 5.9(c) portray the access delay for a 2-sec segment duration with 512 MB cache size. The delay for 2-sec segment duration and 256 MB MEC cache size reduces by 10% and 12% for the *Zipf* and *Iflix* respectively. It is observed that the access delay also reduces as cache size increases which is depicted in Figure 5.7(c).

5.4.6 Video Quality and Jitter

To measure the quality of video content, the PSNR metric is used, denoted by Equation (5.5). Table 5.1 presents a comparison of video quality for *ColabCache* with respect to both heuristics and ML-based strategies. The PSNR improved by almost 5% when compared with *DRL-CCT* for 2-sec segment duration with a MEC cache size of 256 MB for the *Zipf*. As for the *Iflix* dataset, the PSNR improved by 3%. The lower re-buffering and access delay allow streaming at higher bitrates, thus improving overall video quality.

Jitter $\mathfrak{J}$ measures the variations of latency while fetching consecutive segments denoted by Equation (5.6). With reduced congestion at the backhaul links, there is substantial improvement in the jitter value for the proposed collaborative caching model. The jitter value $\mathfrak{J}$ for 2 and 4-sec segment duration for a MEC cache size of 256 MB is tabulated in Table 5.1. The jitter value reduced by almost 12% for the *Zipf* dataset and by 10% for the *Iflix* dataset.

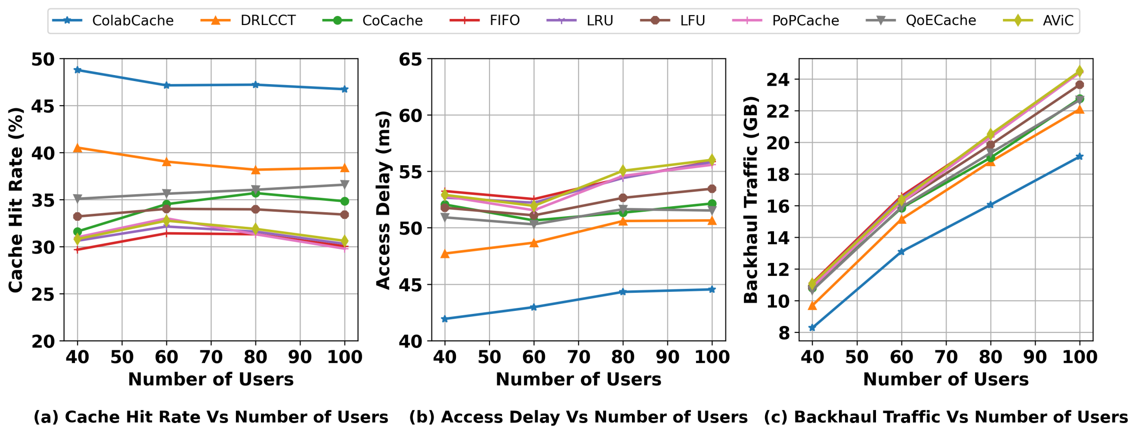


Figure 5.10: *Performance of Proposed ColabCache Against Different Number of Users*

5.4.7 Re-buffering

The performance of *ColabCache* for re-buffering is presented in Table 5.1. *ColabCache* outperformed all the existing caching schemes for various combinations of segment duration and MEC cache size. The reduction of re-buffering can be attributed to the decrease in access delay because lower access times lead to faster content fetches and, thus, decrease the total re-buffering time. With re-buffering significantly better than other strategies, the end-users can enjoy a stutter-free viewing experience. The re-buffering events were observed for both the *Zipf* and *Iflix* dataset. The proposed model, along with improving the caching metrics, also focused on providing a stutter-free viewing experience as the *CHR* increases and *backhaul traffic* and *access delay* reduced substantially.

5.4.8 Impact of Total Number of Users

The results in Table 5.1 and Figure 5.8 and 5.9 correspond to 80 users streaming content in the network. To measure the impact of different numbers of users streaming simultaneously at time t, *ColabCache* is compared against metrics such *CHR, backhaul traffic* and *access delay.* Figure 5.10 illustrates the performance of the proposed model when the total number of users varies from 40 to 100 for 2-sec segment duration and 256 MB cache size on the *Iflix* dataset. The *CHR* reduces slightly for *ColabCache* as the number of users increases, and it tends to flatten from 80 to 100. This reduction in *CHR* is expected because the

Table 5.2: *Comparison of ColabCache With and Without Clustering*

Random Video Request Generated Using Zipf Distribution						
Segment Duration	MEC Cache Size	*Proposed ColabCache*	CHR (%)	Backhaul Traffic (GB)	Access Delay (ms)	Re-buffer (sec)
2 SEC	256 MB	With Clustering	56.24	13.78	38.18	4.29
		Without Clustering	53.48	14.34	41.09	5.81
Iflix Dataset						
Segment Duration	MEC Cache Size	*Proposed ColabCache*	CHR (%)	Backhaul Traffic (GB)	Access Delay (ms)	Re-buffer (sec)
2 SEC	256 MB	With Clustering	47.21	16.05	44.32	4.77
		Without Clustering	40.37	17.89	49.90	6.27

number and diversity of requests are increasing with the increase in users, but the cache size remains fixed. But the decrease is not significant as *ColabCache* adapts to this change. Thus, the access delay and backhaul traffic increase as users increases. The access delay and backhaul traffic for other strategies are significantly higher compared to *ColabCache* as the number of users increases. The RL agents of *ColabCache* efficiently explore and exploits the solution space for caching contents into the cache of collaborating MEC servers. Therefore, *ColabCache* handles the dynamicity arising out of variations in request patterns and maintains a stable and improved *CHR* with the increasing number of users.

5.4.9 Ablation Study

In *ColabCache*, geographically nearby MEC servers collaborate to learn the best caching policy suitable for that geographical region. Clustering the MEC servers according to their geographical proximity helps improve the caching efficiency for collaborative caching as it handles the diversities arising out of the geographical dimension. Each cluster learns the caching policy most suitable for the region it is located in, thus, leveraging the geographical locality of video request patterns. Table 5.2 presents a comparative study of *ColabCache* with and without clustering for 2-sec segment duration with a MEC cache size of 256 MB for both the datasets. The *CHR* for *ColabCache* with clustering improved substantially for both the *Zipf* dataset and *Iflix* dataset. Clustering limits the sample space even when the network grows. The increase in *CHR* of *ColabCache* with clustering also led to the drop in backhaul traffic and access delay, along with re-buffering events for both datasets. The proposed model with clustering handles the diverse video request patterns arriving at the BS.

5.4.10 Popularity Vs Redundancy

Figure 5.11 shows how the redundancy of video content varies with its popularity in our proposed caching policy. Redundancy is the measure of the average number of copies of the content that was found in the cluster. The observed correlation between redundancy and popularity is 84.31%. The high correlation value implies that as the content popularity increases, *ColabCache* increases the redundancy of video content by storing the content in multiple BSs within the cluster. And when the popularity decreases, *ColabCache* reduces the redundancy and reaches zero for highly unpopular content. Low redundancy of video content implies that the content is not stored in multiple locations in the cluster. As a result, it leads to more free space in the caches of BSs within the cluster, allowing other (more popular) content to take its space. Therefore, by reducing the redundancy for less popular video content, *ColabCache* creates space for more popular content, thus increasing the *cache hit rate*. Furthermore, if highly popular video content (Say content c) were not stored redundantly at multiple BSs within the cluster, all requests for the content c would have to be satisfied by a single BS within the cluster. The load at that BS will increase, leading to network congestion at the fronthaul links between the BSs. Hence, by storing highly popular video contents at multiple BSs within the cluster, *ColabCache* reduces both the fronthaul traffic and load on the BSs, which reduces the *Access Delay*.

5.5 Summary

In this chapter, we propose a Collaborative-based caching mechanism *ColabCache* in a clustered MEC mesh network using RL. The clustering of MEC servers is performed based

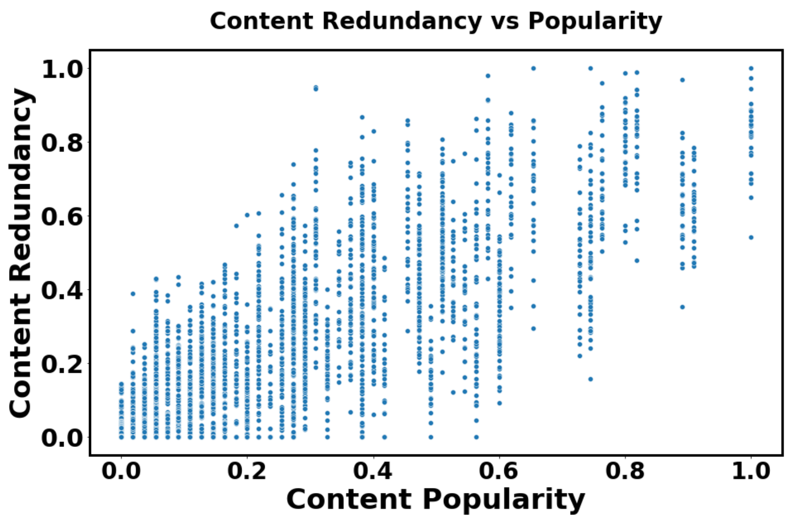


Figure 5.11: *Scatter Plot of Normalized Redundancy and Popularity*

on geographical locations. Collaboration is carried out among $'M'$ MEC servers to service a given video request that arrives at a BS. If the requested content is present in the local MEC server, it is sent directly to the user; else it is brought from the neighbouring MEC server. Whenever the requested content is not present in the clustered MEC network, the content is fetched from the CCS and sent to the user. The decision on caching a particular requested content into the cache of a MEC server is based on the measured Priority Score of the content. The proposed model has been extensively evaluated using the *Zipf* and *Iflix* dataset. To measure the efficacy of the proposed model, *ColabCache* has been compared with various state-of-the-art caching strategies against CHR, backhaul traffic, access delay and re-buffering.

Although, the proposed collaborative caching strategy performs well with heterogeneous end users' demands and requests, it is a centralized approach and may burden the MEC servers if number of end-users becomes high. To make the processing decentralized and a bit user-specific, in the final contributory chapter, we focus on developing a federated-based learning caching strategy, where training is performed on the UEs instead of the central server. Along with federated caching we also explore on the users' viewing experience for an adaptive video streaming session.

"Many of life's failures are people who did not realize how close they were to success when they gave up."

~Thomas A. Edison

6

Federated Caching and Prediction Model for Content Delivery

There are two contributions in this final contributory chapter. In the first part, a hierarchical Federated RL-based Content Caching strategy is presented. In this work, for the first time, a A3C DRL network has been trained in a Federated way for making caching decisions at the edge server. The proposed model offers a scalable solution by transferring the training process to the UEs instead of centrally at the edge server. Further, the second part of this chapter proposes a deep learning based prediction model for adaptive video streaming. In this work, a LSTM-DNN based on DRL has been devised to model the dynamic control rules for handling varying network conditions and end-user demands. The proposed model maximizes the overall QoE by satisfying the three QoE verticals such as overall video quality, re-buffering and video quality switches simultaneously.

6.1 Federated Learning-based Caching

In most of the existing caching systems presented in chapter 2, the centralized server collects user information and local data for training a caching model. Hence, as the number of content requests increases, the diversity in request patterns also increases, thus making the model less scalable and adaptive to the ever-changing dynamic scenarios. Caching models trained in a centralized way over-consumes the network resources during training and transmission of video requests. Federated Learning (FL) [36–38] has been introduced recently to improve the caching performance at the edge server and efficiently offload the computation task from the central server to the end-users. In FL, a model is trained at the user's end instead of the edge server, where the user data is not needed to be transferred to the servers. Although there has been considerable success in using ML techniques for caching schemes in wireless networks, learning-based caching strategies still face several challenges. *Firstly*, in wireless network edge, multimedia popularity is considered unpredictable and dynamic. The spatial-temporal fluctuation in content popularity adds a layer of complexity to content caching. It is also difficult for ML algorithms to reliably and swiftly anticipate content popularity based on user data and content retrieval history. *Secondly*, redundant items may be stored in the edge node cache, which fails to optimize the resource utilization of the global cache. Given the finite and restricted cache sizes at the various levels of edge nodes, deciding where and how to cache is not straightforward. *Finally*, scalability would be another challenge with a centralized training environment. With an increase in the number of users, the data generated at the MEC server increases exponentially. Because of the high computing and communication costs, centralized ML algorithms may struggle to handle such enormous data.

Therefore in this work, we propose a FL-based caching strategy called *FedCache* which deploys a decentralized approach for training a caching model. In *FedCache*, the edge server aggregates the trained parameters sent by various end-users and presents a global model. This caching approach offloads the central server's computation task to the participating end-users. In our proposed work, we present a Federated RL-based caching approach that focuses on the popularity of videos. Users download the initial RL model from the edge server and train the model using its local data. After each training round, the users upload the learned parameters of the trained model to the server and suggests 'n' video files to the server based on popularity. The server then aggregates the learned parameters uploaded by each user using Federated Averaging [103] and finally recommends the most popular files after completing 'N' training rounds. The primary objective of the proposed decentralized training approach using RL is to maximize the cache hit rate at the edge server and simultaneously improve the users viewing experience.

The significant contributions made in this work to achieve the above-mentioned objectives are as follows:

- In this work, we propose a novel RL-based FL model called *FedCache* for efficient network caching. In this work, Actor-Critic (A3C)-based RL architecture is used for the first time to model the caching strategy at the edge server in the federated way. *FedCache* offers a scalable solution for handling a diverse request patterns by deploying the training process to the end user's devices instead of doing it centrally at the edge server.

- In addition, a novel RL-based A3C framework has been proposed for defining state policies π_θ related to cache admission and eviction. Two Phases: I and II are trained sequentially, generating state-action policies for deciding which segments to admit and which to evict.

- Finally, a novel cache admission and eviction policy has been defined, unlike previous strategies, which mostly used simple heuristic-based approaches such as FIFO, LRU and LFU. *FedCache* measures the segment-level popularity and trains the A3C network to learn which and how many segments to be evicted from the local cache to cache a new segment and reward the action taken.

The proposed *FedCache* measures the caching performance against various metrics such as *cache hit rate, backhaul traffic* and *access delay*. A comprehensive set of experiments were performed over the *Iflix* video streaming dataset [46]. *FedCache* outperformed both heuristics and learning-based state-of-the-art strategies substantially.

6.1.1 System Overview

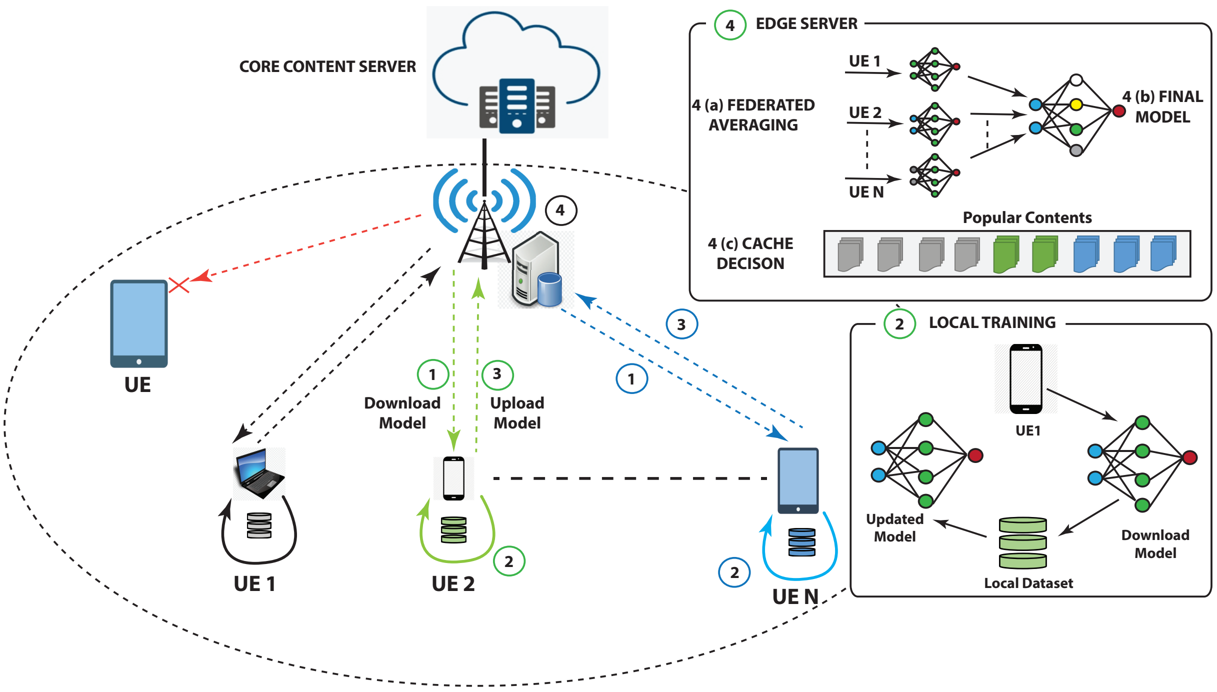


Figure 6.1: *A Hypothetical Scenario of Edge Caching using Federated Learning*

A diagrammatic overview of the system architecture used in our proposed FL-based caching at the edge node is presented in Figure 6.1. FL is defined as a ML setup that includes several distributed user devices (or edge devices) with their local data to prepare a model locally without transporting the data to the edge server. This setup does not reflect the traditional centralized ML approaches, where users' local data are brought to the central server for training purposes. It does not follow traditional distributed approaches, which often assume that data among users are uniformly distributed. The primary challenge with a centralized training environment would be scalability [36]. As the number of users expands, so does the volume of data created by those users. Because of the high computing and communication costs, centralized ML algorithms may struggle to handle such enormous data. That's where FL comes into the picture. In FL, a central or edge server does not have to train that massive data; instead, users are selected to train the caching model on the local in each federated iteration. The server will only collect and aggregate the model updates and take decisions based on that updated model.

Figure 6.1 illustrates the steps performed in a FL communication round. Each communication round consists of the following four steps:

1. **Download Model:** In this work, we assumed that u users are present within the coverage area of a given BS. q users are randomly selected from the pool of u users, which participate in the FL-based training process, consisting of 'N' communication rounds. The users then download the initial global model from the MEC of the BS.

2. **Local Training:** The next step of the proposed *FedCache* is to train the caching model based on the local data of users, as illustrated in step 2 of Figure 6.1. The dataset for every user consisting of the video request pattern is denoted by $\mathcal{D} = \langle D_1, D_2 .....D_q \rangle$. D_q denotes the video request data of q^{th} user and the length of dataset D_q is represented by d_q, where $d_q = |D_q|$. To train our RL-based model in a Federated manner, we employed A3C [99], a state-of-the-art RL technique. Our proposed FL-based RL model aims to improve the cache hit rate at the edge server to minimize the backhaul traffic and access delay. The performance of our proposed model is measured in terms of the overall reward χ_t, where χ_t is a function of cache hit rate, backhaul traffic, access delay, video quality and re-buffering events. The proposed model aims to maximize the overall reward χ_t for every training iteration and communication round.

3. **Upload Trained Model:** As represented in step 3 of Figure 6.1, the next step is to upload the trained local model ω_n^q from the pool of users u. ω^q represents the trained parameters of the n^{th} communication round for q^{th} user. In an FL-based training environment, the time for communication costs dominates the computation cost, as

mentioned in [104].

4. **Model Aggregation:** After individual users upload the models to the edge server, a new global model ω^g_{n+1} is generated by computing the weighted sum of all received local models ω^q_n as shown in Figure 6.1. The newly constructed global model ω^g_{n+1} is downloaded by the users for training in the next round $n+1$. The global model is constructed using **Federated Averaging** [37] algorithm, a common technique for calculating the weighted sum in FL.

   Let us assume that ω^g_n is the initial global model sent to q number of end-users. Next, after training at the users' end, the updated model is then uploaded to the edge server, which is denoted by Equation (6.1):

$$H^q_n := \omega^g_n - \omega^q_n \tag{6.1}$$

   The edge server, after receiving the individual updates, aggregates and generates a new global model ω^g_{n+1} for the next communication round using Federated Averaging given by Equation

   (6.2):

$$\omega^g_{n+1} = \omega^g_n + \psi H^g_n \tag{6.2}$$

   where,

$$H^g_n := \frac{1}{q}\sum_{i=1}^{q} H^q_n \tag{6.3}$$

   where, ψ is learning rate. The training continues for a total of N communication rounds. The final trained model then caches the most popular content in the edge server as depicted by Steps 4 (b) and 4 (c) of Figure 6.1. The FL-based caching model is implemented using A3C model of RL, the primary objective is maximizing the overall reward χ_t given by Equation (6.8) In *FedCache*, we measure the popularity of video segments using short, medium and long-term popularity defined in Section

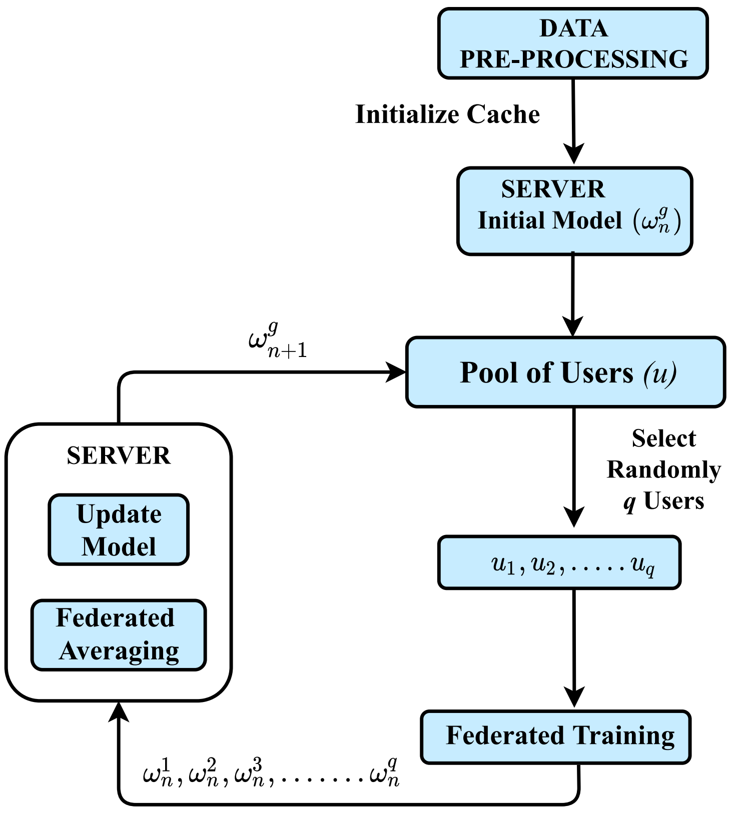

DATA PRE-PROCESSING
Initialize Cache
SERVER Initial Model (ω_n^g)
ω_{n+1}^g
Pool of Users *(u)*
Select Randomly *q* Users
SERVER
Update Model
Federated Averaging
$u_1, u_2, \ldots\ldots u_q$
Federated Training
$\omega_n^1, \omega_n^2, \omega_n^3, \ldots\ldots\ldots \omega_n^q$

Figure 6.2: *Overview of Federated Learning-based Edge Caching*

6.1.2.

6.1.2 Proposed Model

In this section, we present the related problem formulation and architecture of the *FedCache* followed by the training mechanism. Figure 6.2 presents an overview of the proposed caching mechanism using FL. In the proposed *FedCache* mechanism, a stream of video requests is pre-processed from various users. An RL-based Actor-Critic model is defined and initialized to be trained on user devices locally. At the other end, clients await the edge server to broadcast the initial model ω_n^g. From a pool of u users, present within a given BS, q users are selected randomly. The selected users $(u_1, u_2 .... u_q)$ receive the initial actor-critic model, model optimizer, and a learning algorithm broadcasted by the edge server. The q selected users locally estimate the model's update by performing the training algorithm on the initial model using the data on the device. Users then sent the locally trained parameters $\omega_n^1, \omega_n^2 ... \omega_n^q$ to the server for aggregation. The edge server then aggregates the obtained model parameters using federated averaging and finalizes the global model ω_{n+1}^g. This process continues for N communication rounds.

6.1.2.1 Problem Formulation

Let us assume that the MEC server at the edge node consists of a local cache having a capacity of K. u users are present within the purview of BS, where q users are selected randomly to participate in the federated learning-based training procedure. We assume that the participating users are computationally capable of training the proposed RL-based model. The set of DASH video streams is denoted by $\mathcal{V}$ and c_{ij}^b is the requested j^{th} segment of i^{th} video with b^{th} video quality. $b \in L$ where, $L = (240p, 480p, 720p)$ denotes video quality representation. To determine the cache hit rate at the MEC server for a video request c_{ij}^{bq} of q^{th} user, a binary decision variable y_{ij}^{bq} is defined, where $y_{ij}^{bq} \in \{0, 1\}$. If the requested content by the q^{th} user is present in the local cache of the MEC server, it is a hit and $y_{ij}^{bq} = 1$. Whereas, in case of a miss, $y_{ij}^{bq} = 0$, and the content has to be fetched from the CCS, and a decision to cache the requested content is to be made.

Hence, in a given time instance t, the local cache of the MEC server can only accommodate videos that do not exceed the total capacity K and is denoted by Equation (6.4)

$$\sum_{q \in u} \sum_{c_{ij} \in \mathcal{V}} y_{ij}^q \cdot k_{ij}^q \leq K \tag{6.4}$$

k_{ij}^q is the size of requested video segment of the q^{th} user.

The decision to cache the requested segment m_{ij}^{bq} the average popularity P_{ij}^q of the requested segment is measured given by Equation (6.5)

$$P_{ij}^q = \frac{m_{ij}^q}{\sum\limits_{ij \in \mathcal{V}} \sum\limits_{q \in u} m_{ij}^q} \tag{6.5}$$

The average popularity P_{ij}^q is the ratio of requests received for the content m_{ij} to the total number of requests received from q users at a particular time frame. Along with the average popularity value *FedCache* also considers various popularity values such as *short-term* (number of times m_{ij} requested in the last 100 time steps), *medium-term* (number of times m_{ij} requested in the previous 1000 time steps) and *long-term* (number of times m_{ij} requested in the last 10000 time steps) popularity.

The Cache Hit Rate *(CHR)* in the local cache of the MEC server is denoted by Equation (6.6) and is defined as the total number of hits occurring to the total number of requests received in a given time frame.

$$CHR = \frac{\sum\limits_{ij \in \mathcal{V}} \sum\limits_{q \in u} y_{ij}^{q} m_{ij}^{q}}{\sum\limits_{ij \in \mathcal{V}} \sum\limits_{q \in u} m_{ij}^{q}} \tag{6.6}$$

With frequent cache miss at the MEC server, more requests are forwarded to the CCS resulting in higher congestion at the backhaul links (the link between BS and CCS). Congestion at the backhaul link is measured by Equation (6.7) denoted as ϱ

$$\varrho = \sum_{ij \in \mathcal{V}} \sum_{q \in u} \cdot m_{ij}^{q} \cdot k_{ij}^{q} \cdot (1 - y_{ij}^{q}) \tag{6.7}$$

The proposed model minimizes the congestion at the backhaul link by training an efficient caching mechanism at the MEC and reducing access time for the users to retrieve the requested content. The performance of ***FedCache*** is measured against the overall reward χ_t which is a linear combination of video quality $(Q(t))$, *CHR* and re-buffering $(R(t))$ events denoted by Equation (6.8).

$$\chi_t = \beta_1 \cdot Q(t) + \beta_2 \cdot CHR(t) - \beta_3 \cdot R(t) \tag{6.8}$$

Therefore, the objective of *FedCache*, is to maximize the overall reward χ_t subject to the availability of resources S. β_1, β_2, and β_3 are hyper-parameters used for training our model.

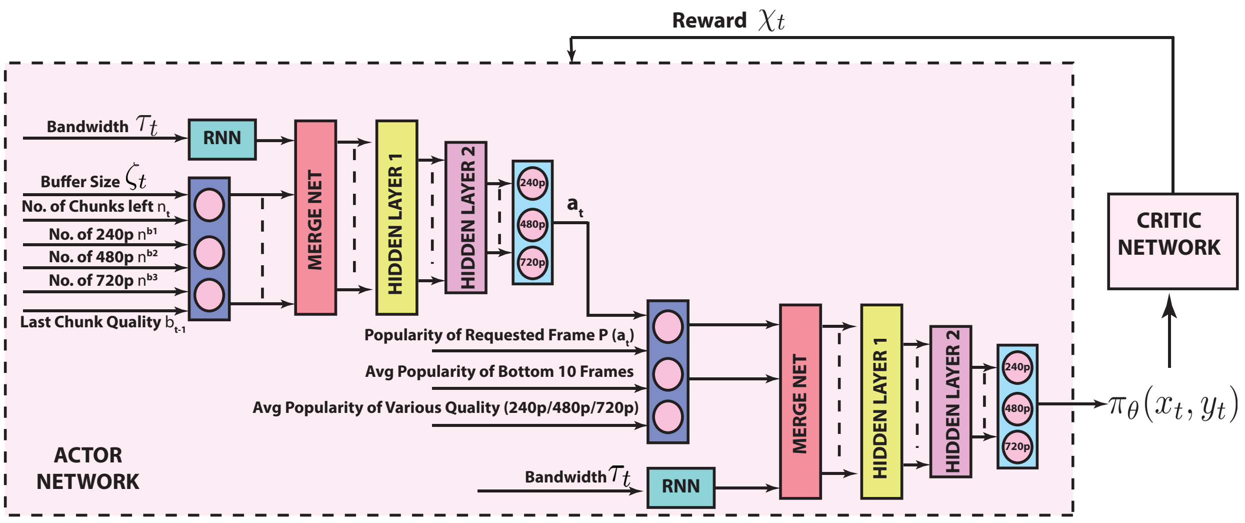


Figure 6.3: *Proposed Architecture of FedCache using A3C Network*

6.1.2.2 Architecture of FedCache

Figure 6.3 illustrates the proposed architecture of RL-based *FedCache Caching Module (CAM)* using A3C network. The architecture presents a detailed overview of the proposed RL-based actor network. The *CAM* module takes various features as input to generate policies π_θ related to the eviction of video segments of various qualities (240p, 480p and 720p). The *CAM* consists of two phases: *Phase I* outputs the quality of the video (240p, 480p and 720p) (a_t) to be sent to the user. *Phase II* determines the quality and number of such quality segments ($\pi_\theta(x_t, y_t)$) to be evicted from the local cache of MEC server. The inputs and outputs related to Phase I and II is mentioned below:

- Phase I of actor-network takes state inputs $s_{t1} = (\tau_t, \zeta_t, n_t, n^{b1}, n^{b2}, n^{b3}, b_{t-1})$ when content m^q_{ij} is requested by the q^{th} user at time t. τ_t represents the bandwidth which is fed to an RNN module. ζ_t is the current buffer size, n_t is the number of chunks left, n^{b1}, n^{b2} and n^{b3} is the number of 240p, 480p and 720p segments, and b_{t-1} is the last chunk quality. The inputs from both sequential and instantaneous are merged using the *Merge_Net*. The Merge_Net is followed by two hidden layers consisting of 64 and 32 neurons and then a FC Layer which gives the output a_t. a_t represents the video quality to be sent to the user.

- Phase II takes input state $s_{t2} = (a_t, P^1_{ij}, P^2_{ij}, P^3_{ij}, \tau_t)$. a_t is the output from Phase I,

$P_{ij}^1, P_{ij}^2, P_{ij}^3$ is the average popularity of video quality 240p, 480p and 720p respectively. The inputs are combined using a Merge_Net followed by two hidden layers (consisting of 32 neurons each) and a final FC Layer, which defines the state-action policies $\pi_\theta(x_t, a_{t1})$. a_{t1} represents the action which states how many video quality segments (240p/480p/720p) need to be evicted from the local cache of the MEC server.

Similar to the Actor-Network, the architecture of the critic network takes the same set of input states and hidden layer configuration with an additional input of $\pi_\theta(x_t, y_t)$. The critic network measures the efficiency of policy being learned by calculating the expected reward $\hat{\chi_t}$. RMSprop (Root Mean Squared Propagation) optimizer with learning rate $\alpha = 0.01$ is used. The discount factor, γ is set to 0.9.

6.1.3 Federated Learning-based Training

The proposed *FedCache* deploys an RL-based A3C agent to train the caching model. The chosen client locally estimates the model's update by performing the training algorithm on the initial model using the data present on the device. The selected user transfers its locally trained model parameters to the edge server for aggregation. The edge server then aggregates the obtained model parameters (i.e., federated averaging) and finalizes the global model. The global model actors are then used to make caching decisions i.e. Cache Admission and Eviction. In Federated Averaging, the amount of computation is controlled basically by three parameters, namely, q, the number of clients taking part in each communication round; I is the number of iterations each client makes during each round; B is the batch size of the local dataset. With Federated averaging, the compression rate is much stronger than other averaging approaches such as SGD (Stochastic Gradient Descent) [105]. This work considers a single MEC server with a cache capacity of S [256 GB or 128 GB, or 64 GB]. Users train the broadcasted model for 50000 iterations. Figure 6.4 illustrates the average reward for randomly selected five users for 50000 iterations in a given communication round. Algorithm 7 presents an overview of the training policy for the proposed work.

Algorithm 7: *FedCache* Training Policy

```
1  Initialize MEC's cache: $B_S$ ;
2  $K \rightarrow$ Local cache capacity (64,128,256 GB) ;
3  Let $m_{ij}^{q}$ is the requested content at the timestamp $t$ ;
4  $\mathcal{V}$ : Set of videos ;
5  $u$: Pool of $u$ users within the BS;
6  if ($m_{ij}^{q} \in \mathcal{V}$ in $B_S$) then
7      $B_S \xrightarrow{m_{ij}^{q}}$ User (q)
8  else
9      CS $\rightarrow B_S$
10     CachingModule($m_{ij}^{q}$)
11     $B_S \xrightarrow{m_{ij}^{q}}$ User (q)
12 Function CachingModule($m_{ij}^{q}$):
13     $a_{t1} = FedCache.output$
14     $L_{ij}$ Least Popularity Score
15     if ($P_{ij} < L_{ij}$) then
16         return $m_{ij}^{q}$                              ▷ Do not Cache $m_{ij}^{q}$
17     else
18         while $P_{ij} > L_{ij}$ & $K'_o < k_{ij}^{b}$ do
19             Evict($L_{ij}$)      ▷ Delete the content from the cache with the least popular
                score
20 Function FedCache($m_{ij}^{q}$):
21     $q =$ rand $(u)$
22     while $i \in q$ do
23         A3C.actor()
24         A3C.critic()
25     $\omega_{n+1}^{g} = \omega_{n}^{g} + \psi H_{n}^{g}$                    ▷ Federated Averaging
26     Calculate $\chi_t$
```

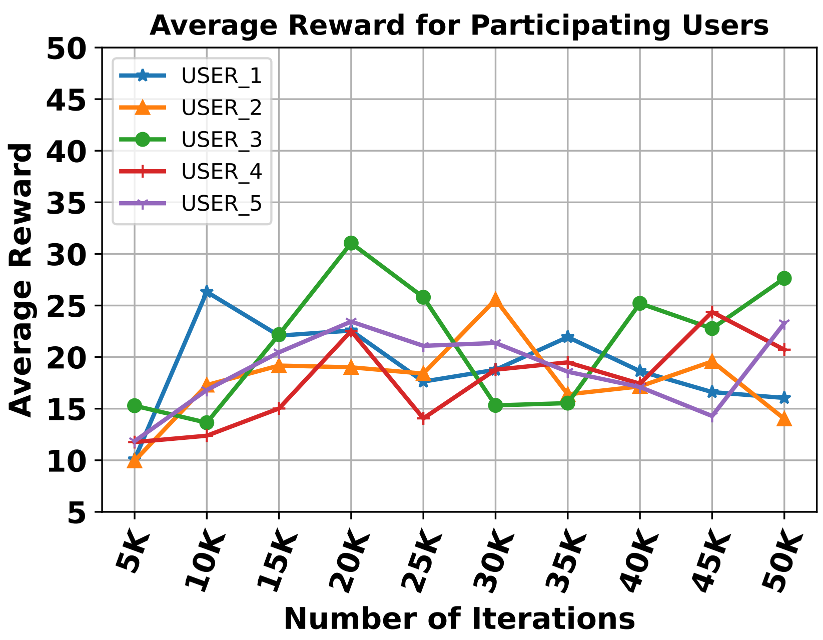


Figure 6.4: *Federated Learning-based Training among 5 different Users*

6.1.4 Experiments and Results

The performance of the proposed *FedCache* is evaluated against various evaluation metrics such as *Cache Hit Rate (CHR), backhaul traffic (GBs)* and *access delay (sec)*. We compared our work against various baseline work such as *LRU* [23], *LFU* [24] and *FIFO* along with state-of-the-art strategies like *PoPCache* [22], *QoECache* [13], *AdaptSize* [31] and *AViC* [25]. We evaluated the performance of *FedCache* using Kaggle's *Iflix* [46] Video Streaming Dataset. The dataset contains information on 17272 media files used as our media library. It also includes 542158 plays (in our case, requests) of 17272 movies made by 110640 users. The video requests dataset is modified using *Zipf* distribution [106].

In our experimental scenario, we evaluated *FedCache* against various cache sizes of 64 GB, 128 GB, and 256 GB. The number of users N within a given BS is fixed at 100. DASH video streams [43] are used with a segment duration set of 10 seconds with four video resolution layers (720p, 480p and 240p). We randomly choose 5-10 users for training in each federated iteration.

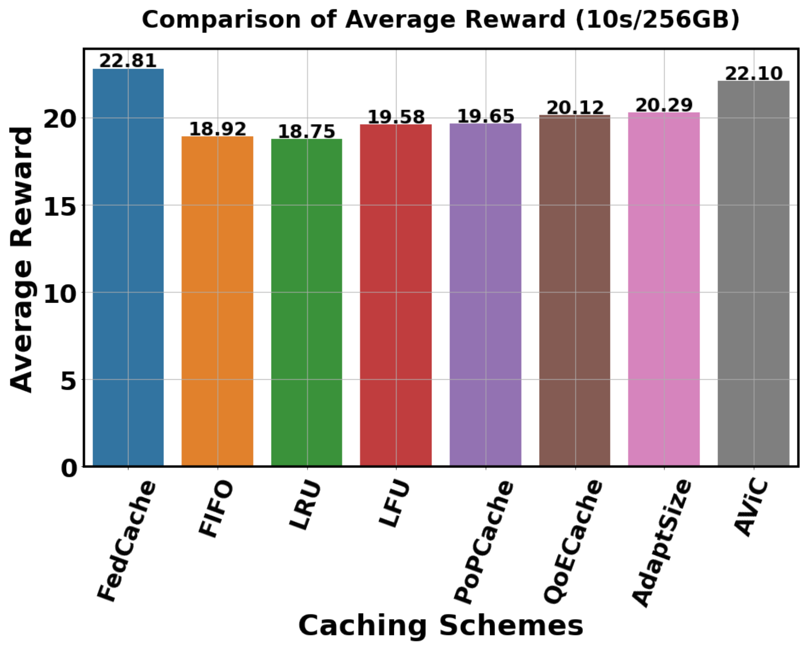


Figure 6.5: *Comparison of Average Reward for various Caching Schemes*

6.1.4.1 Evaluation Against various Cache Size

Figure 6.5 illustrates a comparative study of the Average Reward given by Equation (6.8) for various caching strategies. In this representation, the average reward is measured for a 10-second video representation with a cache size of 256 MB. *FedCache* performance is substantially improved when compared with other caching strategies, with *AViC* [25] being the next best performer in the case of overall average reward. The average reward is a combination of both ABR metrics (video quality and re-buffering events) and Caching metrics (CHR, backhaul traffic and access delay). The performance of *FedCache* further increases with an increase in MEC cache size, as it is evident from Figure 6.6. As the cache size increase from 64 to 256 GB, the local cache of MEC could accommodate more video segments which are frequently requested, thus increasing the cache hit rate. *FedCache* primary objective is increasing the cache hit rate, therefore the performance is evident from Table 6.1. Table 6.1 presents a comparative study of *FedCache* against CHR, backhaul traffic and access delay for various cache sizes of 64 and 128 GB with respect to a 10-sec video segment duration. As the cache size of the MEC server increases from 64 GB to 128 GB, the CHR increases by almost 21%. Therefore, more popular video segments could be stored as the

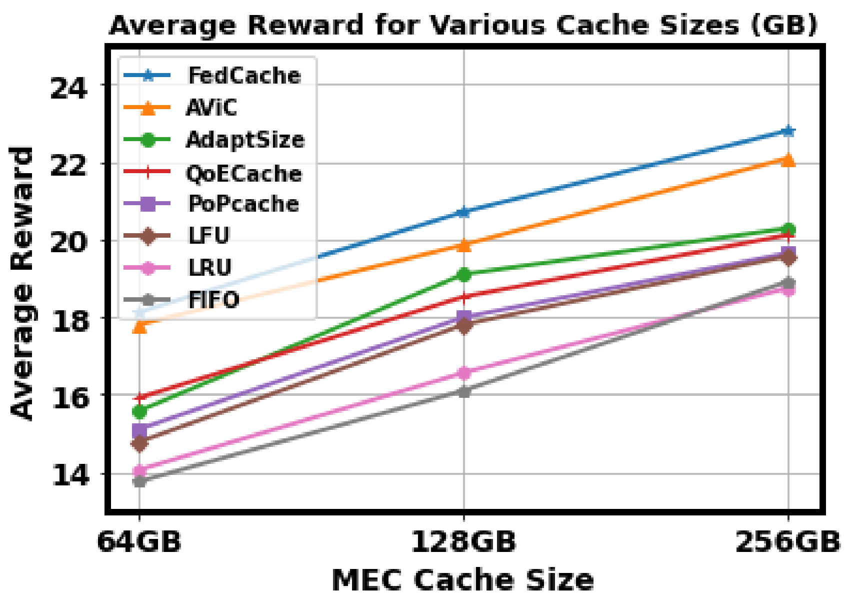


Figure 6.6: *Comparison of Average Reward Against Various MEC Cache Sizes (GB)*

Table 6.1: *Comparison of FedCache Against Various Cache Size. The best results are shown in Red and the second best in Blue*

MEC Cache Size (GB)	Evaluation Metrics	*Proposed FedCache*	FIFO	LRU	LFU	PoPCache	QoECache	AdaptSize	AviC
64 GB	CHR	35.53	18.28	17.01	22.31	23.1	24.23	21.16	33.76
	Backhaul Traffic (GB)	432.28	547.42	557.47	520.46	501.6	491.46	528.17	444.15
	Access Delay (sec)	1429	1811	1839	1722	1704	1679	1747	1468
128 GB	CHR	43.22	26.20	27.25	33.72	34.56	36.22	29.98	41.77
	Backhaul Traffic (GB)	380.67	494.36	469.10	444.01	430.21	422.80	469.10	390.39
	Access Delay (sec)	1258	1635	1612	1469	1450	1413	1552	1290

cache size increases, resulting in higher CHR. *FedCache* performance increases by almost 5% and 12% for cache size of 64 GB and 256 GB respectively when compared with the next best strategy *AViC*.

6.1.4.2 Evaluation Against Cache Hit Rate

The Cache Hit Rate (CHR) is denoted by Equation (5.4) and represents the CHR of randomly selected users from u. The proposed model outperforms all the other caching techniques in terms of cache hit rate. For smaller cache sizes, *AViC* almost performs similarly to our proposed method. But as the cache size increases, the difference in performance keeps getting prominent. *FedCache* achieved a cache hit rate of 35.53%, 43.22%, and 56.35% for cache sizes 64 GB, 128 GB, and 256 GB respectively. Figure 6.7(a) presents the CHR of the proposed model along with other caching strategies. *FedCache* outperforms both heuristics and learning-based strategies as it is more scalable and robust to the changing and diverse request patterns arising out of heterogeneous user demands.

6.1.4.3 Evaluation Against Backhaul Traffic

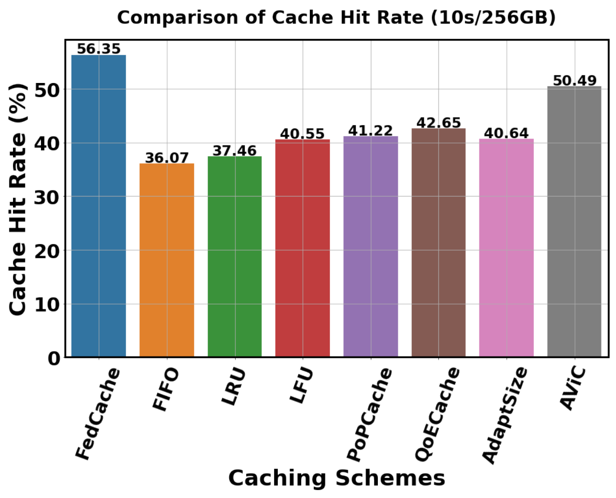
The network traffic at the backhaul link is measured by ϕ represented by Equation (6.7). Table 6.1 and Figure 6.7(c) presents a comparative study of *FedCache* against backhaul traffic which is given by Equation (6.7). The result obtained is for a 10-second video segment duration. An increase in the CHR reduces the backhaul traffic, as is evident from our experimental results. The cache hit rate is inversely proportional to the amount of backhaul traffic generated, as whenever a cache hit occurs, we serve the request directly from the cache. We must only fetch the requested segment from the core network if a cache miss occurs. The proposed approach's high cache hit rate significantly decreases the backhaul traffic generated as fewer requests are forwarded to the CCS through the backhaul links. The backhaul traffic generated in the case of *FedCache* reduces by almost 2% and 11% for 64 GB and 256 GB cache size when compared with its nearest competitor *AViC*.

6.1.4.4 Evaluation Against Access Delay

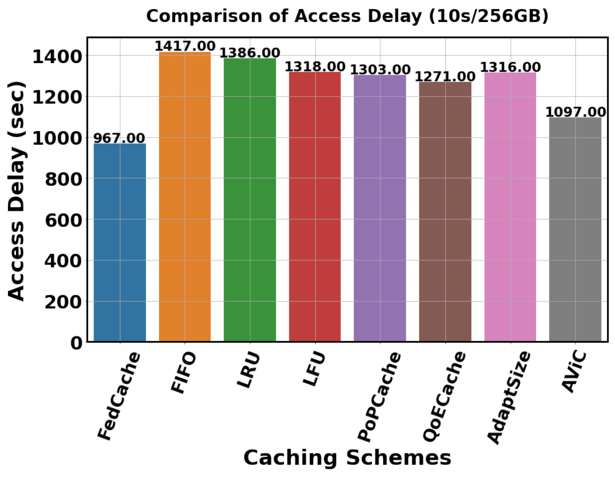
The Access Delay, δ, is the delay between the arrival of request for m_{ij}^{q} at the BS from the q^{th} user and its delivery. It is represented by Equation (6.9)

$$\delta = \sum_{ij \in \mathcal{V}} \sum_{u \in q} \cdot m_{ij}^{q} \cdot (1 - y_{ij}^{q}) \cdot \eta \tag{6.9}$$

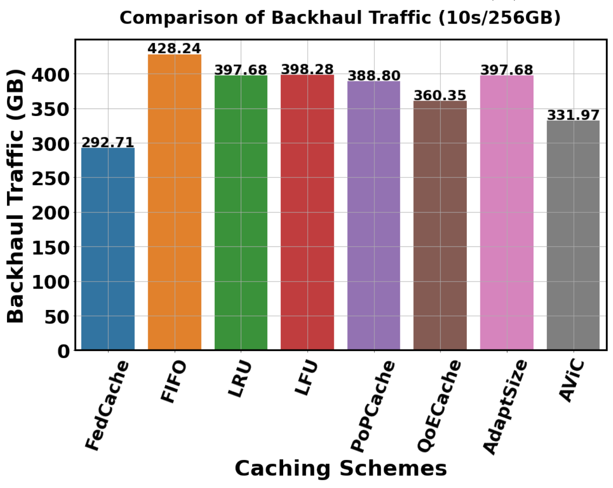
where $\eta = 0.02$ msec is the delay in the backhaul link. Access delay is also inversely proportional to CHR. The increase in the local cache hit signifies more popular content being stored in the server and fewer requests being forwarded to the CCS in case a miss occurs. This leads to lesser congestion at the backhaul link, thereby reducing the overall



(a) *Cache Hit Rate*



(b) *Access Delay (sec)*



(c) *Backhaul Traffic (GB)*

Figure 6.7: *Performance of the proposed FedCache against Cache hit, Access Delay and Backhaul Traffic for 10-sec segment with 256 GB cache size*

time to access the content for a user q. Here, access delay is considered the total of q users for n communication rounds. Access delay for the proposed *FedCache* is presented in Table 6.1 and Figure 6.7(b) for cache sizes of 64, 128 and 256 GB with segment duration of 10 seconds. Access delay for the proposed work outperforms the existing state-of-the-art heuristic and learning-based model. Access delay is reduced by 3% and 12% for 64 GB and 256 GB cache size compared with *AViC*. Therefore, the proposed *FedCache* performance with respect to caching metrics improves substantially as the size of the local cache of the MEC server increases. It is expected from our experiments that as the cache size increases, the performance of the proposed work will further increase. Thus, *FedCache* performs the training process in a decentralized way on the end-user's devices with its local data. FL-based training offloads the computation task from the central server to the end-users making the model more scalable to the increasing volumes of data.

6.2 Prediction Model for Content Delivery

Throughout this dissertation, we have explored different issues regarding caching strategies to improve *CHR* while maintaining a decent QoE. In this final phase, the second part of fourth contributory chapter explores content delivery for a video streaming session. A DNN-based model is then proposed that chooses the proper video bitrates to maximize the user's viewing experience of various QoE models. In order to enhance the overall QoE, the main objective of the adaptation strategies is to balance three important QoE verticals such as bitrate, buffer size and flickering effect. It essentially means that the best appropriate available bitrate (Maximize Video Quality) of the video segments should be selected in such a way that buffering along with the flickering effect (Video Quality Switches) should be minimized. Literature reveals that research based on adaptive video streaming has progressed and primarily grouped into two classes of strategies, namely, (i) heuristics based and (ii) ML-based approaches. For example, heuristics based approaches like [5] [73] [74] [107] and a few ML-based approaches [6] [76] [108] models adaptive bitrate ABR strategies to maintain uninterrupted high quality video viewing experience for adaptive video streaming.

The work presented in [4] has discussed a QoE management module for designing an adaptive bitrate model for video streaming. Figure 6.8 depicts an overview of a general block diagram for QoE management. The QoE management is primarily divided into three modules: *a) QoE Modeling* for a video streaming session is defined as a linear function of overall video quality, buffering events and the number of video quality switches for a given session. *b) QoE Monitoring* determines the QoE reward for every video segment fetched by the client and *c) QoE Optimization and Control* observes the QoE for the video chunks already fetched by the client, and then the QoE reward is maximized by updating the control policy. To devise such a type of QoE management, real-time, accurate and adaptable

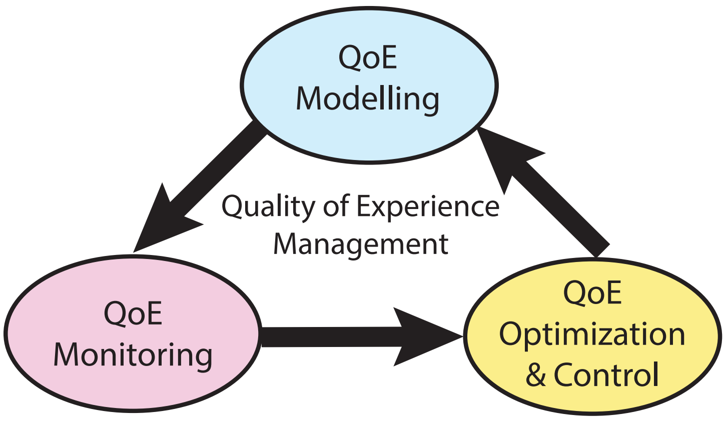


Figure 6.8: *Block Diagram for Performing QoE Management [4]*

QoE parameters are required. In recent times, machine learning ML-based strategies have been incorporated to meet these requirements, which helps to predict more accurate QoE management parameters for optimizing the QoE reward function.

In this work, an LSTM-CNN and RL-based DNN model has been devised, which is trained with a large set of input parameters. These parameters essentially model the dynamic control rules for handling varying network conditions and end-user demands. Thus, a more efficient QoE management model can be achieved. The proposed DNN architecture combined with the flavour of RL starts learning the control policy for an adaptive algorithm. It gradually keeps improving the reward signal measured in terms of QoE for the decisions made for past video segments. The proposed model was able to maximize the overall QoE by satisfying the three QoE verticals such as *overall video quality*, *re-buffering* and *video quality switches* simultaneously. In order to overcome the shortcomings of existing literature, a DNN based on LSTM and CNN *(LASH)* has been proposed. The major contributions made in this work are as follows:

- Identifying a sequential DNN that can work much more effectively than other CNN models. LSTM has the ability to remember the patterns within the input data for longer duration's. Along with the long sequence of inputs, LSTM also remembers the historical contexts of the inputs. Therefore, the model is able to predict the quality level of the videos in a much more efficient way.

- The three QoE metrics *(perceived video quality, buffering time and video quality switches)* were optimized simultaneously. Therefore, the overall QoE improved by $\approx 8.84\%$ when compared with state-of-the-art ML and heuristics-based approach.

- Moreover, the proposed model reduces the flickering artifact significantly for different QoE models.

6.2.1 Proposed Model

The proposed adaptive streaming model for improving QoE is mainly based on DASH framework discussed in chapter 2. In this section, the proposed *LASH* model has been described. This deep neural model consists of LSTM, CNN and a variant of RL architecture to achieve improved QoE for the end-users. In this work, an existing *Actor-Critic Network* or the A3C [6] has been improved by using LSTM architecture. In RL, there is an *agent* which takes an *action* on an *environment* and based on the action taken, the environment gives a reward. The underlying architecture uses two networks, *i) the Actor Network* which describes the action policies and *ii) the Critic Network* that provides a *reward* for the action taken.

A block diagram of the proposed LSTM-CNN based model for Adaptive Video Streaming over HTTP *(*LASH) architecture has been illustrated in Figure 6.9. The underlying problem for adaptive bitrate prediction generally takes two types of inputs. For instance, some inputs are sequential, and others are static or instantaneous. In this model, the *state-action* pair $\pi(s_t, a_t)$ is defined by the underlying *Actor-Critic Network.* The RL agent of the *actor network* selects an action a_t that corresponds to the bitrate of the next video segment upon receiving the input state s_t. Based on the action (a_t) taken, the critic network analyses the performance of the proposed LASH model with the help of the reward function mentioned in Equation 6.10. The detailed *LASH* architecture, along with the training phase, is discussed in the subsequent subsections.

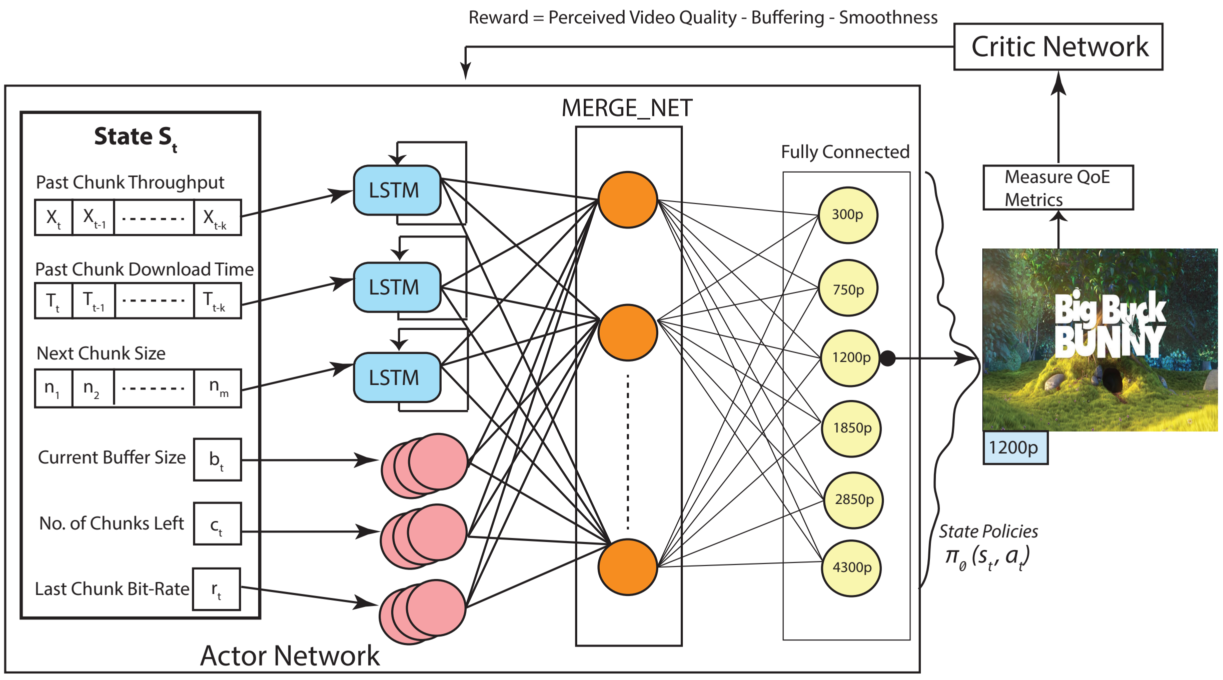


Figure 6.9: *Proposed LASH Deep Neural Architecture with RL*

6.2.1.1 Objectives of LASH ABR Model

The primary objective of the proposed *LASH* model is to select an action a_t for the input state s_t, such that the overall video quality perceived by the end-user is maximized while minimizing re-buffering events and flickering effects simultaneously. A QoE reward function has been devised in Equation (6.10) as given in [6] to measure the overall performance of a video streaming session:

$$\chi_t = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \sum^{N-1} |q(R_{n+1}) - q(R_n)| \tag{6.10}$$

where the first term $q(R_n)$ represents the video quality perceived by a user for N video chunks. μ is the re-buffering penalty for the re-buffering time T_n which is resulted from downloading n video chunks. The sub-expression $|q(R_{n+1}) - q(R_n)|$ denotes the flickering effects occurring while viewing the video segments from chunk n to chunk $n + 1$.

Accordingly, the individual terms of Equation (6.10) are required to be optimized in the following manner so that the overall reward for QoE of a given video streaming session is maximized i.e.

- Maximize the overall perceived video quality: Higher quality video streams are to be provided to the clients i.e. $max(\sum_{n=1}^{N} q(R_n))$
- Minimize re-buffering time T_n: For providing a stutter-free video viewing experience, it is required to minimize the re-buffering time i.e. $min(\sum_{n=1}^{N} T_n)$
- Minimize frequent switching between the corresponding bitrates R_{n+1} and R_n: The unnecessary video quality switches downgrade the QoE because of the flickering effect i.e. $min(\sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)|)$

where N is the total number of video segments for a given video, and n is the current video segment the client fetches. Based on the Equation (6.10), there are three different models depending on $q(R_n)$:

- $QoE_{Linear} : q(R_n) = R_n$ MPC [98] used this metric for the video quality perceived by the end-user that maps the selected bitrate R_k to $q(R_k)$. As stated by the authors in MPC, the video quality $q(.)$ perceived by the end-user depends on the device as well as the content of the video. For example, consider viewing videos of different bitrates on an HDTV and then on a mobile device. The authors claimed that the difference in viewing experience is quite noticeable when the same video with different bitrates is viewed on a HDTV. However, such differences are not observable when the video is viewed on a mobile device with a much smaller display.

- $QoE_{Log} : q(R_n) = log(R/R_{min})$ BOLA [107] used this metric to define the video quality perceived by the end-user. This logarithmic function signifies the fact that whenever there is a switch in the chunk bitrate for lower video qualities, i.e. from 0.5 Mbps to 1.5 Mbps, the gain in the perceived video quality is higher than switching between higher bitrates: for example, from 6 to 7 Mbps.

- QoE_{HD} : To define a metric for HD videos, [6] assigned scores to both the HD and non-HD videos. The function $q(R_n)$ provided a low score for the non-HD videos and a higher score for HD videos. Table 6.2 provides the exact values of HD video quality scores.

Table 6.2: *Video Quality Scores of HD Videos*

Video Bitrate (Kbps)	300	750	1200	1850	2850	4300
Video Quality Score (HD Model)	1	2	3	12	15	20

6.2.1.2 Input Space

The input space s_t considered by *LASH* after the download of each chunk t is categorized into two types. The input data space $s_t = (\hat{x_t}, \hat{\tau_t}, \hat{n_t}, \zeta_t, c_t, b_t)$ is feed to the deep neural network of *LASH*. $\hat{x_t}$ denotes the throughput of past k chunks, $\hat{\tau_t}$ is the past k chunks download time, $\hat{n_t}$ is available sizes of the next m video chunks, ζ_t is the current buffer size, c_t is the number of chunks left, and b_t is the bitrate of the last chunk which was downloaded.

- Sequential Data: In this work, few inputs are assumed as sequential, for example, past *k chunk throughput* as $\hat{x_t}$, *past k chunk download time* as $\hat{\tau_t}$ and *chunk sizes* as $\hat{n_t}$ for state inputs as they may be altered with respect to time. Since LSTM works better than 1DCNN, RNN on sequential data, the sequential state inputs are fed into the LSTM.

- Instantaneous Input Data: Inputs like *current buffer size* as ζ_t, *number of chunks left* as c_t and *last chunk bitrate* as b_t are treated as instantaneous data. Here, instantaneous data represents the exact value of ζ_t, c_t, b_t at time t.

6.2.1.3 Actor-Critic (A3C) Network

LASH uses A3C [109], a state-of-the-art *actor-critic* RL algorithm for the training. A3C consists of two networks, one being the *actor network* and the other as the *critic network*. The input to the actor network is the input space s_t, and the output is the state-action pair defined as the probability distribution over state policy $\pi(s_t, a_t)$ given by Equation (6.11) as mentioned in [6]:

$$\pi : \pi(s_t, a_t) \rightarrow [0, 1] \tag{6.11}$$

where, $\pi(s_t, a_t)$ is the probability that for the input state s_t, action a_t is taken. In real-life scenarios, many [state, action] pairs (like buffer and bandwidth) can be mapped to different bitrates of the same video segment or the same bitrate of different video segments. It is hard to design such a model with so many (s_t, a_t) pairs. So, *LASH* trains its network with several adjustable policy parameters θ [6]. The policy parameter is represented as $\pi_\theta(s_t, a_t)$. The state policies defined by the actor network is evaluated by the critic network using the reward function, which is based on the QoE of the end-user. The input to the critic network is the state-action policies $\pi_\theta(s_t, a_t)$ defined by the actor network. The critic network uses the equation defined in 6.10 to measure the QoE of a given video streaming session for the state policies generated by the actor network. The critic network aids in training the *LASH* model for improving ABR strategies. The primary goal of these agents is to maximize the aggregated overall reward (χ_t) defined in Equation (6.10). The reward χ_t is optimized for a video streaming session while simultaneously considering all the QoE metrics *(perceived video quality, buffering and smoothness).*

6.2.1.4 LASH Architecture

The proposed *LASH* model considers past $(k = 16)$ bandwidth measurement, chunk download time and video chunk sizes in the experimentation. These sequential inputs are fed to a LSTM network as illustrated in 6.9. Instantaneous inputs like *current buffer size* as ζ_t, *number of chunks left* as c_t and *last chunk bitrate* as b_t are fed to a *fully connected layer* with 128 *filters*, each of *size 4* and *stride 1*. Outputs from this first layer are concatenated in the next layer using the *merge_net* having 128 neurons. The last layer is a *fully connected layer* that uses a Softmax Function to select action a_t for state s_t in the state policies $\pi_\theta(s_t, a_t)$. Action a_t is the probability of choosing the appropriate bitrate of the next video segment from a set of available bitrates for the given video. The softmax function calculates the probability of selecting an action a_t from a list of n different actions. The critic network uses the same neural architecture as the actor network. The critic network uses a linear neuron in the last layer instead of the softmax function, which measures the QoE of a given video streaming session using Equation (6.10).

6.2.1.5 Training Phase

In policy gradient strategy [100], the primary motivation is to estimate the gradient of the expected reward (r_t). The total reward χ_t is mentioned in Equation (6.10). In the policy gradient method, an Advantage function is defined as $A^{\pi_\theta}(s, a)$ for the state policies $\pi_\theta(s_t, a_t)$ by the actor network. The advantage function represents how much better a definite action a_t in the state-action policy (s_t, a_t) at time instant t is better compared to the average actions. For a given experience of state-action pair (s_t, a_t), the advantage function $A(s_t, a_t)$ can be estimated by Equation (6.12) as described in [99]

$$A(s_t, a_t) = \chi_t + \gamma V^\pi \theta(s_{t+1}; \theta_v) - V^\pi \theta(s_t; \theta_v) \tag{6.12}$$

As mentioned in [99], the above equation represents the experience for the advantage function for the transition from state s_t to s_{t+1} the obtained reward is χ_t for the action a_t taken. θ_v represents the critic network parameters, $V^{\pi\theta}(\cdot; \theta_v)$ is the estimate of $v^{\pi\theta}(\cdot)$ and γ is the discount factor representing the importance of future rewards. $v^{\pi\theta}(\cdot)$ is the expected total reward that starts at state s and follows the policy π_θ. The critic network learns an estimate of $v^{\pi_\theta}(s)$ from the experimentally observed rewards.

- Softmax Function: The final fully connected layer of *LASH* uses the *softmax function*, which gives a probability distribution of all the actions a_t (selecting the video bitrates) of the state-action policies $\pi_\theta(s_t, a_t)$. The softmax function is defined in Equation (6.13)

$$\rho(a)_i = \frac{e^{a_i}}{\sum\limits_{j=1}^{K} e^{a_j}}, \quad \text{for i = 1,..,K} \tag{6.13}$$

  where K denotes the total number of probabilities of the actions a_t and $a = (a_1, ...a_k)$ is the input vector corresponding to the available video bitrates $(300p, 750p, 1200p, 1850p, 2850p, 4300p)$. The output of the softmax function is selecting the bitrate for the next video segment having the highest probability such that the most appropriate bitrate is selected for a corresponding state s_t.

In the proposed *LASH* architecture, both the *actor* and the *critic* network are used to train the network. While during the testing phase, only the *actor* network is used, and the *critic* network is not required for the testing scenario. In this sub-section, it has been shown how the *actor-critic* network is being used to train the proposed DNN comprising both LSTM and CNN. The applicability of the proposed model will be justified in the next section by a comprehensive set of experiments with the help of a standard dataset.

6.2.2 Experiments and Results

The efficacy of the proposed *LASH* model has been evaluated by conducting a comprehensive set of experiments based on the HSDPA [47] Norway traces. For the simulation of the proposed model, the format of HSDPA dataset is used as $[timestamp(sec), throughput(Mbps)]$. *LASH* ABR model uses the vast collection of HSDPA traces to train the *actor-critic* based DNN. After training, the trained model is deployed in the MEC server. The server delivers the most relevant quality of the video segment to the client for the requested video segment. To train the proposed *LASH* model, we considered $k = 16$ past bandwidth measurements and chunk download time to the LSTM network for the time series data. Whereas, the instantaneous current buffer size, number of chunks left and last chunk bitrate are fed to the FC layer. The actor network's learning rate is 0.0001, and for the critic network, the learning rate is 0.001. Parallel training was employed while training the proposed *LASH* model. A total of 16 agents, controlled by a central agent, were deployed for the training. The loss function used in *LASH* is MSE, which is the average of the squared difference between the calculated and predicted value. The model was trained for 50,000 iterations.

6.2.2.1 Ablation Studies

This section presents an ablation study for the proposed *LASH* model to better understand the motivation behind using LSTM instead of 1DCNN for learning the previous k instances of input features such as throughput, chunk of downloaded segments etc. We analyzed the

Table 6.3: *Comparative study of LSTM and 1DCNN*

HSDPA	LSTM		1DCNN	
Dataset	RMSE	COR.	RMSE	COR.
BUS	0.24	0.74	0.24	0.69
CAR	0.38	0.78	0.50	0.73
FERRY	0.49	0.51	0.66	0.43
TRAIN	0.29	0.71	0.38	0.64
METRO	0.31	0.53	0.34	0.45
TRAM	0.43	0.56	0.44	0.46

performance of both the LSTM and 1DCNN on the HSDPA dataset. A comparative study between LSTM and 1DCNN for prediction accuracy has been carried out using the RMSE and the Correlation metrics. RMSE measures the difference between the predicted values and the actual observed values. At the same time, correlation measures the association between the observed and the predicted variable.

To validate the introduction of LSTM into the proposed *LASH* model, we consider one of the times series data (like network bandwidth) and test it against both the sequential models: LSTM and 1DCNN. The bandwidth traces for public transportation like *Bus, Car, Train, Tram, Ferry and Metro* is then used to compare the efficiency of both LSTM and 1DCNN. A comparative study of LSTM and 1DCNN with respect to the RMSE and the correlation for both the deep models are presented in Table 6.3. It reveals that LSTM produces a better correlation than 1DCNN compared to different bandwidth traces. The RMSE score, which denotes the predicted accuracy of the network bandwidth for a given deep model, is better for LSTM than 1DCNN. LSTM captures the historical context of the time series data instead of only the last input data. For example, consider the bandwidth sequence: $6.4 \rightarrow 7.4 \rightarrow 8.4 \rightarrow ?$. The next expected output in the sequence is $9.4(x + 1.4)$. Also, consider the sequence $2.1 \rightarrow 4.2 \rightarrow 8.4 \rightarrow ?$. We would like to have $16.8(2x)$ as the next output in the sequence. The expected output is different in both sequences, although the last input was 8.4 in both cases. The recurrent nature of LSTM, along with various input gates, can obtain better results than 1DCNN.

6.2.2.2 Evaluation of Proposed LASH Model

The proposed *LASH* model has been compared with two state-of-the-art ABR techniques to assess its performance and correctness. One of the strategies is a heuristic-based approach handling the current playout buffer called *Buffer Based (BB)* [5]. The other is an RL-based Neural Network called the *Pensieve* [6]. These two very recent and state-of-the-art schemes have been chosen for comparison with the proposed scheme because *Pensieve* [6] is an ML-based scheme, and [5] is a heuristics-based scheme. It has been experimentally shown that the proposed model has outperformed both the scheme against three different QoE models, namely, QoE_{HD}*,* QoE_{Linear} *and* QoE_{Log}.

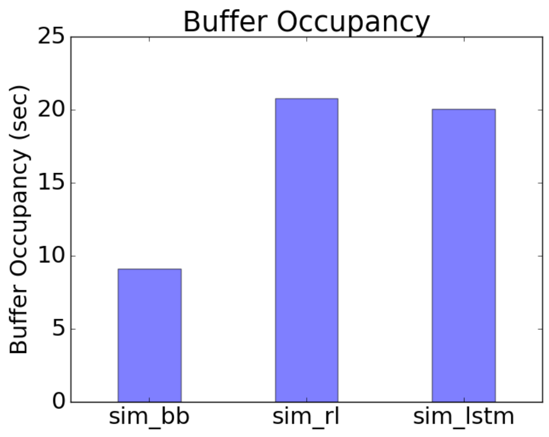
6.2.2.3 Comparison for QoE HD Reward Model

Table 6.4: *QoE_{HD} Reward for Various Transportation. The best results are shown in Red and the second best in Blue*

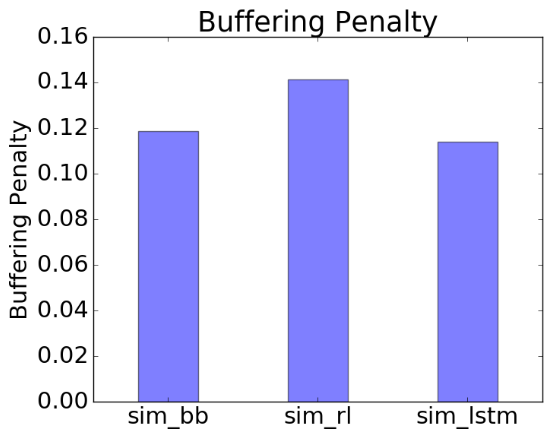
ABR Strategies	QoE_{HD} Reward in the HSDPA Dataset					
	Train	Tram	Car	Bus	Ferry	Metro
Buffer Based [5]	206	53	240.72	377	200	177
Pensieve [6]	442	123	212	386	376	332
Proposed LASH	464	123	255	505	404	354

Table 6.4 presents the total QoE reward obtained for the HD model while streaming a video travelling in various public transportation. The total reward received by the proposed *LASH* model for the individual public commutes like *train, tram, car, bus, ferry and metro* is much higher when compared with *Buffer-based* [5] and *Pensieve* [6]. *LASH* employed LSTM for sequential input data like *past k chunk throughput* ($\hat{x}_t$), *past k chunk download time* ($\hat{\tau}_t$) and *chunk size* ($\hat{n}_t$), because of which the proposed model was able to learn the dependency within the input data in a much efficient way. As a result, *LASH* can generate state-action pair $\pi_\theta(s_t, a_t)$ such that the action a_t taken for the input state s_t obtained a better and improved QoE than its counterpart. The plots given in Figure 6.10(b) and 6.10(c) reveal that the buffering penalty and penalty is less for *LASH* (sim_lstm) in comparison with the *BB* (sim_bb) [5] and *Pensieve* (sim_rl) [6] schemes. For *LASH*, the buffering penalty is

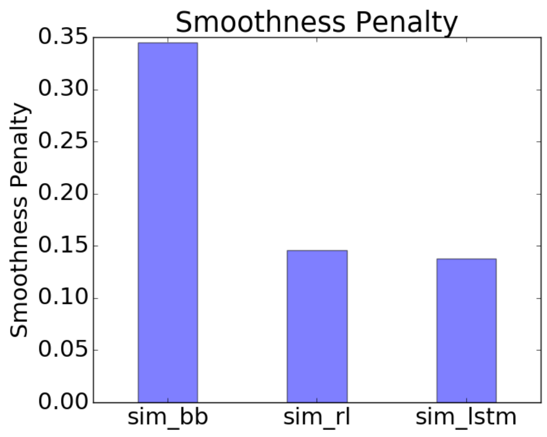
around 0.1140 when compared with *BB* ≈ 0.1185 and *Pensieve* ≈ 0.1414. The smoothing penalty for *LASH* was around ≈ 0.1379, which is less than *BB* ≈ 0.34464 and *Pensieve* ≈ 0.14571596. Further, it is evident from Figure 6.10(a) that both *LASH* and *Pensieve* maintained a safe buffer capacity to avoid video stuttering artifacts.



(a) *Buffer Occupancy*



(b) *Buffering Penalty*



(c) *Smoothing Penalty*

Figure 6.10: *Comparing LASH on QoE metrics with BB [5] and Pensieve [6] QoE$_{HD}$ model*

6.2.2.4 Comparison for QoE Linear Reward Model

A higher QoE reward Table shown in 6.5 implies that the proposed *LASH* model maintained an improved balance between perceived video quality, buffering events and frequent switching between video qualities. The plots depicted in Figure 6.11(b) and 6.11(c) show that the *LASH* model reduces the buffering and smoothing penalty to improve the overall QoE reward. The total QoE reward obtained while streaming a video traveling in different

Table 6.5: *QoE_{Linear} Reward for Various Transportation. The best results are shown in Red and the second best in Blue*

ABR Strategies	QoE_{Linear} Reward in the HSDPA Dataset					
	Train	Tram	Car	Bus	Ferry	Metro
Buffer Based [5]	2.89	9.54	45.52	46.65	68.80	11.62
Pensieve [6]	7.88	6.67	20.87	44.25	65.50	4.11
Proposed LASH	17.25	13.65	48.30	49.15	89	14.35

public commutes is depicted in Table 6.5. It can be observed that the reward obtained for *LASH* outperforms both the heuristics based approach *BB* and machine learning based approach *Pensieve* for all the cases.

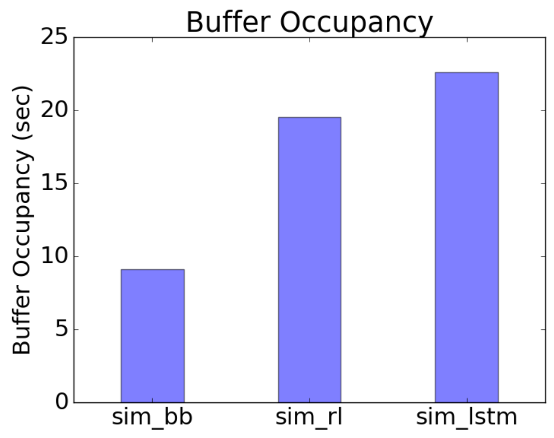
6.2.3 Comparison for QoE Log Reward Model

Figure 6.12 portrays the individual values of the QoE metrics like *buffer occupancy, smoothing penalty and buffering penalty* obtained for a given video streaming session. The plots display that the proposed *LASH* (sim_lstm) model can maintain a better buffer occupancy level when compared with *BB* (sim_bb) and *Pensieve* (sim_rl). With a safe buffer occupancy level, there is less smoothing penalty $\approx$ 0.2004 than *BB* $\approx$ 0.344 and *Pensieve* $\approx$ 0.2802. *LASH* shows almost similar buffering penalty when compared with *Pensieve* but performs better when compared with *BB*.

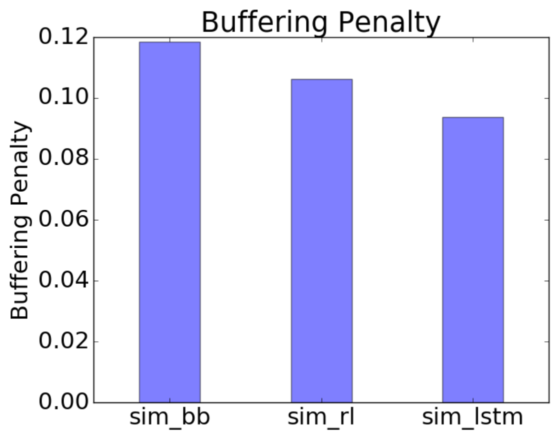
Table 6.6: *QoE_{Log} Reward for Various Transportation. The best results are shown in Red and the second best in Blue*

ABR Strategies	QoE_{Log} Reward in the HSDPA Dataset					
	Train	Tram	Car	Bus	Ferry	Metro
Buffer Based [5]	49.40	27.35	50.40	46.65	50.30	33.04
Pensieve [6]	43.43	32.20	48.12	45.92	52.51	33.95
Proposed LASH	47.04	36.72	53.24	58.09	59.73	35.39

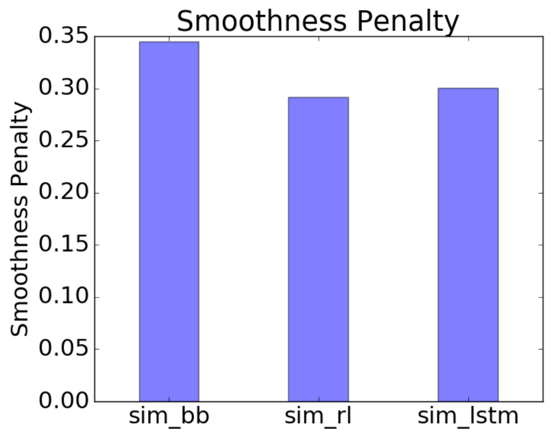
A comparative study of *LASH* with *BB* and *RL* for the total QoE obtained while streaming a video in various public transportation is depicted in Table 6.6. It can be observed that the proposed *LASH* model outperforms both the strategies for the QoE_{Log} model for different scenarios.



(a) *Buffer Occupancy*



(b) *Buffering Penalty*

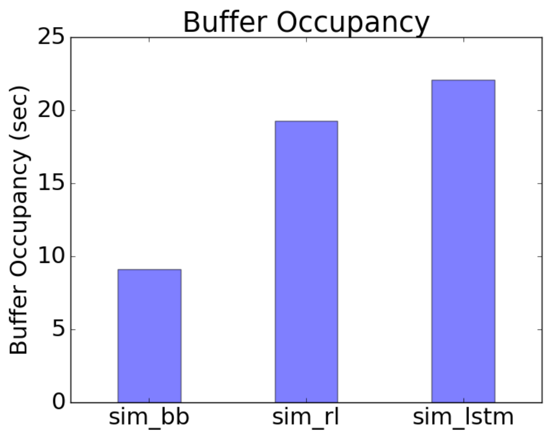


(c) *Smoothing Penalty*

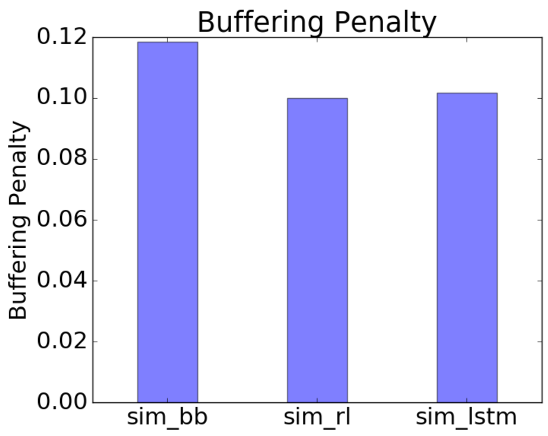
Figure 6.11: *Comparing LASH on QoE metrics with BB [5] and Pensieve [6] QoE_{Linear} model*

6.2.3.1 Comparative study between various QoE models

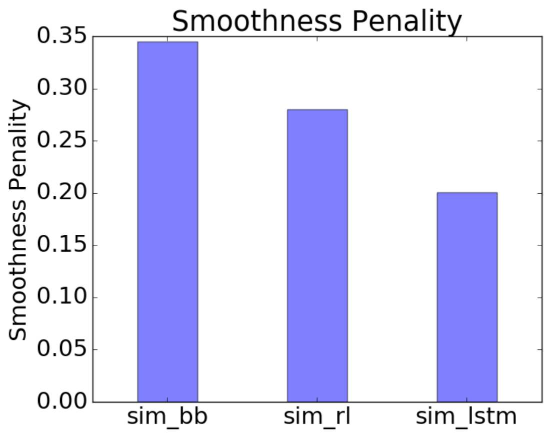
Table 6.7 portrays the comparative results obtained for various QoE models like QoE_{HD}, QoE_{Linear} and QoE_{Log} as discussed in earlier sections. The results in Table 6.7 show the overall aggregated QoE reward obtained for the HSDPA dataset. The QoE reward, a function of perceived video quality, buffering and frequent switching between the bitrates of the video segments, defines the users' satisfaction with a given video streaming session. It is evident from the results presented in Table 6.7 that the proposed *LASH* model outperforms both the heuristics-based approach *BB* [5] and ML-based approach *Pensieve* [6]. The proposed *LASH* obtained better QoE rewards for the three models QoE_{HD}, QoE_{Linear} and QoE_{Log}. In the QoE_{HD} model, there is an improvement of $\approx$ 3.02% over *Pensieve*



(a) *Buffer Occupancy*



(b) *Buffering Penalty*



(c) *Smoothing Penalty*

Figure 6.12: *Comparing LASH on QoE metrics with BB [5] and Pensieve [6] QoE_{Log} model*

for the overall reward obtained. As for the QoE_{Log} model, there was an improvement of over $\approx 8.84\%$ compared with *Pensieve*. In the QoE_{Linear} model, the QoE reward metrics obtained for *LASH* were better than *Pensieve* by $\approx 7.5\%$.

6.3 Summary

In the final contributory chapter of this dissertation, two works are proposed. In the first part of this chapter, we proposed *FedCache* an RL-based FL Caching mechanism at the MEC. Learning-based methods are intelligent and dynamic but not scalable to diverse request patterns. The proposed approach trains the model locally on the client side instead of the server centrally. After training their model, the users upload their trained parameters

Table 6.7: *Comparative Results for various QoE models. The best results are shown in Red and the second best in Blue*

QoE Model	Adaptive Bitrate Strategies		
	Buffer Based [5]	Pensieve [6]	Proposed LASH
QoE HD	140.15	236.12	243.27
QoE Linear	30.04	30.37	32.65
QoE Log	30.04	31.45	34.23

to the server, aggregating the learned parameters using Federated Averaging. *FedCache* introduces intelligence to the caching system with an actor-critic network-based learning model, which helps make intelligent caching decisions. On the other hand, it solves scalability by transferring the training to the users' devices. Extensive experiments were performed on the *Iflix* dataset. The performance of the proposed work was measured against various state-of-the-art caching mechanisms such as *FIFO, LRU, LFU, PoPCache, QoECache, AdaptSize* and *AviC*. *FedCache* outperforms these existing strategies for various evaluation metrics such as CHR, backhaul traffic and access delay.

In the second part of this chapter, we propose a content prediction model for adaptive video streaming. an ABR model named *LASH* is proposed, a LSTM-CNN-based deep neural model for adaptive video streaming. It has been demonstrated that unlike the heuristic-based approaches and other recently introduced machine learning-based approaches, the proposed model can predict a more accurate bitrate of video segments for the end-users to maximize the overall QoE for an entire video streaming session. Using LSTM in the proposed architecture enabled the model to understand the sequential feature more accurately. Thus, it improves the underlying actor-critic network of *LASH* by defining a state policy for the bitrate of the video segments to maximize the overall video quality. It has been experimentally shown that the proposed *LASH* model has outperformed both heuristic and machine learning-based state-of-the-art approaches concerning total QoE reward. Simulation results over the standard HSDPA Norway dataset reveal that the proposed model can minimize the buffering penalty along with frequent switching between the bitrates of the video segments while delivering an improved perceived video quality to the end-user for an entire session. Thus, the proposed *LASH* model optimizes all the QoE verticals, namely, *overall video quality*, *buffering events* and the *flickering effect* simultaneously.

The next chapter concludes the thesis by briefly summarizing the work presented in this dissertation and discussing the future research works.

"Obstacles are those frightful things you see when you take your eyes off your goal."

~Henry Ford

7

Conclusions and Future Perspectives

In this dissertation, DL approaches for efficient MEC concerning content caching and delivery while video streaming is presented. A summary of contributions is narrated as follows, which aims at improving the caching performance and viewing experience for video streaming at MEC.

7.1 Summary of Contributions

7.1.1 Content Aware Caching based on the Users Viewing Profile

In the first contributory chapter, a novel two-step caching model has been proposed that focuses on the users' viewing profiles at different time slots of the day. A time-slot-based hypothetical popularity index has been introduced as a *"Genre Vector"* that signifies what genre (action, comedy, drama, etc.) is prevalent at what time of the day. The ultimate goal is to find a time-slot (of the day) based popularity index of the videos (Movie for our case). The MEC caches are then updated so that a higher number of movie requests can be met. We consider various input features obtained from the standard *MovieLens* dataset to predict the expected request counts in the future. The movies' expected request counts are used for caching the movies, with the most requested movie stored first into the MEC cache server in a slot-wise manner. Experimental results reveal that considering the users viewing profiles and categorizing the movies into various genre improved the overall cache hit rate at the MEC server.

7.1.2 QoE-Aware Adaptive-Bit Rate Caching

In the second chapter of this dissertation, the users preference for viewing a particular video quality based on varying network conditions is taken into consideration while making caching decisions. A joint optimization framework using RL that improves the overall QoE of the end-users by focusing and giving equal weightage to both the ABR and caching mechanism at the MEC is proposed. The RL-based framework, called *ABRCache* uses three types of modules, namely ABR, Planner and Evictor to optimally select the most appropriate and popular bitrate based on segment-level popularity and caches it accordingly. In addition, *ABRCache* handles the variations in bandwidth pertaining to different mobility models. The proposed model, combined with LSTM, can determine and extract patterns from the variable bandwidth input data, which is a type of time-series data. A comprehensive set of experiments demonstrate that the proposed *ABRCache* model improves the overall QoE and reduces network traffic load on the backhaul link simultaneously.

7.1.3 Collaborative Video Caching in a Clustered Edge Network

In the third contributory chapter, to further improve the caching performance a collaborative caching mechanism for clustered edge network is presented. The clustering of MEC servers is performed based on geographical locations. Collaboration is carried out among *'M'* MEC servers to service a given video request that arrives at a BS. The proposed collaborative caching mechanism unlike other collaborative strategies does not consider transcoding a given video in case of a cache miss, which is computationally costly. Rather, the caching strategy focuses on caching those quality of video segment which have a higher popularity score. *ColabCache* collaborates amongst the clustered MEC servers to make caching decisions based on the calculated *Priority Score* of video segments with respect to all MEC servers in the cluster. In addition, *ColabCache* unlike most of the previous work, is independent of the size of media library at the CDN. The proposed model is independent of the total number of videos in the media library, and hence the performance is optimal even when the media library is continuously increasing. The proposed model has been extensively evaluated using the *Zipf* and *Iflix* dataset. *ColabCache* has been compared with various state-of-the-art caching strategies against evaluation metrics such as cache hit rate, backhaul traffic, access delay and re-buffering.

7.1.4 Federated Caching and Prediction Model for Content Delivery

In the initial part of the final contributory chapter, a Federated RL-based caching mechanism, called *FedCache* is proposed. In this approach the model is trained locally on the individual data, rather being trained in a centralized server. For the first time, an A3C RL network has been trained in a Federated way for making caching decisions at the edge server. *FedCache* offers a scalable solution for training diverse request patterns by transferring the training process to the user's device instead of centrally at the edge server. After training their model, the users upload their trained parameters to the server, aggregating the learned parameters using Federated Averaging. Extensive experiments were performed on the *Iflix* dataset and *FedCache* outperforms existing state-of-the-art strategies.

7.1.5 Prediction Model for Content Delivery in Adaptive Video

The last part of the final chapter, focuses on improving the overall QoE of users for a video streaming session. QoE management using a fixed set of rules may not always guarantee optimal bandwidth utilization, video quality enhancement and accurate buffer estimation, especially in the face of severely varying and often unpredictable bandwidth fluctuations. To handle these issues across a wide range of varying network conditions and QoE parameters, an LSTM-CNN-based RL model called, *LASH* has been proposed. *LASH* is trained with a large set of input parameters. These parameters essentially model the dynamic control rules for handling varying network conditions and end-user demands. Thus, a more efficient QoE management model can be achieved. The proposed model maximizes the overall QoE by satisfying the three QoE verticals such as overall video quality, re-buffering and video quality switches simultaneously.

7.2 Future Works

While we have made significant contributions towards the designing and developing various DL approach for content caching and content delivery at the MEC server, some future directions still could be considered for further improving resource management at the MEC.

7.2.1 Content Aware cost efficient caching using Federated Learning

In the first part of Chapter 6 we proposed a FL-based caching mechanism at the MEC server. The proposed model called *FedCache* trains the caching model at the end-users devices, instead of centrally training the model at the MEC server. Such learning-based model shifts the training process from the central server to UE's, which otherwise over-consumes the network resources. However, in *FedCache* it is observed that during every communication round where the UE's send their trained parameters to the central MEC server incurs a significant communication cost. FL may therefore become ineffective or perhaps not feasible if communication bandwidth is limited. In order to solve this problem several compression techniques have been introduced such as *Gradient Dropping* [110] and *signSGD* [111]. However, such compression techniques are not efficient enough for handling larger dataset. Therefore, in the future work we aim to develop a cost efficient caching strategy using FL, that improves the overall QoE along with caching performance.

7.2.2 Deep Learning for Efficient Resource Allocation in V2X Communication Networks

With the recent advancement in cellular communication technologies and the introduction of 5G, it has been possible to achieve high reliability and QoS provisions. This has attracted many diverse application domains in the industry. One such application domain was created with 3GPP release 14 (2017), which allowed device-to-device (D2D) communication and has been severely exploited in V2X communications. Just like any communication network involves handling various resources, Radio Resource Management is central to V2X communications as well. It involves the construction of strategies and algorithms for handling and management of various radio resources such as Transmission power, Transmission frequency, Data rates, Modulation scheme, etc. [112]. With the recent advancements in DL-based methods, more DL-based algorithms have been proposed for the resource allocation problem in V2X networks. Therefore, in the future, we plan to propose a DL-based approach to allocate transmission frequency and power in a 5G V2V network communication such that latency constraints are met, and interference is minimal.

References

[1] M. ETSI, "Mobile edge computing (mec); framework and reference architecture," *ETSI, DGS MEC*, vol. 3, pp. 1–18, 2016.

[2] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin *et al.*, "Mec in 5g networks," *ETSI white paper*, vol. 28, pp. 1–28, 2018.

[3] A. Lekharu, K. Moulii, A. Sur, and A. Sarkar, "Deep learning based prediction model for adaptive video streaming," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS).* IEEE, 2020, pp. 152–159.

[4] M. T. Vega, C. Perra, F. De Turck, and A. Liotta, "A review of predictive quality of experience management in video streaming services," *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 432–445, 2018.

[5] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 187–198, 2015.

[6] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication.* ACM, 2017, pp. 197–210.

[7] H. Pang, J. Liu, X. Fan, and L. Sun, "Toward smart and cooperative edge caching for 5g networks: A deep learning based approach," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–6.

[8] V. Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, March 2020(Updated).

[9] Ericsson, "Ericsson mobility report," *White Paper*, June 2021. [Online]. Available: https://www.ericsson.com/4a03c2/assets/local/mobility-report/documents/2021/june-2021-ericsson-mobility-report.pdf

[10] G. Orsini, D. Bade, and W. Lamersdorf, "Computing at the mobile edge: Designing elastic android applications for computation offloading," in *2015 8th IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE, 2015, pp. 112–119.

[11] S. Kumar, N. Wang, C. Ge, and B. Evans, "Optimising layered video content delivery based on satellite and terrestrial integrated 5g networks," in *2019 European Conference on Networks and Communications (EuCNC)*. IEEE, 2019, pp. 161–166.

[12] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[13] C. Ge, N. Wang, S. Skillman, G. Foster, and Y. Cao, "Qoe-driven dash video caching and adaptation at 5g mobile edge," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016, pp. 237–242.

[14] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[15] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[16] X. Xu, J. Liu, and X. Tao, “Mobile edge computing enhanced adaptive bitrate video delivery with joint cache and radio resource allocation,” *IEEE Access*, vol. 5, pp. 16 406–16 415, 2017.

[17] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa, “The future of mobile cloud computing: Integrating cloudlets and mobile edge computing,” in *2016 23rd International conference on telecommunications (ICT).* IEEE, 2016, pp. 1–5.

[18] D. Satria, D. Park, and M. Jo, “Recovery for overloaded mobile edge computing,” *Future Generation Computer Systems*, vol. 70, pp. 138–147, 2017.

[19] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, “Fog computing: Focusing on mobile users at the edge,” *arXiv preprint arXiv:1502.01815*, 2015.

[20] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, “Mobile-edge computing introductory technical white paper,” *White paper, mobile-edge computing (MEC) industry initiative*, pp. 1089–7801, 2014.

[21] M. A. Maddah-Ali and U. Niesen, “Fundamental limits of caching,” *IEEE Transactions on information theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[22] K. Suksomboon, S. Tarnoi, Y. Ji, M. Koibuchi, K. Fukuda, S. Abe, N. Motonori, M. Aoki, S. Urushidani, and S. Yamada, “Popcache: Cache more or less based on content popularity for information-centric networking,” in *38th Annual IEEE Conference on Local Computer Networks.* IEEE, 2013, pp. 236–243.

[23] S. Podlipnig and L. Böszörmenyi, “A survey of web cache replacement strategies,” *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.

[24] Q. Huang, K. Birman, R. Van Renesse, W. Lloyd, S. Kumar, and H. C. Li, “An analysis of facebook photo caching,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 167–181.

[25] Z. Akhtar, Y. Li, R. Govindan, E. Halepovic, S. Hao, Y. Liu, and S. Sen, “Avic: a cache for adaptive bitrate video,” in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 305–317.

[26] W. Shi, Q. Li, C. Wang, G. Shen, W. Li, Y. Wu, and Y. Jiang, “Leap: learning-based smart edge with caching and prefetching for adaptive video streaming,” in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.

[27] W. Jiang, G. Feng, S. Qin, and Y.-C. Liang, “Learning-based cooperative content caching policy for mobile edge computing,” in *ICC 2019-2019 IEEE International Conference on Communications (ICC).* IEEE, 2019, pp. 1–6.

[28] Y. Jin, Y. Wen, and C. Westphal, “Optimal transcoding and caching for adaptive streaming in media cloud: An analytical approach,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 1914–1925, 2015.

[29] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, “Qoe-driven mobile edge caching placement for adaptive video streaming,” *IEEE Transactions on Multimedia*, vol. 20, no. 4, pp. 965–984, 2017.

[30] I. Sodagar, “The mpeg-dash standard for multimedia streaming over the internet,” *IEEE multimedia*, vol. 18, no. 4, pp. 62–67, 2011.

[31] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, “Adaptsize: Orchestrating the hot object memory cache in a content delivery network,” in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 483–498.

[32] V. Kirilin, A. Sundarrajan, S. Gorinsky, and R. K. Sitaraman, “Rl-cache: Learning-based cache admission for content delivery,” in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, ser. NetAI’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 57–63.

[33] T. X. Tran and D. Pompili, “Adaptive bitrate video caching and processing in mobile-edge computing networks,” *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 1965–1978, 2018.

[34] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, “Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach,” in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications.* IEEE, 2020, pp. 2499–2508.

[35] Z. Wan and Y. Li, “Deep reinforcement learning-based collaborative video caching and transcoding in clustered and intelligent edge b5g networks,” *Wireless Communications and Mobile Computing*, vol. 2020, 2020.

[36] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas, “Federated learning based proactive content caching in edge computing,” in *2018 IEEE Global Communications Conference (GLOBECOM).* IEEE, 2018, pp. 1–6.

[37] X. Wang, C. Wang, X. Li, V. C. Leung, and T. Taleb, “Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching,” *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9441–9455, 2020.

[38] K. Qi and C. Yang, “Popularity prediction with federated learning for proactive caching at wireless edge,” in *2020 IEEE Wireless Communications and Networking Conference (WCNC).* IEEE, 2020, pp. 1–6.

[39] W. Shi, Q. Li, R. Zhang, G. Shen, Y. Jiang, Z. Yuan, and G.-M. Muntean, “Qoe ready to respond: a qoe-aware mec selection scheme for dash-based adaptive video streaming to mobile users,” in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 4016–4024.

[40] Z. Chang, X. Zhou, Z. Wang, H. Li, and X. Zhang, “Edge-assisted adaptive video streaming with deep learning in mobile edge networks,” in *2019 IEEE Wireless Communications and Networking Conference (WCNC).* IEEE, 2019, pp. 1–6.

[41] Grouplens, *Movielens Dataset*, 2019 October. [Online]. Available: https://grouplens.org/datasets/movielens/

[42] F. M. Harper and J. A. Konstan, “The movielens datasets: History and context,” *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, p. 19, 2016.

[43] S. Lederer, C. Müller, and C. Timmerer, “Dynamic adaptive streaming over http dataset,” in *Proceedings of the 3rd multimedia systems conference*, 2012, pp. 89–94.

[44] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, “HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks,” *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[45] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, “Optimal edge user allocation in edge computing with variable sized vector bin packing,” in *International Conference on Service-Oriented Computing.* Springer, 2018, pp. 230–245.

[46] Kaggle, “Movie streaming dataset iflix,” *Dataset*, 2019. [Online]. Available: https://www.kaggle.com/aungpyaeap/movie-streaming-datasets-iflix

[47] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, “Commute path bandwidth traces from 3g networks: analysis and applications,” in *Proceedings of the 4th ACM Multimedia Systems Conference.* ACM, 2013, pp. 114–118.

[48] M. ETSI, “Mobile edge computing (mec); mobile edge management; part 1: System, host and platform management,” *ETSI, DGS MEC*, pp. 010–1.

[49] ETSI, “Mobile edge computing (mec); mobile edge management; part 2: application lifecycle, rules and requirements management: Etsi gs mec 010-2 v1. 1.1,” 2017.

[50] M. ETSI, “Mobile edge platform application enablement,” vol. 11, 2017.

[51] ——, “Mobile edge computing (mec); radio network information, etsi gs mec 012 v1.1.1,” 2017.

[52] ——, “Mobile edge computing (mec); ue application interface, etsi gs mec 016 v1.1.1,” 2017.

[53] DASH, “Dash dataset.” [Online]. Available: https://dash.itec.aau.at/dash-dataset/

[54] DASHEncoder, “Dashencoder.” [Online]. Available: https://github.com/slederer/DASHEncoder

[55] Youtube, “Youtube live encoder settings, bitrates and resolutions.” [Online]. Available: https://support.google.com/youtube/answer/2853702?hl=en

[56] W. Jiang, G. Feng, and S. Qin, “Optimal cooperative content caching and delivery policy for heterogeneous cellular networks,” *IEEE Transactions on Mobile Computing*, vol. 16, no. 5, pp. 1382–1393, 2016.

[57] X. Chen, L. He, S. Xu, S. Hu, Q. Li, and G. Liu, “Hit ratio driven mobile edge caching scheme for video on demand services,” in *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2019, pp. 1702–1707.

[58] S. Li, J. Xu, M. Van Der Schaar, and W. Li, “Popularity-driven content caching,” in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[59] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, “A survey on deep learning for big data,” *Information Fusion*, vol. 42, pp. 146–157, 2018.

[60] M. A. Beyer and D. Laney, “The importance of ‘big data’: a definition,” *Stamford, CT: Gartner*, pp. 2014–2018, 2012.

[61] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, “Deepcache: A deep learning based framework for content caching,” in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, 2018, pp. 48–53.

[62] E. Nygren, R. K. Sitaraman, and J. Sun, “The akamai network: a platform for high-performance internet applications,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[63] 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.

[64] A. Sadeghi, G. Wang, and G. B. Giannakis, “Deep reinforcement learning for adaptive caching in hierarchical content delivery networks,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1024–1033, 2019.

[65] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, “A provably efficient online collaborative caching algorithm for multicell-coordinated systems,” *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1863–1876, 2015.

[66] T. X. Tran, A. Hajisami, and D. Pompili, “Cooperative hierarchical caching in 5g cloud radio access networks,” *IEEE Network*, vol. 31, no. 4, pp. 35–41, 2017.

[67] E. Baccour, A. Erbad, A. Mohamed, K. Bilal, and M. Guizani, “Proactive video chunks caching and processing for latency and cost minimization in edge networks,” in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2019, pp. 1–7.

[68] Y. Guo, F. R. Yu, J. An, K. Yang, C. Yu, and V. C. Leung, “Adaptive bitrate streaming in wireless networks with transcoding at network edge using deep reinforcement learning,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 3879–3892, 2020.

[69] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, “Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks,” *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 247–257, 2019.

[70] Z. Yu, J. Hu, G. Min, Z. Zhao, W. Miao, and M. S. Hossain, "Mobility-aware proactive edge caching for connected vehicles using federated learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5341–5351, 2020.

[71] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 187–198.

[72] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions On Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[73] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. Pham, "Adaptive streaming of audiovisual content using mpeg dash," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 78–85, 2012.

[74] S. Kim, D. Yun, and K. Chung, "Video quality adaptation scheme for improving qoe in http adaptive streaming," in *2016 International Conference on Information Networking (ICOIN).* IEEE, 2016, pp. 201–205.

[75] M. D. F. De Grazia, D. Zucchetto, A. Testolin, A. Zanella, M. Zorzi, and M. Zorzi, "Qoe multi-stage machine learning for dynamic video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 1, pp. 146–161, 2017.

[76] Y.-L. Chien, K. C.-J. Lin, and M.-S. Chen, "Machine learning based rate adaptation with elastic feature selection for http-based streaming," in *2015 IEEE International Conference on Multimedia and Expo (ICME).* IEEE, 2015, pp. 1–6.

[77] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive http streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, 2011, pp. 169–174.

[78] H. T. Le, D. V. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, “Buffer-based bitrate adaptation for adaptive http streaming,” in *2013 International Conference on Advanced Technologies for Communications (ATC 2013).* IEEE, 2013, pp. 33–38.

[79] C. Müller, S. Lederer, and C. Timmerer, “An evaluation of dynamic adaptive streaming over http in vehicular environments,” in *Proceedings of the 4th Workshop on Mobile Video*, 2012, pp. 37–42.

[80] A. Lekharu, S. Kumar, A. Sur, and A. Sarkar, “A qoe aware lstm based bit-rate prediction model for dash video,” in *2018 10th International Conference on Communication Systems & Networks (COMSNETS).* IEEE, 2018, pp. 392–395.

[81] M. Chen, W. Saad, C. Yin, and M. Debbah, “Echo state networks for proactive caching in cloud-based radio access networks with mobile users,” *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3520–3535, 2017.

[82] K. Thar, N. H. Tran, S. Ullah, T. Z. Oo, and C. S. Hong, “Online caching and cooperative forwarding in information centric networking,” *IEEE Access*, vol. 6, pp. 59 679–59 694, 2018.

[83] 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.

[84] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[85] D. Wu, H. Xu, Z. Jiang, W. Yu, X. Wei, and J. Lu, “Edgelstm: Towards deep and sequential edge computing for iot applications,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1895–1908, 2021.

[86] F. M. Harper and J. A. Konstan, “The movielens datasets: History and context,” *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[87] M. Ahmed, S. Traverso, P. Giaccone, E. Leonardi, and S. Niccolini, "Analyzing the performance of lru caches under non-stationary traffic patterns," *arXiv preprint arXiv:1301.4909*, 2013.

[88] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 60–71, 2010.

[89] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, pp. 1–6, 2011.

[90] B. Chen and C. Yang, "Caching policy for cache-enabled d2d communications by learning user preference," *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6586–6601, 2018.

[91] L. Teng, X. Yu, J. Tang, and M. Liao, "Proactive caching strategy with content-aware weighted feature matrix learning in small cell network," *IEEE Communications Letters*, vol. 23, no. 4, pp. 700–703, 2019.

[92] M.-C. Lee, M. Ji, A. F. Molisch, and N. Sastry, "Throughput–outage analysis and evaluation of cache-aided d2d networks with measured popularity distributions," *IEEE Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5316–5332, 2019.

[93] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep multi-agent reinforcement learning based cooperative edge caching in wireless networks," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[94] J. Du, F. R. Yu, G. Lu, J. Wang, J. Jiang, and X. Chu, "Mec-assisted immersive vr video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9517–9529, 2020.

[95] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 272–285.

[96] J. Choi, A. S. Reaz, and B. Mukherjee, “A survey of user behavior in vod service and bandwidth-saving multicast streaming schemes,” *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 156–169, 2011.

[97] H. Zhao, J. Wang, F. Liu, Q. Wang, N. Luo, and W. Zhang, “Resource allocation for virtual streaming media server cluster in cloud-based multi-version vod,” in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*. IEEE, 2018, pp. 313–318.

[98] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, “A control-theoretic approach for dynamic adaptive video streaming over http,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 325–338.

[99] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[100] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[101] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, “Utility maximization in peer-to-peer systems with applications to video conferencing,” *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1681–1694, 2012.

[102] E. Altman and T. Jiménez, “Measuring audience retention in youtube,” in *Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2019, pp. 79–85.

[103] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.

[104] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics.* PMLR, 2017, pp. 1273–1282.

[105] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.

[106] Y. Navrotsky and N. Patsei, "Zipf's distribution caching application in named data networks," in *2021 IEEE open conference of electrical, electronic and information sciences (estream).* IEEE, 2021, pp. 1–4.

[107] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications.* IEEE, 2016, pp. 1–9.

[108] M. D. F. De Grazia, D. Zucchetto, A. Testolin, A. Zanella, M. Zorzi, and M. Zorzi, "Qoe multi-stage machine learning for dynamic video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 1, pp. 146–161, 2018.

[109] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[110] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv preprint arXiv:1704.05021*, 2017.

[111] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," in *International Conference on Machine Learning.* PMLR, 2018, pp. 560–569.

[112] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep reinforcement learning based resource allocation for v2v communications," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3163–3173, 2019.

Publications Related to Thesis

Journals

1. **Anirban Lekharu**, Mitansh Jain, Arijit Sur and Arnab Sarkar, ***"Deep Learning Model for Content Aware Caching at MEC Servers"***, in IEEE Transactions on Network and Service Management **(IEEE TNSM)**, vol. 19, no. 2, pp. 1413-1425, June 2022. [Chapter 3]

Conferences

1. **Anirban Lekharu**, K. Y. Moulii, A. Sur and A. Sarkar, ***"Deep Learning based Prediction Model for Adaptive Video Streaming"***, 2020 International Conference on COMmunication Systems & NETworkS **(IEEE COMSNETS)**, 2020, pp. 152-159. [Chapter 6]

Under Review

Journals

1. **Anirban Lekharu**, Chouhan, PS A., Sur, A., Patra, M.:, ***"Reinforcement Learning based Adaptive Bit-Rate Caching at MEC Server."***, In IEEE Transactions on Network and Service Management **(IEEE TNSM)** ***(Major Review Submitted, July 2023)***. [Chapter 4]

2. **Anirban Lekharu.**, Gupta, P., Sur, A., Patra, M.:, ***"Collaborative based Video Caching in the Edge Network using Deep Reinforcement***

Learning", In ACM Transactions on Internet of Things **(ACM TIOT)** ***(Under Review)***. [Chapter 5]

3. **Anirban Lekharu**, Samanta, A., Sur, A., Patra, M.:, ***"Content-Aware Caching at the Mobile Edge Network using Federated Learning"***, In IEEE Transactions on Emerging Topics in Computational Intelligence ***(Major Review)***. [Chapter 6]

Publications Outside Thesis

Conferences

1. **Anirban Lekharu**, Satish Kumar, Arijit Sur, and Arnab Sarkar. 2018. ***"A QoE Aware SVC Based Client-side Video Adaptation Algorithm for Cellular Networks"***. In Proceedings of the 19th International Conference on Distributed Computing and Networking **(ICDCN '18)**. Association for Computing Machinery, New York, NY, USA, Article 27, 1–4.

2. **Anirban Lekharu**, S. Kumar, A. Sur and A. Sarkar, ***"A QoE aware LSTM based bit-rate prediction model for DASH video"***, 2018 10th International Conference on Communication Systems & Networks **(IEEE COMSNETS)**, 2018, pp. 392-395.

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Guwahati 781039, India