
Electrical Engineering Theses

Electrical Engineering

Fall 12-9-2022

EFFECT ON 360 DEGREE VIDEO STREAMING WITH CACHING AND WITHOUT CACHING

Md Milon Uddin

Follow this and additional works at: https://scholarworks.uttyler.edu/ee_grad



Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Uddin, Md Milon, "EFFECT ON 360 DEGREE VIDEO STREAMING WITH CACHING AND WITHOUT CACHING" (2022). *Electrical Engineering Theses*. Paper 51.

<http://hdl.handle.net/10950/4130>

This Thesis is brought to you for free and open access by the Electrical Engineering at Scholar Works at UT Tyler. It has been accepted for inclusion in Electrical Engineering Theses by an authorized administrator of Scholar Works at UT Tyler. For more information, please contact tgullings@uttyler.edu.

EFFECT ON 360 DEGREE VIDEO STREAMING WITH CACHING AND WITHOUT
CACHING

by

MD MILON UDDIN

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering
Department of Electrical Engineering

Jounsup Park, Ph.D., Committee Chair

College of Engineering

The University of Texas at Tyler

December 2022


The University of Texas at Tyler
Tyler, Texas

This is to certify that the Master's Thesis of

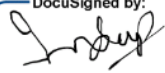
MD MILON UDDIN

has been approved for the thesis requirement on
November 10th, 2022
for the Master of Science in Electrical Engineering

Approvals:

DocuSigned by:

8628FBE73AD249F...

Thesis Chair: Jounsup Park, Ph.D.

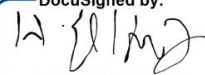
DocuSigned by:

0EE7E374D15143B...

Member: Premananda Indic, Ph.D.

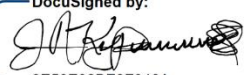
DocuSigned by:

A22ED704553E410...

Member: Arun Kulkarni, Ph.D.

DocuSigned by:

7E4401EB0BCF49D...

Chair: Department of Electrical Engineering

DocuSigned by:

3E50E32BE8F046A...

Dean: College of Engineering

© Copyright 2022 by Md. Milon Uddin
All rights reserved.

ACKNOWLEDGEMENT

I would like to express my sincerest thanks and appreciation to my thesis supervisor, Dr. Jounsup Park for his immense support and guidance. I am very appreciative of his patience, guidance and encouragement that led to the successful completion of this thesis. I would like to thank my thesis committee members, Dr. Premananda Indic and Dr. Aurun Kulkarni for their support and counsel despite their busy schedules. I am very lucky to have a graduate coordinator like Dr. Indic who helped me a lot in my master's program. I want to express my gratitude to Dr. Hassan El-Kishky for giving me research funds for publishing my research papers. I am highly indebted to every person of the Department of Electrical Engineering for their support in completing this endeavor. My thanks and appreciation also go to my friend Tharun Kumar Bethi who helped and encouraged me during my master's program. I would like to express my gratitude to my other friends Joshua Stapp, Kazi Arfat and Vivek Kumar Tiwari for their support throughout this journey. Finally, I'm extremely pleased to have received enormous support from my father, mother, and other family members to pursue my master's degree in the United States. A special thanks to my elder brother Jasim Uddin for this tremendous support for my higher study.

Thank you everyone for being a part of my life and making this possible for me.

TABLE OF CONTENTS

LIST OF TABLES.....	ii
LIST OF FIGURES.....	iii
Abstract.....	iv
Chapter One INTRODUCTION	1
1.1 Aim/Objectives.....	4
1.2 Motivation.....	4
1.3 Organization of Thesis.....	5
Chapter Two LITERATURE SURVEY.....	6
Chapter Three SYSTEM ARCHITECTURE.....	8
Chapter Four LRU AND LFU ALGORITHM.....	9
4.1 Least Recently Used Algorithm.....	9
4.2 Least Frequently Used Algorithm	10
4.3 Experimental Set Up for LRU and LFU Algorithm.....	11
Chapter Five EXPERIMENTAL RESULTS FOR LRU AND LFU ALGORITHM	12
Chapter Six MACHINE LEARNING MODEL EVALUATION FOR 360 DEGREE VIDEO CACHING	18
6.1 Feature Extraction.....	18
6.2 Algorithm.....	19
6.3 Experiment Setup.....	21
Chapter Seven EXPERIMENTAL RESULTS FOR MODEL DEVELOPMENT	22
Chapter Eight TEST BENCH FOR THE DEVELOPED ALGORITHM AND COMPARISON OF RESULTS.....	29
8.1 Test Bench.....	29
8.2 Comparison of Results.....	30
Chapter Nine DISCUSSIONS AND FUTURE WORK	39
9.1 Discussions	32
9.2 Future Work	33
References	34
Appendix	38

TABLE OF TABLES

Table 1: Simulation of LRU and LFU-50% cache size (video-1).	16
Table 2: Simulation of LRU and LFU-50% cache size (video-2)	16
Table 3: Comparison of average cache hit rate for LRU and LFU	17
Table 4: Video meta data	19
Table 5: Video segments.....	21
Table 6: Summary of three models.....	28

LIST OF FIGURES

Figure 1: Virtual Reality.....	3
Figure 2: Cache with the client and the video server	8
Figure 3: Flow chart of a LRU algorithm	9
Figure 4: Flow chart of a LFU algorithm.....	10
Figure 5: Comparison of average cache hit rate with cache size for video 1.....	12
Figure 6: Comparison of cache miss with different cache size for video 1	13
Figure 7: Comparison of cache hit for each individual user with different cache size for video 1.....	14
Figure 8: Comparison of average cache hit rate for 70% cache size for video 2	14
Figure 9: Comparison of cache miss for 70% cache size for video 2.....	15
Figure 10: Comparison of cache hit rate for random user's arrival for video 1 and 2	17
Figure 11: Decision tree for the random forest algorithm.....	19
Figure 12: Comparison of total actual vs total predicted freq. for a video seg. for RF algorithm	22
Figure 13: Comparison of total actual vs total predicted freq. for a video seg. for LR algorithm	23
Figure 14: Comparison of total actual vs total predicted freq. for a video seg. for Bayesian algorithm ...	24
Figure 15: Comparison of coefficient of determination of three ML models	26
Figure 16: Scatter plot diagram for Random Forest Regression	27
Figure 17: Scatter plot diagram for Linear regression	27
Figure 18: Scatter plot for Bayesian Regression.....	28
Figure 19: Comparison of average cache hit rate for LRU and Machine learning algorithm.....	30
Figure 20: Comparison of average cache miss rate for LRU and Machine learning algorithm.....	31

ABSTRACT

EFFECT ON 360 DEGREE VIDEO STREAMING WITH CACHING AND WITHOUT CACHING

Md Milon Uddin

Thesis Chair: Jounsup Park, Ph. D.

The University of Texas at Tyler

November 2022

People all around the world are becoming more and more accustomed to watching 360-degree videos, which offer a way to experience virtual reality. While watching videos, it enables users to view video scenes from any perspective. To reduce bandwidth costs and provide the video with less latency, 360-degree video caching at the edge server may be a smart option. A hypothetical 360-degree video streaming system can partition popular video materials into tiles that are cached at the edge server. This study uses the Least Recently Used (LRU) and Least Frequently Used (LFU) algorithms to accomplish video caching and suggest a system architecture for 360-degree video caching. Two 360-degree videos from 48 users' head movements are used in the experiment, and caching between the LRU cache and LFU cache is compared by changing the cache size. The findings demonstrate that, for varied cache sizes, utilizing LFU caching outperforms LRU caching in terms of average cache hit rate.

In the first part of the research, we compared LRU and LFU caching algorithm. In the second part of the research, a suitable caching strategy model was developed based on user's field of view. Field of view (FoV) is the term used to describe the portion of the 3600 videos that viewers typically see when watching 3600 videos. Edge caching can be a smart way to increase customer satisfaction while maximizing bandwidth usage (QoE). A 3600-video caching strategy has been developed in this study using three machine learning models that use random forest, linear regression, and Bayesian regression. As features, tiles' frequency, user's view prediction probability, and resolution were used. The created machine learning models are designed to decide the caching method for 360-degree video tiles. The models can forecast the frequency of viewing for 3600 video tiles (subsets of a full video). With a predictive R^2 value of 0.79, the random forest regression model performs better than the other suggested models when the outcomes of the three developed models are compared.

In the third part of the research, to compare our machine learning algorithm with LRU algorithm, a python test bench program was written to evaluate both algorithms on the test set by varying the cache size. The results demonstrate that our machine learning approach, which was created for 360-degree video caching, outperforms the LRU algorithm.

CHAPTER ONE

INTRODUCTION

A human-computer interface called virtual reality (VR) enables people to communicate effectively in a three-dimensional virtual world [1]. The foundation of virtual reality is the idea of illusion. Users can fully experience virtual reality (see figure 1). Virtual reality has attracted a lot of interest recently from a range of sectors, including academia, entertainment, military training, and scientific visualization. Many virtual content producers have been investigating 360-degree videos in recent years to improve the quality of the user experience (QoE). To do this, edge servers might cache popular video content with high resolution. 360-degree videos are typically viewed with a head mounted display, such as the Oculus Rift S, HTC Vive, and others. Head mounted displays give pleasant and immersive feelings for end users [2]. By 2021, there will be 100 million Virtual Reality (VR) headsets in use, with 50 million of those predicted to be mobile VR headsets, according to Cisco Visual Networking Index [3].

Omni-directional cameras or numerous cameras can record 360-degree videos, which result in spherical panoramas of that size [4]. Segments, which are rectangular-shaped partitions, can be created from 360-degree films. Each segment is broken up into tiles that are stored in the content servers in a variety of resolutions. 360-degree videos have a high bitrate and require a huge storage system, requiring up to 4-5 times the storage when compared to normal videos, so delivering them to virtual reality (VR) headsets would require a lot of network traffic [5]. Furthermore, compared to streaming traditional videos, these videos take more CPU, GPU, and energy from the end user [6].

A method for sending video to end users while using the least amount of bandwidth is called video tiling. The only tiles in a 360-degree video system that must be provided in high-resolution format are those that are in the user's field of view (FoV), whereas all other tiles may be sent in low-resolution or not at all [7-9].

Caching is one of many research efforts that have been done to lower the amount of bandwidth needed for 360-degree videos [10–12]. Popular 360-degree video content can be cached to reduce network traffic. However, it must be utilized well due to the cache size restriction.

A cache between a core network, also known as a video server, and a user can significantly reduce bandwidth consumption by reducing the number of queries users make that must be transmitted to the core network in order to obtain the desired contents [13].

The main advantages of caching may be divided into four categories: 1) bandwidth reduction, 2) network load reduction of core network, 3) latency reduction, 4) cost reduction, and 4) effective resource management.

The process of caching a video stream involves temporarily keeping it in cache so that it can be played back later. Users experience a cache hit if a CDN cache server keeps the tiles they have requested. A cache miss, on the other hand, happens when the cache must get the requested tiles from the CDN's original server because the requested tiles were not found in the cache. Because of the high bandwidth required to fetch the tiles from the original server and the distance to the server, the delivery time may be prolonged. The cache server must choose whether to save fetched tiles in the cache after a cache miss. Additionally, the cache server must choose which tile has to be discarded in order to create room for the most recent arrivals of tiles if there is no room in the cache to add newly fetched tiles. DeepCache [14], PopCache [15], and Greedy-Dual-Frequency caching policy [16] are only a few of the many cache eviction methods that have been created. A content popularity prediction model is employed by DeepCache to forecast the likelihood of upcoming requests. PopCache is a decision policy that allows an individual ICN router to cache content roughly in accordance with the popularity characteristic of the content. PopCache hasn't looked into contrasting their outcomes with those of other cache replacement policies. Researchers recommended including the most crucial aspects of the file and its accesses, such as file size, file access frequency, and recentness of the previous access, in their Greedy-Dual-Size-Frequency caching strategy. The user's field of view (FoV), which can be an intriguing topic to examine for caching 360-degree videos, has not been taken into account for view prediction above the given techniques.

Numerous academics have looked into the popularity of videos for video caching [17]. Their study aims to decrease bandwidth loss, shorten content delivery time, boost cache hit ratio, and enhance user experience. Deep reinforcement learning-based caching, feedforward neural networks for caching, and deep learning for 360-degree video caching are just a few of the machine learning techniques that have been introduced.

In the first part of the research, we design a system architecture of edge server (cache) and apply the least recently used (LRU) and least frequently used (LFU) algorithms to cache contents at the cache server in order to meet the wireless Virtual Reality applications, increase cache hit rate, and improve user quality of experience. The decision of which tiles to remove from the cache when the cache is full is one of the issues in 360-degree video. Our strategy is to change the cache size and see how the number of cache hits changes. For experimental purposes, we examine the cache hits and cache misses between LRU and LFU for various cache sizes, and we contrast the consequences of these size changes using the LRU and LFU algorithms. The 48 users' 360° videos used in this study came from a publicly accessible data source [18].

In the second part of our research, machine learning (ML) models are used in this study to suggest a caching technique for 360-degree films. Our three machine learning models, which we constructed using Random Forest regression, Linear regression, and Bayesian regression, are compared. These models are designed to identify which tile should be cached to increase the overall cache hit ratio, decrease video delivery delay, and make optimal use of bandwidth. In the third part of the research, we have used a test bench to compare our machine learning algorithm with LRU algorithm,

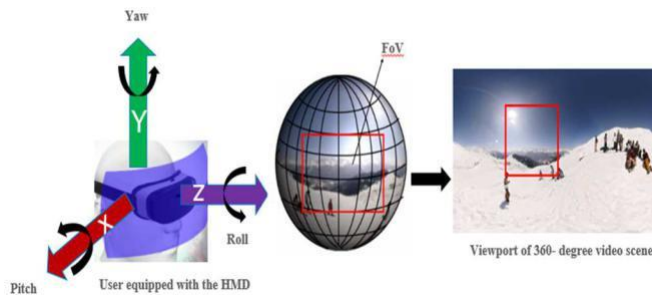


Figure 1: Virtual Reality.

1.1 Aim/Objectives

The aims and objectives of this research is

- 1) to address some of the some of the limitations of the current 360degree video caching system.
- 2) to maximize overall cache hit ratio
- 3) to reduce video delivering latency and
- 4) to minimize bandwidth loss

The boarder impact of the research is to improve user's quality (QoE) of experience.

1.2 Motivation

Utilizing a cache between users and distant content servers can significantly reduce network latency, network load, and bandwidth usage by reducing the number of queries sent to content servers. Machine learning model for 360-degree video caching could be an effective way to maximize cache hit rate at the user end.

1.3 Organization of Thesis

The novel contributions of this research are developing a machine learning model for caching 360-degree videos, comparison of our developed model and other model and effect on cache hit ratio by varying cache size for LRU and LFU algorithm. The research in this Thesis is explained in the following nine chapters: Chapter 2 looks at a brief overview of previous research in 360-degree video caching. Chapter 3 describes the system architecture. Chapter 4 LRU and LFU algorithm. Chapter 5 illustrates experimental results for LRU and LFU algorithm. Chapter 6 indicates machine learning model evaluation for 360-degree video caching. Chapter 7 describes experimental results for model development. Chapter 8 detailed out the test bench for the developed model algorithm and comparison of results. Chapter 9 describes discussion and future work of the research.

CHAPTER TWO

LITERATURE SURVEY

Let's talk about the present research being done in fields that serve as the foundation for our proposed framework before we delve into the research for it.

A Macro cell base station (MBS) and a small cell base station (SBS) that work together to prevent adjacent SBSs from storing the same tile have been presented as an effective method to cache 360-degree videos in cellular networks [19]. The light-wave collaborative field of vision (FoV) prediction algorithm in [20] combines the prediction from the trajectories of users in the flock with the prediction from its own user's previous trajectory for caching at the edge. Reinforcement learning (RL), a machine learning-based cache admission technique, was proposed in [21], although they only achieved mediocre cache hit results when cache space was plentiful. Reinforcement learning uses a variety of factors, like object size, recentness, frequency, etc., to determine cache entry. DeepCache uses popularity prediction to prefetch things into a cache [22]. Dynacache reduces the amount of cache misses by more than 65% by using recency as a cache eviction strategy [23]. The CFLRU algorithm, which employs cost and recency for cache eviction, lowers mean replacement costs in swap systems and buffer caches by 26.2% and 26.4%, respectively [24]. For live 360-degree video, a tile-based caching mechanism has been suggested in [25]. When a new user request pops up in the cache, the authors of [26] suggest an online MaxMinDistance algorithm to replace the cache. In our study, we altered several cache sizes and contrasted these size changes with the LRU and LFU caching algorithms. Our research is distinct from previous research methods due to the approach we used, and the high cache hit rate it produced when compared to other research works.

Edge caching has been suggested as a practical way to improve the quality of material served while also bringing content closer to end users [27-28]. The authors of [29] developed a tile-based caching method that aims to increase the coverage of the tile set while simultaneously reducing the disparity between the requested and cached tile resolutions across viewports. The light-wave collaborative field of vision (FoV) prediction approach for edge caching combines predictions from the past trajectory of its own user with predictions from the trajectories of other users in the flock [30].

Caching popular information at mobile edge servers and lowering network operating costs can both lessen the requirement for pricey backhaul cables [31–34]. When the cache size is big for the reinforcement learning (RL) model, the authors of [35] achieved a good percentage of cache hit rate.

Most of the research studies described above assumed that the popularity profiles of the videos were known, however in some circumstances, this was not the case. The popularity of a piece of content is predicted using reinforcement learning algorithms that profit from demand history in [36–39] to get around this restriction. To increase the overall quality of videos for end viewers, authors in [40] proposed a view-port aware deep reinforcement learning technique for 360-degree video, in which the most popular videos were employed at base quality with a virtual viewport. To prefetch items into a cache, DeepCache uses popularity prediction [14]. According on server hit rate and predicted round-trip time, the creators of PopCache established an analytical methodology to evaluate the effectiveness of various caching choice strategies [15].

They compared the proposed caching policy to the benchmark caching choice policies, such as the always, fixed probability [41], and path-capacity-based probability [42], using their analytical model. Our study differs from other research methods in that we employed a machine learning model to identify which tiles are likely to be requested by users and which tiles needed to be cached to achieve a high cache hit rate.

CHAPTER THREE

SYSTEM ARCHITECTURE

Figure 2 depicts a standalone caching architecture, which serves as the initial point of contact for user requests. The terms "cache hit" and "cache miss" refer to two potential outcomes in a content caching architecture. Depending on the tile request, a user can receive a cache hit or cache miss. If a requested tile is found in the cache, the user receives a cache hit; otherwise, the user receives a cache miss. In the event of a cache miss, the cache server requests the desired tile from the remote content server. The cache server provides the requested tile to the user after retrieving it. It is crucial to send the requested tiles as soon as possible. When a cache hit occurs, distributing tiles to users proceeds more quickly than when a cache miss occurs. Due to the limited cache size, it is crucial to make informed decisions about which tiles should be added to the cache and which tiles should be removed from the cache server in order to increase cache hit rate. By boosting the cache hit rate, delivery time and bandwidth usage can be significantly decreased [43].

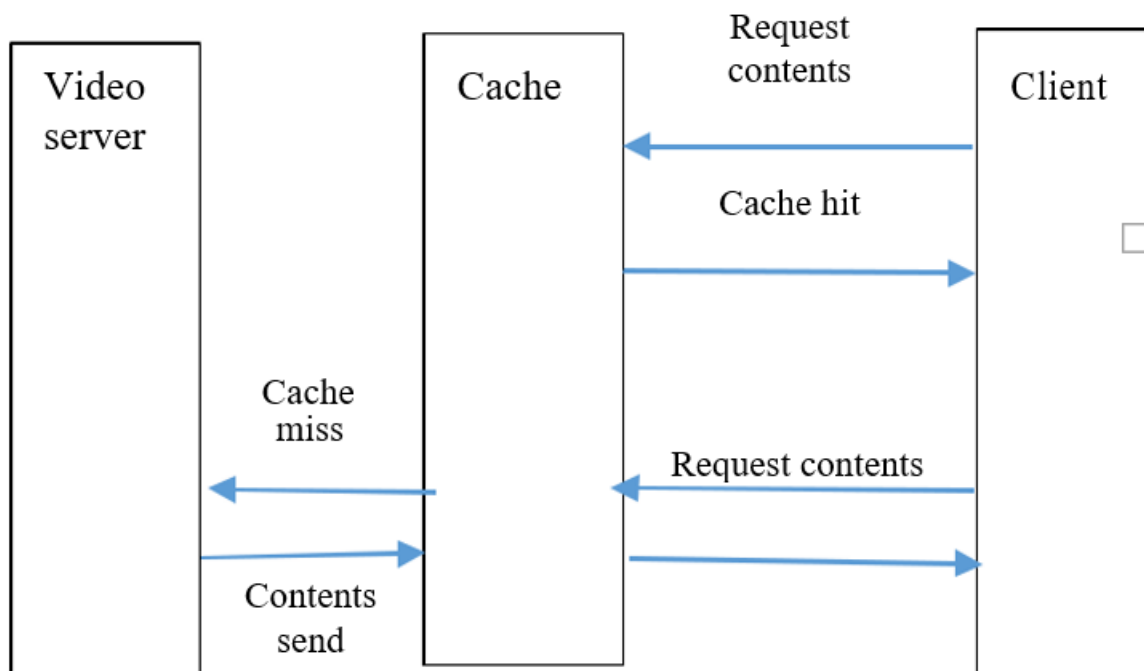


Figure 2: Cache with the Client and the Video Server.

CHAPTER FOUR

LRU AND LFU ALGORITHM

4.1 Least Recently Used (LRU) Algorithm

When the cache capacity is full according to the LRU caching policy, the tile with the longest time since the last access gets the highest priority to be removed from the cache in order to make room for new tiles [44]. When a user requests a new tile that is not already in the cache, LRU removes the least recently used video tiles that the user has viewed from the cache. The cache module replaces the least recently used tile when the cache's maximum size is reached with the newly requested tile. In order to test the cache capacity, we employed 50%, 80%, and 100% of the entire video data kept on a video server. Additionally, we took into account 70% of the overall cache size.

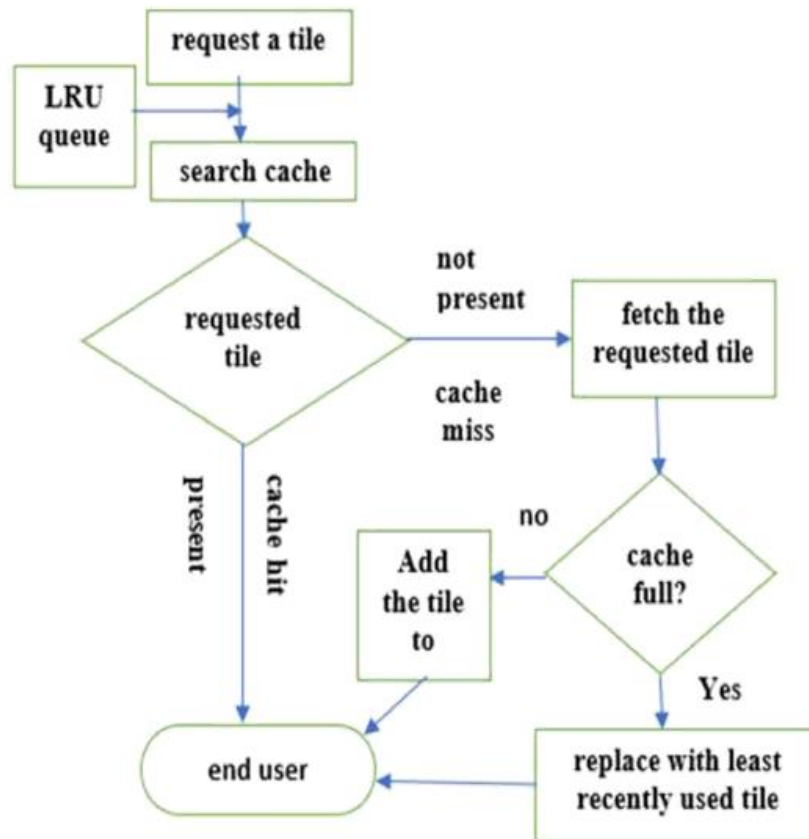


Figure 3: Flow Chart of a LRU Algorithm.

4.2 Least Frequently Used (LFU) Algorithm

When the cache capacity is full, according to LFU caching policy, a tile has the highest priority to be removed from the cache to create room for new tiles if it has experienced the fewest accesses since the last access [44]. When a new tile that is not already in the cache is requested by the users, LFU removes the least frequently used video tiles that were viewed by the users from the cache. For the purpose of clearing the cache, the frequency data for tiles is stored in the LFU cache. The frequency information for each tile in the cache is examined in the event of a cache miss to identify the tile that has been accessed the fewest times. The newly desired tile is subsequently put in its place. A first-in, first-out (FIFO) strategy is utilized to replace the first tile entered to the cache when the frequencies of the two least recently used tiles are equal.

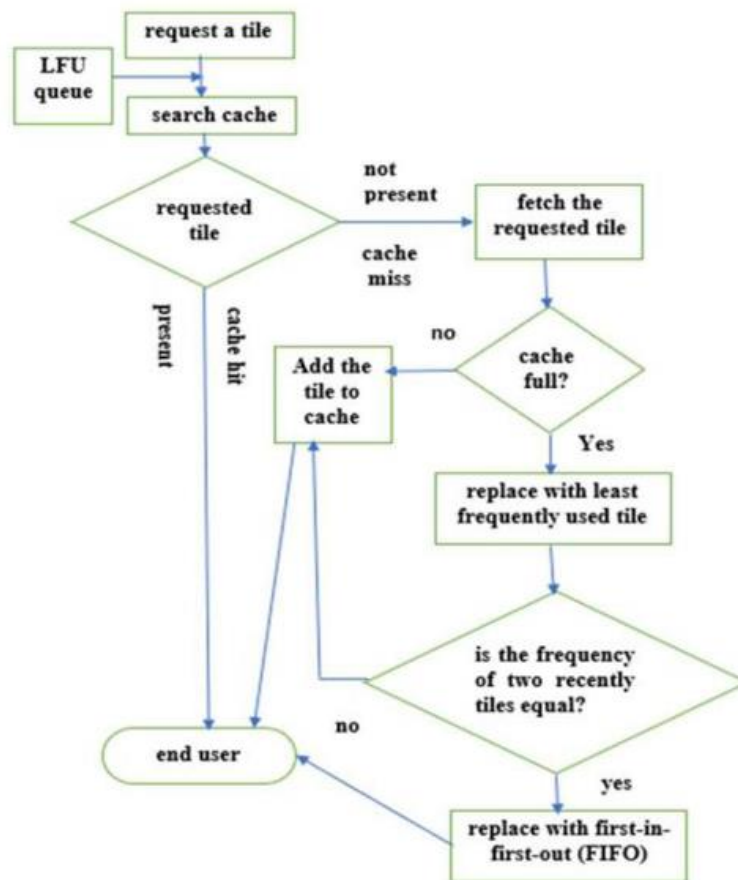


Figure 4: Flow Chart of a LFU Algorithm.

4.3 Experimental Setup for LRU and LFU Algorithm

We used video 1 and 2 that are 360-degree panoramas. Conan's talk show in video one features the most items from the video on stage. Freestyle skiing, which is featured in Video 2, features numerous skiers who alternately become visible and invisible. There are 48 viewers for each video. Each segment in videos 1 and 2 is broken geometrically into four-by-four tiles to create an equirectangular shape, totaling 164 1-sec long video segments. Due to its excellent storage and compression capabilities for 360-degree films, we adopted an equirectangular format [45]. The maximum cache size was set at the number of tiles in each video as a whole. The segments are of varying quality, and each tile has a distinct byte size. 48 individuals come sequentially in experiment 1's video-1 and video-2 (starting with user 1, followed by user 2, and so on). We are not considering delay time in this situation. We viewed the film with 48 people in a random order in experiment 2. The arrival of users is subject to a 1 to 5 second random time delay. User requests for tiles are presented in order for experiment 2. By doing this, the experiment's frequency, and recency of tile requests, which are used for the LFU and LRU cache eviction processes, respectively, are effectively randomized.

CHAPTER FIVE

EXPERIMENTAL RESULTS FOR LRU AND LFU ALGORITHM

Figure 5 compares the average cache hit rates for the LRU and LFU algorithms for various cache sizes for video 1 in Experiment 1 (Sequential). The average cache hit rate for LRU caching at 50% cache size is 68.67%, but the average cache hit rate for LFU caching is 71.49%. Thus, when comparing the LFU algorithm's cache hit rate to the LRU algorithm's, the average cache hit rate for video 1 from 48 users increased by 3%. When examining the outcomes for 48 users' viewing data for video 1, the LFU algorithm's cache hit rate improved for 80% cache size by an average of 0.5% greater than the LRU algorithm's cache hit rate. The error margin for this finding is rather narrow, and the pattern might not hold for other videos.

The size of the cache at 100% is equal to the size of the entire video. As a result, the cache never entirely fills to the point where the LFU or LRU algorithms must replace old data in the cache with new data. Since there is always available cache space, any cache misses are tiles that no prior user has requested. As a result, they are put into the cache without altering any existing data. The hit rates for the LRU and LFU algorithms are therefore presented as being equal. The cache hit rate at 100% cache size for initial loading is 96.69%.

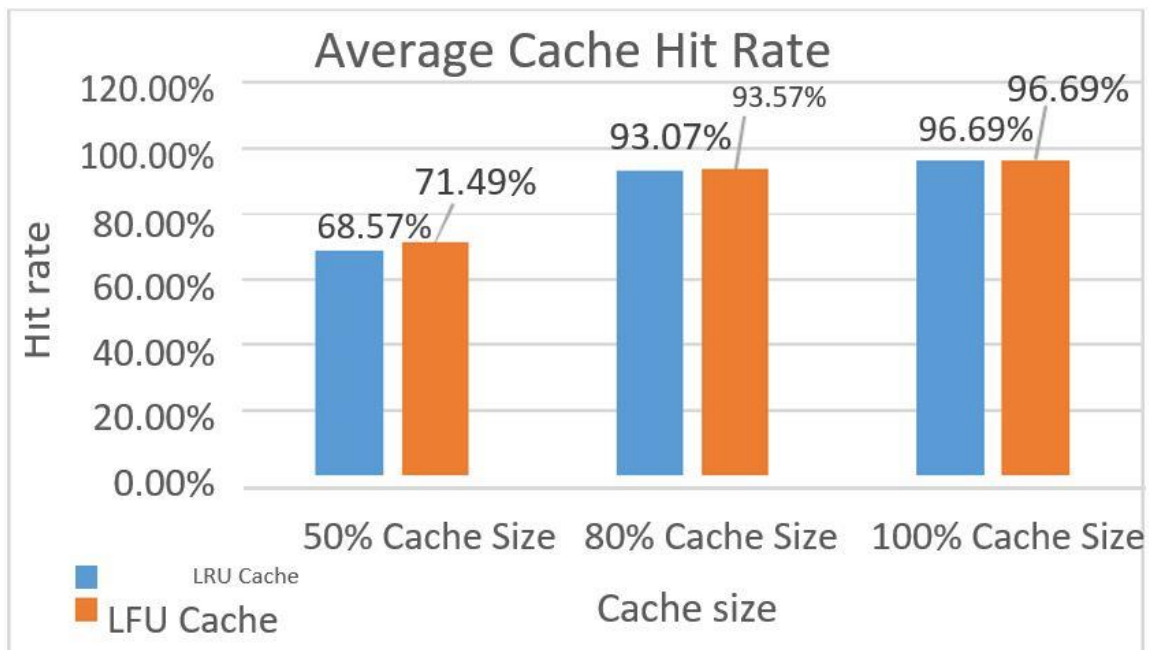


Figure 5: Comparison of Average Cache Hit Rate with Different Cache Size for Video 1.

Figure 6 compares LRU and LFU caching for total cache misses at 50%, 80%, and 100% of the cache sizes, respectively. When the cache size is reduced by 50%, the LRU cache experiences 2,218 more cache misses than the LFU cache, whereas the LRU cache experiences 412 more cache misses than the LFU cache. Both algorithms have the same number of cache misses at 100% cache size. Even when the cache is initially fully loaded, cache misses still happen. Since there is always adequate cache space, any cache misses—tiles that have not been requested by any previous users—are loaded into the cache without altering any existing data.

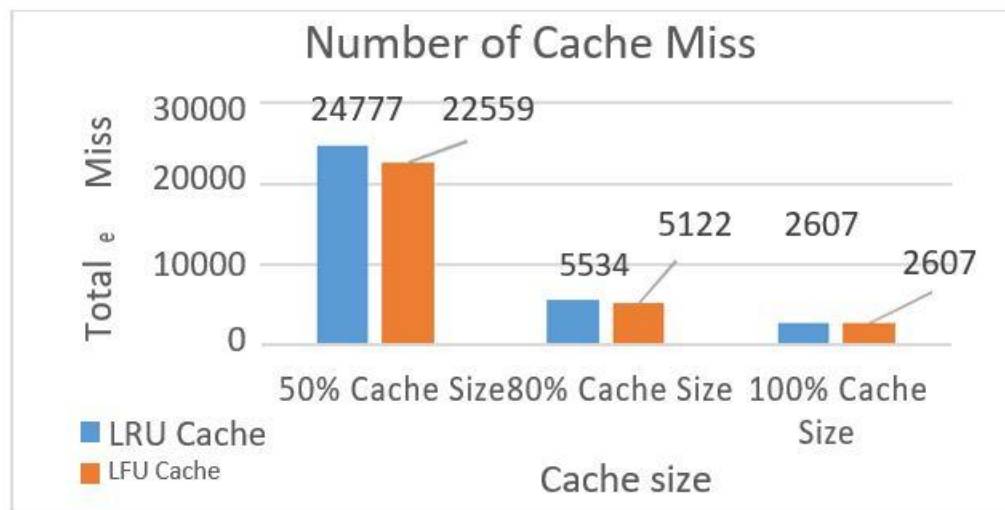


Figure 6: Comparison of cache miss with different cache size for video 1.

When viewing video 1, Figures 7 illustrates the cache hit rate for individual users. It is clear that LFU cache performs better than LRU cache for cache sizes of 50% and 80%, respectively. Both the LFU and LRU algorithms have the same cache hit rate for each individual user at 100% cache size.

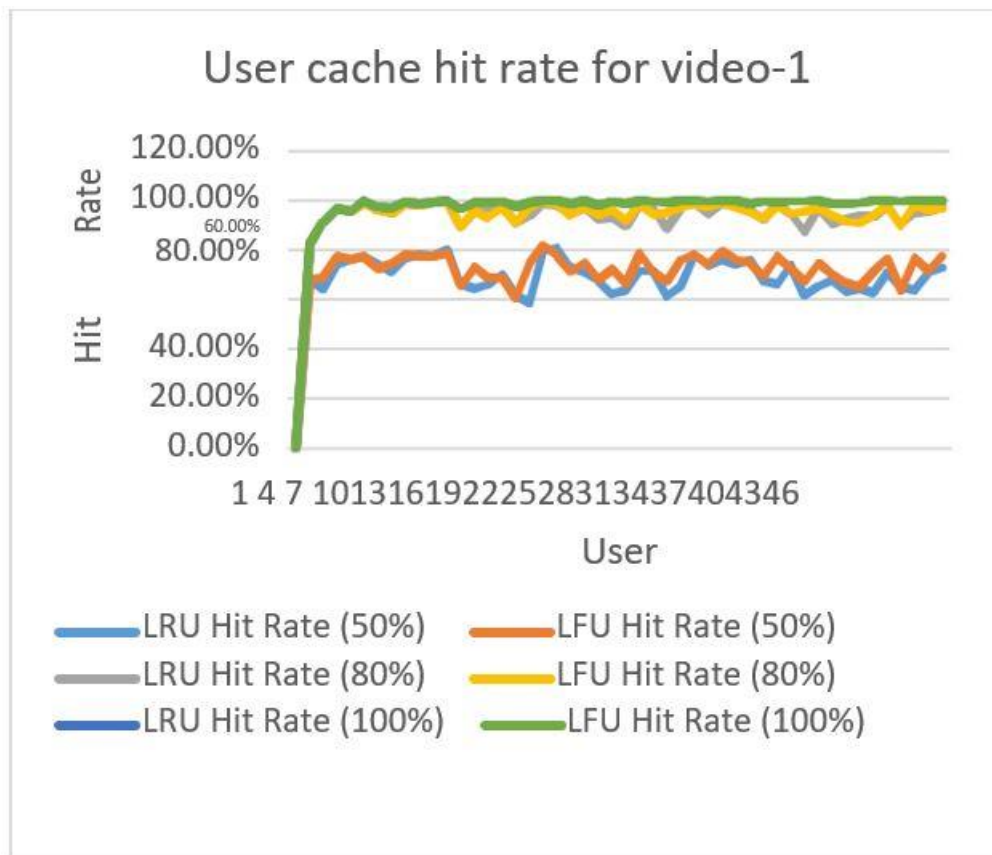


Figure.7. Comparison of Cache Hit for Each Individual User with Different Cache Size for Video 1.

Figure 8 shows that for 70% of the total cache size, LRU caching has an average cache hit rate of 79.38% whereas LFU caching has an average cache hit rate of 82.92%. When comparing LFU caching to LRU caching for video 2's 70% cache size, the average cache hit rate increased by 3.54%.

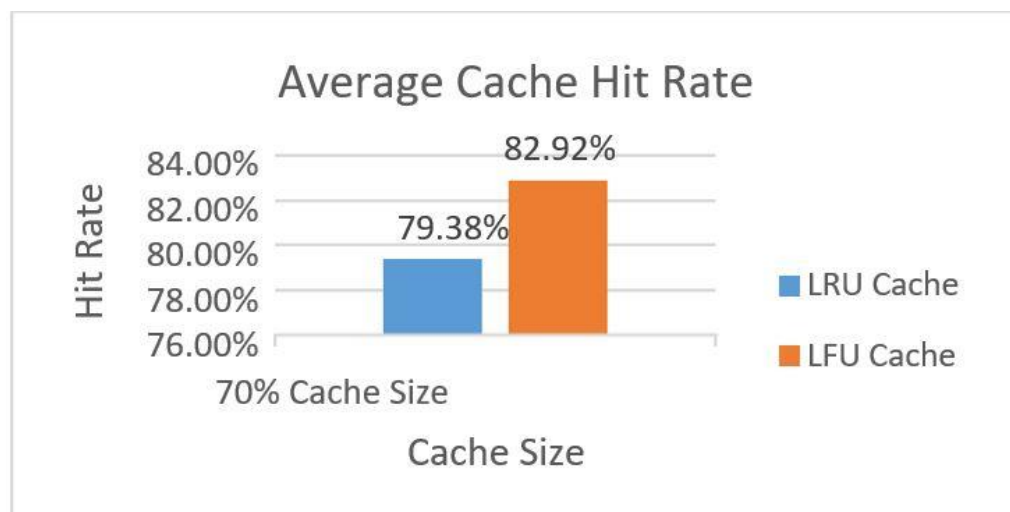


Figure 8: Comparison of Average Cache Hit Rate for 70% Cache Size for Video-2.

Figure 9 shows that using the LFU technique to cache video 2 resulted in less cache misses than using the LRU algorithm. Regarding this, there have been a total of 18648 cache misses for LRU caching and 14655 for LFU caching. Nearly 4000 tile queries make up the gap between the two techniques' cache misses.

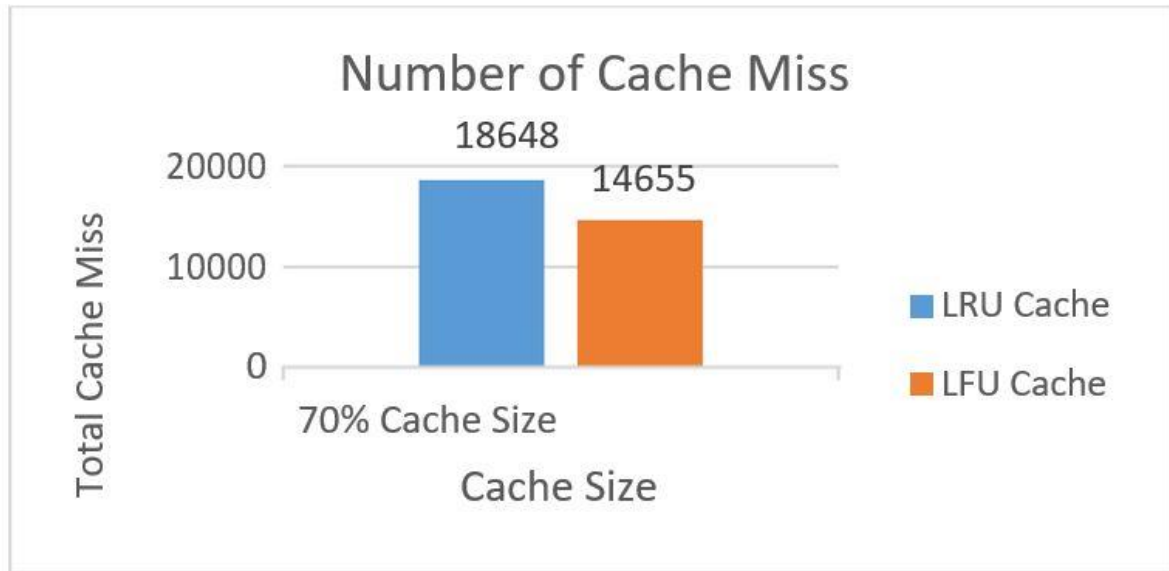


Figure 9: Comparison of cache miss for 70% cache size for video 2.

Experiment 2 (Random Arrival of People): For both videos 1 and 2, experiment 2 was conducted for random arrival of users. For each movie, five simulations were run using a randomly assigned user order. Table-1 displays the LRU and LFU simulation results for video-1 when the cache size is set to 50%. Table 2 displays the LRU and LFU simulation results for video-2 at a cache capacity of 70%.

Table I. Simulation of LRU and LFU- 50% cache size (Video-1)

Simulation number	Simulation Average Cache Hit Rate (LRU)	Average Hit Rate	Simulation Average Cache Hit Rate (LFU)	Average Hit Rate
1	54.35%	52.53%	59.62%	55.47%
2	50.57%		52.6%	
3	50.74%		53.27%	
4	52.27%		55.39%	
5	54.74%		56.43%	

Table II. Simulation of LRU and LFU- 70% cache size (Video-2).

Simulation number	Simulation Average Cache Hit Rate (LRU)	Average Hit Rate	Simulation Average Cache Hit Rate (LFU)	Average Hit Rate
1	68.03%	68.19%	70.94%	69.77%
2	67.25%		68.04%	
3	66.90%		68.02%	
4	67.76%		68.95%	
5	71.03%		72.91%	

According to Figure 10, the average hit rate for caching at a 50% cache size is 52.53% for LRU caching and 55.47% for LFU caching, resulting in a difference of 2.94% in favor of LFU caching. Average hit rates for LFU caching for the 70% cache size are 69.77% compared to 68.19% for LRU caching, a difference of 1.58% favoring LFU caching (shows in Figure 10). When a random user arrives, LFU caching once more produces marginally superior outcomes to LRU caching algorithm.

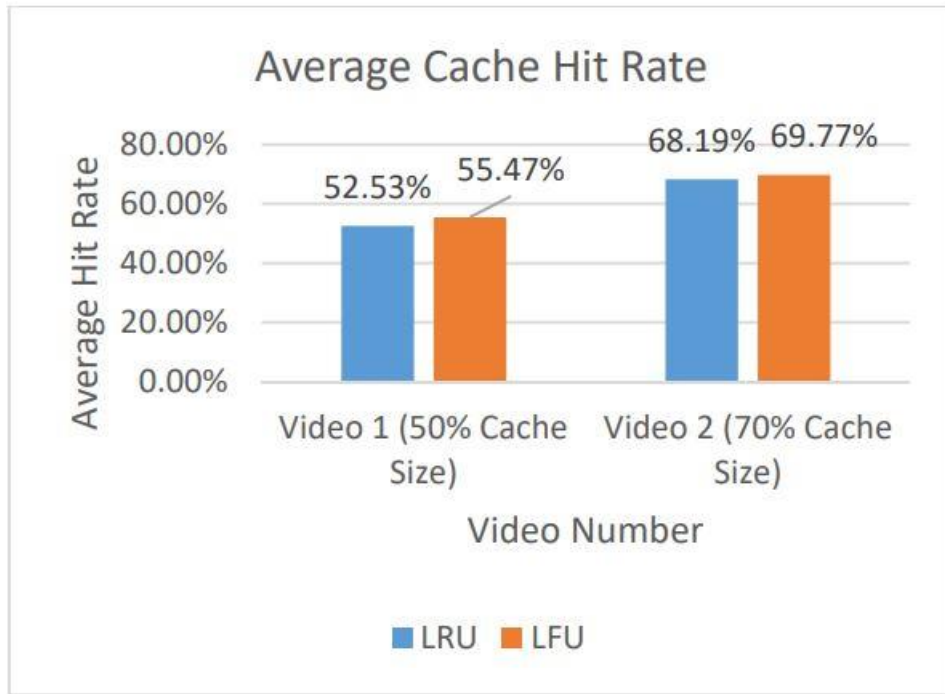


Figure10: Comparison of Cache Hit Rate for Random Users' Arrival for Video 1 and 2.

Table III. Comparison of average cache hit rate for LRU and LFU.

Algorithm	Average cache hit for 50% cache	Average cache hit for 80% cache	Average cache hit for 70% cache	Comments
LRU	68.67%	93.07%	79.38%	When users come sequentially
LFU	71.49%	93.57%	82.92%	
LRU	52.53%		68.19%	When users come randomly
LFU	55.47%		69.77%	

CHAPTER SIX

MACHINE LEARNING MODEL EVALUATION FOR 360 DEGREE VIDEO CACHING

6.1. Feature Extraction

For the training and testing datasets of the machine learning model, features were extracted from a data set consisting of 9 videos with 48 users. Tile frequency, view prediction probability for each tile for 1 second, and tile resolution for each video are features used in the data set. The following equation [46] calculates the forecast accuracy.

$$\sum_{t=1}^T (wt, gt) \dots \dots \dots (1)$$

Here, g_t = normalized ground truth which has 0 for tiles which are not visible and $1/(\text{number of visible tiles})$ for tiles which are visible.

w_t = The probability that tile t will be exhibited in a future segment is projected.

The segment frequency score is the result of adding the ground truth values from all users. The average individual segment frequency score in the data set was taken to be 16 tiles per video segment.

$$\text{Segment frequency score} = \frac{fT1 + fT2 + fT3 + \dots + fT16}{\text{Total number of tiles}}$$

Here,

$fT1$ = frequency of tile 1,

$fT2$ = frequency of tile 2,

$fT3$ = frequency of tile 3

$fT16$ = frequency of tile 16

The video's metadata is used to determine the tile's resolution. Using a transition probability matrix, P , the users' view prediction has been computed. As indicated in [47], we adopted this state transition probability matrix because of its simplicity. Assuming that there are 1, 2, and k states, the state transition matrix is

$$P = \begin{bmatrix} p_{11} & p_{12} \dots p_{1k} \\ p_{21} & p_{22} \dots p_{2k} \\ \vdots & \vdots \dots \vdots \\ p_{k1} & p_{k2} \dots p_{kk} \end{bmatrix}$$

Table IV. Video meta data

Video No.	Video contents	Video length	Video category
1	Conan	2'44"	Performance
2	Ski	3'21"	Sport
3	Help	4'53"	Film
4	Conan	2'52"	Performance
5	Tahiti Surf	3'35"	Sport
6	The fight of Falluja	10'55"	Documentary
7	Cooking Battle	7'31"	Performance
8	LOSE Football	2'44"	Sport
9	The Last of the Rhinos	4'53"	Documentary

6.2. Algorithm

Three regression algorithms-random forest, linear regression, and Bayesian regression are used. Random forest is a supervised learning method that utilizes the ensemble learning method for regression. A random forest performs its operation by building various decision trees (shown in figure 3) during the training period and taking the mean of the classes as the prediction of all the trees.

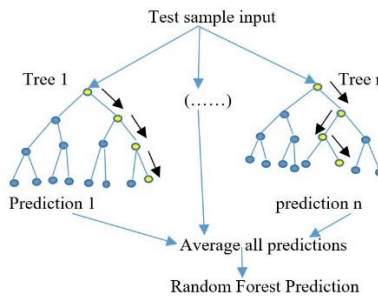


Figure11: Decision Tree for the Random Forest Algorithm.

In our model, we randomly selected k data points from the training set (row of the data set indicating vector of attributes, such as tile frequency, view prediction, and tile resolution) and created a decision tree linked with these k data points. Each of our N-trees (the number of decision trees in the model) will forecast the value of Y for the data points (segment frequency score) for a new data point, and we will then assign the new data points to the average of all the anticipated Y values.

Another kind of regression procedure is linear regression (LR). The linear regression model is used to create a link between features (independent variables) and a continuous target variable (dependent variable). When there is just one feature, simple linear regression is employed; when there are numerous features, multi-linear regression is used. The linear regression equation can be expressed as follows because our dataset has numerous features:

$$Y = mx_1 + mx_2 + \dots + mx_n + b \dots \dots \dots (2)$$

Y= dependent variable

m= slope

x_1 =1st independent variable

x_2 =2nd independent variable

x_n =nth independent variable

b= constant

From a Bayesian perspective, we have defined linear regression using probability distributions rather than point estimates. Instead of being estimated as a single value, the output, Y, is thought to be taken from a probability distribution. The answer used in the Bayesian linear regression model comes from a normal distribution.

$$Y \sim N(\beta^T X, \sigma^2 I)$$

Here,

β =coefficient

X=predictor

I=identity matrix

σ =standard deviation

The output Y is derived from a normal (Gaussian) distribution with a mean and variance. The mean has been calculated by multiplying the weight matrix by the predictor matrix. The variance has been found by squaring the standard deviation.

6.3. Experimental Setup

Nine videos from a variety of genres, including performance, athletics, documentary, and film, have been used. Two of these videos were used as a training data set, while seven of them were used for training. Title frequency, view prediction probability for each tile over a one-second period, and tile resolution are among the characteristics of the training and testing data set. Conan's talk program, video-1, features a lot of on-stage displays of objects. In video 2, a free-style skiing activity is demonstrated, and many skiers emerge in the frame at random. A monster chases a woman and a police officer in video 3. Since most of the objects in videos 1 and 4 appear on the stage, viewers are more likely to focus on the tiles in the video's center. In the documentary in Video 9, the rhino serves as the main protagonist. Here, the rhino is the main focus of spectators' attention. There were 48 users who watched each video. Each video's section lasts for one second and is broken into 16 tiles. Each tile in this case has a distinct number of bytes, and each video's resolution is unique.

Table V. Video segments

Video number	Video segment
Video-1	154
Video-2	191
Video-3	162
Video-4	195
Video-5	195
Video-6	645
Video-7	441
Video-8	154
Video-9	282

CHAPTER SEVEN

EXPERIMENTAL RESULTS FOR MODEL DEVELOPMENT

The Random Forest Algorithm uses the video resolution and view prediction probability for 16 tiles in 1 segment as input characteristics. Frequency is the anticipated result. The projected frequency of segments for this method shows a closer relationship to the actual frequency of the segments.

Actual frequency= [32 31 32 ...36 35 36]

Predicted frequency= [31.90 30.68 32.03...31.67 30.47 30.57]

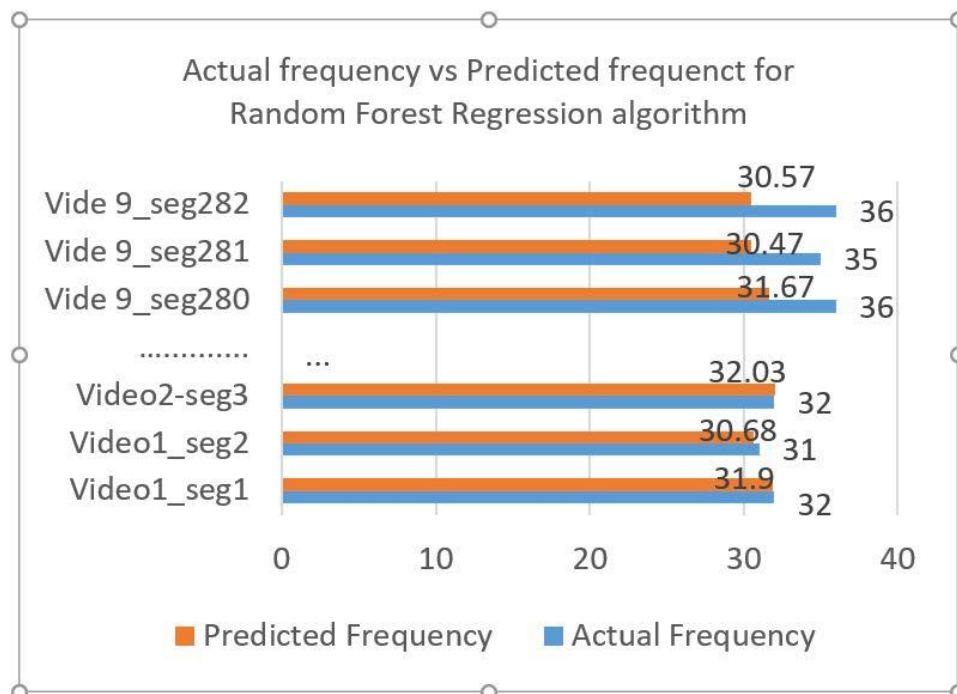


Figure12: Comparison of Total Actual vs Total Predicted Frequency for a Video segment for Random Forest Algorithm.

Figure 12 compares the total expected frequency of the video segment that the user will watch to the total actual frequency of the video segment that the user has actually viewed. While the model predicts a frequency of 31.90 for the identical video 1 segment 1, the actual frequency of video 1 segment 1 is 32. Out of 48 viewers, the model predicts that roughly 32 will watch a comparable amount of segment 1 of the film. For video 9 of section 280, there are significantly less predicted frequencies than actual frequencies. Using this frequency data, it is possible to load the tiles or segments that many viewers will watch into the cache.

For the Linear Regression algorithm, the predicted frequency is given below:

Actual frequency= [32 31 32 ...36 35 36]

Predicted frequency= [31.90 30.68 32.03...31.67 30.47 30.57]

Figure 13 compares the overall actual frequency of a video segment watched by users to the overall anticipated frequency of the segment that the user would watch. The model predicts a total frequency of approximately 31, which is 1 frequency less than the actual total frequency of video 1 segment 2. The real frequency of video 1 segment 1 is 32. Out of 48 viewers, the model predicts that about 31 will watch the first part of the film. The estimated frequency for video 9 of segment 282 has been lowered from the total of 36 projected frequencies to roughly 31. Here, the model shows almost 5 out of 48 viewers will not watch segment 282.

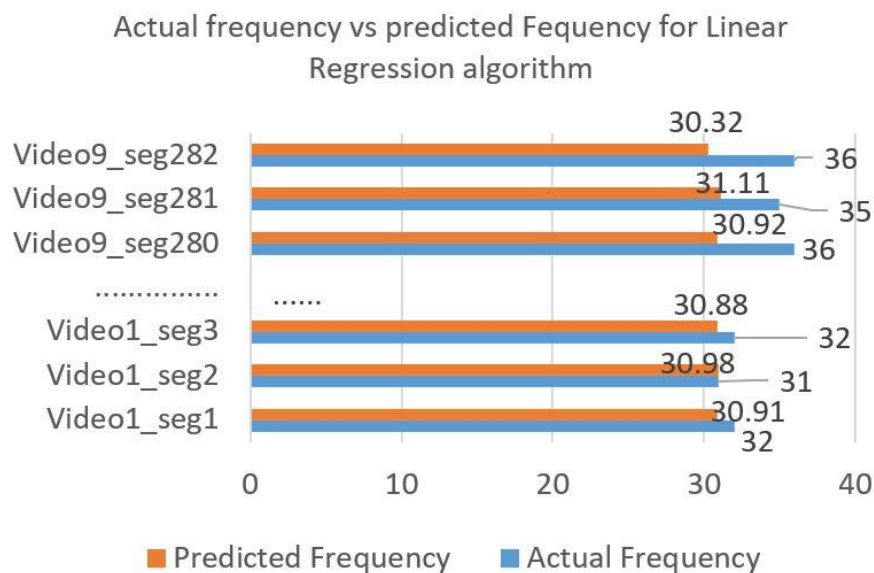


Figure13: Comparison of Total Actual vs Total Predicted Frequency for a Video Segment for Linear Regression Algorithm.

The Bayesian Regression algorithm's total predicted frequency and total actual frequency are shown in Figure 14 respectively. Segments 280, 281, and 282 of video 9 have different frequencies than the total projected frequency for segments 11, 12, and 13.

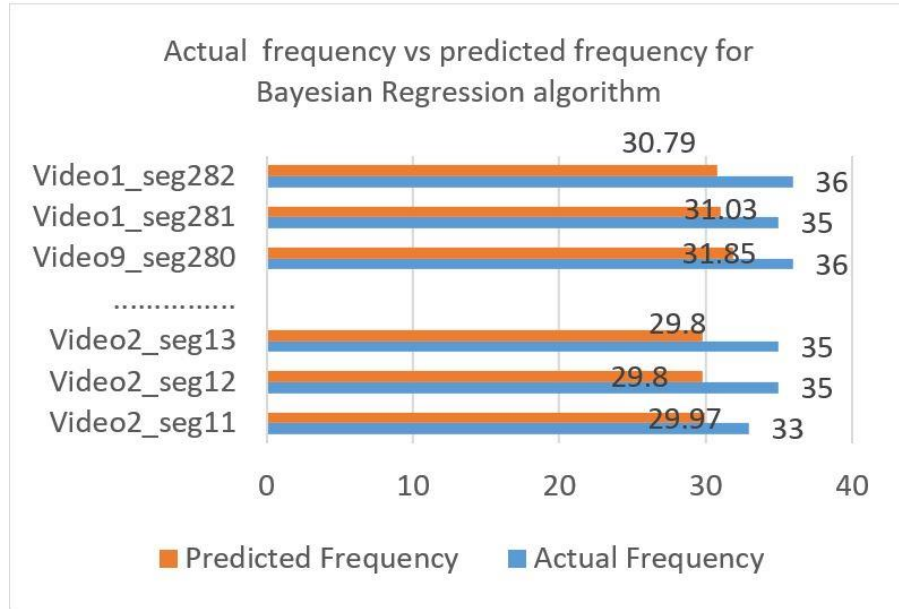


Figure14. Comparison of Total Actual vs Total Predicted Frequency for a Video Segment for Bayesian Regression.

The loss function for the regression, also referred to as the mean squared error (MSE), measures how closely a regression line resembles a set of data points. It is calculated by averaging the squared differences between the estimated and actual values.

$$MSE = \frac{1}{N} \sum_{i=1}^n (Y_i - Y_j)^2 \quad \dots\dots\dots(3)$$

N= number of data points

Y_i= observed values

Y_j= predicted values

K-fold cross-validation offers a way to test a model's performance on hypothetical data, enabling assessments of how well a model avoids overfitting and functions generally. The method operates by dividing the dataset into k-folds. The dataset is then trained using k-1 folds, and this sub-model is tested using the final fold. K times are needed to complete this operation, one for each fold.

An aggregated assessment of the overall model's performance will be calculated by averaging the outcomes from each of these sub-models. Results of random forest regression using cross-validation when five folds of data are taken into account:

[-3.0530 -2.9752 -2.8716 -3.0231 -2.9165]

MSE= 2.97

So, Root Mean Square Error (RMSE)= $\sqrt{\text{MSE}}$ = 1.72

When making predictions about actual data points, our generated model is typically 1.72 frequency off, according to this Root Mean Square Error (RMSE), but the RMSE is rather small, which seems to be a good thing.

Cross-validation results when considering 5 folds of data set of linear regression:

[-4.5384 -2.9659 -2.4855 -1.9474 -3.2928]

MSE =3.408

So, Root Mean Square Error (RMSE)= $\sqrt{\text{MSE}}$ = 1.84

Cross-validation results when considering 5 folds of data set of Bayesian regression:

[-4.3109] -3.0404 -2.425 -1.8992 -3.3320]

MSE =3

So, Root Mean Square Error (RMSE)= $\sqrt{\text{MSE}}$ = 1.73 which is less than the random forest and the linear regression.

A coefficient of determination, or R², is used to assess the extent to which fluctuations in the independent variable can account for variations in the dependent variable. R-squared provides the precise percentage of fluctuations in Y that are interpreted by X variables. ranges from 0% to 100%, or from 0 to 1. The correlation coefficient's formula [48] is as follows:

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n \sum x^2 - (\sum x)^2][n \sum y^2 - (\sum y)^2]}} \dots\dots\dots(4)$$

Here,

n = Total number of observations

Σx = Total of the First Variable Value

Σy = Total of the Second Variable Value

Σxy = Sum of the Product of first & Second Value

Σx^2 = Sum of the Squares of the First Value

Σy^2 = Sum of the Squares of the Second Value

Now the coefficient of determination can be written as $= (\text{correlation coefficient})^2 = r^2$.

If the r-squared value is between 0.5 and 0.7, it is generally seen as having a moderate effect size, and if it is greater than 0.7, it is often regarded as having a strong impact size [48].

The regression model is most likely not going to accurately forecast the genuine number because it predicts a segment's total frequency as 43.4 instead of the segment's true total frequency. In this situation, the threshold serves as a tolerance for error within which predictions provided by machine learning models can be accepted as accurate forecasts for all models. An acceptable margin of error for determining tile quality modifications for caching in the case of segment frequency is 2.5 frequency units.

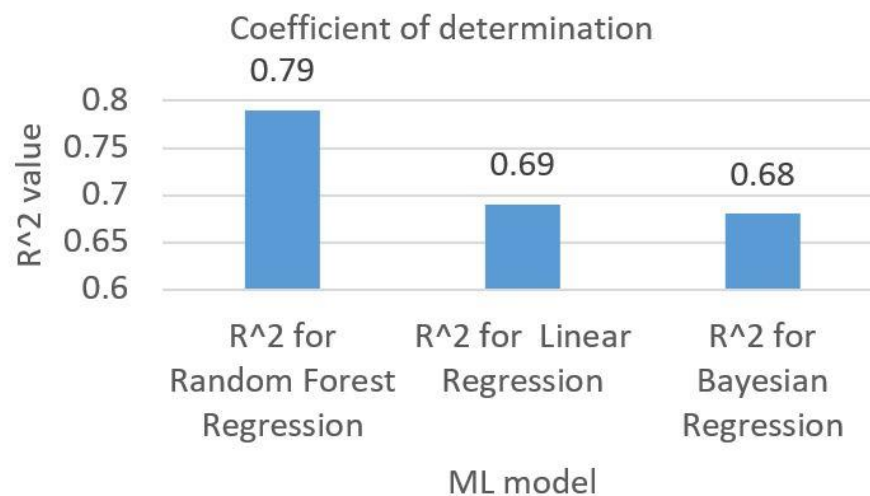


Figure15: Comparison of Coefficient of Determination of Three ML Models.

The coefficient of determination for three models—random forest regression, linear regression, and Bayesian regression—is compared in figure 15. With a coefficient of determination of 0.79, random forest regression has the highest correlation with linear and Bayesian regression models having roughly 0.1 less.

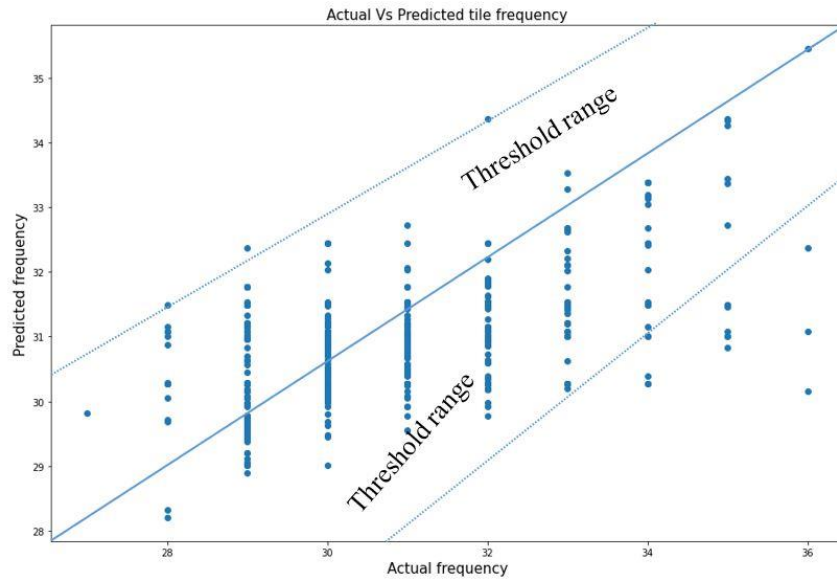


Figure16: Scatter Plot Diagram for Random Forest Regression.

The relationship between variables is visually shown and observed using a scatter plot. Figure 16's solid line represents the performance of the ideal, perfect model with a forecast R^2 value of 1. A notable departure from the ideal model can be seen in figure 16's data point distribution. The dotted line in the illustration illustrates how this might be overcome by giving the model a modest threshold. 79% of predictions on the test dataset came inside the threshold range for the random forest regression with a threshold value of 2.5 frequency units. As shown in figure 17, with a threshold value of 2.5 frequency units, 69% of predictions on the test dataset fell within the threshold range for the linear regression

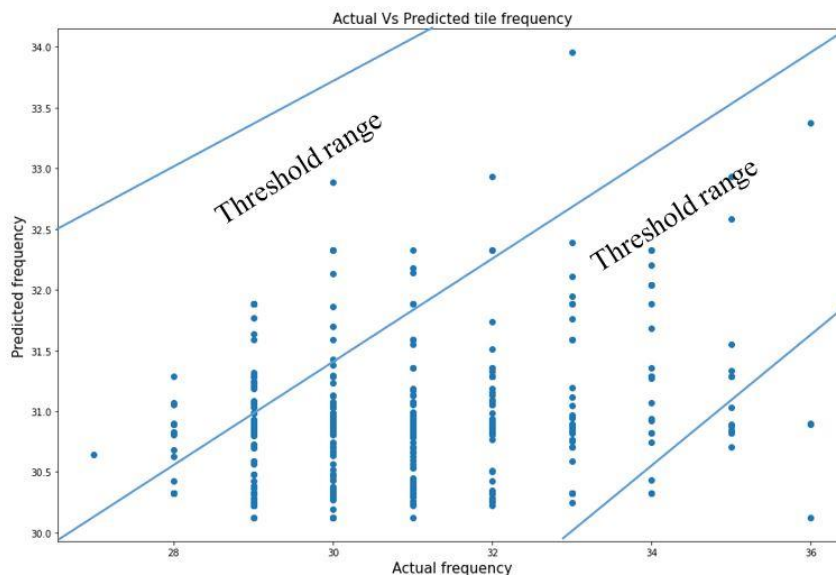


Figure 17: Scatter Plot Diagram for Linear Regression

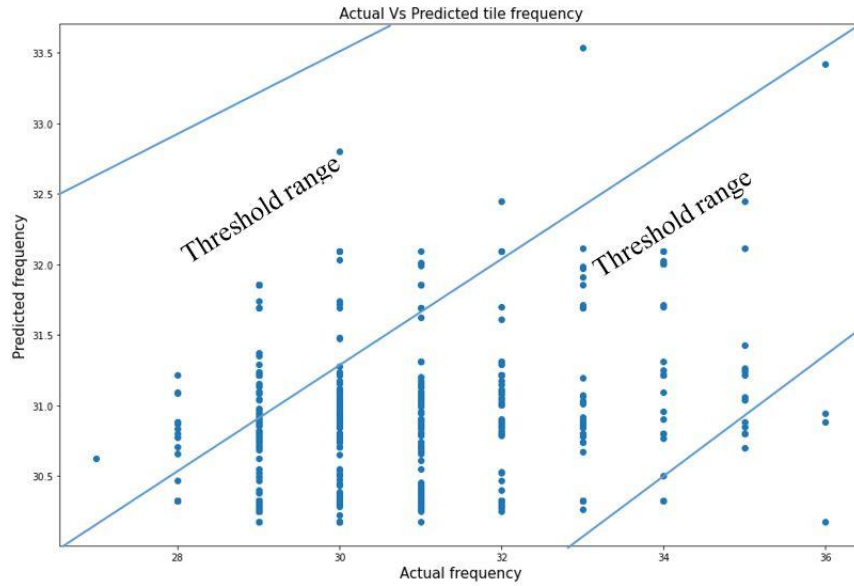


Figure18: Scatter plot diagram for Bayesian Regression.

Figure 18 demonstrates that, with a threshold value of 2.5 frequency units, 68% of predictions on the test dataset fell inside the linear regression's acceptable range.

Table VI. Summary of three models

Model	MSE	RMSE	R^2
Random forest regression	2.97	1.72	0.79
Linear regression	3.40	1.84	0.69
Bayesian regression	3	1.72	0.68

With a predictive R^2 value of 0.79, random forest regression seems to perform better when compared to the other two models using the criteria in table VI.

CHAPTER EIGHT

TEST BENCH FOR THE DEVELOPED ALGORITHM AND COMPARISON OF RESULTS

8.1 Test Bench

We discussed about the development of the three-machine learning model in the previous chapter. The goal of developing those models is to determine which model provides the most accurate forecast of viewing a tile based on segment resolution and tile probability. Out of three models, Random Forest regression model performs better compared to other two models and we have used this trained model for the algorithm development. In the final algorithm the random forest model would be used to determine the frequency score of the segment of the 360 degree video. A higher frequency score indicates tile within the field of view (FoV) has the high probability of being viewed by the users, while a low frequency score indicates the tile is most likely the outside of FoV. For this reason, tiles outside of FoV are loaded into cache at half resolution, while tiles predicted to be inside the FoV are loaded at full resolution. Once the cache is completely filled up, an LRU cache eviction method is utilized to evict the tile from the cache to make space for the new tile.

To test the LRU and Machine learning caching algorithm, a test set was extracted two 360 videos (video 1 and video 2). Features used in the data set are tile probability and tile resolution. The test data set is passed through the model to generate frequency score prediction. Since 360-degree video caching system is designed to be used by multiple users, the test file is setup in a way that we can simulate multiple users watching the same segment at the same time of the same video or users coming and watching videos sequentially for example user 1 watches video 1 and user 2 watches video after user 1 with a time delay between 2 minutes by repeating calls different segments throughout the test file for different users. 716 segments were used in the test bench for two videos. We have considered a full segment size is 5MB.

To compare our machine learning algorithm with LRU algorithm, a python test bench program was written to evaluate both algorithms on the test set by varying the cache size. The size of cache is defined as varying sizes in megabyte. After defining the data cache size, we loaded the testing data set and imported the trained Random Forest machine learning model.

We set requested segment's resolution to full resolution into the cache if frequency score predicted by the model surpasses the threshold, otherwise sets requested segments resolution to 50% of the original segment into the cache. With the resolution reduced for frequency scores less than the threshold, tiles loaded into the cache would take less space in the cache than those at full resolution. In the test bench, if there is a cache miss, a cache miss counter is incremented, and, if there is a cache hit, the hit counter incremented.

The test bench prints the average cache hit and average cache miss

The average cache hit rate is calculated:

$$\text{Average cache hit rate} = \frac{\text{Cache hit}}{\text{cache hit} + \text{cache miss}} * 100 \dots \dots \dots (5)$$

8.2 Comparison of Results

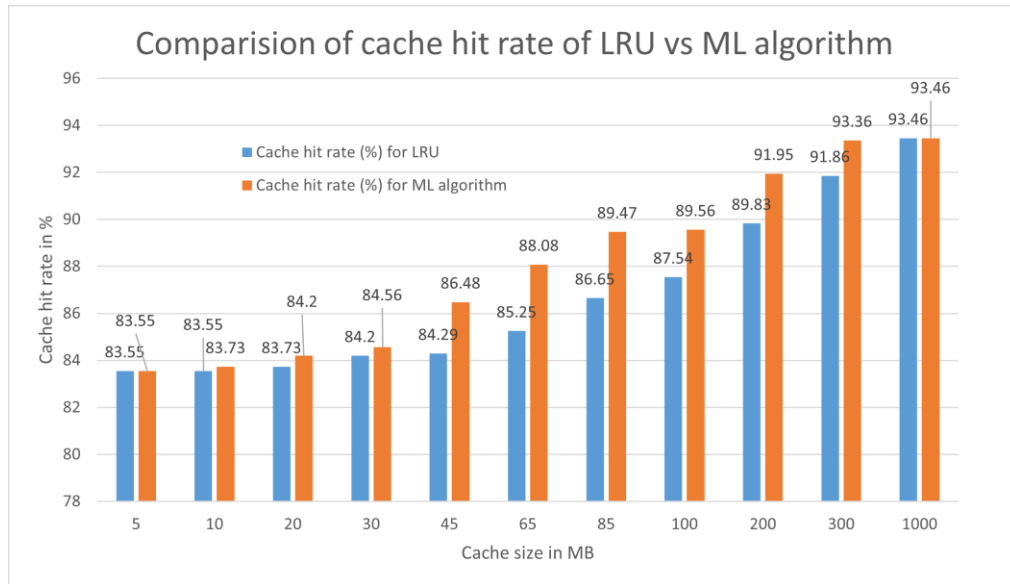


Figure 19: Comparison of Average Cache Hit Rate for LRU and ML algorithm.

The figure 19 shows comparison of cache hit rate of LRU and machine learning algorithm. When cache size is 5MB, the cache hit rate for LRU (least recently used) algorithm, and developed machine learning algorithm is similar (i.e., 83.55%) and almost similar cache hit rate is found for the 10 MB too. With a max tile size of 5 MB, the cache size of 5 MB and 10 MB only allows for a maximum of 2 tiles at full size.

With the tile request order tested, the cache was depopulated in both algorithms evenly due to the low cache size. When cache size is 54 MB, the machine learning model shows 2.19% higher cache hit rate more than the LRU algorithm. For the 100 MB the average cache hit is 89.56% for the machine learning and 87.54% for LRU algorithm, resulting in a difference of 2.2%, favoring our ML algorithm. The highest average cache hit rate, which is 93.36%, is found for machine learning algorithm when cache size is 300 MB whereas the LRU algorithm exhibits 91.86% average cache hit rate. Each model shows same cache hit rate at cache size of 1000 Mb because at this the cache size is big enough that it can load every segment for both algorithms. Between 45 MB to 300 MB the machine learning performs well.

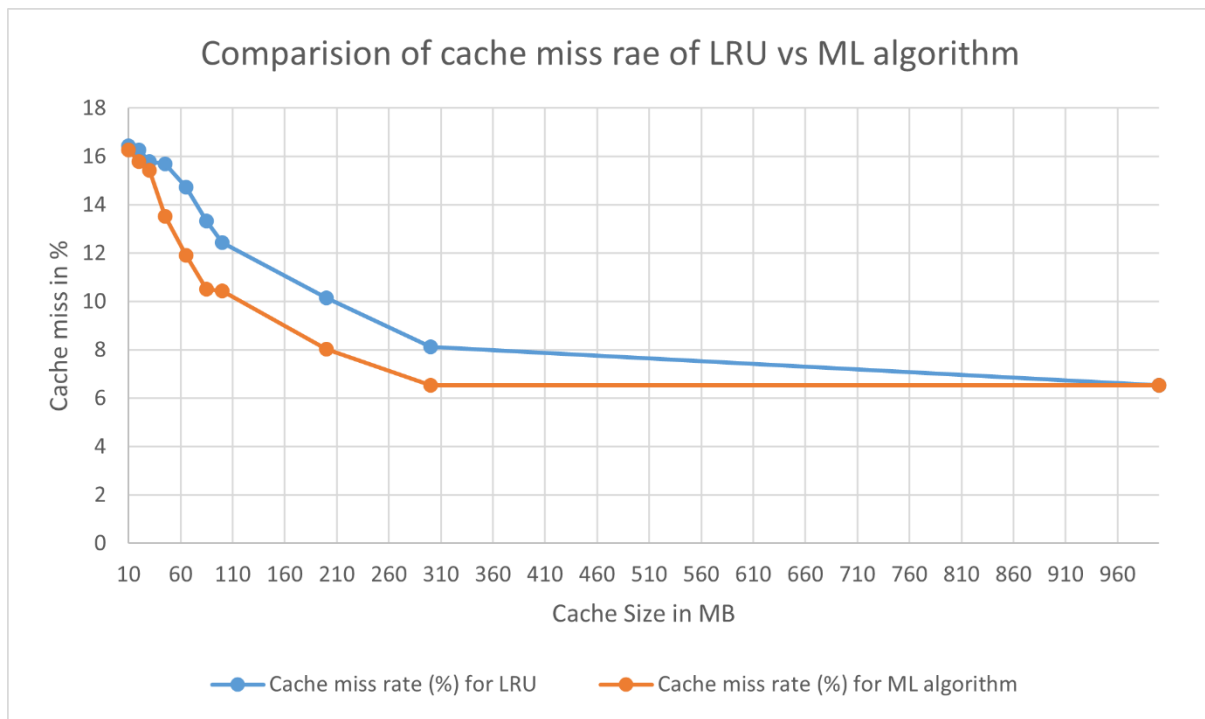


Figure 20: Comparison of Average Cache Miss Rate for LRU and ML algorithm.

Figure 20 illustrates comparison of average cache miss rate between LRU and ML algorithm. The lower average cache miss rate indicates that the machine learning model l performs better than the LRU cache algorithm. The lowest average cache miss rate which is 6,64% is found for the machine learning algorithm whereas the average cache miss rate for the LRU algorithm is 8.14%. With the increases of cache size both algorithm exhibits the lower average cache hit rate.

CHAPTER NINE

DISCUSSION AND FUTURE WORK

9.1 Discussion

In the first of the research, we proposed caching 360-degree video using the LRU and LFU caching algorithms to reduce network latency and provide 360-degree videos to users effectively. According to our testing findings, when the cache size is set to 50%, the average cache hit rate for video 1 shows a nearly 3% improvement over LRU caching when users arrive consecutively. In contrast, for video 2, when the cache size is set to 70%, the average cache hit rate for LFU caching shows a 3.54% improvement over LRU caching. The sequential experiment revealed that the majority of users have similar viewing habits. As a result, it will be simple for future users to save the tiles that were watched by previous users in the cache. According to the results of the randomized trial, LFU caching looks to perform better overall than LRU caching. Additionally, when employing a reduced overall cache size for the LFU caching mechanism, the hit rate performance improvement is much more noticeable. When designing a tiny cache, it is practical to take the LFU caching algorithm into account. By decreasing the number of times video tiles need to be retrieved from the core network, an enhanced rate of cache hits can conserve bandwidth, lower network load, reduce network latency, and cut costs. The outcomes could vary if the scenario were scaled up to include more users, longer videos, and more tiles. In our 2nd part of the research, we'll have the results of the FoV prediction to create an even more effective machine learning-based caching technique for removing unwanted tiles from the cache.

In second part of the research, we developed three machine learning models for caching 360-degree videos, using random forest, linear, and Bayesian regression. For the random forest, linear, and Bayesian regression models, the predictive R² values are 0.79, 0.69, and 0.68, respectively. 79% of predictions on the test dataset came within the threshold range for random forest regression with a threshold value of 2.5 frequency units. On the test dataset, however, 69% of predictions for linear regression and 68% of predictions for Bayesian regression, respectively, came below a threshold value of 2.5. The random forest regression model outperformed the other two models overall.

In the third part of the research, we have utilized our developed our trained machine learning model for the test bench. A Python test bench program was created to assess both methods on the test set while adjusting the cache size in order to compare our machine learning approach with the LRU algorithm. Our Machine learning model removes unwanted tiles from the cache while also improving the cache hit rate at the user end. Performance-wise, the machine learning model outperforms the LRU approach. The model shows a cache hit rate of 93.46% at 300 MB of cache size.

9.2 Future Work

In the future it will be good to set up an actual 360-degree video system which includes a real test bench, a real cache server and main server to cache the tiles requested by the users. The hypothetical test bench might not be 100% similar in the real world. We can explore more features for example point of interest from the video segment and incorporate them in the dataset.

REFERENCES

- [1] J. Dai, Z. Zhang, S. Mao, and D. Liu, “A view synthesis-based 360° VR caching system over MEC-enabled C-ran,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 10, pp. 3843–3855, 2020.
- [2] S. Afzal, J. Chen, and K. K. Ramakrishnan, “Viewing the 360° future: Trade-off between user field-of-view prediction, network bandwidth, and Delay,” 2020 29th International Conference on Computer Communications and Networks (ICCCN), 2020.
- [3] Cisco Visual Networking Index. 2016. Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper. link: <http://goo.gl/yITuVx> (2016).
- [4] G. Papaioannou and I. Koutsopoulos, “Tile-based caching optimization for 360° videos,” *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019.
- [5] Y. Bao, T. Zhang, A. Pande, H. Wu, and X. Liu, “Motion-prediction-based multicast for 360-degree video transmissions,” 2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2017.
- [6] N. Jiang, V. Swaminathan, and S. Wei, “Power evaluation of 360 VR video streaming on Head Mounted Display Devices,” *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2017.
- [7] M. Graf, C. Timmerer, and C. Mueller, “Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video over HTTP,” *Proceedings of the 8th ACM on Multimedia Systems Conference*, 2017.
- [8] M. Hosseini and V. Swaminathan, “Adaptive 360 VR Video Streaming: Divide and Conquer,” 2016 IEEE International Symposium on Multimedia (ISM), 2016.
- [9] P. Rondao Alface, J.-F. Macq, and N. Verzijp, “Interactive Omnidirectional Video Delivery: A Bandwidth-Effective Approach,” *Bell Labs Technical Journal*, vol. 16, no. 4, pp. 135–147, 2012.
- [10] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang, “Pano,” *Proceedings of the ACM Special Interest Group on Data Communication*, 2019.
- [11] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, “Flare,” *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018.

- [12] M. Xiao, C. Zhou, Y. Liu, and S. Chen, "OpTile," Proceedings of the 25th ACM international conference on Multimedia, 2017.
- [13] H. Ahlehagh and S. Dey, "Video caching in Radio Access Network: Impact on delay and capacity," *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, 2012.
- [14] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "DeepCache," Proceedings of the 2018 Workshop on Network Meets AI & ML - NetAI'18, 2018.
- [15] K. Suksomboon *et al.*, "PopCache: Cache more or less based on content popularity for information-centric networking," *38th Annual IEEE Conference on Local Computer Networks*, 2013
- [16] L. Cherkasova, "Improving www proxies performance with greedy-dual-size-frequency caching policy", Hewlett-Packard Laboratories, 1998.
- [17] H. S. Goian, O. Y. Al-Jarrah, S. Muhaidat, Y. Al-Hammadi, P. Yoo and M. Dianati, "Popularity-Based Video Caching Techniques for Cache-Enabled Networks: A Survey," in *IEEE Access*, vol. 7, pp. 27699-27719, 2019.
- [18] W. Chenglei, T. Zhihao, W. Zhi, and Y. Shiqiang, "A dataset for exploring user behaviors in vr spherical video streaming," in Proceedings of the 8th ACM on Multimedia Systems Conference, ser. MMSys'17, pp. 193–198, 2017.
- [19] P. Maniotis, E. Bourtsoulatze, and N. Thomos, "Tile-Based Joint Caching and Delivery of 360° Videos in Heterogeneous Networks," 2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP), 2019.
- [20] L. Sun, Y. Mao, T. Zong, Y. Liu, and Y. Wang, "Flocking-based live streaming of 360-degree video," Proceedings of the 11th ACM Multimedia Systems Conference, 2020.
- [21] V. Kirilin, A. Sundarajan, S. Gorinsky, and R. K. Sitaraman, "RL-Cache: Learning-Based Cache Admission for Content Delivery," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2372–2385, 2020.
- [22] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "DeepCache," Proceedings of the 2018 Workshop on Network Meets AI & ML - NetAI'18, 2018.
- [23] C. Ge, N. Wang, W. K. Chai and H. Hellwagner, "QoE-Assured 4K HTTP Live Streaming via Transient Segment Holding at Mobile Edge," in *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1816-1830, Aug. 2018.

- [24] C. Li, L. Toni, J. Zou, H. Xiong and P. Frossard, "QoE-Driven Mobile Edge Caching Placement for Adaptive Video Streaming," in *IEEE Transactions on Multimedia*, vol. 20, no. 4, pp. 965-984, April 2018.
- [25] G. Papaioannou and I. Koutsopoulos, "Tile-based caching optimization for 360o videos," in *Proc. of the 20th ACM Int. Symp. on Mobile Ad Hoc Networking and Computing, Mobihoc '19*, Catania, Italy, Jul. 2019.
- [26] L. Sun, Y. Mao, T. Zong, Y. Liu, and Y. Wang, "Flocking-based live streaming of 360-degree video," *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020.
- [27] N. Jiang, V. Swaminathan, and S. Wei, "Power Evaluation of 360 VR Video Streaming on Head Mounted Display Devices," *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2017.
- [28] M. Graf, C. Timmerer, and C. Mueller, "Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video over HTTP," *Proceedings of the 8th ACM on Multimedia Systems Conference*, 2017.
- [29] M. Hosseini and V. Swaminathan, "Adaptive 360 VR Video Streaming: Divide and Conquer," *2016 IEEE International Symposium on Multimedia (ISM)*, 2016.
- [30] P. Rondao Alface, J.-F. Macq, and N. Verzijp, "Interactive Omnidirectional Video Delivery: A Bandwidth-Effective Approach," *Bell Labs Technical Journal*, vol. 16, no. 4, pp. 135–147, 2012.
- [31] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: design aspects, challenges, and future directions," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, Sept. 2016.
- [32] J. Poderys, M. Artuso, C. M. O. Lensbl, H. L. Christiansen, and J. Soler, "Caching at the mobile edge: A practical implementation," *IEEE Access*, vol. 6, pp. 8630–8637, Feb. 2018.
- [33] T. X. Vu, S. Chatzinotas, and B. Ottersten, "Edge-caching wireless networks: Performance analysis and optimization," *IEEE Trans. on Wireless Communications*, vol. 17, no. 4, pp. 2827–2839, Apr. 2018.
- [34] S. Zhang, N. Zhang, P. Yang, and X. Shen, "Cost-effective cache deployment in mobile heterogeneous networks," *IEEE Trans. on Vehicular Technology*, vol. 66, no. 12, pp. 11 264–11 276, Dec. 2017.
- [35] L. Sun, Y. Mao, T. Zong, Y. Liu, and Y. Wang, "Flocking-based live streaming of 360-degree video," *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020.

- [36] P. Blasco and D. Gunduz, "Multi-armed bandit optimization of cache content in wireless infostation networks," in Proc. of IEEE Int. Symp. on Information Theory, Honolulu, HI, USA, Jun. 2014.
- [37] S. Muller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," IEEE Trans. on Wireless Communications, vol. 16, no. 2, pp. 1024–1036, Feb. 2017.
- [38] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in Proc. of the 52nd Annual Conf. on Information Sciences and Systems (CISS), Princeton, NJ, USA, Mar. 2018.
- [39] Y. Wei, Z. Zhang, F. R. Yu, and Z. Han, "Joint user scheduling and content caching strategy for mobile edge networks using deep reinforcement learning," in Proc. of IEEE Int. Conf. on Communications Workshops (ICC Workshops), Kansas City, MO, USA, May 2018.
- [40] P. Maniotis and N. Thomos, "Viewport-Aware Deep Reinforcement Learning Approach for 360° Video Caching," in IEEE Transactions on Multimedia, vol. 24, pp. 386–399, 2022.
- [41] G. Rossini and D. Rossi, "A drive into the caching performance of content centric networking," IEEE CAMAD'12, Wkshps., pp. 304–309, 2012.
- [42] I. Psaras, W. K. Chai and G. Pavlou, "Probabilistic in-network caching for information-centric networks," ACM ICN'12, Wkshps., 2012.
- [43] A. Kabir, G. Rehman, S. M. Gilani, E. J. Kitindi, Z. Ul Abidin Jaffri, and K. M. Abbasi, "The role of caching in next generation cellular networks: A survey and research outlook," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 2, 2019.
- [44] H. Chen and Y. Xiao, "Cache access and replacement for future wireless Internet," *IEEE Communications Magazine*, vol. 44, no. 5, pp. 113–123, 2006.
- [45] Y. Zhou, Z. Chen, and S. Liu, "Fast sample adaptive offset algorithm for 360-degree video coding," *Signal Processing: Image Communication*, vol. 80, p. 115634, 2020.
- [46] J. Park et al., "SEAWARE: Semantic Aware View Prediction System for 360-degree Video Streaming," 2020 IEEE International Symposium on Multimedia (ISM), 2020, pp. 57–64.
- [47] https://www.probabilitycourse.com/chapter11/11_2_2_state_transition_matrix_and_diagram.php (Access on 18th April 2022)
- [48] <https://byjus.com/maths/coefficient-of-determination/> (Accessed on 31st January 2021).

Appendix 1

Code for the Machine learning Model (Random Forest Algorithm)

```
import pandas as pd
import numpy as np
import sklearn as SK
data_df=pd.read_csv("/Daraset_revised_2.csv")
data_df.shape

data_df.size

X = data_df.iloc[:,18:33].values
Y = data_df.iloc[:,17].values
print(Y)
print(X)

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test=train_test_split(X,Y,test_size=0.2, random_state=0)

from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor()
rfr.fit(X_train, Y_train)
Y_pred=rfr.predict(X_test)
print(Y_pred)

Y_pred[0:2507]

from sklearn.metrics import r2_score
r2_score(Y_test, Y_pred)

r2_score(Y_test, Y_pred)
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.scatter(Y_test, Y_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual Vs Predicted')
```


Appendix 1(Continued)

```
Difference=Y_test-Y_pred
```

```
Difference=Y_test-Y_pred
```

```
row=len(Difference)
```

```
col = len(Difference[0])
```

```
Y_temp = Y_pred.copy()
```

```
threshold=2.5
```

```
for i in range(row):
```

```
    for j in range(col):
```

```
        if abs(Difference[i][j]) < threshold:
```

```
            Y_temp[i][j] = Y_test[i][j]
```

```
r2_score(Y_test, Y_temp)
```

```
plt.figure(figsize=(15,10))
```

```
plt.scatter(Y_test, Y_temp)
```

```
plt.xlabel('Actual')
```

```
plt.ylabel('Predicted')
```

```
plt.title('Actual Vs Predicted')
```

Code of the test bench:

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Created on Mon Jan 24 15:35:55 2022
```

```
@author: 19033
```

```
"""
```

```
# Imported libraries
```

```
from LRUCache_1 import LRUCache
```

```
import sklearn as SK
```

```
import pandas as pd
```

```
import numpy as np
```

```
import joblib
```

```
from VideoSegment import VideoSegment
```

```
# Creates an LRU cache for caching video tile and segment data
```

```
#cache_store = 0          # Size in MB of filled space in the cache
```

```
cache_size = 1000# Size in MB
```

```
cache = LRUCache(cache_size)
```

Appendix 1(Continued)

```
# Reads testing data file for cache simulation
data = pd.read_csv("C:\\Users\\19033\\OneDrive\\Desktop\\RA-Milon\\algorithm\\Cache-Algorithms-
main\\Cache-Algorithms-main\\CacheTest_1.csv")

# Imports trained RFR machine learning model
joblib_model = joblib.load("rfr.pkl")

# Initializes hit count and miss count to 0
num_users=20
miss_count = 0
hit_count = 0
user_miss=[0] * num_users
user_hit=[0] * num_users
# For each row in the testing data file,
for index in range(len(data)):
    user_num=data.iloc[index, 52]
    user_num=round(user_num)
    # Extracts maximum resolution of the requested tiles
    resolution = data.iloc[index, 35]

    # Extracts features for input to the machine learning algorithm
    X = data.iloc[index, 19:34].values
    X = X.reshape(1, -1)

    # Makes predictions on the provided features and rounds the result to the nearest whole number
    Ypredict = joblib_model.predict(X)

    Ypredict = round(Ypredict[0])
    print(Ypredict)

# Sets requested resolution to full resolution if predicted frequency surpasses threshold
if Ypredict >31:
    segment_size = 5
    #print(segment_size)

# Otherwise, sets the requested resolution to 50%
else:
    segment_size = 2.5
    #print(segment_size)

# Sets item cache key to tile number
key = data.iloc[index, 51]
```

Appendix 1(Continued)

```
# Sets item cache content to segment size
content=VideoSegment(1,segment_size)
print (content.getSize())
#content = segment_size # eventually will replace with actual video size in MB

# Checks if the requested tile is in the cache
tile = cache.get(key)

    # If there is a cache miss, the requested tile is added to the cache
if tile == -1:
    cache.put(key ,content) # Change 'content' to actual video tile in the future
    miss_count += 1        # Increments miss count by 1
    user_miss[user_num-1] +=1
    # Otherwise, increments hit count by one
else:
    hit_count += 1
    user_hit[user_num-1] +=1

# If the new value of cache storage exceeds the cache size,
#if cache_store > cache_size:
    # Adjusts pre-incremented cache_store value to the current value of filled cahce
    # cache_store = cache_store - segment_size

# Prints hit count and miss count
print(hit_count)
print(miss_count)
print(user_hit)
print(user_miss)
cache_hit_sum=0
cache_miss_sum=0
for i in range (num_users):
    cache_hit_rate=(user_hit[i]/(user_hit[i]+user_miss[i]))*100
    cache_miss_rate=100-cache_hit_rate

print(cache_hit_rate)
print(cache_miss_rate)

cache_hit_sum +=cache_hit_rate
cache_miss_sum +=cache_miss_rate
print(cache_hit_sum/num_users) #average cache hit rate for the ..... users
print(cache_miss_sum/num_users) #average cache miss rate for the .... users
```