Deep Learning based Predictive Analytics for Decentralized Content Caching in Hierarchical Edge Networks

by

Dhruba Chakraborty (18101028)
Mahima Rabbi(18101563)
Maisha hossain (18201184)
Saraf Noor Khaled (18141009)
Maria Khanom Oishi (17301029)

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
January 2022

© 2022. Brac University
All rights reserved.

Deep Learning based Predictive Analytics for Decentralized Content Caching in Hierarchical Edge Networks

Submitted by

Dhruba Chakraborty (18101028)
Mahima Rabbi(18101563)
Maisha hossain (18201184)
Saraf Noor Khaled (18141009)
Maria Khanom Oishi (17301029)

Submitted on
20 January,2022

A thesis submitted to the
Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Supervisor:
Md. Golam Rabiul Alam
Co-Supervisor:
Arif Shakil

Department of Computer Science and Engineering
Brac University

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at BRAC University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Dhruba Chakraborty
18101028

Mahima Rabbi
18101563

Maisha hossain
18201184

Saraf Noor Khaled
18141009

Maria Khanom Oishi
17301029

Approval

The thesis/project titled "Deep Learning based Predictive Analytics for Decentralized Content Caching in Hierarchical Edge Networks" submitted by

1. Dhruba Chakraborty (18101028)
2. Mahima Rabbi(18101563)
3. Maisha hossain (18201184)
4. Saraf Noor Khaled (18141009)
5. Maria Khanom Oishi (17301029)

Of Fall, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 20, 2022.

Examining Committee:

Supervisor:
(Member)

Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
BRAC University

Co-Supervisor:
(Member)

Arif Shakil
Lecturer
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Content centric network is a state-of-the-art networking architecture for content distribution and content caching. However, it is inefficient to cache every content in each network device. The modern edge computing technology opens the door for content caching in the edge of the network. However, still we have to decide which contents we should cache and which content we should replace from the cache. Deep learning based predictive analytics can play an important role in selecting contents for caching purposes. In this research, we will use Long short-term memory(LSTM) based Recurrent Neural Network(RNN) for decentralized content caching at the hierarchical edge of the network.

Keywords: Content, Caching, Edge networking, Deep learning, Recurrent Neural Network(RNN), Long short-term memory(LSTM),Decentralized,Hierarchical.

Dedication

We would like to dedicate our thesis to our beloved parents and respected faculty members for whom we are able to successfully come up with the idea and implementation of our research work in the field of technology through the knowledge of Computer Science and Engineering.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.
Secondly, we would like to sincerely thank our honorable thesis supervisor Dr. Md. Golam Rabiul Alam who constantly supported us and guided us through a challenging topic. We were able to overcome all obstacles and hurdles faced through his exquisite recommendations and regular feed backs. Despite an on-going pandemic, Sir always managed to spare time for us, even extremely late at times and we will forever be grateful for the gesture shown to us.
Thirdly, our co-supervison Arif Shakil who has guided us with the paper for the betterment of writing.
Due to this pandemic situation, we have successfully completed our thesis paper because of their immense support.
Also, We would like to mention, one of our dearest senior for his unconditional support throughout the thesis period, Eialid Ahmed Joy.
And finally to our parents without their throughout sup-port it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

Chapter 1

Introduction

Soon after the invention of the first computer ENIAC in 1946, one of the most significant lacking it had was networking. People could do many things with the computer. But, it was impossible to share their works with others who were miles away. From this hunger of sharing, people started to think about making a system by which they could share their works with others. From this consequence, in 1960 ARPANET (The Advanced Research Projects Agency Network) was built in order to create a network with thousands of computers. And, thus the journey of networking had started. Simply, the linking of computers to allow them to operate interactively is networking.

In the very first era of networking, it was just a connection between computers for sharing mostly research data or important files. Only some of the sophisticated researchers and high-level people got to have the benefit of networking. But, in modern times, the concept of networking has changed a lot. Nowadays, there are thousands of fields in networking. Computer networking connects devices and endpoints on a LAN or a wider network, such as the internet or a private wide area network (WAN). This function allows service providers, enterprises, and consumers all over the world to exchange resources, use or give services, and interact. From phone conversations to text messages to streaming video to the internet of things, networking makes everything easier (IoT). People from every stage in society get help from networking in their day to day life. In this context, content has become the most powerful weapon in the networking field. Any content that is stored as digital data is considered digital content. Content, also referred to as digital media, is saved in specific formats on digital or analog storage. Information broadcast, transmitted, or stored in computer files are examples of digital content. People use content to get their job done in their daily life. Starting from media streaming sites, social networking sites, online news portals and many others are spreading digital well being to human beings through content.

Content centric networks are getting richer day by day with the help of thousands of content providing sites and its users. However, this wouldn't have been this rich, if it wouldn't have been efficient. Efficiently caching the contents is so important in networking. Caching a content means fetching the content from the server. It might be any server all over the world. But, that might be problematic

as the server from which the files are being cached, might be far away from the user. That's where efficient content caching comes in handy. In efficient content caching, files get fetched from the closest server. As a result, lots of time gets saved.

However, there is a significant issue when deciding which content we should cache and which we should replace from the cache because of limited cache memory. We need to cache contents that are more important to the users. But, it is harder to decide which content is more important to the user. Again, another issue might be where to cache the files. Whether we would need to cache the files in the regional cache or the central cache. To make the purpose easier, we can use deep learning based predictive analytics. Predictive analytics can help us to decide which file to cache and which file to replace from the cache depending on its importance. Moreover, predictive analytics helps us to cache the contents in a decentralized way. Thus, the prediction will run on the edges but the central cache will benefit from that.

Predictive analytics is essential for storing content at the network's edge efficiently. Businesses may be proactive and forward-thinking using predictive analytics, predicting events and behaviors based on data rather of guesswork or assumptions. The same can be said of content caching. Information providers can use predictive analytics to cache the most popular content at the network's edge, allowing users to access it faster and with less latency.

1.1 Research Problem

With [6] the mass availability of devices like mobile phones, computers etc. the use of the internet is increasing rapidly day by day. And, content providing sites like YouTube, Netflix, Prime Video etc. are becoming so popular among the users. However, people want to stream their contents faster from the sites with less latency. If the requested files are available on the cache server, they are delivered to the users extremely faster. Which is why caching is necessary. Assume a [6] Netflix subscriber in London wants to stream the show House of Cards. To ensure fast access and minimum buffering time, Netflix copies the videos from their origin servers in Los Gatos, CA, to the caching server closest to London. Because of this, all subscribers in London can quickly access the show and avoid a transatlantic file transfer. However, it is impossible to keep every movie in the closest caching server of London because of space limitation. To save the space of the cache server, the not so popular movies are needed to be replaced from the cache server with new ones. As a result, there comes a decision between what movies to keep and what movies need to be replaced from the cache. Therefore, a question might arise:
"Which contents we should cache and which contents we should replace?"
This research will answer the above question through the usage of Deep Learning based Predictive Analytics Algorithm (in our case, LSTM based RNN).

1.2 Research Objectives

We are going to build a system using deep learning based predictive analytics so that we can decide which contents need to be cached and which contents need to be replaced from the server. The contents that are trending should be kept in the cache and others should be replaced from the cache. The objectives of the research are:

- To understand, what content caching is and how it works
- Importance of efficient content caching and its mechanism
- Importance of edge computing and edge network in efficient content caching
- To develop a model for connection between predictive analytics and efficient content caching
- To evaluate the model

Chapter 2

Literature Review

As the blessings of modern technologies like mobile phones, tablets, computers etc. are becoming more affordable and easier to get, people are getting more and more used to these devices' day by day. And, people are getting more comfortable with content providing sites like Netflix, YouTube, Prime Video and so on. And, the number of users is rapidly increasing day by day. According to [14] Statista, the number of Netflix users in 2020 is 195.15 million by Q3. However, in a recent article of [16] TNPS (The New Publishing Standard), in 2030 the number of Netflix users is expected to increase up to 500 million. There might be one problem with the loading time of the contents that are far away from the user. To solve that issue, the concept of caching comes in handy. But, the amount of cache memory is limited. That's why there is a trade-off between which content to cache and which to replace. To efficiently cache data, predictive analytics is so necessary.

2.1 Decentralized Content Caching

Content caching is a performance optimization mechanism in which data is delivered from the closest servers for optimal application performance. According to [3] 'interserver', when a system accesses the website, the contents in that site will be provided by a nearby cache server rather than the original server which is remote. As a result, it will decrease the latency. However, while caching the movies on the central cache server, a large amount of computational power is needed to predict the movies. Moreover it takes a lot of time to predict the movies for the central cache. So, some solution is needed to cache the contents on the central cache with least computational power. That's where decentralized content caching comes in. Decentralization helps to predict the movies on the regional cache servers and use the prediction to cache the movies on the central cache server. Again, it is impossible to cache each and every content in the cache server. That's why efficient content caching is needed too. In efficient content caching, most popular contents are cached and least important contents get replaced from the cache server. Decentralization and efficient content caching reduces server traffic and the performance of the application gets improved.

2.2 Edge Computing and Edge Network

Edge networking is a distributed computing paradigm that brings computation and data storage as close to the point of request as possible in order to deliver low latency and save bandwidth. However, [19] edge computing is a modern technology on data center and cloud computing architectures to help create efficiencies. Edge computing is significantly important outside the cloud, at the edge of the network, and more significantly in applications where real-time data processing is required. Due to the proximity of the analytical resources to the end users, sophisticated analytical tools and Artificial Intelligence tools can run on the edge of the system. According to [7] 'The Emergence of Computing', this placement at the edge helps to increase operational efficiency and contributes many advantages to the system.

2.3 Predictive Analytics

Predictive analytics is the use of data, statistical algorithms and machine learning techniques to identify the likelihood of future outcomes based on historical data. The goal is to go beyond knowing what has happened to providing a best assessment of what will happen in the future [21]. Though predictive analytics has been around for decades, it's a technology whose time has come. More and more organizations are turning to predictive analytics to increase their bottom line and competitive advantage. According to PredictiveAnalyticsToday [15], it uses a number of data mining, predictive modeling and analytical techniques to bring together the management, information technology and modeling business process to make predictions about the future. The patterns found in historical and transactional data can be used to identify risk and opportunities for the future.

2.4 Related Works

This part aims to critically review previous relevant works in the field of Predictive Analytics in the context of Efficient Content Caching at edge networks. Observing different techniques used in different relevant research works, we found many challenges in efficiently caching the contents through prediction.

Content caching on the edge of the network is so important because if not cached, the data will be accessed by the user directly from the main server through the cloud. Which will increase the latency. According to [4], popular content and objects can be stored and served from edge locations, which are closer to the end users. This operation is also beneficial from the end user perspective since edge caching can dramatically reduce the overall latency to access the content and increase the sense of overall user experience.

Again, edge computing is another factor in terms of content caching. According

to [10], using the cloud as a centralized server simply increases the frequency of communication between user devices, such as smartphones, tablets, wearable and gadgets, referred to as edge devices, and geographically distant cloud data centers. This is limiting for applications that require real-time response. Hence, there has been a need for looking 'beyond the clouds' towards the edge of the network, referred to as edge computing. Computing on edge nodes closer to application users could be exploited as a platform for application providers to improve their service. Although, the cache memory at the edge of the network is limited. So, we have to make a decision about what content to cache and what content to replace from the cache. That's where deep learning based predictive analytics comes in useful.

Recurrent Neural Network (RNN) is significantly useful for solving the efficient content caching prediction problem because it not only utilizes the current state but also uses the previous state data using sequence. According to [4], Unlike the hidden neuron in FNN, the output of RNN depends on both the current output of the previous layer and the last hidden state. However, using RNN might not be appropriate in some cases as there might be data vanishing gradient problems. Recurrent Neural Net- works work just fine when we are dealing with short-term dependencies[8]. To solve that issue, LSTM (Long Short-Term Memory) comes handy. Long Short- Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems[5]. According to [10], LSTM models are quite popular due to their special design property related to carefully avoiding vanishing and exploding gradient problems when building deep layer neural network models. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things[8].

Hierarchical LSTM that considers both check-in time and event taxonomy structure from check-in sequences to provide accurate predictions on a user's future check-in location category. Each category is also projected into an embedding via hierarchical LSTM, resulting in new representations with greater semantic implications. The efficiency of the suggested Hierarchical LSTM is set to be demonstrated by experimental results that Hierarchical LSTM increases Accuracy by 4.22 percent on average, and Hierarchical LSTM learns a superior taxonomic representation for clustering categories, culminating in a 1.5X increase in Silhouette Coefficient.[12]

Several cloud-based apps use a data center as a centralized computer to analyze data from edge devices like smartphones, tablets, and wearable. This strategy puts ever-increasing requirements on communication and computing resources, thereby lowering Quality of Service and Experience. Edge Computing is based on the idea of transferring part of this computing burden to the network's edge in order to take use of computational capabilities that are presently underutilized in edge nodes such base stations, routers, and switches. This position paper examines the difficulties and possibilities that come as a result of this new computing path.[7]

2.5 Loss functions

The error (also known as "the loss") between both the output of our methods and the supplied goal value is calculated using loss functions. The loss function, in layman's terms, describes how far our estimated output is off the mark.[17] It's a way of determining how effectively your algorithm models the data.[9] Loss functions are used in optimization problems in order to reduce the loss. Loss functions are used in regression to find the best fit line by reducing the total losses of all the points that fall inside the line's prediction. Loss functions are used to control how perception and neural network weights are altered during training. The magnitude of the loss is proportional to the size of the update. The accuracy of the model is improved by lowering the loss.However, in these machine learning applications, the trade off between update size and low loss must be considered.[17]

2.6 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems[5]. Long Short Term Memory networks, or "LSTMs," are a kind of RNN that can learn long-term dependencies. LSTMs are specifically developed to prevent the problem of long-term reliance. They don't have to work hard to remember knowledge for lengthy periods of time; it's like second nature to them[1].LSTM networks are well-suited to classifying, processing and making predictions based on time series data[18]. LSTM is an RNN architecture specifically designed to address the vanishing gradient problem [5]. LSTM works tremendously well on a large variety of problems, and are now widely used [1].

Here is the struc- ture of the Long Short-Term Memory(LSTM) unit which shows its workflow. A cell, an input gate, an output gate, and a forget gate are the components of an LSTM unit. The three gates control the flow of information in and out of the cell, and the cell remembers values across arbitrary time periods. A LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

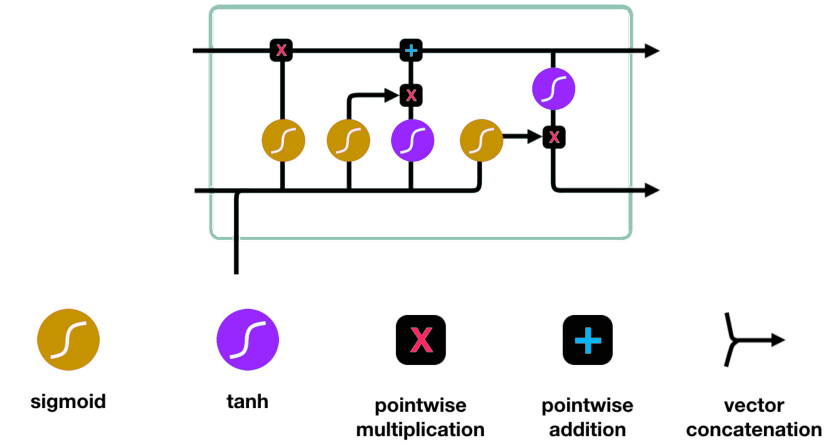


Figure: Structure of a LSTM Unit [11]

Now,flowchart of LSTM has been given here to illustrate the work process and its steps:

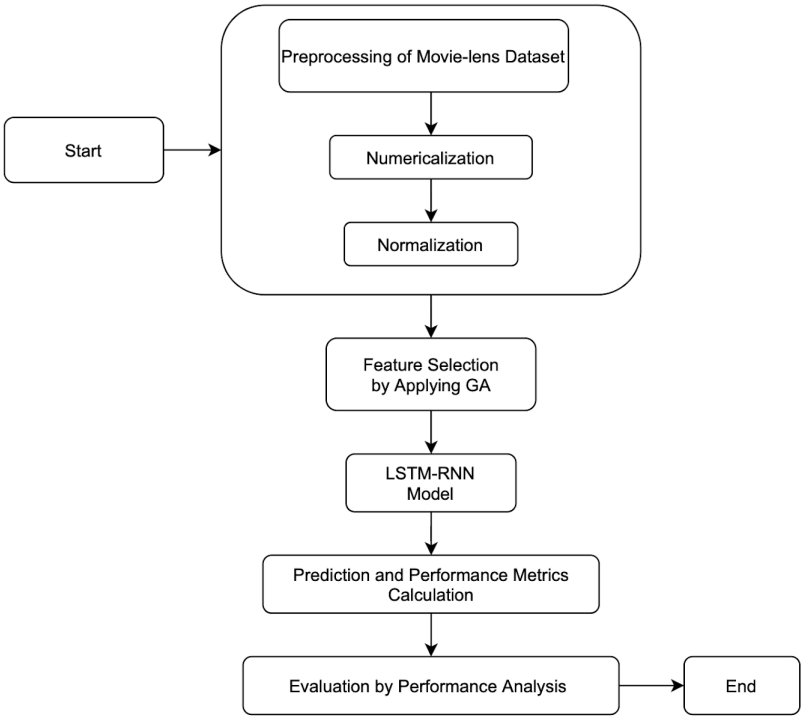


Figure: Flowchart of LSTM

2.6.1 Understanding LSTM

The LSTM cell is similar to a conveyor belt. With only a few tiny linear interactions, it flows straight down the entire chain. It's incredibly easy for data to

simply travel along it unaltered. Gates allow information to pass across the cell if desired. A sigmoid neural net layer plus a pointwise multiplication operation make them up. The sigmoid layer produces integers ranging from zero to one, indicating how much of each component should be allowed to pass. "Allow nothing through!" signifies a value of zero, while "let everything through!" means a value of one. Three of these gates are present in an LSTM to protect and govern the cell state. The first stage in our LSTM is to decide which information from the cell state will be discarded. The "forget gate layer," a sigmoid layer, makes this judgment. The LSTM layer checks h_{t1} and x_t and gives a value between 0 and 1 for each value in the cell state C_{t1}. A value of 1 indicates to completely keep this and 0 indicates to completely delete this.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

The next stage is to figure out what new data we'll store in the cell state. There are two components to this. The "input gate layer," a sigmoid layer, chooses which values we'll update first. Then a tanh layer generates a vector of new candidate values. In the next step these two layers get merged to create an update to the state.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$C' = tanh(W_C.[h_{t-1}, x_t] + b_C)$$

It's now time to switch from C_{t1} to C_t. It multiplies the previous state by ft, forgetting what it had previously opted to ignore. it*Ct is then added. This is the new candidate values, scaled according to how much each state value was updated.

$$C_t = f_t * C_{t-1} + i_t * C'_t$$

Finally, we must determine what we will produce. This output will be based on the state of our cells, but it will be filtered. First, we run a sigmoid layer to determine which aspects of the cell state will be output. The cell state is then passed through tanh (to force the values to be between 1 and 1) and multiplied by the output of the sigmoid gate, resulting in only the parts we choose to output.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

However, the above-mentioned LSTM is a very basic one. There are a lot more types of LSTMs. Another type of LSTM can was created by adding 'peephole connections' to all the gates.

$$f_t = \sigma(W_f.[C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i.[C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o.[C_{t-1}, h_{t-1}, x_t] + b_o)$$

Another type of LSTM uses coupled forget and input gates. It makes the forget and keep decisions together.

$$C_t = f_t * C_{t-1} + (1 - f_t) * C'_t$$

However, a much updated version of LSTM, Gated Recurrent Unit or GRU was introduced in 2014. It's a single "update gate" that combines the forget and input gates. It also modifies the cell state and hides the state, among other things. The resulting model is easier to understand than ordinary LSTM models, and it is gaining popularity.

$$z_t = \sigma(W_z.[h_{t-1}, x_t])$$
$$r_t = \sigma(W_r.[h_{t-1}, x_t])$$
$$h'_t = tanh(W.[r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t + h'_t$$

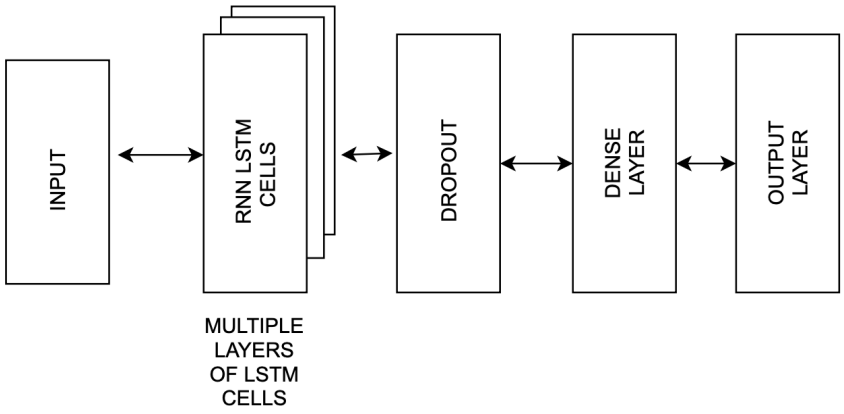


Figure: Various layers of the RNN model layer as used in this paper

Chapter 3

Methodology

The aim of using predictive analytics for decentralized content caching at the hierarchical edge network is to cache the most popular contents at the edge of the network and thus decrease the latency. With a view to doing so, the model requires designing a process that takes data from the activity of the users as an input. Then it systematically processes the input data and outputs either of the different results: 'stream from the caches' or 'stream from the cloud' or 'don't cache'. The below figure provides a generalized view of the model design for the first time slot:

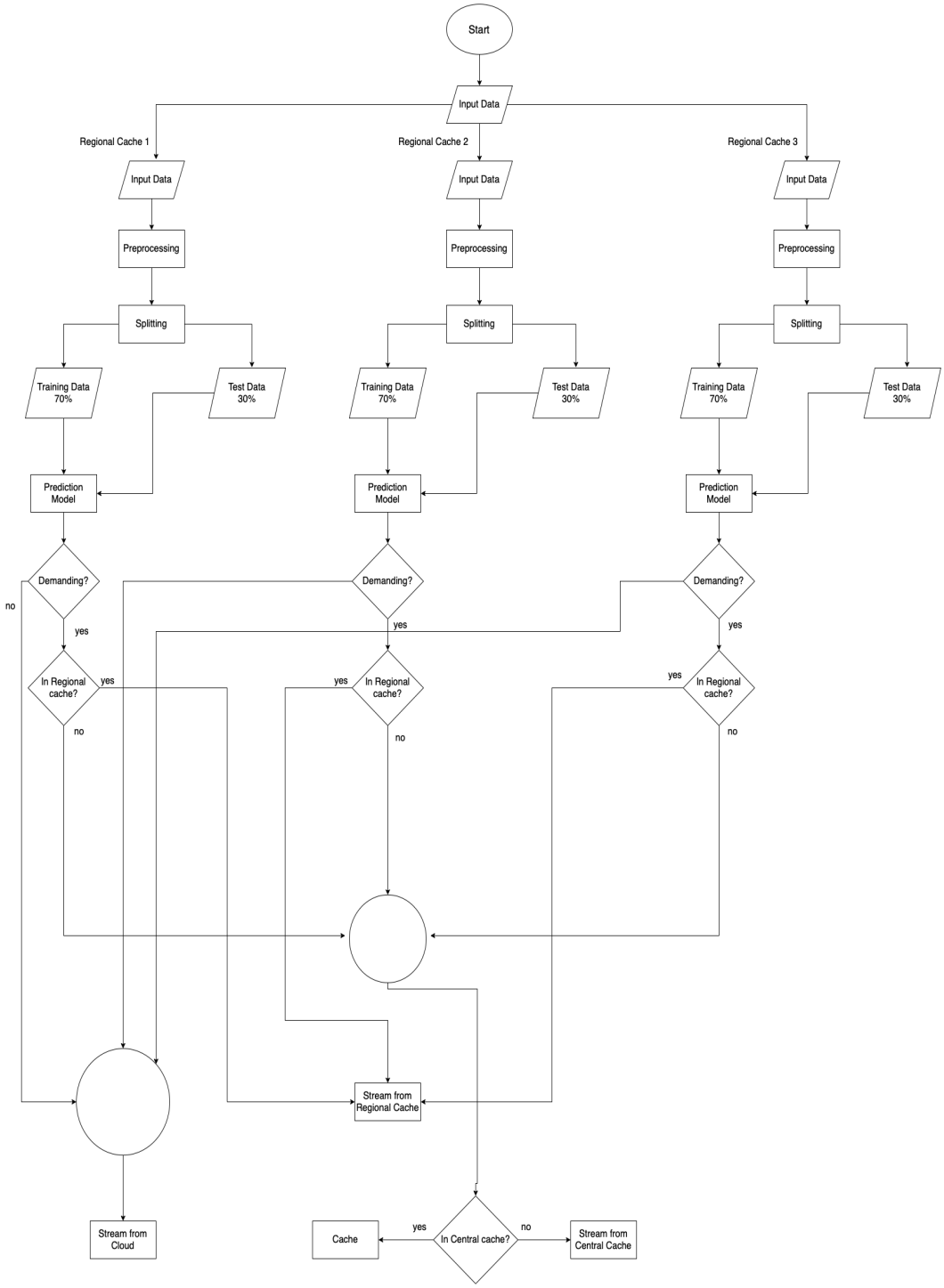


Figure 3.1: The flow chart of the predictive analytics model

We are using LSTM based Recursive Neural Network (RNN) for solving our problem. We could use other Deep Learning based models for this work. But, unlike many other algorithms, LSTM based RNN remembers the previous sequence by keeping them in memory. As a result, the output gets more and more accurate day by day. The workflow will be done in the following stages:

1. Input data:
   In this stage the program takes activity data from the users as input.

2. Input data pre-processing: In this stage the input data gets formatted in such a way that LSTM can use it to process easily.

3. Splitting: In this stage the formatted input data gets split into two parts. One is for training and another is for testing. We have taken 70% for training and 30% for testing. The model will get trained based on the training data and we will check the accuracy of the model based on the testing data.

4. Predictions: In this stage prediction model is used for prediction to decide whether to cache or replace.

5. Cache or Replace: In this stage the system will decide whether to cache or replace a movie based on their predicted hit count.

3.1 System Architecture

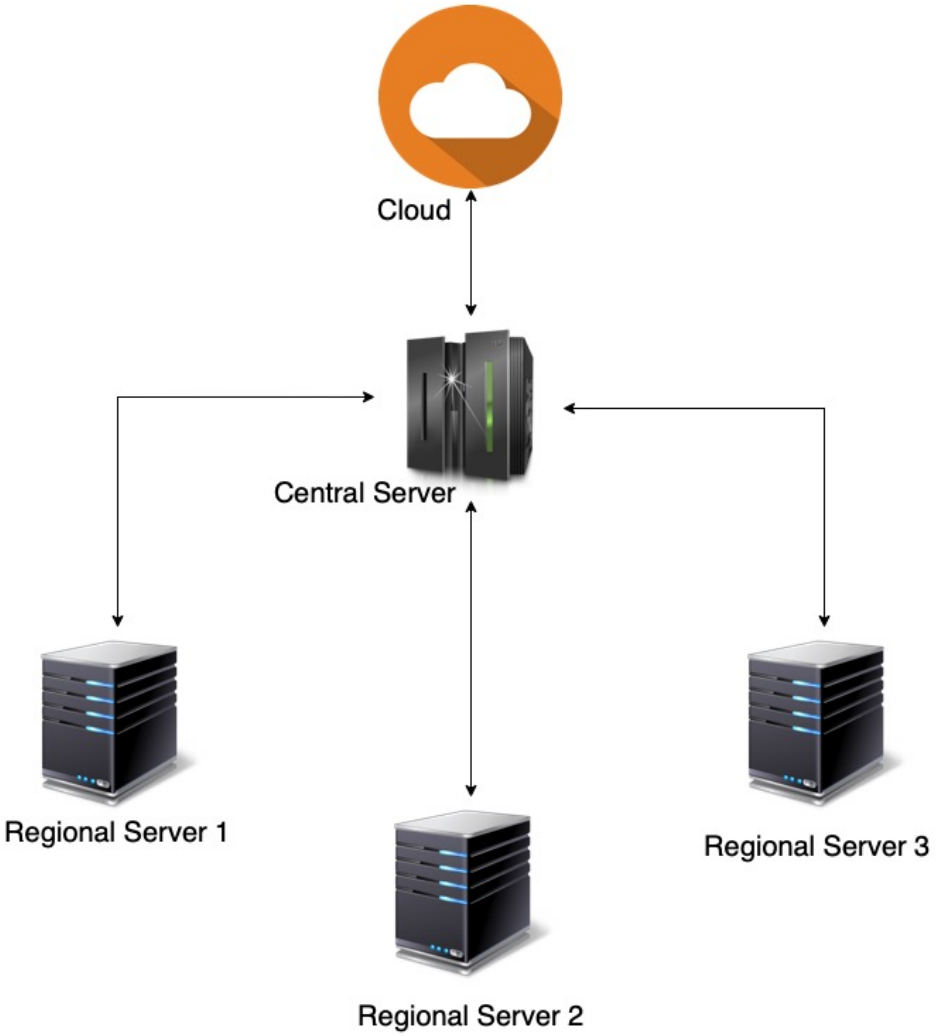


Figure 3.2: System Architecture

For an efficient content centric caching, we needed better prediction which we achieved using a multi-layered system which can be seen in the figure above. In our proposed system, we have 2 layers that are interconnected and highly

optimised for efficient content caching. The first layer or the Regional Servers have the lowest hop count from the end users. Which means that the regional servers have the lowest latency with respect to the end users. In the second and the final layer or the central cache server, is the closest server to the cloud.

3.1.1 Regional Cache Servers

In the very beginning of the caching time-frame, the most demanding movies are stored in the regional cache servers based on the highest predicted hit counts. The process gets repeated on all the regional cache servers. After filling up all the cache servers with their maximum capacity, they send back the remaining popular movies to the central server. The rest of the caching happens on the central server.

3.1.2 Central Cache Servers

In the beginning stage, movies with predicted hit counts come from the regional cache servers. The most popular movies get stored in the central cache server with the maximum capacity and the rest are dropped.
In the next time frame, when a new movie comes, the system checks if the movie is available in the regional cache server or not. If the movie is available, then it simply streams from the cache server at a faster speed. If the movie is not available in the regional cache, it looks for the movie into the central cache server. If the movie is available on the central server, it will stream from there. If the movie is not available in the central cache server too, then it will cache the movie to the regional cache server or the central cache server based on the predicted hit count value of the movie. Thus, the regional cache server and the central cache server makes the streaming job much faster.

3.2 Model Architecture

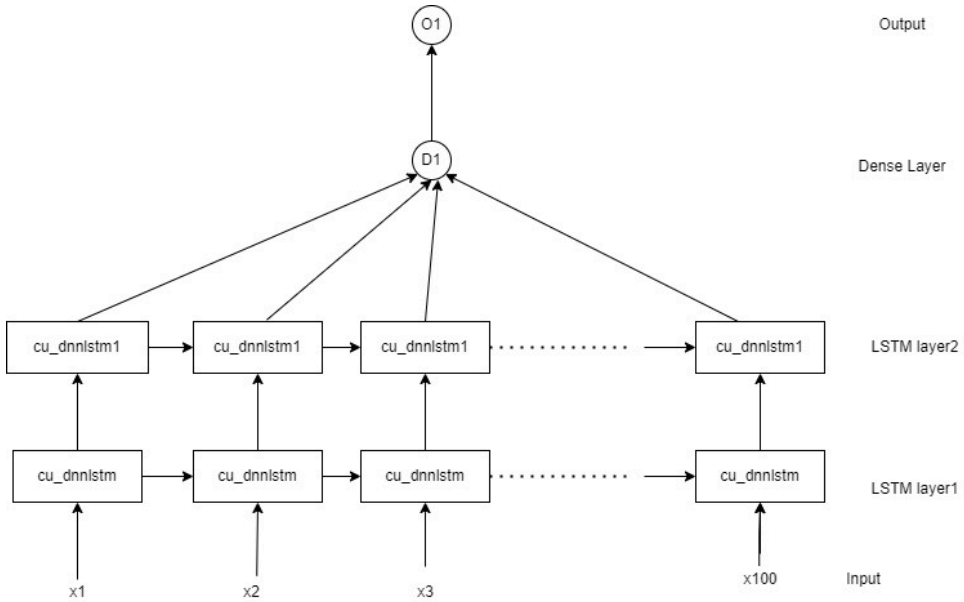


Figure 3.3: Architecture of our Model

We have built our LSTM model to predict hit counts of movies in a time series manner. The LSTM model is a many-to-one RNN model because our model takes movieID and timestamp as input and gives us hit count as predicted output. As LSTM is a recurrent neural network model, we converted our dataset into a series and for our model's input we are feeding sequences. In the model we have 100 LSTM layers (here we have used CuDNNLSTM as it executes much faster with the help of GPU), 1 dropout layer and 1 dense layer. We haven't used redundant dense layers because they use too much computational power. In the dense layer there is only one cell. Because of the one cell in the dense layer the output layer also has a single output. The output is predicted hit count for a particular movie at a particular timestamp. Based on the predicted hit count, we can know how many times the movie has been watched. And, thus we can determine which movie to cache and which to replace.

3.3 Dataset Description

In predictive analytics, the most important tool is data. To know what to cache and what not to cache, we need a lot of data based on the user's ratings. That is why we have chosen MovieLens dataset which consists of various datasets among which, we will primarily be using movies and ratings datasets. The movie dataset consists of movie id, title, genres and rating dataset consists of userid, movie id, rating and timestamp. It contains 27753444 ratings and 1108997 tag applications across 58098 movies. These data were created by 283228 users between January 09, 1995 and September 26, 2018. This dataset was generated on September 26, 2018 [2]. This dataset can be downloaded from [20].

3.4 Dataset Pre-processing

Data preprocessing is a data mining technique to turn the raw data gathered from diverse sources into cleaner information that's more suitable for work. In other words, it's a preliminary step that takes all of the available information to organize it, sort it, and merge it [13].

- Data set clean: Not all the data of a data set are necessary for each and every research. For that reason, data set cleanup is necessary for preparing the data for pre-processing. Data cleaning is required for smooth noisy data and standardizing the data. By cleaning up the data set movies, we are categorizing the genres into integer values and adding another field called release date which gets derived from the title field. In the case of rating data set, we are sorting the data set by userId keyword. Also, a field called daily (seconds a day).

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy

userId	movieId	rating	timestamp
1	307	3.5	1260000000
1	481	3.5	1260000000
1	1091	1.5	1260000000
1	1257	4.5	1260000000
1	1449	4.5	1260000000

Table: Before Data Cleanup

	movieId	title	genres	releaseDate
0	1	Toy Story (1995)	1.00	1995
1	2	Jumanji (1995)	2.00	1995
2	3	Grumpier Old Men (1995)	3.00	1995
3	4	Waiting to Exhale (1995)	4.00	1995
4	5	Father of the Bride Part II (1995)	5.00	1995

timestamp	userId	movieId	rating	daily
23237827	237556	21	3	9140
5510411	56769	1176	4	9140
23237876	237556	1079	3	9140
23237833	237556	47	5	9140
23096009	236139	28	5	9524

Table: After Data Cleanup

- Then we are joining the data sets (movies, rating) that are coming from our previous step, data set cleanup.
- Then we are dividing the data set into three parts for building prediction models on three regional cache servers. However, to reduce redundancy, we are showing output from only the first part of the three.
- Then we are sorting our joined data sets in ascending order based on timestamp, userId, movieId.

	timestamp	userId	movie Id	rating	daily	tstamp_hour	tstamp_day	tstamp_year	genres	releaseDate
0	23237827	237556	21	3	9140	6455	269	1	17	1995
2	23237876	237556	1079	3	9140	6455	269	1	38	1988
3	23237833	237556	47	5	9140	6455	269	1	30	1995
51498	23193844	237134	21	5	9609	6443	269	1	17	1995
51499	23193845	237134	150	5	9609	6443	269	1	58	1995

Table: Before Sorting

	timestamp	userId	movieId	rating	daily	tstamp_hour	tstamp_day	tstamp_year	genres	releaseDate
64	23184175	237014	1	3	9618	6441	269	1	1	1995
65	23184176	237014	4	4	9618	6441	269	1	4	1995
66	23184177	237014	10	3	9618	6441	269	1	9	1995
63	23184178	237014	11	5	9618	6441	269	1	4	1995
62	23184179	237014	19	1	9618	6441	269	1	5	1995

Table: After Sorting

- After sorting the datasets, we are creating label from tstamp day and movieId. At the very beginning we are creating a merged string with tstamp day and movieId separated by '-'. Then we are counting how many times the merged string is there in the data set. This indicates to us the hit count of the particular movie thus we can know the popularity of that movie.

	timestamp	userId	movieId	rating	daily	tstamp_hour	tstamp_day	tstamp_year	genres	releaseDate
0	23237827	237556	21	3	9140	6455	269	1	17	1995
1	5510411	56769	1176	4	9140	1531	64	1	93	1991
2	23237876	237556	1079	3	9140	6455	269	1	38	1988
3	23237833	237556	47	5	9140	6455	269	1	30	1995
4	23096009	236139	28	5	9524	6416	268	1	15	1995
5	2619774	26999	60	4	9524	728	31	1	2	1995
6	23096013	236139	58	5	9524	6416	268	1	4	1994

Table: Before Preprocessing

	timestamp	userId	movieId	rating	daily	tstamp_hour	tstamp_day	tstamp_year	genres	releaseDate	label
0	23237827	237556	21	3	9140	6455	269	1	17	1995	18
2	23237876	237556	1079	3	9140	6455	269	1	38	1988	1
3	23237833	237556	47	5	9140	6455	269	1	30	1995	23
51498	23193844	237134	21	5	9609	6443	269	1	17	1995	18
51499	23193845	237134	150	5	9609	6443	269	1	58	1995	40

Table: After Preprocessing

- After the preprocessing, we are only keeping the movieId, tstamp day and label in our final data sets because the other entities are not useful for our use case. So, we are dropping the rest.

	timestamp	userId	movieId	rating	daily	tstamp_hour	tstamp_day	tstamp_year	genres	releaseDate	label
0	23237827	237556	21	3	9140	6455	269	1	17	1995	18
2	23237876	237556	1079	3	9140	6455	269	1	38	1988	1
3	23237833	237556	47	5	9140	6455	269	1	30	1995	23
51498	23193844	237134	21	5	9609	6443	269	1	17	1995	18
51499	23193845	237134	150	5	9609	6443	269	1	58	1995	40

Table: Before Dropping

	movieId	tstamp_day	label
0	21	269	18
2	1079	269	1
3	47	269	23
51498	21	269	18
51499	150	269	40

Table: After Dropping

Chapter 4

Implementation

This section describes the implementation of the proposed model for predicting contents for the users in the edge of the network. This model was implemented and tested using Jupyter Notebook. The implementation of the model consists of dataset collection, input data pre-processing and testing. Among them, we have already described the first two parts previously. Now, we are only describing the testing part. This section also delivers the result of running the implementation of the pro- posed model for predicting contents. Jupyter Notebook is used to run the test file. Jupyter Notebook is a powerful tool for running python codes. We could have used languages like Java or C. However, python is much more efficient and much less time consuming compared to those languages. Also, most of the ma- chine learning libraries are easily accessible compared to Java or C. That is why we have chosen python as our primary programming language.

4.1 Constructing the Model

The proposed model consists of three files. The files are described in table below:

File Name	Description
dataset_preprocessing.ipynb	Cleans up the dataset and categorises genres into integers, Joins two datasets (movies.csv and ratings.csv), labels up the data and prepares for applying prediction based algorithms, Ascendingly sorts the preprocessed dataset based on timestamp, userId, movieId.
Thesis_Draft.ipynb	Applies LSTM on the dataset and provides prediction models.
Content_Caching.ipynb	Caching and replacing movies using prediction models provided by Thesis_Draft.ipynb for caching on regional cache servers and the central cache server.

In the very beginning we are importing the three datasets that were produced in the dataset preprocessing phase. After importing the datasets, we are dropping the duplicate values as the duplicate values will cause an under fitting problem in our models. Then we are splitting the datasets into train and test datasets. We are taking 70% for the train and 30% for the test dataset. After splitting

the datasets, we are reshaping the train and test datasets because we will need to transform our datasets in a shape so that our model grants them. In the next step we are creating our LSTM model. Here we have a 64 cell LSTM layer, a 32 cell LSTM layer, a dropout layer and finally a 1 cell dense layer where we used rectified linear activation function or ReLU as activation function. To compile the model we have used adam optimizer and mean squared error as our loss function. In the dense layer we have only used one cell because our LSTM model is many to one and we want a single output in the output layer. Thus the model takes movieId and timestamp day as input and outputs a single output which is hit count. Then we are saving our model for future caching.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 cu_dnnlstm_2 (CuDNNLSTM)     (None, 2, 64)             17152

 cu_dnnlstm_3 (CuDNNLSTM)     (None, 32)                12544

 dropout_1 (Dropout)          (None, 32)                0

 dense_1 (Dense)              (None, 1)                 33

=================================================================
Total params: 29,729
Trainable params: 29,729
Non-trainable params: 0
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 cu_dnnlstm (CuDNNLSTM)       (None, 2, 64)             17152

 cu_dnnlstm_1 (CuDNNLSTM)     (None, 32)                12544

 dropout (Dropout)            (None, 32)                0

 dense (Dense)                (None, 1)                 33

=================================================================
Total params: 29,729
Trainable params: 29,729
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 cu_dnnlstm_2 (CuDNNLSTM)     (None, 2, 64)             17152

 cu_dnnlstm_3 (CuDNNLSTM)     (None, 32)                12544

 dropout_1 (Dropout)          (None, 32)                0

 dense_1 (Dense)              (None, 1)                 33

=================================================================
Total params: 29,729
Trainable params: 29,729
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.1: Summary of the LSTM Models

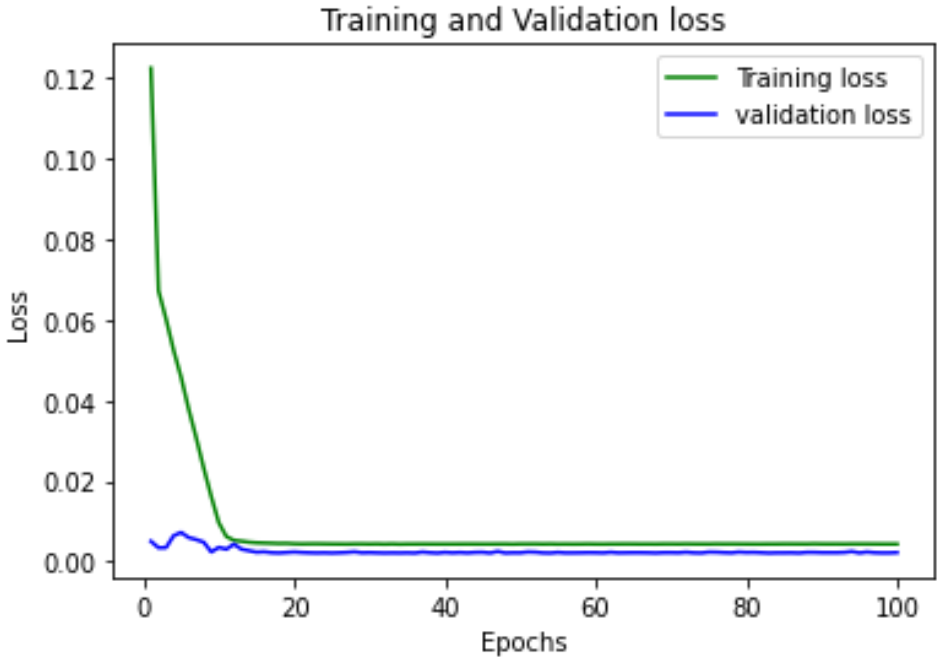
4.2 Content Caching and Replacing

In the very beginning of the caching, we loaded 3 models that were saved in the implementation stage. For real life scenarios, in spite of using the test data of the existing dataset, we have created a synthetic dataset made with 60 movies and corresponding movie size for those movies. Then we have trained the 3 models individually with the same dataset which refer to 3 regional cache servers. We have 15000 MB space for each regional cache server and 25000 MB space for the central cache server. Then we ran knapsack on all the regional servers using predicted hit count as value and movie size as weights. The items returned by the knapsack algorithm are sent to the edges respectively.
After fulfilling the regional caches with their maximum capacity, the rest of the movies from all the regional caches that could not be cached will be sent to the central cache server. However, there is also a storage scarcity. The central cache server can only store 25000 MB movies. So, we need to choose which movie to cache. To ensure the most popular movies on the central server, we ran knapsack with the regional cache excluded movies and their predicted hit counts. However, there are some movies which are in more than one regional cache server. For those movies, we have summed up the predicted hit counts from the common regional caches. After running knapsack on the central server, the returned movies will be stored along with their hit counts.
In the second time slot, when there will be a new movie request by a user in a particular region, the system will search for the movie in the regional cache server. If it finds the movie in the regional cache, it will simply load the movie from the regional cache server much faster compared to the cloud.
If the movie is not in the regional cache, it will search in the central cache server. If the movie is available in the central cache, it will stream the movie from there. It will be slower than regional cache but still much faster than cloud. However, if the movie is not in the central cache too, then the model will run with movieId and corresponding timestamp and it will output a predicted hit count. Then it will look into the central cache. If the predicted hit count is greater than any of the movies in the central cache, then the movie will be replaced by the new movie. However, if the hit count is less than all the movies, then the movies won't be cached.

Chapter 5

Result Analysis

The main goal of our prediction model was to predict the hit count of the movies so that we can accurately predict the popularity of the movies in an autonomous way. Predicting the hit count accurately helps us efficiently cache most popular movies on the regional and central cache servers. And, our prediction result here is highly satisfactory. We ran our models with Adam optimizer and mean squared error loss function individually on all three regional cache servers respectively. After completing 100 epochs on each of the regional servers, the prediction accuracy was 99.79%, 99.70% and 99.49% on the three servers respectively.
By evaluating the three LSTM models we can see that in model 1, the training loss value is 0.0022 and the validation loss is 0.0043. Below is the the training loss vs validation loss graph for model 1:

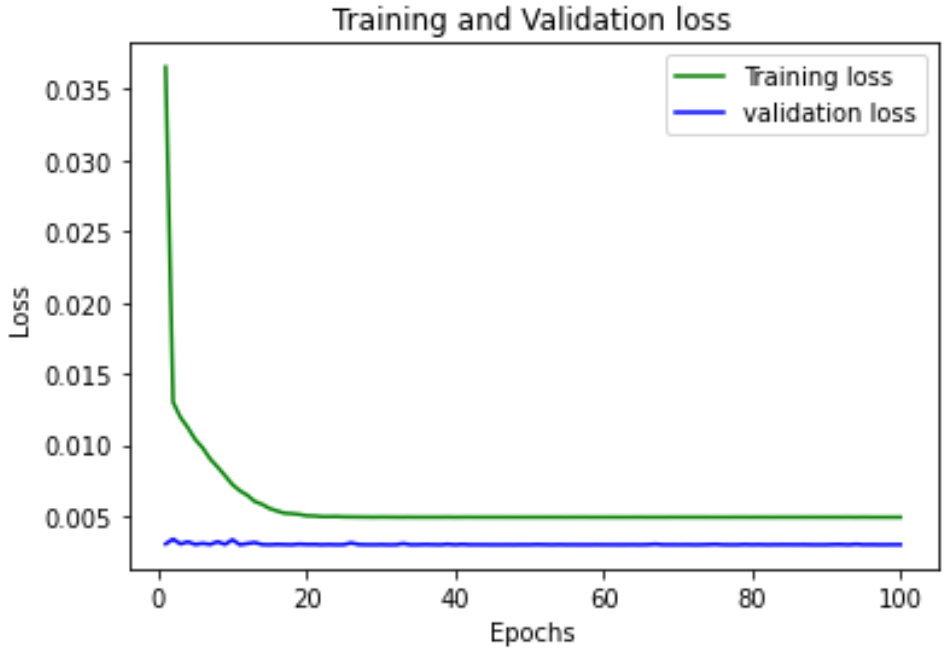


And the accuracy of our first model is 99.79% which is achieved after running the model for 100 epochs.

Epoch	Runtime per step	Loss	Accuracy	Val.loss	val.accuracy
96	2s	0.0043	0.9958	0.0021	0.9979
97	2s	0.0043	0.9958	0.0021	0.9979
98	3s	0.0043	0.9958	0.0022	0.9979
99	2s	0.0043	0.9958	0.0021	0.9979
100	2s	0.0043	0.9958	0.0022	0.9979

Table: Accuracy of model 1 achieved by epochs

Similarly in model 2, the training loss value is 0.0030 and the validation loss is 0.0049. Below is the the training loss vs validation loss graph for model 2:

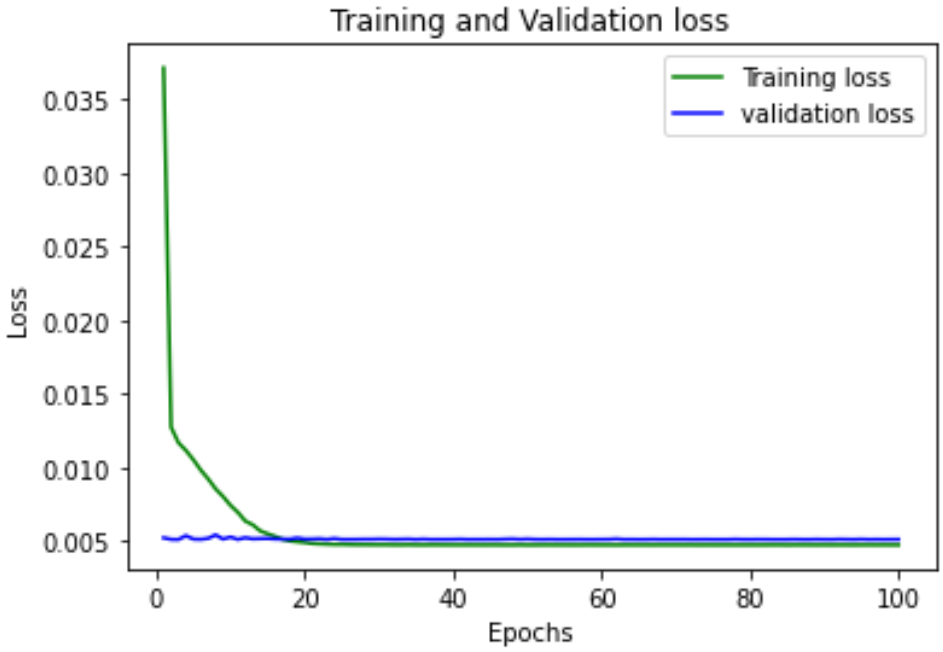


And the accuracy of our first model is 99.70% which is achieved after running the model for 100 epochs.

Epoch	Runtime per step	Loss	Accuracy	Val.loss	val.accuracy
97	2s	0.0049	0.9952	0.0030	0.9970
98	3s	0.0049	0.9952	0.0030	0.9970
99	2s	0.0049	0.9952	0.0030	0.9970
100	2s	0.0049	0.9952	0.0030	0.9970

Table: Accuracy of model 2 achieved by epochs

From model 3, it can be observed that the loss value is 0.0051 and the validation loss is 0.0047. Below is the the training loss vs validation loss graph for model 3:



And the accuracy of our first model is 99.49% which is achieved after running the model for 100 epochs.

Epoch	Runtime per step	Loss	Accuracy	Val.loss	val.accuracy
96	2s	0.0047	0.9955	0.0051	0.9949
97	2s	0.0047	0.9955	0.0051	0.9949
98	2s	0.0047	0.9955	0.0051	0.9949
99	2s	0.0047	0.9955	0.0051	0.9949
100	2s	0.0047	0.9955	0.0051	0.9949

Table: Accuracy of model 3 achieved by epochs

It can also be seen that the time required to stream a content from the caches is much lower compared to cloud:

Figure: Time required to stream a content (in ms)

Here is the bar diagram which illustrates the time required to stream content in milliseconds. We can see clouds st6ream needs around 4200 milliseconds. Assuming we need almost 21 hubs to reach cloud station. Compared to that, it takes only 400 milliseconds to stream a content from central cache. When it comes to regional, the required time calms down to 200 milliseconds only. According to the graph, regional cache server undoubtedly takes less time to

stream a content which is praiseworthy.
Moreover, the accuracy is considerably higher compared to conventional caching methods:

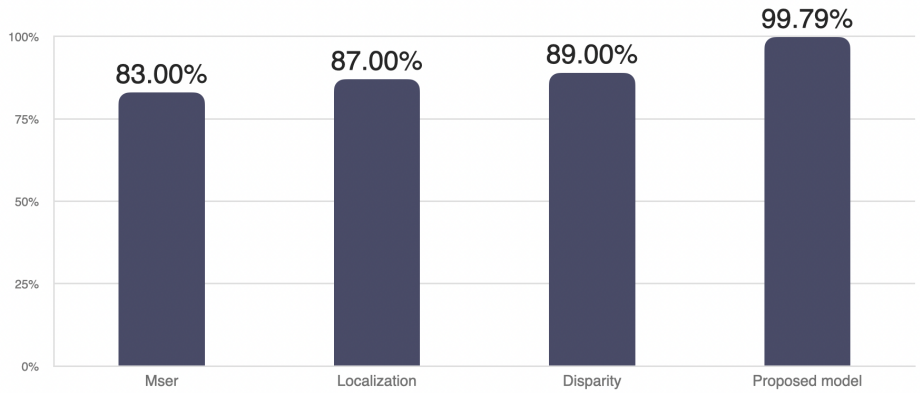


Figure: Comparison with conventional caching methods

This diagram illustrates the accuracy comparison of existing caching methods with our proposed one. Some of the existing models are Mser, Localization, Disparity etc. Accuracy of Mser is 83% where localization has 87% accuracy. Beside them, Disparity has 89% accuracy. On the other hand decentralised content caching model has accuracy of 99.79% which is considerately higher than mentioned existing models. In other words, compare to other models, decentralized content caching method predicts better accurately which is really close to 100%.
Finally, it is visible that using the central cache server in a decentralized way uses a lot less power compared to not using in a decentralized way.

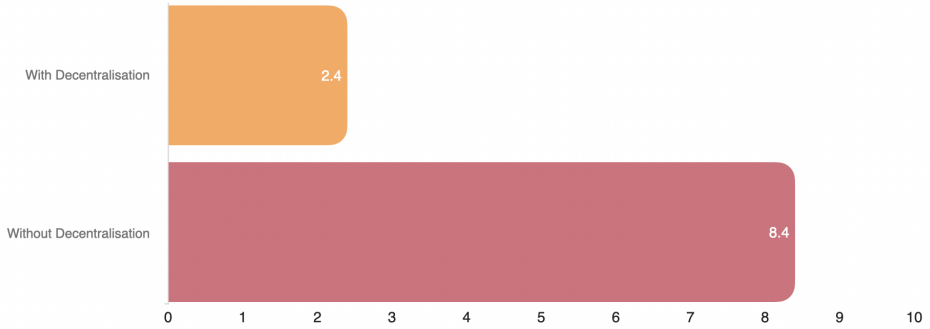


Figure: Power usage comparison (in kWh per day)

This diagram clearly shows that, the central cache server's daily power usage. It shows, when its running with decentralisation the central cache server uses only 2.4 kilowatt-hour per day. Whereas, without decentralisation it uses up-to 8.4 kilowatt-hour per day. Now, we can say that with decentralisation, we need almost one-third of computational power than without decentralised one.
So we can confidently claim that, in terms of accuracy, streaming and power usage, with decentralisation, central server caching is way better than the traditional caching methods.

Our proposed model improves the conventional caching in the following ways:

- Using our proposed model, the system can cache the movies much faster than conventional models. Because, when streaming from the regional server, the end user only needs to traverse a single hop and when streaming from the central cache server, the end user needs to traverse two hops whereas when streaming from the cloud, the end user needs to traverse so much more than that of cache servers. Thus the users can stream contents much faster with the least latency.
- Moreover, our proposed system does not need to make any prediction in the central cache server. As a result, a major amount of computational power and a huge amount of time gets saved.

Chapter 6

Conclusion

6.1 Challenges Faced and Solutions

On our journey to prepare our dataset and implement the algorithm, we had faced a lot of challenges and we had tried a bunch of approaches to get rid of those challenges. However, the final approach helped us to solve most of the issues that arose. The issues and solutions are discussed here.

At the very beginning, when we were preparing our dataset, we faced a major issue. After the preprocessing, when we were using the dataset in our LSTM model, we were facing an under fitting issue and the accuracy was very low(around 23%). Then we looked into our preprocessing and found out that there was a lot of duplicate data which was causing this problem. So, we dropped the duplicate values and our problem got solved. Our accuracy went above 99%.

The next problem we were faced with was in spite of getting 99% accuracy, our prediction value was giving us invalid values. Then we looked into our mode and changed our loss function from sparse categorical crossentropy to many other loss functions. But finally we switched to mean squared error. And, our problem got solved.

While caching the movies on regional servers as well as the central cache server, we were planning to do prediction on both layers. But, it takes additional time to predict on the central cache server and then cache the movies. So, we finally stuck to the plan to do prediction only on the regional servers and then use those hit count values to run knapsack on the central cache server. However, there we were facing another major problem. The movies that edge 1 could not cache, we planned to cache them on the central server. But, there might be movies that could not be cached on edge 1, edge 2 and edge 3. For those movies, the prediction values would be different. Which will create problems with running knapsack on the central cache server. We successfully solved this problem by simply summing up the predicted hit count values from the common edges. And, finally used the summed up hit count for the central cache server.

6.2 Summary and Future Work

The aim of our thesis was to build a complete system model which accurately predicts contents by determining efficiently what to cache and what to replace from the cache. Our system model will reduce the total hop count and thus decrease the latency to access the contents. Our work also helps to decrease computational power used in conventional ways. However, there are a lot of fields that we can improve in future.

- For our thesis, we have only worked on three Regional servers. If we can increase the regional cache server count, the overall caching on the central server would be more efficient and accurate.

- We haven't considered the base stations under the regional servers. In future if we can consider the base stations under the regional servers, there will be three layers for caching contents and the overall caching would be more efficient and contents would be more accessible.

Bibliography

[1] Colah. "Understanding lstm networks." (2015), [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[2] F. M. Harper and J. A. Konstan. "The movielens datasets: History and context." (2015), [Online]. Available: https://www.kaggle.com/grouplens/movielens-latest-full/version/1.

[3] Jithin. "What is content caching?" (2016), [Online]. Available: https://www.interserver.net/tips/kb/what-is-content-caching/#:~:text=A%20content%20cache%20is%20a,the%20requests%20for%20the%20application..

[4] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. Nikolopoulos, "Challenges and opportunities in edge computing," Nov. 2016. DOI: 10.1109/SmartCloud.2016.18.

[5] J. Brownlee, "A gentle introduction to long short-term memory networks by the experts," *Machine Learning Mastery*, vol. 19, 2017.

[6] Johnson. "What is content caching?" (2017), [Online]. Available: https://blog.stackpath.com/glossary-content-caching.

[7] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017. DOI: 10.1109/MC.2017.9.

[8] Srivastava. "Essentials of deep learning : Introduction to long short term memory." (2017), [Online]. Available: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/.

[9] "Introduction to loss functions." (2018), [Online]. Available: https://algorithmia.com/blog/introduction-to-loss-functions#whats-a-loss-function.

[10] 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.

[11] M. Nguyen, "Illustrated guide to lstm's and gru's: A step by step explanation," *Online] Towards Data Science*, 2018.

[12] C.-H. Liu, D.-C. Juan, X.-A. Tseng, *et al.*, "Hierarchical lstm: Modeling temporal dynamics and taxonomy in location-based mobile check-ins," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2019, pp. 217–228.

[13] Miller. "Data preprocessing: What is it and why is important." (2019), [Online]. Available: https://ceoworld.biz/2019/12/13/data-preprocessing-what-is-it-and-why-is-important/.

[14] "Netflix: Number of subscribers worldwide 2020 — statista." (2020), [Online]. Available: https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide.

[15] Y. Wang and V. Friderikos, "A survey of deep learning for data caching in edge network," vol. 7, no. 4, p. 43, 2020.

[16] Williams. "Netflix targets 500 million subscribers by 2030. rolls out mobile-only subscription option for nigeria's 126 million internet users. publishers look the other way." (2020), [Online]. Available: https://thenewpublishingstandard.com/2020/10/05/netflix-targets-500-million-subscribers-by-2030-rolls-out-mobile-only-subscription-option-for-nigerias-126-million-internet-users-publishers-look-the-other-way/#:~:text=A%20half%20billion%20subscribers%20by,from%20Digital%20TV%20Research%20shows..

[17] "Loss function." (), [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/loss-function.

[18] Sinha. "Long short term memory (lstm)." (), [Online]. Available: https://iq.opengenus.org/long-short-term-memory-lstm/.

[19] Unknown. (), [Online]. Available: http://www2.dac.com/events/videoarchive.aspx?confid=170&filter=keynote&id=170-103--0&#video.

[20] ——, (), [Online]. Available: http://files.grouplens.org/datasets/movielens/ml-latest.zip.

[21] "What is predictive analytics ? (predictiveanalyticstoday.com)," [Online]. Available: https://www.predictiveanalyticstoday.com/what-is-predictive-analytics/.