LSTM Based Content Prediction for Edge Caching Using Federated Learning Approach

by

Shafkat Ahmed Mazumder
17101093
Piash Paul
17101040
DIN MOHAMMAD ZUBAIR
17101168
Maksudul Haque
17101084
Jidni Mayukh
17101139

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
June 2021

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:

Arnob	Maksudul Haque
Shafkat Ahmed Mazumder 17101093	Maksudul Haque 17101084
JIDNI MAYUKH	piash
Jidni Mayukh 17101139	Piash Paul 17101040

DM Zubair

DIN MOHAMMAD ZUBAIR
17101168

Approval

The thesis titled "LSTM Based Content Prediction for Edge Caching Using Federated Learning Approach" submitted by

1. Shafkat Ahmed Mazumder (17101093)
2. Piash Paul (17101040)
3. DIN MOHAMMAD ZUBAIR (17101168)
4. Maksudul Haque (17101084)
5. Jidni Mayukh (17101139)

Of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on June 6, 2021.

Examining Committee:

Supervisor:
(Member)

[illegible]

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

Head of Department:
(Chair)

[illegible]

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

Abstract

With rapid expansion and worldwide penetration of internet usage, there has been a rapid growth and development in the field of communication technology. To meet a never ending demand of excellence in quality and computation, a relatively new and effective computation theory called Edge computing is making its mark. Edge computing basically means the computing which is done at or near the data source instead of relying on the cloud to do all the work which enhances network performance by reducing latency. With Edge computing and Edge caching we seek to integrate federated learning approach by training the model across multiple edge nodes that have thier own local environment, without exchanging them which will eventually turn into Edge Intelligence by increasing system level optimization making content delivery faster than before. In a whole in this research topic we aim to investigate service provisioning in edge computing which will make our daily used devices more efficient in terms of performance and keep our personal data secured with the help of federated learning approach. Accurate content prediction combined with optimized caching promises to be a future-proof solution. We adopt a hierarchy based three layer system architecture in which we integrate federated learning with LSTM for predicting content based on view count. With our FedPredict algorithm we intend to maximize cache hit so that the network flow remains optimized. Lastly, we look into potential optimization our algorithm and address some areas of improvement regarding distributed learning systems.

Keywords: Federated Learning; Edge Computing; Edge Caching; Content Prediction; Long Short Term Memory; Decentralized Learning System; Cache-Hit Ratio

Dedication

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

Acknowledgement

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 ongoing 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.

Table of Contents

Declaration i

Approval ii

Ethics Statement iii

Abstract iii

Dedication iv

Acknowledgment v

Table of Contents vi

List of Figures viii

Nomenclature ix

1 Introduction 1
- 1.1 Edge Caching and Federated Approach Background 1
- 1.2 Aims and Objectives 4
- 1.3 Thesis Organization 5

2 Related Work 6
- 2.1 Edge Caching and Content Prediction Best Practices 6
- 2.2 Federated Learning and Deep Learning Approach 9
  - 2.2.1 Conventional Content Prediction Practices 10
  - 2.2.2 Neural Networks and LSTM 12
  - 2.2.3 Federated Average 13
  - 2.2.4 Stochastic Gradient Descent 14

3 Proposed Model and Methodology 15
- 3.1 System Architecture 15
  - 3.1.1 Federated learning layer 1 (Base Station) 16
  - 3.1.2 Federated learning layer 2 (Regional Server) 16
  - 3.1.3 Federated learning layer 3 (Continental Server) 17
- 3.2 Flow Diagram 17
- 3.3 Model Architecture 19
  - 3.3.1 Dataset 19
  - 3.3.2 Data Preparation for LSTM 20

3.3.3 Data Preparation for Federated Learning Simulation 20
3.3.4 Proposed FedPredict Algorithm 20

4 Comparative Study 22
4.1 Optimizer Comparison 22
4.2 Loss Function Comparison 23

5 Result Analysis 24

6 Conclusions 27
6.1 Attempted Approaches and Challenges Faced 27
6.2 Summary and Future Work 28

Bibliography 31

List of Figures

1.1 Netflix Popularity On The Rise [1] 2
1.2 Brief structure of Federated Learning 3

3.1 System Architecture For Proposed Model 15
3.2 Flow Diagram of the Proposed Model 18
3.3 Architecture of the Proposed Model 19
3.4 Input Overview 20

4.1 Comparison of Various Optimizers 22
4.2 Comparison of Various Loss Functions 23

5.1 Training Accuracy and Validation Accuracy Graph 24
5.2 Training Loss and Validation Loss Graph 25
5.3 Prediction result for continental server 25
5.4 Regional server computation round 26
5.5 Federated vs Non-Federated Approach 26

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

CNN	Convolutional Neural Network
$FedAvg$	Federated Averaging
$LSTM$	Long Short Term Memory
NN	Neural Network
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

1.1 Edge Caching and Federated Approach Background

Edge computing has recently risen up the ranks of communication technology as the usage of internet grows at a phenomenal rate. Edge computing basically means the computing which is done at or near the data source instead of relying on the cloud to do all the work which enhances network performance by reducing latency. With Edge computing and Edge caching we seek to integrate federated learning approach by training an algorithm across multiple decentralized edge devices or servers holding local data samples. In a whole in this research topic we aim to investigate service provisioning in edge computing which will make our daily used devices more efficient in terms of performance and keep our personal data secured with the help of federated learning approach.

The internet is filled with content that is consumed by millions of people and based on the popularity of content, the rush in network flows is often determined. Accurate prediction of content popularity combined with an optimized solution using edge caching and federated learning, has the potential to maximize user satisfaction whilst maintaining flawless quality.

Edge caching is a promising solution for various cases where it is necessary to maintain a low latency despite extremely vigorous volume in traffic. As the global population increases, the number of mobile devices or devices that connect to the internet are increasing at an exponential rate. Thus, having high traffic is a common occurrence in a multitude of cases. As a result, optimizing networks flows and properly managing this influx of traffic which travels through various data channels and networks. We can take the example of Vehicular Content Networks (VCN) which requires very low latency regardless of having high volume of data. Another relevant and recent example would be the streaming industry where it is mandatory that we ensure fast communication and transmission of large volumes of data in the form of 4K videos and images. In streaming platforms such as Twitch.tv, real time transmission of 4k data or high resolution videos is a norm these days. In cases like this and many others, edge caching has given promising and groundbreaking results which is why continuous development of this practice is happening day by day.

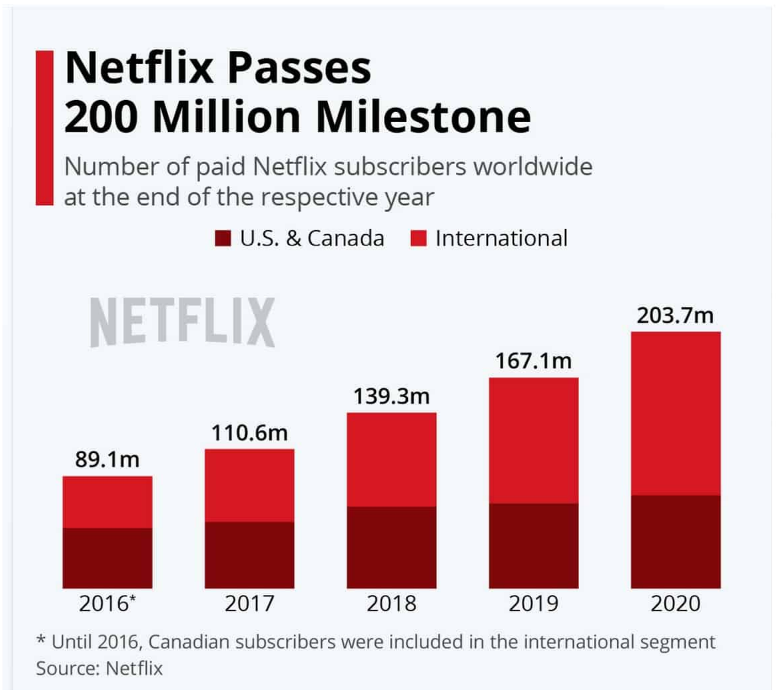


Figure 1.1: Netflix Popularity On The Rise [1]

If we only take a look at the popular streaming service Netflix's data, we will see the the rising demand and increase of monthly active users of such high quality services is quite apparent. The numbers will continue to grow over time as more people have access to affordable yet high quality services due to the expansion of internet access worldwide. So, the necessity and requirement of having uninterrupted and delay less communication channels are definitely apparent. As previously discussed, edge caching is one way of handling this problem but there are other methods as well such as edge computing and software defined networking. Our research primarily addresses practices related to caching and aims to improve communication flow with the integration of federated learning or a federated approach. Federated approach is a completely decentralized way of training Machine Learning Models and provides security clearance unlike any other.

The concept federated approach was introduced by the tech Giant Google where the primary idea is that the training of a model is done in a decentralized way where it happens over a scattered out and distributed learners. The key point to note is that this is a decentralized inference approach, which largely differs from conventional centralized approaches in machine learning. In the paper [2], the authors explain that, what federated learning is trying to do is to optimize the training by keeping training dataset within its origin or source and perform the learning mechanism in a local environment in each individual device or learner in the federation. After learning is achieved in a local environment, instead of passing raw dataset to an intermediary or aggregating unit, it passes the local model parameters which can then be utilized to update a larger global model. Eventually, the updated global model data is fed back to the individual local units which they can later use. Here, high level of privacy is introduced and maintained as local learners improve their performance through the global model without accessing each-others private data.

This is done through the intermediary aggregator model. Still if concerns regarding privacy persists as some form of data may still be shared, the local learners actually can share encrypted and protected versions of their model to the aggregator which ensures privacy preservation. This may raise concerns regarding computation power as encrypted models need to be decrypted but the aggregation algorithm used makes it possible to process the models without decryption [2]. Figure 1.2 below gives an illustration of how the entire process functions.

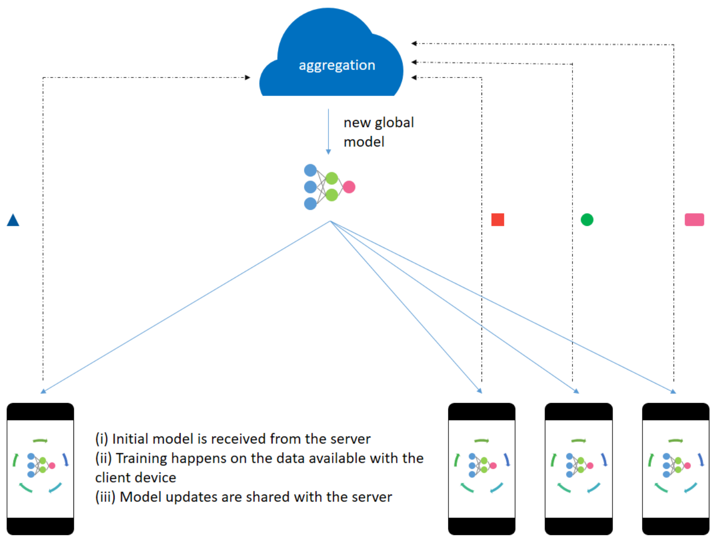

aggregation
new global model
(i) Initial model is received from the server
(ii) Training happens on the data available with the client device
(iii) Model updates are shared with the server

Figure 1.2: Brief structure of Federated Learning
[3]

Moreover, they identified that current conventional and pre-existing distributed systems can be differentiated from federated learning through some key points. In a decentralized approach such as federated ML, one can easily think that the datasets of the learners are basically realized through identically distributed (iid) random variables. In this case that isn't exactly true. As various learners may observe of process various parts of the system, the generated data samples aren't always the exact distribution of the entire global dataset. So if we consider the example of human hand written digit recognition, local learners of this task will have samples of different digits they encountered, each within their own domain. Moreover, in this approach the datasets are usually unbalanced and unmatching in size. Some parts of the process may be significantly less impactful than others. If we consider the digit recognition example, dataset gained from different systems may largely differ because of the number of digits they encounter. One last interesting differentiation is that total number of samples in each local environment will be smaller [2].
As discussed earlier Federated Learning in many ways differ from other decentralized approaches. In this section we take a detailed look at why optimization through this method is possible.

In generic distributed learning schemes, the aggregator usually organizes localized data which provide a robust estimation of the parameters that are being taken into consideration [2]. The collected data is achieved in the form of locally trained models. So quite evidently, in learning systems like this, local learners are basically data collectors at the edge and are independent in nature due to the fact that the global model isn't giving any feedback to them directly or through the aggregator. If we look at Wireless or (WSN), each and every single sensor in the network maintains communication with the locally trained model from it's own dataset to the global center. The fusion center then synthesizes the local information to make a seemingly accurate global estimation of the temperature of the field [2].

Now, another practice in this field is through parallel learning. Which mainly refers to the scheme where our main objective is to scale up the algorithm in a way that it accelerates the learning process. In fact, both scaling and acceleration can be done depending on the requirements. What basically happens in that the entire available training set at a central or global parameter is divided into subsets and each are assigned to worker machines on the edge. It is possible that the data-sets that are assigned to the worker machines have the same distribution. Simultaneously, the parameters are fed back to the primary server and training is performed in parallel. This is usually performed in data centers where they have a shared storage, which results into them having same distribution samples unlike Federated Learning. In Federated learning, the data is massively and in large scale distributed. However, in this case, the average samples per worker is increasingly larger than the number of worker machines in the training process. So in Federated Approach we have a wide-spread data distribution which is crucial to solve many security challenges. Another approach is distributed learning which combines multiple learners to make significant performance boost to the model. Here certain parts of the dataset are used to train different portions of the models. In this method, what basically happens is that instead of making improvements to the global model, learning is done from a mixture of well trained models. [2].

1.2 Aims and Objectives

In this paper, we aim to optimize edge computing even more by implementing the federated learning approach. Moreover, we aim cache contents with respect to the popularity based on the regional and sub-regional context.

Merging these two methods we want to achieve:

1) **Minimization of latency**: Round Trip Time (RTT) Will be significantly lower than the cloud and delay of offloading tasks will be comparably much lower.

2) **Maximization of network capacity**: Data offloading as well as context-aware computation offloading are combinations of technologies that are expected to address some of the challenges in the field of edge caching. MEC and content caching could assist the network capacity by caching popular content to the edge and BSs, and by saving the back-haul bandwidth.
In the proposed scheme we will use Federated Learning on the edge nodes to predict the most accessed contents based on geographical context and what are the possi-

bilities of accessing them within a specific time frame in the particular the area and lastly, storing the most popular contents.

To predict the most popular contents we will use different parameters tor different areas to predict the most likely contents accessed in those areas. By implementing the model, we aim to cache the most popular contents to reduce overhead and data energy consumption thus minimizing the latency. We also aim to reduce the internet traffic as the streaming platforms use most of the internet data. By caching them in local edges the traffic can me reduced significantly Another objective we want to assure is to provide data security. As the data are stored locally, it is less likely to face and kind of threat comparing to data that has to perform multiple hops to reach the user from the cloud.

1.3 Thesis Organization

We have divided our work into a few sections. Intitially, **Chapter 2** talks about the relevant and contemporary reserach relevant to our topic. The chapter is again sub-divided into two sections, where we first talk about Edge Caching and Content Prediction best practices and then move onto recent approaches in federated learning and deep learning.

Afterwards in **Chapter 3**, we explain our proposed model, the entire flow of the system and how we prepared the dataset for simulation. Lastly in this chapter we deep dive into the proposed FedPredict algorithm that we have implemented.

Next, in **Chapter 4**, we do a comparative study based on the various optimizers and loss functions that we used. The differentiation is shown via graphical illustration in this chapter.

Later on, in **Chapter 5**, we do a analysis of the results that we got from implementing the FedPredict Algorithm combined with the multi-layered proposed system.

Lastly, in **Chapter 6**, we talk about the challenges we faced and how we maneuvered around them. We briefly explain how to avoid certain pitfalls and how we got our end result after multiple iterations of attempts. We end the chapter by giving concluding remarks and recommending future work in this particular area of research.

Chapter 2

Related Work

In this section we take a loot at some of the recent developments regarding edge caching and federated learning. We dive deep into best practices, challenges and notable outcomes of the aforementioned technologies. As research in this field is considerably new, we went through papers that have the highest impact and relevant to our method.

The research work is divided into two parts. Firstly, we go over conventional practices regarding edge caching and edge computing. We covers aspects such as where, how and what to cache, how caching can impact content prediction, how caching optimization is being done and also go over some limitations of these processes.

In the second part we go over federated learning and it's application with neural networks. Federated on it's own as a decentralized method has it's perks but when we combine it with pre-existing industry standard solutions for prediction modelling, we get insightful findings. We try to dissect how federated learning is different compared to other distributed computing systems, what impact we get by applying it with deep learning approaches, significance of Long Short Term Memory and Neural Networks, how to avoid challenging problems such as vanishing gradient and lastly some federated learning approaches and algorithms.

2.1 Edge Caching and Content Prediction Best Practices

In this section we go over some conventional ways of caching and predicting content popularity.

In the paper [4], the authors have identified how caching usually works in the modern era. They have also highlighted some relevant examples and benefits of edge caching. Edge caching is more time efficient and saves up on bandwidth and other resources. The authors mention benefits such as reduction of latency, optimization and reduction of network traffic, energy efficiency improvements. As the distance between content and most frequently accessed content can be reduced the overall speed and latency can significantly be improved especially for streaming services which takes up a large chunk of mobile traffic. Moreover, they clarify where, how and what to cache. We take a look at where caching commonly happens.They highlight that caching can be done in network edges, in the last devices as well. Moreover, the edge in wired network and edge in wireless network. This includes caching at UE or user end devices, caching at base stations which is utilized by smart homes and smart offices. Edge routers can also be a good option for caching contents.They duly mentioned that video and image content that are highly revisited must be prioritized when caching. As content can be cached in many ways, the authors point out the intricacies of proactive and reactive caching. They say that when it comes to caching, a few things are taken into consideration such as content type, content popularity, user profiles and these would be updated based on various caching policies. When it comes to content placement, in order to maximize cache hit-rate it is extremely important that caching is done consistently. Usually, there is a content delivery mechanism involved for checking validity if the user request exists or not. They also mention that cloud processing isn't ideal when it comes to identification of fast image processing. This largely happens due to network hops and computation time. They gave an example of traffic engineering information based networking increases content delivery speed through prefetching of data and data distribution [4].

In regards to IoT devices and edge computing, the field has seen significant progress with various interesting models proposed recently. We go over a few that closely resonates with our own proposal. At present we can see the extensive up rise of the population of Internet of Things (IoT) devices, artificial intelligence, industrial robotics, face recognition, natural language processing and many new real time services which are constantly emerging and in order to efficiently use these services we require faster data transmission and faster computational methods. As these services are data intensive while IoT and other edge devices have challenges when it comes to computational power, storage services and of course longevity limitations due to lower battery life. In order to overcome these limitations the concept of mobile cloud computing (MCC) emerged which supported IoT and other edge devices by enabling them to offload heavy computing service requests to the centralized cloud or base stations which also provides huge data storage capabilities to the services. The authors mention that this architecture enabled higher computation but as more devices came into fruition, more problems arose [5].To address these issues the concept of Mobile Edge Computing (MEC) was proposed as the extension of centralized cloud computing to the edge of the networks. MEC is proposed as a novel paradigm for easing the burden of backbone networks by pushing the computation or storage resources towards the end user's edge devices.This basically reduces the user to base station distance problem and in practice this aims to maximize efficiency especially for IoT technology. The authors of [5],proposed a Edge Ai framework where they basically combined mobile edge computing and federated learning to manage resources better in the entire system.

A survey was recently presented on the use of MEC technology for the realization of IoT applications [6]. The static and dynamic architecture of an edge cloud network was accounted for by researchers, thus taking into account the absence and existence of device mobility [7]. Therefore, they proposed an approach to column generation to decide the sites on which the cloudlets would be mounted. They then decided the classification of the BSs to cloudlets. Finally, in deciding the placement of each VM needed by an end system with regard to its usability conditions and latency requirements, the researchers discussed the resource allocation issue. Although the researchers took the cost of deployment of cloudlets into account, the sharing of VMs among multiple end devices was not taken into account. The question of optimum positioning of cloudlets in a network in the metropolitan region, where it is believed that cloudlets are collocated with Aps are evident in many researches. The researchers proposed two heuristic solutions, considering the complexity of the problem. In order to reduce the response time, they have found the users to have a cloudlet assignment problem, and acknowledged that routing traffic normally to the nearest cloudlet may not always have a sufficient solution in terms of response times, so it is important to coordinate the workload between the cloudlets deployed. Multi-access edge computing (MEC) has recently become a successful model to provide IoT applications with resource-intensive and latency-sensitive resources by moving computing capabilities away from the main cloud to the edge of networks [7].

Now we go over more real life applications of the technology. There are many sectors which are directly being impacted by this and we highlight some authors who have identified various novelties that are currently being used.

Recently, edge computing is being employed in a variety of sectors, including real-time data processing, driver-less vehicles, virtual reality, and home automation [8]. Having low latency and high bandwidth enables edge computing to easily address the issues of delay of data transmission from user devices to the cloud servers [9]. Caching approach solves the problem of latency due to excessive real-time interaction and content delivery in time by the cloud servers. Wu, Luo and Li proposed a cache prefetching algorithm in their paper which is based on the model of user classification based on Bayesian network and Markov chain (UCBM) suited for edge computing and in order to speed up end-user access, it stores the desired files in the cache of edge servers. Moreover, they a suggested cache replacement method based on the model of file heat and re-access probability (HFAP)[9]. A collaborative caching and processing architecture that supports Adaptive Bitrate video streaming was proposed by Tuyen X. Tran et al. in their paper [10]. Moreover, they also defined the decision of video variant placement in the cache as an Integer Linear Problem and suggest efficient methods whether video popularity is available or unavailable [10]. The collaborative caching challenge in terms of reducing the overall cost paid by content providers were proposed by Ammar Gharaibeh et al. in their paper [11]. and they defined the problem as an Integer Linear Problem and present an online caching technique that does not require information of data popularity. Video caching has been proposed by Jimy George et al. in their paper [12] where they provide a criterion for selecting videos to cache and a cache replacement algorithm based on the popularity distribution of movies in each cell site. In their work, active users and user preferences are examined to determine local video popularity. To compute video popularity, they employ a chance that a certain user requests videos from each category, and then divide it into two groups based on a threshold and the process of cache replacement depends greatly on the groups. Shan Zhang et al. in their research paper [13] studies delay-optimal cooperative edge caching, in which content placement and cluster size are improved depending on bandwidth allotment, channel quality, and popularity of given data. Edge computing delivers computation/storage resources and services to the end users at the network's edge which is a special form of cloud computing. To increase the size of edge computing, prefetching and replacement techniques have been shown to be effective methods [14]. Wu, Luo and Li showed in their paper [9], the techniques of cache replacement on the basis of file heat where files with high-heats are replicated and stored in the distribution nodes and in edge computing, this technique helps to diminish the latency of access. The cache replacement policy's speed is maintained by maintaining files with high user access needs during the cache replacement procedure and the popularity of a file during a given time period may be determined, and the closer the time, the more current the popularity of the file.

Despite progress and numerous benefits of these methods, certain challenges and security concerns can be seen as highlighted by various authors. We now go over some common challenges and obstacles faced when it comes to dealing with cyber security issues.

The authors of [4] have identified the common security concerns related to content caching on edge devices. With federated learning combined, many of the security challenges can be addressed. hey mention exploitation of network edges with denial of service attacks, wireless jamming and more. The authors then dig deep into the real of privacy protection and try to identify some real-word practical solutions to possible attacks. They talk about imminent threats such as cache pollution, malware attacks and cache deception attacks. These basically prevent services from going to the user and create resource wastage. They highlight that cache pollution attacks, which basically means that false contents are inserted into an edge to deceive requesting users, causes remote servers to crash which results in false content delivery. Common practices are error ridden HTML pages or CSS files or javascript files. On the other hand, they talk about cache pollution attacks which deals with balancing issues. This results in caching unpopular content and hit rate of real users or real popular content goes down. Lastly, they shed light into deception attacks which basically manipulate users to send request to restricted content which is then capitalized by the attacker to gain private information of the users [4].

2.2 Federated Learning and Deep Learning Approach

This section aims to deep dive into common practices of Federated Learning and Deep Learning for content prediction and caching. We also take a look at some of the common problems certain authors faced while dealing with implementation. We also differentiate between other decentralized systems with our federated learning approach.
In the paper [15] , the authors propose a content prediction model based on deep learning using only the title of the content. In our case we have used view count but this provides an overview of how deep learning helps to create prediction models.

Moreover, LSTM was utilized which we use in our research.

They say that, textual content sees a trade-off in consumer interaction, as it can be shared rapidly but engages attention for only a brief period of time. This makes naming articles appropriately very important, as that can determine how many consumers' attention it can grab initially. This phenomenon manifests itself in the form of click-bait - brief snippets of text that can often be misleading or sensationalized, whose primary function is to attract clicks.They aptly mentioned that even though the identification of click-baits is a separate research topic, this paper deals with the more generalized issue of estimating the demand for online content based solely on their titles. Estimating the popularity of online content is a complex operation that hinges on various aspects such as the end user's social and exterior context, relevance of the content etc.

A process for online content popularity prediction based on a bidirectional recurrent neural network called BiLSTM was proposed by their research paper [15]. In contrast to the previous approaches, their method gives highly accountable results by aiming to model intricate correlation between the title of an article and its popularity using novel deep network architecture. Finally the suggested BiLSTM method outperforms the existing state-of-the-art on two separate datasets with over 40,000 samples, providing a considerable performance improvement in terms of prediction accuracy over the typical shallow techniques [15].

2.2.1 Conventional Content Prediction Practices

In recent times, the majority of research works in the field of information technology concentrate on the arrangements of efficient networking along with enhancing the best possible way to distribute information effectively, attempting to achieve maximum benefit from social media marketing and networking technology. The analysis of language effects can be easily predicted with the use of tweet contents that are used over social media throughout the world. Various technological advancements have enabled the impact of hashtag duration on hashtag frequency to be analyzed in the research works. Researches related to Twitter hashtags include use of graph topology to identify the most trending topics. The study of the contrast between 'Persistence' and 'Stickiness' which indicates that certain groups are more enduring than others, explore characteristics inherent in memes, going beyond the most basic topical dissemination. This says a lot about the trends of interest and prediction of trending memes or happening events in the Twitter are possible and for the purpose of accuracy in the analysis, annotators should not be used to categorize hashtags. Except the uncommon ones, the distribution of popular hashtags are predicted most often in this field [16].

For movie popularity, technical prediction of popularity is very important. The aim of popularity prediction is to use knowledge at an early stage to forecast this indicator of public interest. Existing methods for predicting popularity are classified into four types: analyzing using time series, feature engineering, analyzing using cascade process, and methods oriented to deep learnings. At an early level, data relevant to popularity evolution are required by approaches of Time series analysis.

To forecast popularity, early experiments used a basic linear function. Assumptions made by such approaches say that previous popularities have varying or constant relationships with success. Characterization of popular processes are made using trends, and thereby render predictions using the most comparable popularity evolution processes. Many recent studies have modeled the distribution of contents as stochastic methods. Methods focused on feature engineering are mostly concerned with feature architecture. The relevant works used various information of users for proper analysis and prediction. Researchers recommend a variety of features to forecast the success of news stories, including surface features (such as publishing time, subject duration and many other relevant data. Usually, popularity prediction using methods of cascade process is simulated in a topology of consumers. On the basis of deep neural network, few prediction methods are used. Although limited to predicting video popularity, LRCN method extracts functionality from video material, which are then used to forecast the amount of view counts in coming days. For deep learnings such as neural networks: DeepHawkes is one of them which considers user knowledge and cascade direction in-formation. Furthermore, LSTM and GRU are one of the types of RNN. However, it is believed by RNN that the dependence varies monotonously in the chain, while in social media, the time period between acts can be changed. Furthermore, it is difficult to cover the standard RNN for fluctuating time levels [17].

Video popularity prediction has been an emerging and innovative idea in the research arena of Information Technology. Based on the social news contents and YouTube video contents, scientists have found out rising growth in the count as logarithmic curve. According to recent studies, video portals such as YouTube have substantially different popularity growth characteristics than conventional internet streaming channel. Contrary to the existing logarithmic model, studies suggest that improvements in popularity arise in bursts of widely scattered amplitude and time separation. Certainly such spikes in content interest could be attributed to the social visibility (trending nature) of a similar subject in the form of the film. The substantial rise reflects the mutual interest of users and fuels unexpected visibility. In addition, video categorization has also been flourishing in this sector of research where attempts to categorize videos using similar tags and comments are widely seen. [18].

However, despite progress there are numerous drawbacks and content prediction has various existing drawbacks with some of it's relevant methods. The Feature-based method requires much proficient knowledge to execute which is time consuming and also become complex when features are not relevant. While in the Time series analysis method, it requires only considering time series information and depend on solid theories of how popularity develops. Some other deep learning based techniques consider only user information and avoid collaborating various kinds of information during implementation. [17]

2.2.2 Neural Networks and LSTM

Conventional feed-forward neural networks are not the same as recurrent neural networks. This variation in addition to complexity arrives with the possibility of unforeseen behaviors that older approaches are unable to provide.
As stated in the paper [19] , the authors refereed RNN learning as a promising solution to contemporary problems. They said that Recurrent neural networks... have an internal state that may be used to describe the context. They store knowledge about previous inputs for a period of time that is determined by the weights and the input data rather than being defined beforehand. A recurrent net with non-fixed inputs that form an input sequence may be used to convert an input sequence into an output sequence while taking contextual information into account in a flexible manner [19].

In another publication [20] , the authors, mentioned that, Recurrent neural networks have cycles that feed prior time step network activations as inputs to the network to impact predictions at the current time step. These responses are recorded in the network's internal states, which may theoretically preserve long-term temporal contextual information. This approach enables RNNs to take advantage of a dynamically changing contextual window across the history of the input sequence [20].
LSTMs' started to develop from their claim to be one of the first tools to surmount technical challenges and fulfill the promise of recurrent neural networks.

In the paper [21], the authors stated that, Typical RNNs fail to train when there are more than 5 - 10 discrete time steps between relevant input events and target signals. The disappearing error problem calls into question if either typical RNNs can truly outperform time window-based feedforward networks in terms of practical benefits. This issue does not impact a newer model known as "Long Short-Term Memory" (LSTM). By imposing continuous error flow-through "constant error carrousels" (CECs) within special units called cells, LSTM may learn to bridge minimum time gaps over 1000 discrete time steps [21].

LSTMs solve two technical problems: vanishing gradients and exploding gradients, both of which are connected to how the network is taught.

In the paper [22], the authors mentioned that, Regrettably, the range of contextual information that standard RNNs can access is quite limited in practice. The issue is that the influence of a particular input on the hidden layer, and hence on the network output, either deteriorates or explodes exponentially as it cycles across the network's recurrent connections. This flaw, known as the vanishing gradient problem in the literature. Long Short-Term Memory (LSTM) is an RNN architecture that was created particularly to resolve the vanishing gradient problem [22].

Perhaps of going into the mathematics that governs how LSTMs are fitted, an analogy is a valuable technique for easily understanding how they function. Accordig to [23], the authors say that, they deploy networks consisting of one input layer, one hidden layer, and one output layer. Memory cells and related gate units are found in the (fully) self-connected hidden layer. The internal architecture of each memory cell ensures constant error ow inside the constant error carousel CEC.This is the foundation for bridging extremely lengthy gaps. Within each memory cell's CEC, two gate units learn to open and close error access ow. The multiplicative input gate protects the CEC from being perturbed by unnecessary inputs. Similarly, the multiplicative output gate prevents other units from being interrupted by currently inappropriate memory contents [23].

It's worth noting that, even after more than two decades, the simple (or vanilla) LSTM may still be the perfect way to start when implementing the approach.

Again, the authors of [24], say that on many datasets, the most commonly used LSTM architecture (vanilla LSTM) performs reasonably well. The most important tunable LSTM hyperparameters are learning rate and network size. This means that the hyperparameters can be modified separately. The learning rate, in particular, can be tuned first using a relatively tiny network, saving a significant amount of trial time [24]. It is important to get a handle on exactly what type of sequence learning problems that LSTMs are suitable to address.

2.2.3 Federated Average

In recent works, federated averaging has been very much famous and successful. Generally, the theoretical analysis and the design of federated learning algorithm might be more difficult when the availability of client's data along with their distribution substantially fluctuate from one client to the next [25]. Efficiency of communication is greatly enhanced for federated learning in high scale applications by using federated averaging approach . On the basis of widely used distributed Stochastic Gradient Descent (SGD), FedAvg is implemented [26]. Several Stochastic Gradient Descent (SGD) steps are performed on clients which are arbitrarily chosen on a tiny scale and then in the central parameter server, the averages of the local models are made [27]. It has been found in [27], that because of the use of two techniques which includes involvement of partial client and updates of numerous local SGD, may result in increased communication efficiency over the distributed SGD using the FedAvg algorithm. The distributed global model is hosted by the central server Wt that is trained by the Federated Averaging (FedAvg) algorithm and t represents number of communication rounds [28]. The authors say that FedAvg has five parameters which includes the proportion of clients to be trained represented as C, mini-batch size of local (B), quantity of local epochs (E), rate of learning () and rate of learning decay (). The algorithm begins the operation by randomizing the global model's initialization (W0). During the communication rounds of FedAvg, subset of clients (St) is chosen by the server St = K.C which is greater or equal to 1 and the recent global model (Wt) is shared to all the clients in the subset (St). In the shared model, when the local models (wkt) are updated every client divide their private data into batches of size B and executes epochs of Stochastic Gradient Decent (SGD). Moreover, they mention that, At last, the trained local models (wkt+1) are added to the server which then bring about the new global model (Wt+1) by calculating the weighted average of all submitted local models. From mathematical analysis from the paper [28], we find the following expression:

$\mathrm{w}_{t+1} = \sum_{k \in S_t} \frac{n_k}{n_\sigma} w_{t+1}^k, \quad n_\sigma = \sum_{k \in S_t} n_k$

However, there are still risks of user's data privacy being breached and messages might be accessed in an unauthorized manner through sophisticated assaults [29].

2.2.4 Stochastic Gradient Descent

For classification methods based on machine learning such as logistic regression and neural networks, Stochastic Gradient Descent (SGD) is regarded as an efficient and conventional optimization approach. To overcome the problems of data inconsistency, approach of greedy selection of datasets are applied which are consistent and have very low variations [30]. It is one of the principal category of Gradient Descent Algorithms (GDA). Classification is a key approach when dealing with datasets and neural networks and logistic regression are some popular models of classification. Accuracy and efficiency of the algorithms for optimization are highly required for reliable classification of models. By having substantially faster convergence, Stochastic Gradient Descent (SGD) effectively resolves the issue of high computing cost and it solely differs in the amount of data utilized to calculate the gradient of the required function. In addition, a compromise is made between the accuracy of weight updates and the time it takes to make an update, depending on the amount of data available [30].

Chapter 3

Proposed Model and Methodology

This chapter contains in depth explanation of our system architecture, model architecture, the applied algorithm and relevant approaches that we took.

3.1 System Architecture

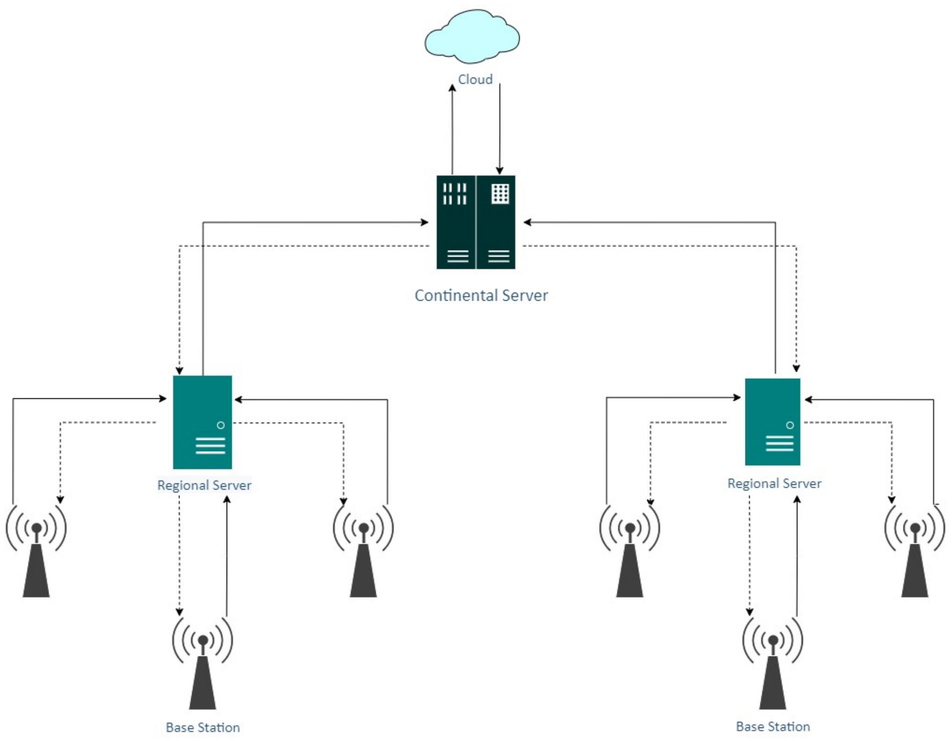


Figure 3.1: System Architecture For Proposed Model

We propose a multi-layered system to optimize content prediction and caching globally. Our proposed model primarily consists of three distinct layers which operate hierarchically. The key entities of our system are the local base stations, the regional servers and lastly the continental server which is connected to the cloud. All of these entities are dynamic in nature and have their own instances.

Layer 1 is where the base stations are which are the closest connected entities with the end users. In the 2nd layer we have regional servers where every regional server will have multiple base stations connected with them. In the final layer we have continental servers which will have multiple regional servers connected with it. Firstly in initializing stage, model parameter Wc will be initialized for continental server. This parameter Wc will be transferred to the regional servers connected to it. So each of the regional servers will now have the same parameters Wr as the continental server. Now from each regional server, Wr will be transferred to the base stations connected to it. As a result each base station will have the same model parameter Wb as the connected regional server and this is how the initialization phase will end. After the model parameters for base station being set our prediction model will start operating from layer 1.

3.1.1 Federated learning layer 1 (Base Station)

The model at it's basic state consisting the parameters received from the regional servers does not have useful values, but as time passes and users watch content and the base station connected to the end user record these movies and after enough data is collected, the model is fit to this new viewing data. The model trains itself and sets new model parameters for better prediction and updates Wb and total view count Vb. After updating base stations transfer their updated model parameters to the associated regional server.

3.1.2 Federated learning layer 2 (Regional Server)

Weight scaling factor: When the updated weights are sent back to regional server not all the weights from all the base stations will have equal effect on the Regional server. In our simulation, we are comparing the amount of data compared to the total amount of data for each base station (Vb). This ratio will be used as the weight scaling factor for the base stations.

Scale model weights: The weight scaling factor will be then multiplied with the weights of the correspondent base station updated model to produce scale model weight Ws.

Weight Averaging: Scaled model weight Ws from each of the base station is now used to calculate Wavg. Summation of all the scaled model weights is divided by the number of base station which gives us Wavg.

Update Wr and predict content: After calculating the Wavg of a particular regional server that servers Wr will be updated to the calculated Wavg. This weight will be used to update the prediction model of the regional server to predict contents for the next timespan and cache contents accordingly.

Transfer Wr and Vr: After updating Wr, the updated Wr will be sent to the connected continental server. Also the total view count of each server Vr will be sent too.

3.1.3 Federated learning layer 3 (Continental Server)

Weight scaling factor: Similarly to layer 2 the updated weights from regional servers are sent back to continental servers and in the same way using total view count of each regional server (Vr) we calculate the weight scaling factor.

Scale model weights: The weight scaling factor will be then multiplied with the weights of the correspondent regional server's updated model to produce scale model weight Ws.

Weight Averaging: Scaled model weight Ws from each of the regional server will be used to calculate Wavg. Summation of all the scaled model weights is divided by the number of regional servers which will produce Wavg.

Update Wc and predict content: After calculating the Wavg of the continental server, Wc will be updated to the calculated Wavg. This weight will be used to update the prediction model of the continental server inorder to predict contents for the next timespan and cache contents accordingly and also to request contents from the cloud based on the prediction.

3.2 Flow Diagram

Figure 3.2 explains the entire process flow in one diagram. Staring from model initialization in the Continental server to the Continental server caching loop.

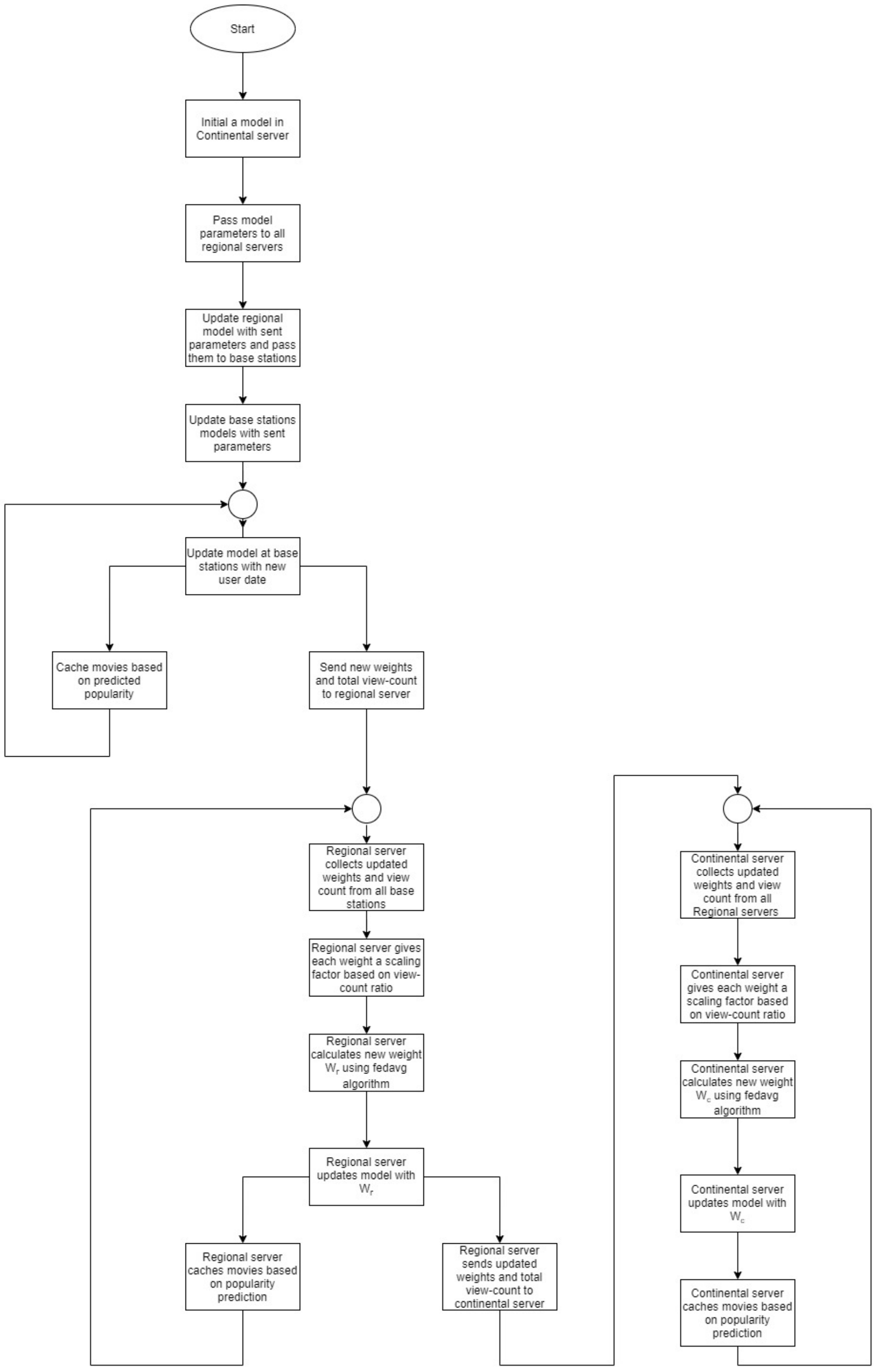

Start
Initial a model in Continental server
Pass model parameters to all regional servers
Update regional model with sent parameters and pass them to base stations
Update base stations models with sent parameters
Update model at base stations with new user date
Cache movies based on predicted popularity
Send new weights and total view-count to regional server
Regional server collects updated weights and view count from all base stations
Regional server gives each weight a scaling factor based on view-count ratio
Regional server calculates new weight W_r using fedavg algorithm
Regional server updates model with W_r
Regional server caches movies based on popularity prediction
Regional server sends updated weights and total view-count to continental server
Continental server collects updated weights and view count from all Regional servers
Continental server gives each weight a scaling factor based on view-count ratio
Continental server calculates new weight W_c using fedavg algorithm
Continental server updates model with W_c
Continental server caches movies based on popularity prediction

Figure 3.2: Flow Diagram of the Proposed Model

3.3 Model Architecture

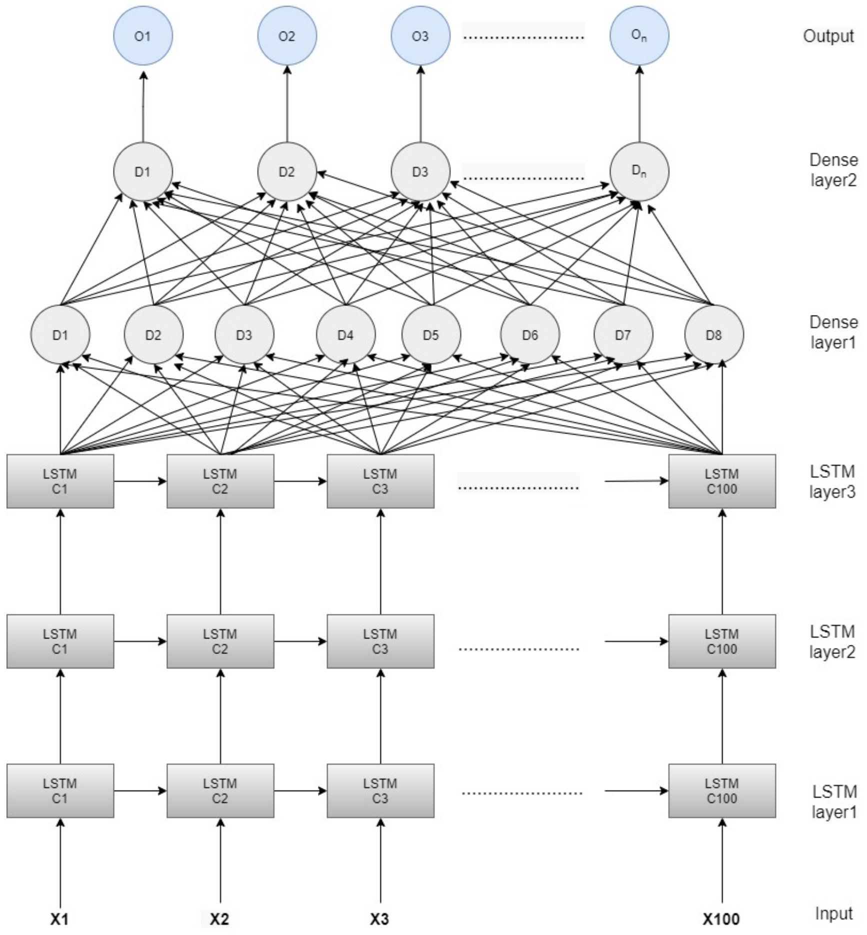


Figure 3.3: Architecture of the Proposed Model

The model we have built for predicting contents in a time series manner is a many to many type rnn model because we are receiving multiple outputs for multiple inputs. LSTM is a recurrent neural network model, which means we have to make our dataset as a series. So, we are using sequences for our model's input. In our model we have 3 layers of Lstm with 100 cells each, 2 dense layers in which the first dense layer includes 8 cells and the second dense layer includes n number of cells where n derives the number of movies we are using in our dataset. The output would be a view-count ratio for movies, relative to one another. So the highest watched movie will have the highest value; the second highest watched movie will have the second highest value and so on. The summation of all the values will be equal to 1. Figure 3.3 gives us a breif overview of how the model is structured.

3.3.1 Dataset

The dataset we have used is a Netflix rating records for different movies collected from Kaggle. But we could not use this data in it's raw form as we intended to predict contents in a time series manner using RNN. Thus, we processed it into a dataset, containing the count of views of the movie got every day. Then it's all compiled into a csv file as input, along with the week of the day. For this, we gave each day a value, from 1 to 7, which will later get normalized. For output, we take a day and give each movie a numeric value based on how much it has been watched. For the highest viewed content, it'll be N where N is the number of movies that can be cached, and the second viewed content will be N-1, and so on. N will be decided based on the amount of movies that can be cached in the server, and all the movies out of the range (N) with low view counts will share the value 0. After that, the values will be divided by N(N+1)/2, so the sum of all the values will be 1. We can later use a softmax function on the final layer to make sure we can match this pattern in the prediction.

[[$M_{1(t-100)}$ $M_{2(t-100)}$ $M_{3(t-100)}$ .... $M_{n(t-100)}$ (day of week=1)]

[$M_{1(t-99)}$ $M_{2(t-99)}$ $M_{3(t-99)}$ .... $M_{n(t-99)}$ (day of week=2)]

[$M_{1(t-98)}$ $M_{2(t-98)}$ $M_{3(t-98)}$ .... $M_{n(t-98)}$ (day of week=3)]

[$M_{1(t-97)}$ $M_{2(t-97)}$ $M_{3(t-97)}$ .... $M_{n(t-97)}$ (day of week=4)]

.

.

.

[$M_{1(t)}$ $M_{2(t)}$ $M_{3(t)}$ .... $M_{n(t)}$ (day of week=d)]]

Figure 3.4: Input Overview

3.3.2 Data Preparation for LSTM

LSTM is a recurrent neural network model, which means we have to make our dataset as a series. For this, we wrote a function, the makes a queue with capacity of n previous values. Then we iterate through the rows in the dataset and keep adding to the queue, and every time we have enough values, we keep adding the view-counts till time (t) to the X list, and the expected output value for t+1 would be added to the Y list. Then it'll all be converted into NumPy arrays and returned.

3.3.3 Data Preparation for Federated Learning Simulation

For this, we take the entire dataset and divide them among the base stations. To do this, we wrote a function, that takes the dataset and the number of base stations we are looking for. Then it makes a dictionary of the base stations with it's own set of input and output data, and returns the dictionary. The data is then batched for the model to train with.

3.3.4 Proposed FedPredict Algorithm

This is entire summary of the algorithm we implemented. We are referring it as the FedPredict Algorithm. At first the variables are initialized and transferred for further computation.

- Initialize W_c for Continental Server. Initialize Continental Prediction Model M_c
- Transfer W_c to Regional Server where $W_r \leftarrow W_c$. Initialize Regional Prediction Model M_r
- Transfer W_r to Base station where $W_b \leftarrow W_r$. Initialize Base Station Prediction Model M_b

```
Base station executes:
1: for each round t = $1, 2_{....}$ do
2:     $p^{t+1} \leftarrow$ Predict ($M_b$)
3:     Cache ($p^{t+1}$) // Cache movies based on available storage.
4:     PassToRegional ($W_b, V_b$)
5: end for

Regional server executes:
6: PassToRegional ($W_b$,$V_b$) / / Collect weights and view-counts from BS
7: for each round t = $1, 2_{....}$ do
8:     for $W_b \epsilon W_{bl}$ do
9:         $S_b \leftarrow (V_b)$ / Sum ($V_b$)
10:        $W_{bs} \leftarrow W_b * S_b$
11:    end for
12:    $W_r \leftarrow$ Sum ($W_{bs}$) / Count ($W_{bs}$)
13:    $M_r \leftarrow$ Update ($W_r$)
14:    $P^{t+1} \leftarrow$ Predict ($M_r$)
15:    Cache ($p^{t+1}$)
16:    PassToContinental ($W_r, V_r$)
17: end for

Continental server executes:
18: PassToContinental ($W_r, V_r$) / / Collect weights and view-counts from RS 19:
for each round t = $1, 2, \ldots$ do
20:    for $W_r \epsilon W_{re}$ do
21:        $S_r \leftarrow (V_r)$ / Sum ($V_r$)
22:        $W_{rs} \leftarrow W_r{}^* S_n$
23:    end for
24:    $W_c \leftarrow$ Sum ($W_{rs}$) / Count ($W_{rs}$)
25:    $M_c \leftarrow$ Update ($W_c$)
26:    $P^{t+1} \leftarrow$ predict ($M_c$)
27:    Cache ($p^{t+1}$)
28: end for
```

Chapter 4

Comparative Study

In this section we aim to compare various approaches, optimizers and loss functions that we have utilized. Attempting things multiple ways have given us a better end result as we have a better overview of how the model performs. We have tried to optimize performance through utilization of various conventional and available methods.

4.1 Optimizer Comparison

The ultimate goal of a neural network is to reduce the amount of errors in order to produce more accurate results. For this objective optimizer in a neural network model plays a very vital role in producing the best result possible for that particular model. For our content prediction lstm model we used different types of optimizers such as Nadam, Adam, Adamax, ftrl and RMSprop and compared the results of training and validation accuracy to determine which optimizer we should use for our prediction model.

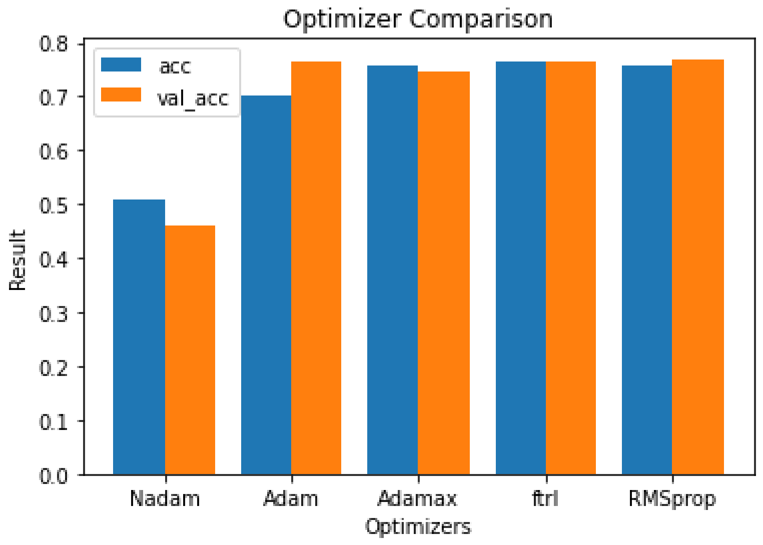


Figure 4.1: Comparison of Various Optimizers

For the above comparison shown in the Figure(4.1) we set the learning rate initially to 0.00005 and ran 25 epochs for each optimizer in our prediction model and generated the training accuracy (acc) and validation accuracy (valacc). From the above results we can see that RMSprop produces the highest number of validation accuracy of 0.7695. As a result we choose RMSprop as the optimizer of our prediction model.

4.2 Loss Function Comparison

Along with Optimizers, Loss function is a vital part of any neural network model. Typically we seek to minimize error or loss of any neural network in order to find a satisfactory result. In our prediction model we used different loss functions like Huber loss, Mean squared error, Categorical hinge, Mean absolute error, Squared hinge and compared the results to choose which loss function works best for our model.

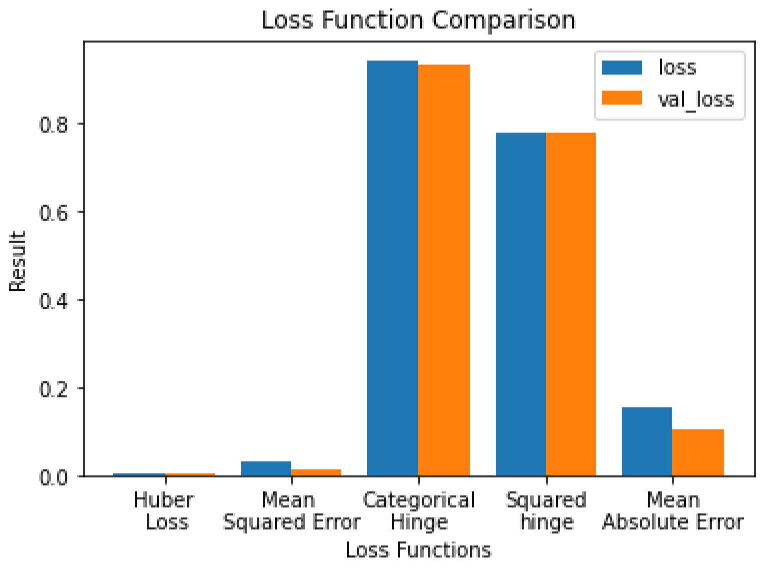


Figure 4.2: Comparison of Various Loss Functions

For the above comparison shown in the Figure(4.2) we used the optimizer as RMS prop and set the learning rate initially to 0.00005 and ran 25 epochs for each functions using our model for producing both training loss and validation loss. For Huber loss as we can clearly observe from the fig that the model produced minimum amount of loss which is 0.0336 training loss and 0.0156 validation loss. As a result we chose to use Huber loss as the loss function of our prediction model.

Chapter 5

Result Analysis

The main objective of our prediction model merged with Fedarated learning was to predict contents accurately for a particular timespan and pass our model parameters to the next layers without sharing any user data. The prediction results of our proposed models are satisfactory. According to our system model, the initial weight generated by the continental server is transferred down to the regional server and from there to the base stations at the initializing state. Using that generic weight, our base station prediction model trained and updated itself to predict movies for time(t+1) at time(t). Using RMSprop as optimizer and huber_loss as loss function, after 25 epochs our base station prediction model was able to produce 71% validation accuracy which we can observe from the Figure(5.1) below

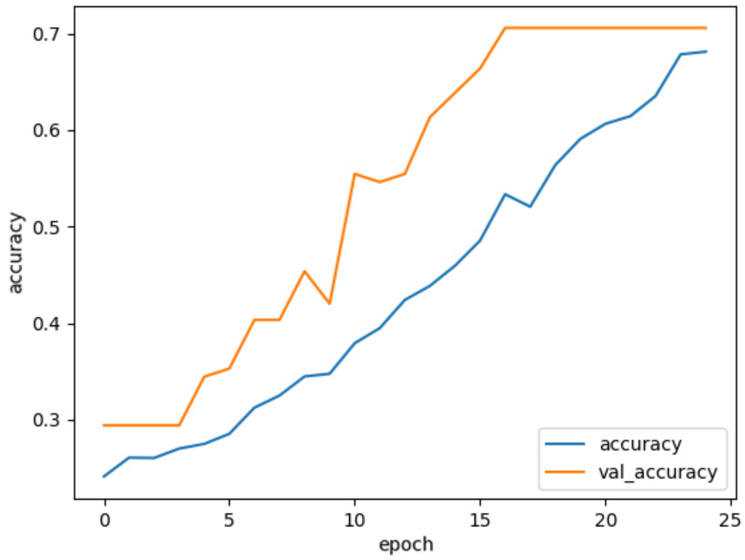


Figure 5.1: Training Accuracy and Validation Accuracy Graph

From the figure(5.2) below we can see that our base station prediction model is quite accurate and stable as both validation and training accuracy seem to converge smoothly.
After the base station model is updated, the updated parameters are passed to the regional servers where the regional server initially using those parameters updates its own prediction model and after 3 communication rounds for each regional server,

those updated parameter models are passed to the Continental Server. With those parameters generated by the regional servers our continental server predicts movies for time(t+1) in time(t).

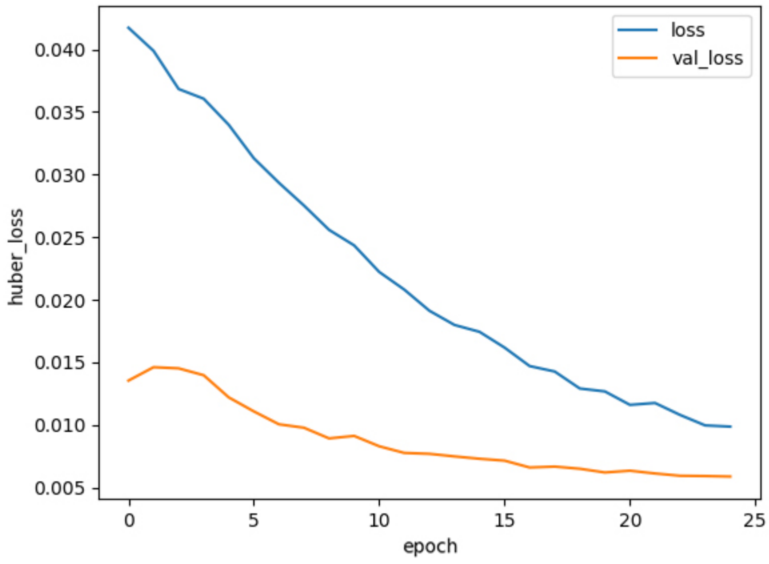
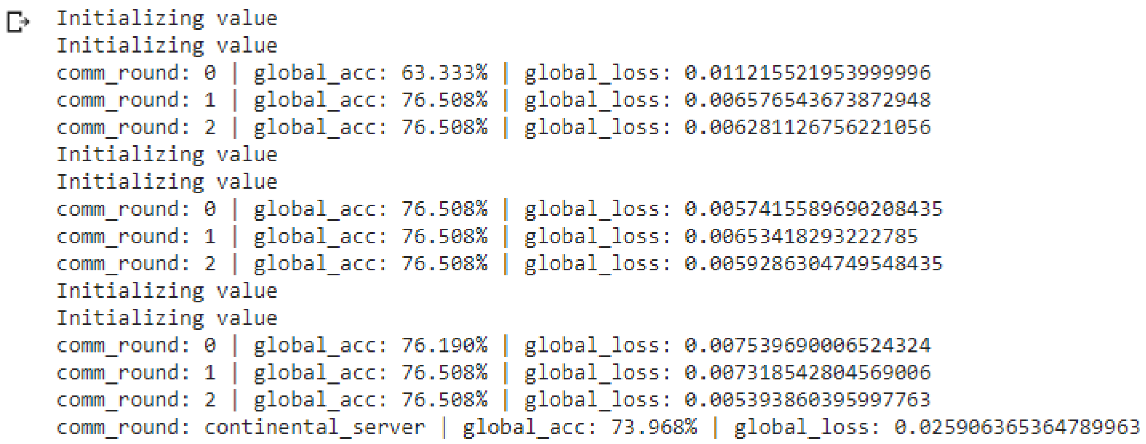


Figure 5.2: Training Loss and Validation Loss Graph

From the figure(5.3) below we can see that our Continental server is able to produce 73.968% global accuracy in just one complete cycle. So it is evident that with high number of computational cycles and larger and versatile dataset our prediction model can produce even better results.



```
Initializing value
Initializing value
comm_round: 0 | global_acc: 63.333% | global_loss: 0.011215521953999996
comm_round: 1 | global_acc: 76.508% | global_loss: 0.006576543673872948
comm_round: 2 | global_acc: 76.508% | global_loss: 0.006281126756221056
Initializing value
Initializing value
comm_round: 0 | global_acc: 76.508% | global_loss: 0.0057415589690208435
comm_round: 1 | global_acc: 76.508% | global_loss: 0.00653418293222785
comm_round: 2 | global_acc: 76.508% | global_loss: 0.0059286304749548435
Initializing value
Initializing value
comm_round: 0 | global_acc: 76.190% | global_loss: 0.007539690006524324
comm_round: 1 | global_acc: 76.508% | global_loss: 0.007318542804569006
comm_round: 2 | global_acc: 76.508% | global_loss: 0.005393860395997763
comm_round: continental_server | global_acc: 73.968% | global_loss: 0.025906365364789963
```

Figure 5.3: Prediction result for continental server

If we take a look at the figure(5.4) below we can see that on average each regional server computation round has an average accuracy of more than 70

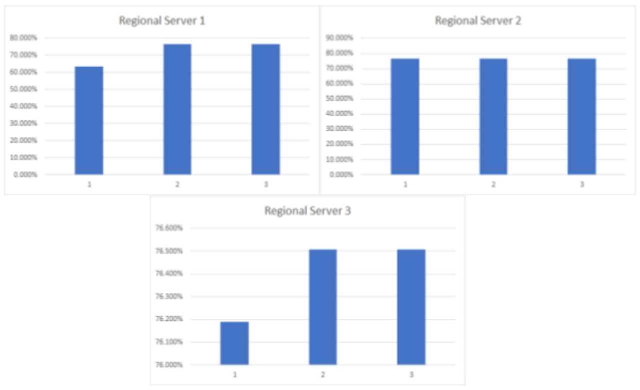


Figure 5.4: Regional server computation round

Now to compare our federated approach we ran similar tests in a non-federated approach. This basically gave us an overview of the entire system. We can see that without applying federated learning, our global model accuracy is significantly lower than our previous one. We get around 67% accuraccy whereas previously the global accuracy stood around 73%.

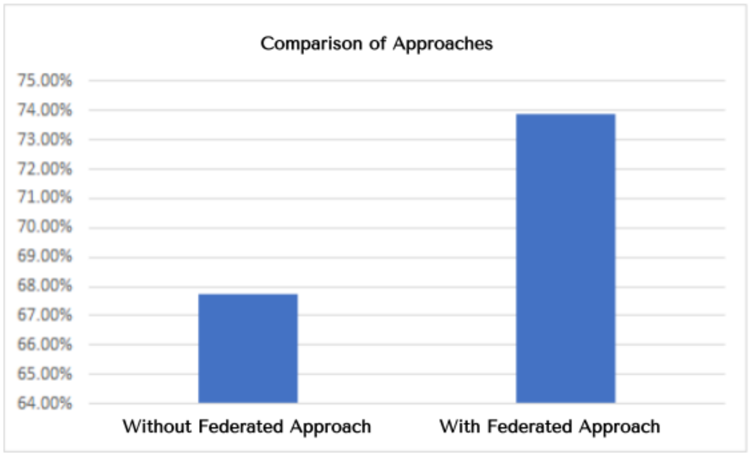


Figure 5.5: Federated vs Non-Federated Approach

Chapter 6

Conclusions

6.1 Attempted Approaches and Challenges Faced

While preparing the dataset and implementing our algorithm, we faced multiple challenges so we made adjustments on the way. We divide our approach into four separate parts. In each phase we have made a distinct modification to achieve our target goal. At the end we reached a point where we got satisfactory results amidst the obstacles faced. We were able to successfully maneuver around the problems that arose. Further modifications and improvements can be made which is discussed later on.

At first, we tried to predict the view-count for a movie. Our plan was to at first make a sequence of view-count for a movie and run that through RNN to predict how many people will watch the movie at time t+1. But while trying this approach, we faced a few difficulties. First of all, it'll be tough to make one model for each movie and then try to predict a count for every single movie, it'd take a lot of time for no good reason. Another big problem that we faced was that, our model was struggling to predict the extremes in values. And if were to compare the predicted view count of movies, it was important to be as precise as possible with the predictions, which wasn't working out as we hoped for.

Failing at approach 1, we decided to try a second approach. For this, we took a bunch of movies to train and test our theory on. At first, we took the view counts of the movies for each day, and made a list of lists out of it. The outer list containing lists in a time series and the inner list containing the view counts. For output, we used a one-hot vector to predict the most popular movie. Then we make a sequence of this data using queue and pass it thorough our model. It worked okay, but the problem remained that we were getting only one movie, and using this approach to figure out top, most viewed movies would be hard.

So,to deal with this issue, we decided to not use one hot vector and give each movie a numeric value based on view-count. The highest viewed movie gets the value n, where n is the number of movies we can cache. The number will slowly get reduced to 1 as the view-count for the movie decreases, and the last few movies will have the value 0, as in they won't get cached. This approach worked pretty well, the predicted values didn't quite output an exact zero, but we could just cache the top valued movies from the output.

This is our final approach that we used for our model. For this approach we just added one extra feature on top of approach 3. We added the day of the week. We figured, the current day of the week can have an effect on the movie that gets popular, so we gave each day a numeric value from 1 to 7 and scaled it to stay between 1 and 0.

6.2 Summary and Future Work

The aim of our thesis was to build a complete system model which accurately predicts contents while ensuring user privacy and determines efficiently what to cache. Our system model will enable faster content delivery by reducing total network hop count and maximize cache hit ratio. The structure and approach of our model enables us to reduce computational complexity and communication overhead. Though there are many aspects that we can improve in future.

• While implementing there were some obstacles that we had to face like scarcity of time series based content dataset for which we needed to modify the dataset for our implementation purpose. With better time series based dataset and federated enabled datasets we can achive greater result for our model.

• We used the Fedavg algorithm for our federated learning purpose. Though Fedavg algorithm works well in practice but it is not perfect. There are very recent upgrades to this algorithm like q-FedAvg and FedProx which if intigrated in our system might produce better results.

Bibliography

[1] F. Richter, *Infographic: Netflix passes 200 million milestone*, Jan. 2021. [Online]. Available: https://www.statista.com/chart/3153/netflix-subscribers/.

[2] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 46–51, 2020.

[3] *Fedmd: Heterogeneous federated learning via model distillation.* [Online]. Available: https://towardsdatascience.com/tagged/privacy-preserving.

[4] J. Ni, K. Zhang, and A. V. Vasilakos, "Security and privacy for mobile edge caching: Challenges and solutions," *IEEE Wireless Communications*, 2020.

[5] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[6] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[7] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1818–1831, 2017.

[8] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, e3493, 2018.

[9] H. Wu, Y. Luo, and C. Li, "Optimization of heat-based cache replacement in edge computing system," *The Journal of Supercomputing*, pp. 1–34, 2020.

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

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

[12] J. George and S. Sebastian, "Cooperative caching strategy for video streaming in mobile networks," in *2016 International Conference on Emerging Technological Trends (ICETT)*, IEEE, 2016, pp. 1–7.

[13] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2017.

[14] C. Li, J. Tang, H. Tang, and Y. Luo, “Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment,” *Future Generation Computer Systems*, vol. 95, pp. 249–264, 2019.

[15] W. Stokowiec, T. Trzciński, K. Wołk, K. Marasek, and P. Rokita, “Shallow reading with deep learning: Predicting popularity of online content using only its title,” in *International Symposium on Methodologies for Intelligent Systems*, Springer, 2017, pp. 136–145.

[16] O. Tsur and A. Rappoport, “What’s in a hashtag? content based prediction of the spread of ideas in microblogging communities,” in *Proceedings of the fifth ACM international conference on Web search and data mining*, 2012, pp. 643–652.

[17] G. Chen, Q. Kong, N. Xu, and W. Mao, “Npp: A neural popularity prediction model for social media content,” *Neurocomputing*, vol. 333, pp. 221–230, 2019.

[18] S. D. Roy, T. Mei, W. Zeng, and S. Li, “Towards cross-domain learning for social video popularity prediction,” *IEEE Transactions on multimedia*, vol. 15, no. 6, pp. 1255–1267, 2013.

[19] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[20] H. Sak, A. W. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling,” 2014.

[21] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with lstm,” *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[22] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, “A novel connectionist system for unconstrained handwriting recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 855–868, 2008.

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

[24] K. Greff, R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[25] J.-H. Ahn, O. Simeone, and J. Kang, “Wireless federated distillation for distributed edge learning with heterogeneous data,” in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, IEEE, 2019, pp. 1–6.

[26] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, “Parallelized stochastic gradient descent.,” in *NIPS*, Citeseer, vol. 4, 2010, p. 4.

[27] Y. Li, T.-H. Chang, and C.-Y. Chi, “Secure federated averaging algorithm with differential privacy,” in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, IEEE, 2020, pp. 1–6.

[28] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, “A performance evaluation of federated learning algorithms,” in *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, 2018, pp. 1–8.

[29] C. Dwork, A. Roth, *et al.*, “The algorithmic foundations of differential privacy.,” *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[30] A. Sharma, “Guided stochastic gradient descent algorithm for inconsistent datasets,” *Applied Soft Computing*, vol. 73, pp. 1068–1080, 2018.