SMART TRAFFIC PREDICTION AND CACHING USING MACHINE LEARNING TECHNIQUES IN FOG RADIO ACCESS NETWORKS

Report submitted to the SASTRA Deemed to be University as the requirement for the course

BCSCCS708: **MINI PROJECT**

Submitted by

RAGURU SAI SANDEEP
(Reg. No.: 121003216, B.Tech - Computer Science and Engineering)
VEERAVALLI SATYA MANASA
(Reg. No.: 121003303, B.Tech - Computer Science and Engineering)
VEGULLA NAGA SAI SOWMITRI
(Reg. No.: 121003304, B.Tech - Computer Science and Engineering)

December 2021

SASTRA
ENGINEERING - MANAGEMENT - LAW - SCIENCES - HUMANITIES - EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**Smart Traffic Prediction and Caching using Machine Learning Techniques in Fog Radio Access Networks**" submitted as a requirement for the course, BCSCCS708**: MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Raguru Sai Sandeep (Reg. No. 121003216, B.Tech - Computer Science and Engineering**), **Mr. Veeravalli Satya Manasa (Reg. No. 121003303, B.Tech - Computer Science and Engineering**) and **Mr. Vegulla Naga Sai Sowmitri (Reg. No. 121003304, B.Tech - Computer Science and Engineering**) during the academic year 2020-21, in the School of Computing, under my supervision.

Signature of Project Supervisor : MEENA V
Name with Affiliation :
Date :

Mini Project *Viva voc*e held on ________________

Examiner 1 Examiner 2

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING THANJAVUR–613 401

DECLARATION

We declare that the thesis titled "**Smart Traffic Prediction and Caching using Machine Learning Techniques in Fog Radio Access Networks**" submitted by us is an original work done by us under the guidance of **MEENA V** AP-III School of Computer Science and Engineering, SASTRA Deemed to be University during the sixth semester of the academic year 2020-21, in the School of Computer Science and Engineering. The work is original and wherever We have used materials from other sources, We have given due credit and cited them in the text of the thesis. This thesis has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Name of the candidates :

1. Raguru Sai Sandeep (121003216)
2. Veeravalli Satya Manasa (121003303)
3. Vegulla Naga Sai Sowmitri (121003304)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING THANJAVUR–613 401

ACKNOWLEDGEMENT

First of all, we express our gratitude to **Prof. Dr. S Vaidhyasubramaniam**, **Vice-Chancellor, SASTRA Deemed University** who provided all facilities and necessary encouragement during the course of our study. We extend our sincere thanks to **Prof. Dr. R. Chandramouli**, **Registrar, SASTRA Deemed University** for Providing the opportunity to pursue this mini project.

It is our privilege to express our sincerest regards to our **Dr.A.Uma Maheswari, Dean(SOC) , Dr.Shankar Sriram VS, Associate Dean (CSE) and Dr.B.Shanthi, Associate Dean (Sponsored Research)** who motivated us during the mini project work.

We owe a debt of deepest gratitude to our mentor **Meena V** for her valuable inputs, able guidance, encouragement, wholehearted cooperation and constructive criticism throughout the duration of our mini project on the topic "**Smart Traffic Prediction and Caching using Machine Learning Techniques in Fog Radio Access Networks**". We have to appreciate the guidance given by the panels especially in our mini project presentation that has improved our presentation skills thanks to their comments and advice.

List of Figures

SMART TRAFFIC PREDICTION AND CACHING USING MACHINE LEARNING TECHNIQUES IN FOG RADIO ACCESS NETWORKS

ABSTRACT

There is a rapid increase in use of wireless communication all over the world in the past ten years. Due to this rapid development of wireless communication and smart devices, the traditional architecture of cloud is unable to meet the increase in the requirements day to day. To meet the requirements efficiently and to provide good quality of service to the end users F-RAN has become a solution. But there are some challenges to be resolved regarding infrastructure, network traffic and caching mechanism of F-RANs.

This paper provides a network traffic flow prediction algorithm based on LSTM mechanism to predict real time network traffic of different data types. This paper also proposed a collaborative filtering method and cognitive caching based on LSTM to reduce the total communication delay.

KEY WORDS: LSTM, Traffic, Communication, Caching, Challenges, Cloud.

SUMMARY OF BASE PAPER

Cloud data access is increasing day by day. To meet user demand with better quality of service to the end user Fog-Random Access networks (F-RANs) are introduced. To make F-RANs more efficient in real time, the real time traffic prediction is to be done and better caching strategies are to be used for effective quality of service to the end user. In this project we used LSTM (long short term memory) algorithm for network traffic prediction and we compared different caching strategies. The basic cloud architecture is a centralized one. Usage of cloud data is increasing now-a-days and might drastically increase in near future. Since the basic cloud architecture is a centralized one it may not be efficient in terms of quality of service from cloud as user demand increases. While F-RANs make the basic cloud architecture a decentralized one by storing the frequently accessed cloud data in local cloud.

The data to be stored in local cloud cache is determined based on network traffic and caching strategy that is used to store the data requests in the local cloud. We performed LRU (least recently used) and LFU (least frequently used) caching strategies of which LFU that is least frequently used is found to be best strategy over long term. This local cloud and caching strategies are actually helpful when the real time traffic crosses the threshold traffic rate. How often the network traffic crosses the threshold traffic rate in future is predicted using LSTM(long short term memory) which is an extension of recurrent neural networks and is found to be best one in predicting real time traffic. F-RANs are the best solution to improve quality of services by cloud to the end user and they can be improved with cognitive learning in real time like traffic prediction which might be helpful in future forecast of traffic and caching strategies to store the requested cloud data.

ABBREVIATIONS

F-RAN	- Fog Radio Access Network
QoS	- Quality of Service
QoE	- Quality of Experience
D2D	- Device to Device Communication
C-RAN	-Cloud Random Access Network
RRH	- Radio Remote Heads
CBS	- Cloud Based Service
BS	- Base Station
LSTM	- Long short term memory
LFU	- Least frequently used
LRU	- Least recently used
F-AP	- Fog access pointers

Table of Contents

Literature Survey

1. Throughput Maximization in C-RAN System (Cloud Random Access Network System)-2015

   - Maximizing the total throughput of cloud radio access networks (CRANs) in which multiple radio remote heads (RRHs) are connected to a central computing unit known as the cloud.
   - Cloud radio access networks (CRANs) total throughput is maximized by a central computing unit known as cloud to which multiple radio remote heads (RRHs) are connected.
   - Hence, the throughput is maximized by this paper which is defined as the number of correctly received bits.

2. Content Delivery in D2D Networks-2017

   - Evaluating the throughput performance of a mobile network cell where device to device (D2D) communication, using part of the resource blocks of the considered cell, is used to offload the base station (BS) of part of its content delivery traffic.
   - A multi-class processor sharing model is developed to determine the optimal resource partition between D2D and BS subsystems and evaluate the resulting throughput performance.

3. CBS Offloading in F-RAN System (Cloud based Service)-2018

   - Maximizing cloud offloading by finding an efficient schedule of coded file transmissions from the eRRHs and the cloud base station (CBS).

4. Cooperative F-RAN and D2D Communications (Device to Device Communication)-2018

   - Device-to-Device (D2D) communication can support the operation of cellular systems by reducing the traffic in the network infrastructure.
   - Offline caching, out-of-band D2D communication, and an F-RAN with two edge nodes and two user equipments, an information-theoretically optimal caching and delivery strategy is presented that minimizes the delivery time in the high signal- to- noise ratio regime.
   - The delivery time accounts for the latency caused by fronthaul, downlink, and D2D transmissions.

CHAPTER 1

Architecture

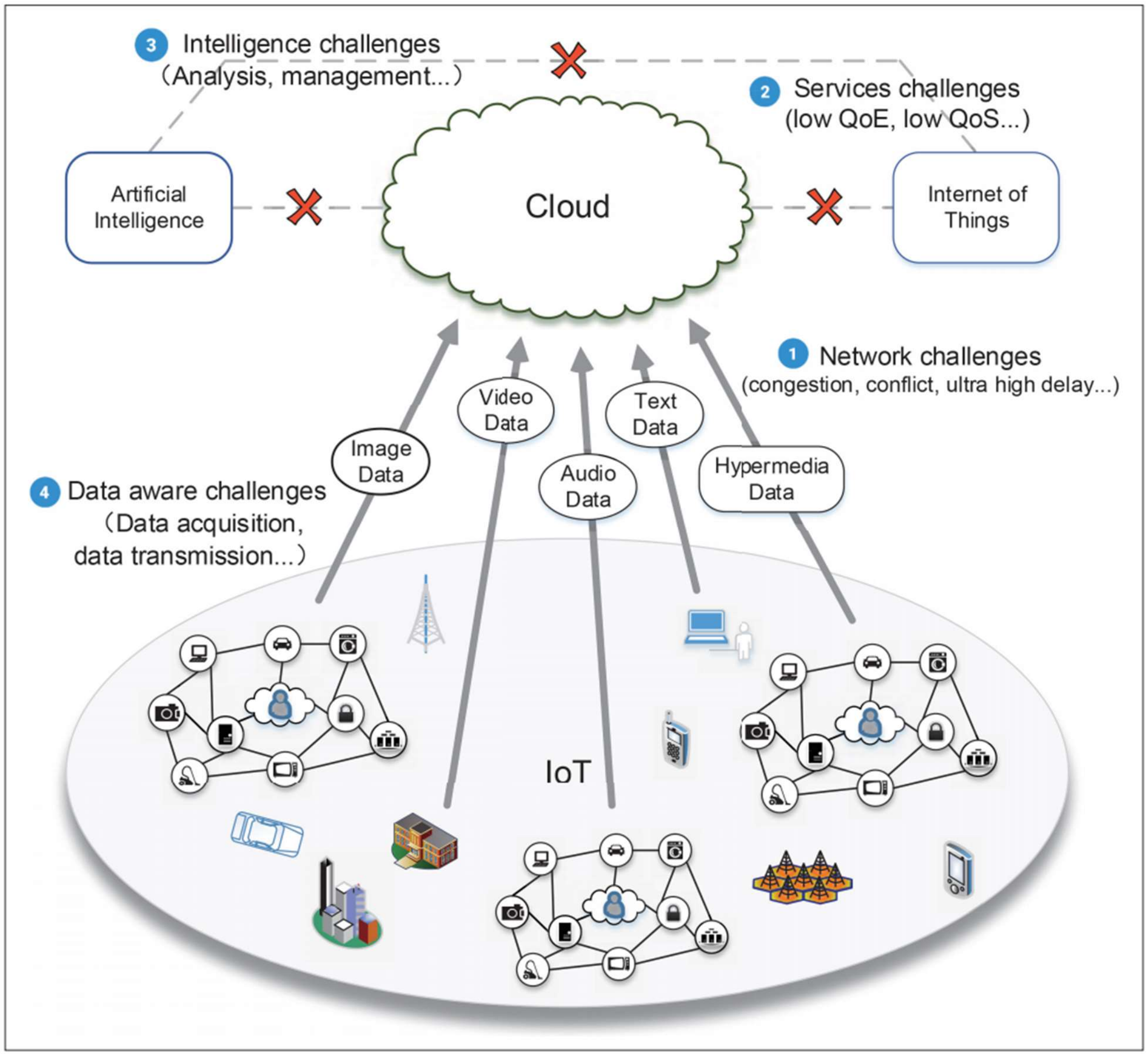


Fig 1.1 Traditional cloud computing based IoT architecture

- <u>Challenges Faced:</u>
  1.Network challenges
  2.Services challenges
  3.Intelligence challenges
  4. Data aware challenges

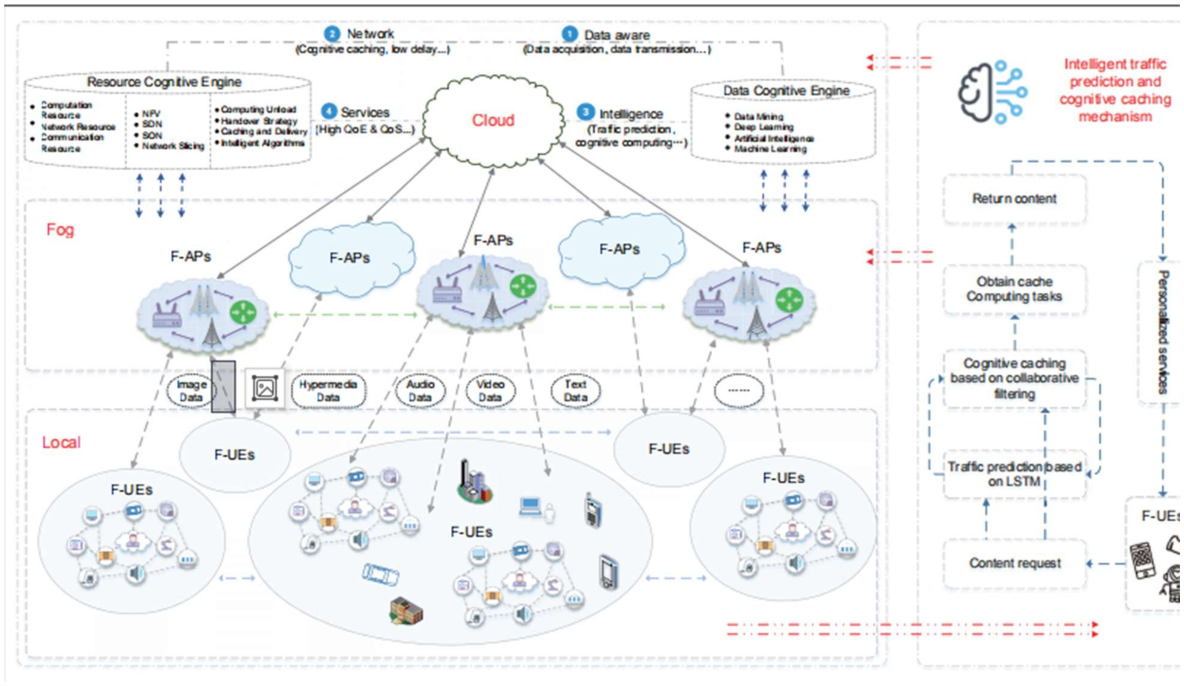

Network (Cognitive caching, low delay...)
Data aware (Data acquisition, data transmission...)
Resource Cognitive Engine
Services (High QoE & QoS...)
Cloud
Intelligence (Traffic prediction, cognitive computing...)
Data Cognitive Engine
Intelligent traffic prediction and cognitive caching mechanism
Fog
F-APs
Local
F-UEs
Image Data
Hypermedia Data
Audio Data
Video Data
Text Data
Return content
Obtain cache Computing tasks
Cognitive caching based on collaborative filtering
Traffic prediction based on LSTM
Content request
Personalized services

Fig 1.2 Proposed Fog architecture

F-APs are fog access points which store the local frequently requested data and makes retrieval of data easy whenever requested by the user.

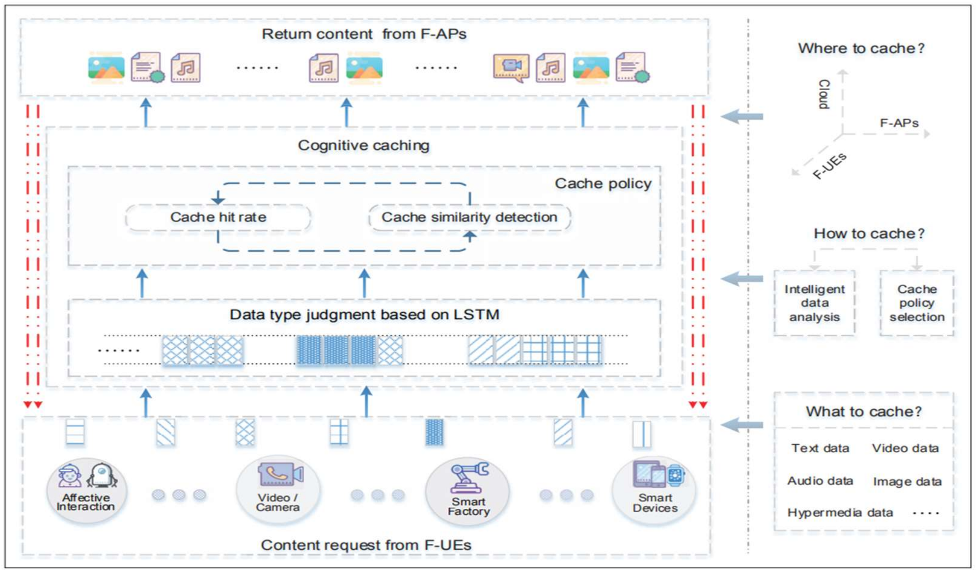


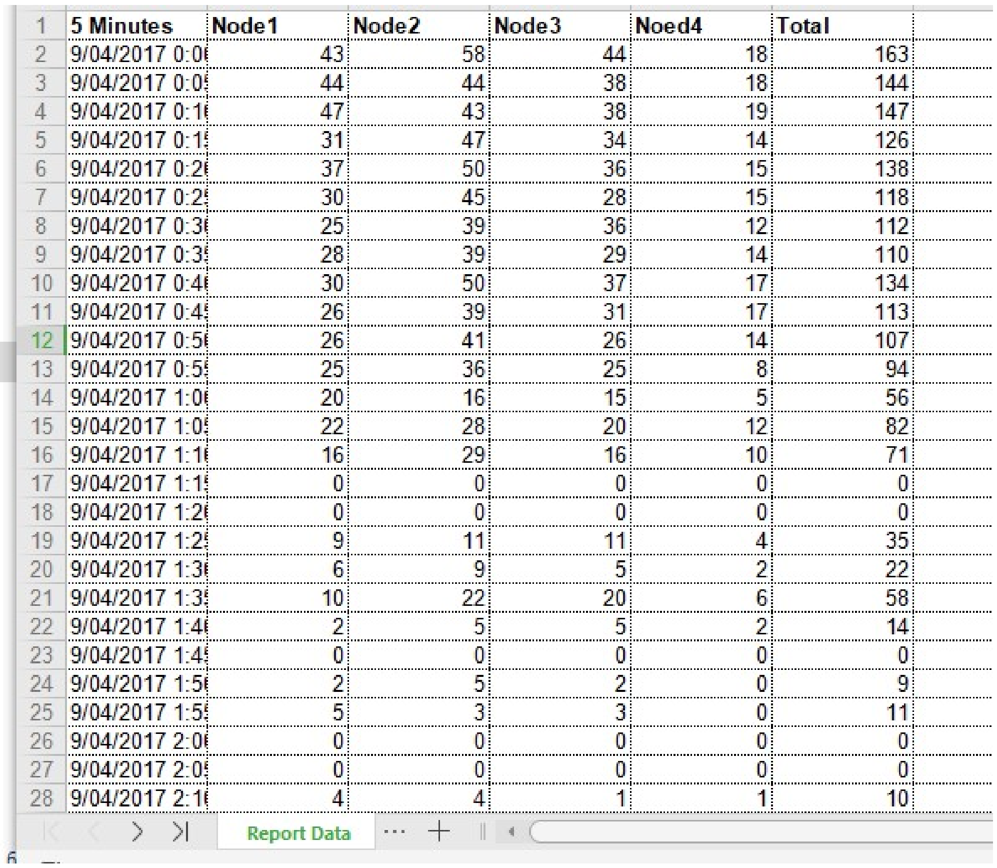
Fig 1.3 Mechanism of traffic prediction and caching strategies

User requests for data and data type of requested data is found based on LSTM. The requested data is searched in the local cache of F-RAN. If the data is found then the data is returned to the requested user directly from F-RAN without the intervention of the main cloud. If the data is not present in the fog cache then the data is requested by F-RAN from the central cloud and is stored in cache to satisfy the user request.

CHAPTER 2

DATA SET

Data collected contains 4 node request rates to a cloud for every timestamp of 5 minutes over two months period.



1	5 Minutes	Node1	Node2	Node3	Noed4	Total
2	9/04/2017 0:0	43	58	44	18	163
3	9/04/2017 0:0	44	44	38	18	144
4	9/04/2017 0:1	47	43	38	19	147
5	9/04/2017 0:1	31	47	34	14	126
6	9/04/2017 0:2	37	50	36	15	138
7	9/04/2017 0:2	30	45	28	15	118
8	9/04/2017 0:3	25	39	36	12	112
9	9/04/2017 0:3	28	39	29	14	110
10	9/04/2017 0:4	30	50	37	17	134
11	9/04/2017 0:4	26	39	31	17	113
12	9/04/2017 0:5	26	41	26	14	107
13	9/04/2017 0:5	25	36	25	8	94
14	9/04/2017 1:0	20	16	15	5	56
15	9/04/2017 1:0	22	28	20	12	82
16	9/04/2017 1:1	16	29	16	10	71
17	9/04/2017 1:1	0	0	0	0	0
18	9/04/2017 1:2	0	0	0	0	0
19	9/04/2017 1:2	9	11	11	4	35
20	9/04/2017 1:3	6	9	5	2	22
21	9/04/2017 1:3	10	22	20	6	58
22	9/04/2017 1:4	2	5	5	2	14
23	9/04/2017 1:4	0	0	0	0	0
24	9/04/2017 1:5	2	5	2	0	9
25	9/04/2017 1:5	5	3	3	0	11
26	9/04/2017 2:0	0	0	0	0	0
27	9/04/2017 2:0	0	0	0	0	0
28	9/04/2017 2:1	4	4	1	1	10

Fig 2.1 Sample dataset

CHAPTER 3

LSTM (Long Short Term Memory)

LSTM is an artificial recurrent neural network used for time series prediction. It has feedback connections unlike standard feed forward neural networks. As a result of the feedback connections in LSTM neural network the problem of vanishing node over long time periods that usually occur with standard recurrent neural networks can be avoided. This makes LSTM more efficient in time series predictions.

In this project LSTM is used for network traffic prediction by giving a series of network traffic over time as input to the LSTM model. The collected network traffic records over 2 months time are split into train and test data in 70-30 ratio. This data is normalized and is given to the LSTM model created.

LSTM model is created with 16 nodes with look back, a dense layer is added to the model to get normalized result of the model. Error metric used is mean squared error and ‘Adam’ optimizer is used as it uses an adaptive weights estimation of nodes in the model.

RMSE (root mean square error) is the metric used for efficiency calculation. The formula for RMSE calculation is

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} \left(x_i - \hat{x}_i\right)^2}{N}}$$

RMSD	= root-mean-square deviation
i	= variable i
N	= number of non-missing data points
x_i	= actual observations time series
$\hat{x}_i$	= estimated time series

The average RMSE value is found to be 0.01 for the network traffic prediction.

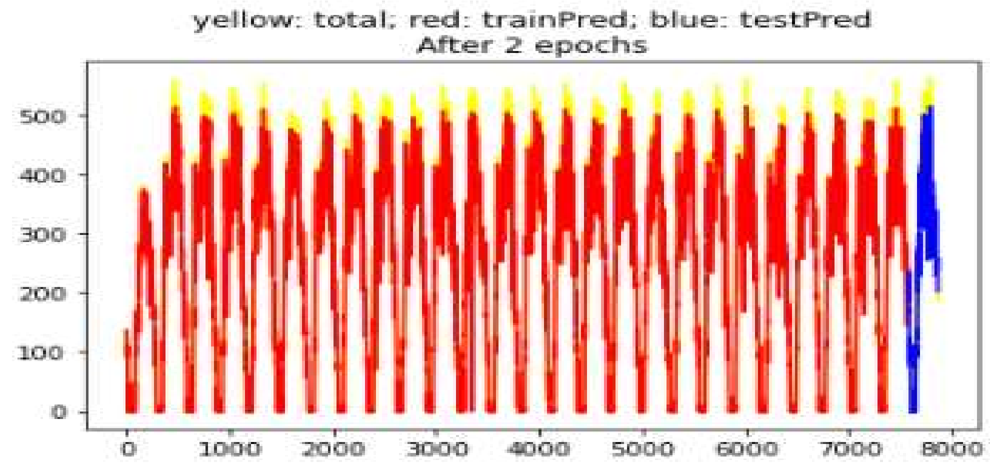


Fig 3.1 LSTM traffic prediction

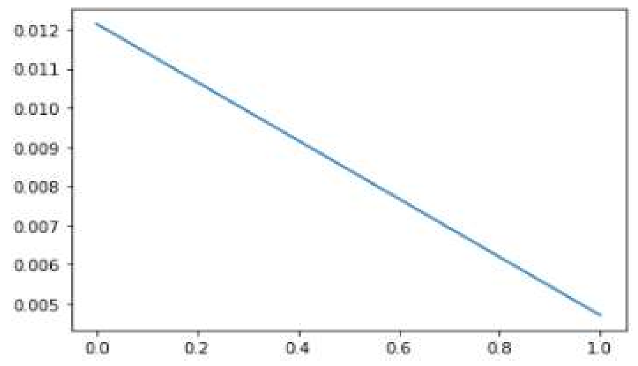


Fig 3.2 Loss plot over records of the dataset

These are the experimental outcomes of traffic prediction using LSTM(long short term memory).

CHAPTER 4

LRU (Least Recently Used)

Least Recently Used is a caching strategy in which the data used long back that is less recently is removed from the cache and that free space is allotted to the new request of data made by user. In this project the least recently requested node locations of cache are allocated to the current request. That is initially all nodes are given equal space in the cache to store the data requests made by users from that local nodes. As time goes on if a node requires more space to satisfy user requests at a particular time stamp then the least recently used node data space will be allocated for the current node request.

For this estimation of least recently used and most recently used an array named index is maintained whose size is equal to number of slots in the cache. Each time if there is a cache hit then the value at that index location is incremented and each time a slot being allocated to new node data that particular location of index array is also incremented as it is recently used one. To find least recently used location or node data in the cache the location of index array with minimum value is found out.

Hit ratio and fault ratio are calculated over time to estimate efficiency of the caching algorithm. The formula used is:

Hit Ratio: $$HR = \frac{hits}{hits + misses} = 1 - MR$$

Miss Ratio: $$MR = \frac{misses}{hits + misses} = 1 - HR$$

The variation of hits and faults of different caching strategies are used to compare caching policies.

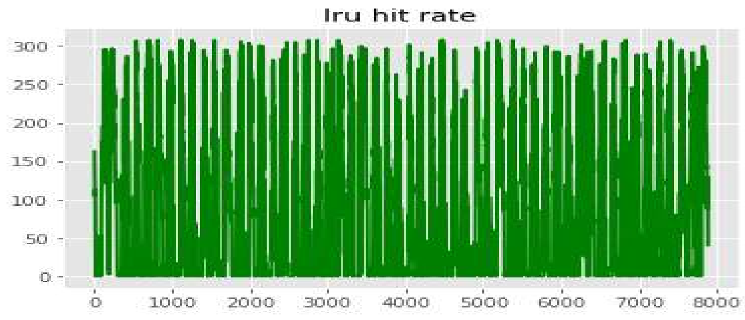


Fig 4.1 Variation of hit rate over time using LRU

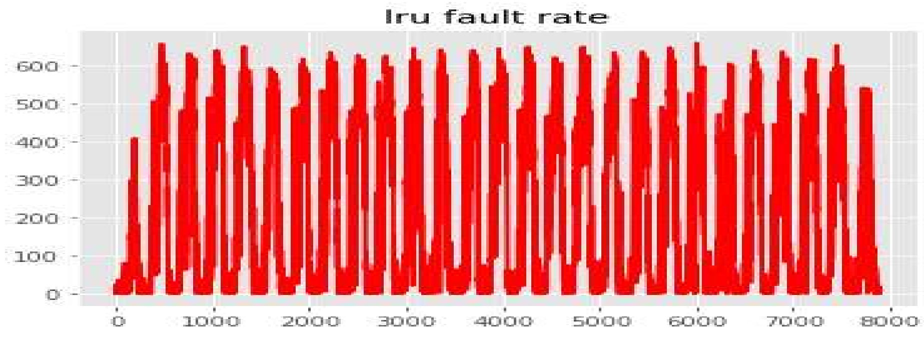


Fig 4.2 Variation of fault rate over time using LRU

CHAPTER 5

LFU (Least Frequently Used)

Least Frequently Used is a caching strategy in which less frequently requested data is removed from cache and the free space is allocated to the new request of the data made by the user. In this project the least frequently requested node requests are allocated to the current node request. . That is initially all nodes are given equal space in the cache to store the data requests made by users from that local nodes. As time goes on if a node requires more space to satisfy user requests at a particular time stamp then the least recently used node data space will be allocated for the current node request.

For this estimation of least frequently requested node an array named freq is created to store the frequency of requests made by each node from the beginning. Each time when a new location is required to satisfy requests at current timestamp the node with least frequent requests made is found and that node location in the cache are allocated to the current request made by the user.

Hit ratio and fault ratio are calculated over time to estimate efficiency of the caching algorithm. The formula used is:

Hit Ratio: $$HR = \frac{hits}{hits + misses} = 1 - MR$$

Miss Ratio: $$MR = \frac{misses}{hits + misses} = 1 - HR$$

The variation of hits and faults of different caching strategies are used to compare caching policies.

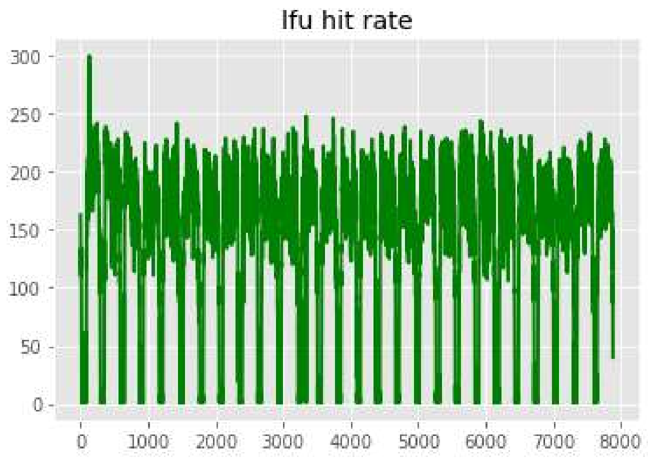


Fig 5.1 Variation of hit rate over time using LFU

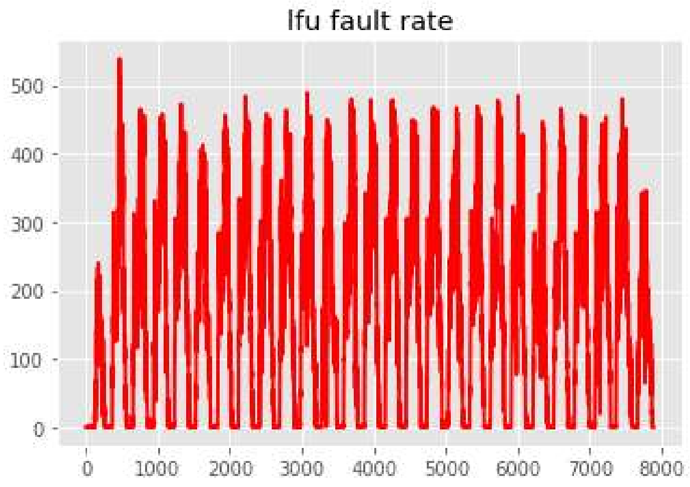


Fig 5.2 Variation of fault rate over time using LRU

CHAPTER 6

Comparison of LRU and LFU

To find the efficient caching strategy the cumulative cache hits and faults plot of both LRU and LFU is made in the same graph. Also the variation of hit rate and fault rate of LRU and LFU is made as a plot of which LFU is found to be more stable one in terms of hits while LRU hit rate has lot of fluctuations which shows that the algorithm is not efficient compared to LFU.

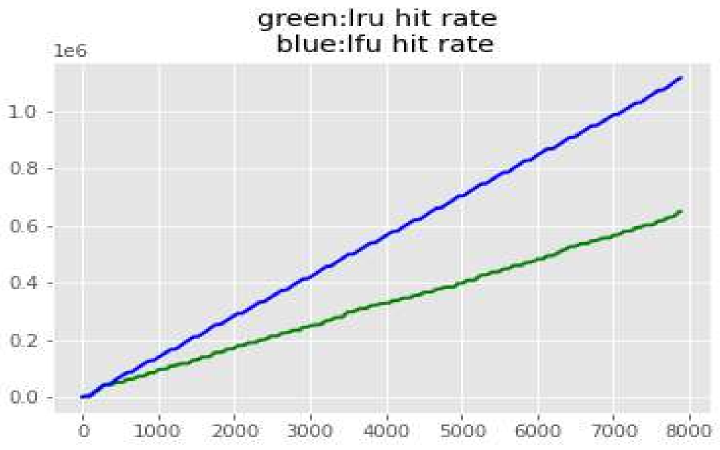


Fig 6.1 Hit rate comparison (cumulative)

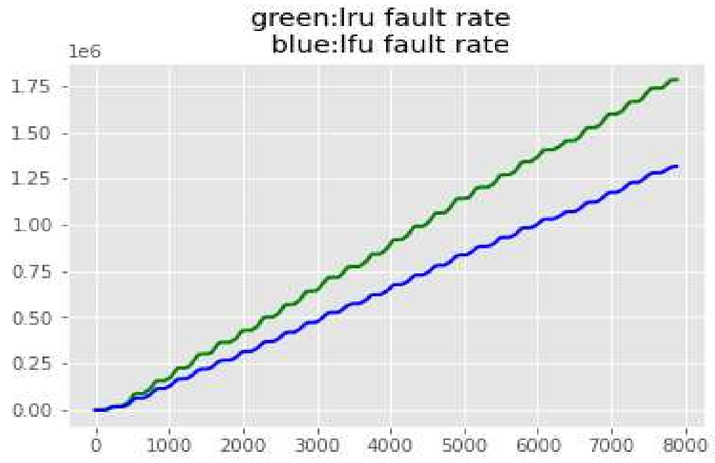


Fig 6.2 Fault rate comparison (cumulative)

From the plots of cumulative hit and fault rates of LRU and LFU, comparatively LFU is found to be best caching policy.

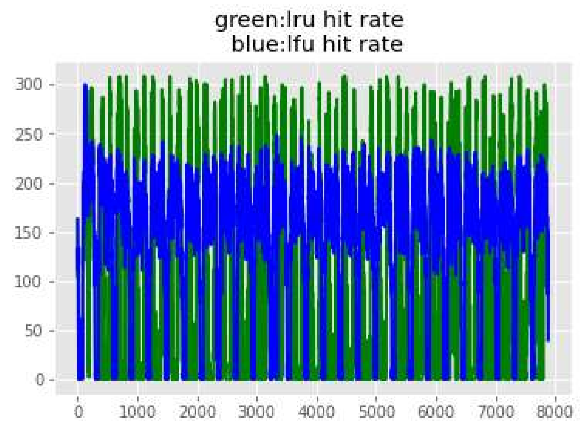


Fig 6.3 Variation of hit rate

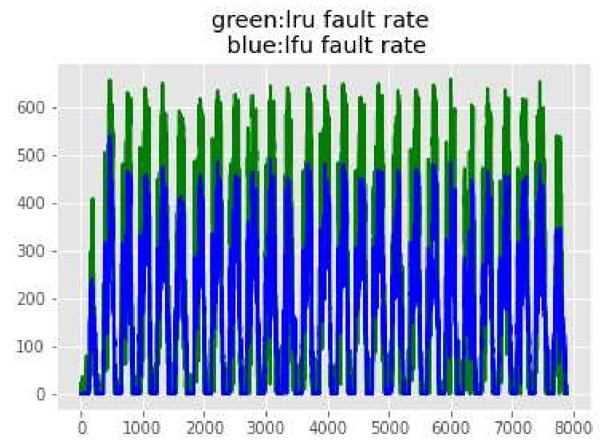


Fig 6.4 Variation of fault rate

CHAPTER 7

CONCLUSION

First, the traditional cloud-based IoT architecture is discussed, and the development history of F-RANs in view of its shortcomings is summarized. To meet the requirements efficiently and to provide good quality of service to the end users F-RAN has become a solution. But there are some challenges to be resolved regarding infrastructure, network traffic and caching mechanism of F-RANs. So an attention-based LSTM algorithm is used to predict the service traffic flow with different requested data types. Then a cognitive caching strategy based on LSTM and collaborative filtering is designed. The performance of the proposed LSTM-based traffic prediction algorithm and cognitive caching strategy is evaluated using simulation of IF-RAN architecture in a real environment.

The results prove that LFU is efficient compared to LRU and the idea of caching becomes more efficient on predicting real time traffic that is caching becomes more useful when the traffic crosses a particular cloud request traffic rate.

CODE IMPLEMENTATION

1)LSTM(Long Short Term Memory) PREDICTION

```
import numpy as np
import matplotlib.pyplot as plt
import math
#import read_traffic_flow
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM



files_dir = "/content/merged_data.xlsx"
epoch_num = 2

data_train, data_test = \
    read_traffic_flow(2)
#converting data column to float for calculations purpose
data_train_ori = data_train.reshape(data_train.shape[0], -1).astype('float32')
data_test = data_test.reshape(data_test.shape[0], -1).astype('float32')


def create_dataset(dataset, look_back=1):
    data_x, data_y = [], []
    for i in range(len(dataset)-look_back-1):
        data_x.append(dataset[i:(i+look_back)])
        data_y.append(dataset[i + look_back])
    return np.array(data_x), np.array(data_y)


# fix random seed for reproducibility
np.random.seed(7)
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
data_train = scaler.fit_transform(data_train_ori)
data_test = scaler.fit_transform(data_test)

look_back = 1
train_x, train_y = create_dataset(data_train, look_back)
test_x, test_y = create_dataset(data_test, look_back)
```

```
train_x = train_x.reshape(train_x.shape[0], 1, train_x.shape[1])
test_x = test_x.reshape(test_x.shape[0], 1, test_x.shape[1])

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(16, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
history=model.fit(train_x, train_y, epochs=epoch_num, batch_size=1, verbose=2)

# make predictions
trainPredict = model.predict(train_x)
testPredict = model.predict(test_x)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
train_y = scaler.inverse_transform([train_y.ravel()])
testPredict = scaler.inverse_transform(testPredict)
test_y = scaler.inverse_transform([test_y.ravel()])
train_y = train_y.T
test_y = test_y.T

trainScore = math.sqrt(mean_absolute_error(train_y, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_absolute_error(test_y, testPredict))
print('Test Score: %.2f RMSE' % (testScore))

l = data_train.shape[0] + data_test.shape[0]
# # shift train predictions for plotting
trainPredictPlot = np.empty((l, 1))
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

 # shift test predictions for plotting
testPredictPlot = np.empty((l, 1))
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+look_back*3:l-1, :] = testPredict


test_y = test_y + 1

loss_test = np.sum(np.abs(testPredict-1 - test_y) / test_y) / test_y.shape[0]

# plot baseline and predictions
plt.plot(scaler.inverse_transform(
    np.vstack((data_train, data_test))), 'yellow')
plt.plot(trainPredictPlot, 'red')
plt.plot(testPredictPlot, 'blue')
plt.title("yellow: total; red: trainPred;" +
          " blue: testPred\n" +
          "After {} epochs".format(epoch_num))

print("Loss of data_test = {}".format(loss_test))
#loss_lst = np.abs(testPredict - test_y).ravel()
#with open('./loss.txt', 'w') as fout:
 #   fout.writelines('\n'.join(loss_lst.astype(str).tolist()))
node2=list()
for i in trainPredict:
  node2.append(int(i[0]))
for i in testPredict:
  node2.append(int(i[0]))

plt.show()
```

2)LRU(Least Recently Used)

```
#lru implementation
#for finding index and min in lists
import collections
#number of nodes
nodes=4
cache=list() #to store lements of cache
#making initial cache elements as 0
#making cache size fix to avg requests i.e., m calculated earlier
for i in range (0,m):
  cache.append(i%nodes+1)
index=list() #to store indexes so that least recently requested node can be kno
wn based on elements of the list
#similar to cache making index list with 'm' size and making all elements as 0'
s
for i in range(0,m):
  index.append(0)
#to store hit number at each timestamp(5minutes)
hit=list()
#to store fault number at each timestamp(5minutes)
fault=list()
#loop over each timestamp requests
for ind,row in frames.iterrows():
  #hit and fault in that particular timestamp
  hit1=0
  fault1=0
  #loop over each node for each timestamp
  for node in range(1,nodes+1):
    nodestr="Node"+str(node)
    #current required number of requests
    req=row[nodestr]
    #number of slots in cache already allocated to that particular node
    curr=cache.count(node)


    #three cases are possible
    #req>curr(1) , req=curr(2) , req<curr(3)

    #case:1,2

    if(req>=curr):
      hit1=hit1+curr
      fault1=fault1+req-curr
      #loop over current elements to update index list(hits)
      for i in range(0,m):
        if(cache[i]==node):
          index[i]=index[i]+1

      #loop over remaining requests to be alloted(faults)->req-curr
      while(req!=curr):
        #there is possibility for allocation


        #finding target location where the current node request is to be alloca
ted
        tar=index.index(min(index))
        #changing cache and index values at that target location
        cache[tar]=node
        index[tar]=index[tar]+1
        #current req is satisfied
        #looper(curr incrementation)
        curr=curr+1

    #case:3
    else:
      #deallocate node from cache for (curr-req) times
      hit1=hit1+req
      while(curr!=req):
        #to get all indices of node in cache
        cacheindices = [ind for ind, x in enumerate(cache) if x == node]
        #getting indices in index
        indexindices=list()
        for m in cacheindices:
          indexindices.append(index[m])
        cache[cacheindices[indexindices.index(min(indexindices))]]=0
        curr=curr-1
  minimi=min(index)
  index = [x - minimi for x in index]
  hit.append(hit1)
  fault.append(fault1)
```

3)LFU(Least Frequently Used)

```
#lfu implementation
#for finding index and min in lists
import collections
#number of nodes
nofnodes=4
# to store current allocation in cache
lfucache=list()
#to store node frequencies
lfufreq=list() #to store frequencies so that least frequently requested node ca
n be known based on elements of the list
#similar to cache making frequency list with 'nofnodes' size and making all ele
ments as 0's
for i in range(0,nofnodes):
  lfufreq.append(0)
for i in range(0,m):
  lfucache.append(i%nofnodes+1)
#to store hit number at each timestamp(5minutes)
lfuhit=list()
#to store fault number at each timestamp(5minutes)
lfufault=list()

#loop over each timestamp requests
for ind,row in frames.iterrows():
  #hit and fault in that particular timestamp
  hit1=0
  fault1=0
  #loop over each node for each timestamp
  for node in range(1,nofnodes+1):
    lfucounter=list()
    for i in range(0,nofnodes):
      lfucounter.append(lfucache.count(i+1))

    nodestr="Node"+str(node)
    #current required number of requests
    req=row[nodestr]
    #number of slots in cache already allocated to that particular node
    curr=lfucounter[node-1]
    lfufreq[node-1]+=req

    #three cases are possible
    #req>curr(1) , req=curr(2) , req<curr(3)

    #case:1

    if(req>curr):
      hit1=hit1+curr
      fault1=fault1+req-curr
      #loop over remaining requests to be alloted(faults)->req-curr
      while(req!=curr):

        #get other node frequencies into freq1
        freq1=list()
        #for remaining node data
        nn=list()
        for i in range(0,nofnodes):
          if(i+1!=node and lfucounter[i]!=0):
            freq1.append(lfufreq[i])
            nn.append(i)
        #finding target node whose request is to be given to current node reque
st
        tar=nn[freq1.index(min(freq1))]
        lfucounter[tar]-=1
        lfucounter[node-1]+=1
        tarloc=lfucache.index(tar+1)
        lfucache[tarloc]=node
      #  freq[node-1]=freq[node-1]+1
        #current req is satisfied
        #looper(curr incrementation)
        curr=curr+1


    #case:2,3
    else:
      hit1=hit1+req

  minimi=min(lfufreq)
  lfufreq = [x - minimi for x in lfufreq]
  lfuhit.append(hit1)
  lfufault.append(fault1)
```

REFERENCES

[1] Ministry of Information Industry, “China’s Telecommunications Industry in the First Half of 2018,” 2018; http:// www.199it.com/archives/751171.html.

[2] M. Chen et al., “Opportunistic Task Scheduling over Co-Located Clouds in Mobile Environment,” IEEE Trans. Service Computing, vol. 11, no. 3, 2018, pp. 549–61.

[3] J. MSV, “Is Fog Computing the Next Big Thing in the Internet of Things,” Forbes Mag., 18 Apr. 2016; https://www.forbes. com/sites/janakirammsv/2016/04/18/is-fogcomputing-thenext-big-thing-in-internet-of-things/2/#1971ac3a34c9.

[4] M. Peng et al., “Fog-Computing-Based Radio Access Networks: Issues and Challenges,” IEEE Network, vol. 30, no. 4, 2016, pp. 46–53.

[5] M. Chen et al., “Cognitive Information Measurements: A New Perspective,” Info. Science, vol. 505, Dec. 2019, pp. 487–97.

[6] P. G. Vinueza Naranjo, E. Baccarelli, and M. Scarpiniti, “Design and Energy-Efficient Resource Management of Virtualized Networked Fog Architectures for the Real-Time Support of IoT Applications,” J. Supercomputing, vol. 74, no. 6, June 2018, pp. 2470–2507.

[7] X. Zhang and M. Peng, “Testbed Design and Performance Emulation in Fog Radio Access Networks,” IEEE Network, vol. 33, no. 3, May/June, 2019 pp.49–57.

[8] Y. Ashkan et al., “On Reducing IoT Service Delay via Fog Offloading,” IEEE Internet of Things J., 2018.

[9] T. Dang and M. Peng,“Joint Radio Communication, Caching, and Computing Design for Mobile Virtual Reality Delivery in Fog Radio Access Networks,” IEEE JSAC, vol. 37, no. 7, 2019, pp.1594–1607. [10] M. Chen and Y. Hao, “Label-less Learning for Emotion Cognition,” IEEE Trans. Neural Networks and Learning Systems, 2019. DOI: 10.1109/TNNLS.2019.2929071.

[11] M. Chiang and T. Zhang, “Fog and IoT: An Overview of Research Opportunities,” IEEE Internet of Things J., vol. 3, no. 6, 2016, pp. 854–64.

[12] P. Verma and S. K. Sood, “Fog Assisted-IoT Enabled Patient Health Monitoring in Smart Homes,” IEEE Internet of Things J., vol. 5, no. 3, 2018, pp. 1789–96.

[13] M. Klum, “Innovation Potentials and Pathways Merging AI, CPS, and IoT,” Applied System Innovation, vol. 1, no.1, 2018, pp. 5–23.

[14] K. Greff et al., “LSTM: A Search Space Odyssey,” IEEE Trans. Neural Networks & Learning Systems, vol. 28, no. 10, 2015, pp. 2222–32.

[15] Y. Hao et al., “Smart-Edge-CoCaCo: AI-Enabled Smart Edge with Joint Computation, Caching, and Communication in Heterogeneous IoT,” IEEE Network, vol. 33, no. 2, Mar./Apr. 2019, pp. 58–64.