# Federated Learning Based Proactive Content Caching in Edge Computing

Zhengxin Yu [1], Jia Hu [1*], Geyong Min [1*], Haochuan Lu [2], Zhiwei Zhao [3], Haozhe Wang [1], Nektarios Georgalas [4]
[1]Department of Computer Science, University of Exeter, UK
[2]School of Computer Science, Fudan University, China
[3]School of Computer Science and Engineering, University of Electronic Science and Technology of China
[4]Research and Innovation Department, British Telecom, UK
Email: {zy246, j.hu, g.min, h.wang3} @exeter.ac.uk, luhc17@fudan.edu.cn, zzw@uestc.edu.cn, nektarios.georgalas@bt.com

***Abstract*—Content caching is a promising approach in edge computing to cope with the explosive growth of mobile data on 5G networks, where contents are typically placed on local caches for fast and repetitive data access. Due to the capacity limit of caches, it is essential to predict the popularity of files and cache those popular ones. However, the fluctuated popularity of files makes the prediction a highly challenging task. To tackle this challenge, many recent works propose learning based approaches which gather the users' data centrally for training, but they bring a significant issue: users may not trust the central server and thus hesitate to upload their private data. In order to address this issue, we propose a Federated learning based Proactive Content Caching (FPCC) scheme, which does not require to gather users' data centrally for training. The FPCC is based on a hierarchical architecture in which the server aggregates the users' updates using federated averaging, and each user performs training on its local data using hybrid filtering on stacked autoencoders. The experimental results demonstrate that, without gathering user's private data, our scheme still outperforms other learning-based caching algorithms such as m-$\epsilon$-greedy and Thompson sampling in terms of cache efficiency.**

## I. Introduction

According to the Cisco Visual Networking Index, mobile data traffic will increase sevenfold between 2016 and 2021, reaching 48.3 EB per month by 2021 [1]. Especially, video traffic is expected to become the dominant data traffic due to the rapid development of smart devices [2] and the growing success of video streaming services. The steep rise of mobile traffic causes the increase of user latency and places a heavy burden on backhaul links which connect local base stations and the Internet. Content caching has been considered a promising approach in edge computing to improve performance, alleviate backhaul link congestion [3] and reduce user delay by storing popular files at local base stations that may be frequently requested by users.

Due to the storage limitation of cache entity, it is important to estimate the future popularity of contents and proactively cache the most popular files. However, traditional caching algorithms such as First-In-First-Out (FIFO), Least Recently Used (LRU) and Least Frequently Used (LFU) do not consider the popularity of contents in the future [4], which leads to low cache efficiency. Many recent caching schemes have been devoted to learning content popularity trends, which is the primary challenge of proactive caching. For instance, [2] uses a multi-armed bandit (MAB) method to study the content popularity distribution for content caching. [5] proposes a collaborative filtering based caching algorithm for small cell networks. However, the existing proactive caching schemes are designed for highly controlled environments, where users need to upload their local data to the central server that may bring privacy and security risks. Moreover, scalability is an issue for those designs as the number of users and the amount of users' data grow.

In order to tackle the above challenges, we propose a Federated learning based Proactive Content Caching (FPCC) scheme with a hierarchical architecture where the bottom layer includes the users requesting for contents, and the top layer contains the central server with a cache entity. Each user downloads the stacked autoencoder model from the server, trains the model using its local data, uploads the updates of model parameters to the server at each communication round and finally recommends $N$ files (i.e., contents) to the server. The recommendation is calculated by hybrid filtering, which uses the users' and the files' similarity based on their latent features extracted using the stacked autoencoder. The server aggregates the uploaded model parameters from each user by the federated averaging algorithm [6] and selects the most popular files from the ones recommended by all the users. The FPCC could inherently reduce the security and privacy risk, since the training data are kept locally and only the updates of model parameters are sent to the central server [6].

The major contributions of this paper are as follows:

1) To the best of our knowledge, the FPCC is the first learning based proactive content caching scheme that can keep the training data locally at each user to reduce the privacy risk.
2) The FPCC is based on a hierarchical architecture where the server aggregates user-side updates to construct a global model and selects the most popular files. Each user performs training on its local data using hybrid filtering based on a stacked autoencoder.
3) Experimental results based on real-world datasets verify that the FPCC outperforms other reference algorithms (Random, m-$\epsilon$-Greedy, and Thompson Sampling) in terms of cache efficiency.

The rest of the paper is structured as follows: the related work is described in Section II. Section III presents the system model. In Section IV, we describe the proposed FPCC scheme in details. The performance evaluation of FPCC is provided in Section V. Finally, Section VI concludes this paper.

## II. Related Work

Due to the limited cache storage, it is essential to place the contents that are most likely to be requested by users in the local cache. Traditional caching schemes [7] update cache contents based on static rules such as FIFO, LRU and LFU. However, they are not adapted to the dynamically changing content popularity. Recent research has put in effort to develop dynamic cache schemes based on the popularity of contents. They can be generally classified into two categories: the cache algorithms with or without the prior knowledge of content popularity distribution. We start by briefly introducing related work which assumes the content popularity with prior knowledge. In some cases, the contents request from users are modeled by a Zipf distribution. With knowing the demand of users, Maddah-Ali et al. [8] exploits the broadcast nature of the wireless medium by coded caching to improve cache efficiency. The aim of improving downlink energy efficiency of proactive content caching has been derived in [9] which supposes the user requests can be predicted.

Additionally, some content caching schemes without prior knowledge of content popularity have been developed. Machine learning techniques could be used in caching algorithms to estimate the popularity of files, such as reinforcement learning and collaborative filtering. Bastug et al. [5] proposes a caching algorithm for small cell networks based on collaborative filtering (CF). It provides the estimation of content's popularity after training phase by using sparse training data, whereas multi-armed bandit (MAB) as another caching algorithm learns popularity of files online by firstly observing demands of cached content and then updating the content of cache at a fixed time [2]. Sengupta et al. [10] proposes a coded caching scheme, where the base station is based on demand history to estimation the popularity of files via a combinatorial multi-armed bandit formulation. It combines the popularity estimation and content placement scheme. Besides, because of different users contributes content popularity, a contextual MAB algorithm [2] is used to learn the content's popularity with considering different users' information. It is an extended work of [11] which aggregated context information, such as user density and request file time. However, above methods all designed for the central environment where server gather all data, which may raise user privacy fears. Users do not trust the server and hesitate to upload their private data.

Therefore, we propose a novel proactive content caching method which firstly joint collaborative stacked auto-encoder and federated learning to predict the popularity of the content while protecting user privacy.

## III. System Model

The system model of FPCC is depicted in Fig. 1. We consider the base station (BS) as a wireless caching entity

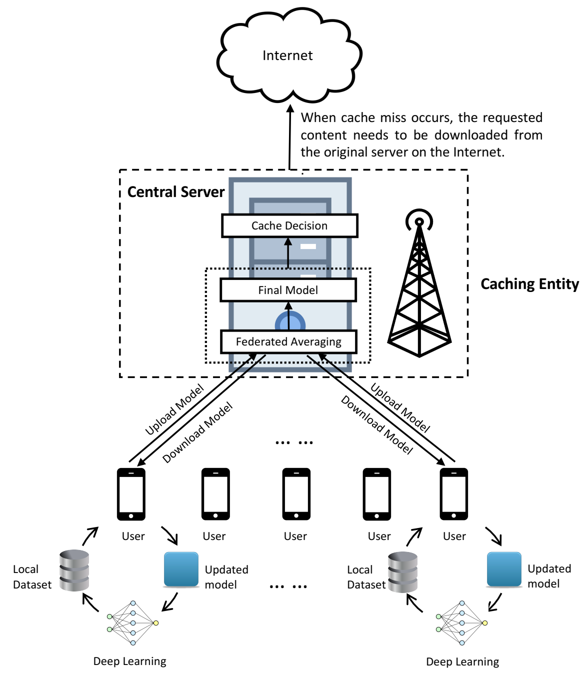

Fig. 1. System model

with a reliable backhaul link to the Internet. The caching entity has limited storage and a learning model to decide what and how to cache on the storage. We assume the cache entity can store up to $m$ same size files. Mobile users send content requests to the BS when they are within its coverage area. If the requested content is cached in the BS (i.e., the cache hit), the content file would be directly delivered from the BS; Otherwise (i.e., the cache miss), the content needs to be downloaded from the original server on the Internet. Hence, the primary goal of the cache enabled BS is to improve the cache efficiency and reduce the users' service response time. In order to achieve this goal, the FPCC is designed based on the federated learning framework where the users independently compute an update to the current global model by using its local data and communicate these updates to the central server to aggregate a new global model.

In our system model, each user equips with a mobile device. The user connects the BS when he/she locates in the coverage area. Each connected user is based on its local data to compute an update of the global model which is downloaded from the server. Then, each user sends the update back to the server. Typically, the local training dataset are generated from the usage of user's device, such as video demands in the daily life. Different places, different times of a day, different current activities and even different types of portable devices [2] may lead users to request different contents. Therefore, the past user requests under different situations form a part of the local training dataset. The local training dataset also includes user's contextual information. However, each connected user may have an individual context space with different context dimension which depends on the personal setting of sensors

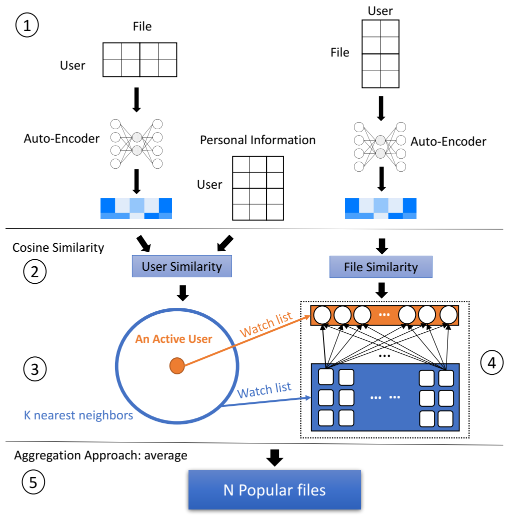

Fig. 2. Hybrid Filtering Model

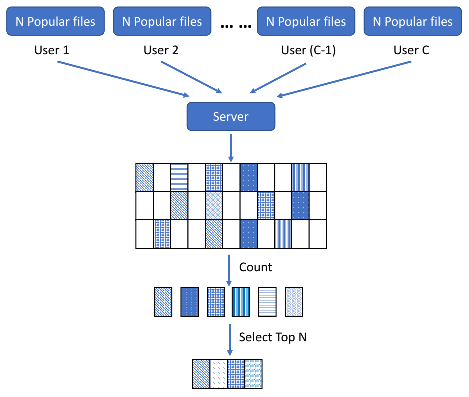

Fig. 3. The selection of the most popular files on Server

on the mobile device.

A shared global model is maintained in the central server, which will be sent to each connected users. Users are responsible for learning the updates of the global model and recommending popular files by their local data. Next, the server aggregates all the user-side updates to construct an improved global model by using federated averaging. However, some users may make the heavy use of some particular services or apps, leading to varying amounts of user's local data. To address this problem, our proposed method reflects the weight proportional to data size when the server aggregates the global model. It reduces complexity and improves the cache efficiency. After that, the improved model will be sent to user-side again. We refer to the above steps as a communication round.

In each communication round, the costs of sending and downloading updates are considered as communication costs, which dominate [12] in federated optimisation, while computing costs are relatively small as modern mobile device have fast processors. Therefore, the number of communication rounds could be reduced with more participating users and/or more computation at each user. Also, the communication rounds are carried out until the results tend to be stable. Finally, according to the recommendation list from each user, the $m$ most popular files are selected in the server for caching. The model in each user's equipment and central server is collaborative stacked autoencoder based on hybrid filtering which will be described in the Section 4. It predicts the popularity of files more precisely.

## IV. The FPCC Scheme

The proposed proactive content caching algorithm consists of three procedures: encoding, hybrid filtering and federated learning. For content caching, a critical consensus is that users' requests together with their contextual information can be exploited to learn caching decisions in the future. Therefore, given the users' contextual information, a caching entity should learn context-specific content popularity in order to cache the most popular files for these users proactively. The core problem is to find the latent features from a complex dataset, which is obtained from the users' contextual information and make use of these essential correlations to work out the similarities of different sets of contents for the user and that between files. This problem is naturally suitable to be solved by using neural network model, and thus we will adapt and extend a neural network model called stacked autoencoder based on hybrid filtering to resolve the problem.

### A. *Stacked Autoencoder*

The stacked autoencoder is an unsupervised learning model, which trains an one-hidden layer neural network to reconstruct input data from the latent representation [13]. In recent years, neural network models [14] show a great potentials in learning hidden representations. Given a set of data instances $\left\{x^{(1)}, x^{(2)}, x^{(i)}, \ldots, x^{(m)}\right\}$ where $i$ = 1,2,...,m, $x^{(i)} \in \mathbb{R}^d$ as input, an encoder is used to map the input to a hidden representation $y^{(i)} \in \mathbb{R}^d$ through an activation function $h(x)$. Then the latent representation is mapped back with a decoder to reconstruct $x$. Essentially, it is to learn a function $h_{W,b}(x) \approx \hat{x}$, where $\mathbf{W}$, $\mathbf{b}$ are weight matrices and the bias vectors respectively [15].

After the training of the autoencoder neural network, the hidden features of the training data can be obtained which are exploited for hybrid filtering and federated learning afterwards.

### B. *Hybrid Filtering*

The users' similarity and files' similarity are calculated by the features of users and files that are extracted from stacked autoencoders. The recommendation list of popular files for caching is generated based on the files' similarity of the active user's watch history and their $K$ neighbour users' watch history. We suppose an active user represents the user's requests in particular scene. $K$ neighbour users' watch history stand for similar scenes for this specific scene. Hybrid filtering combines content-based, demographic and collaborative filtering. It is mainly based on similarity measures to obtain the distance between two files or two users according to their files' ratings and personal profile.

Fig. 2 describes the whole process of estimating the most popular files for one user corresponding to one specific scene. The following five steps are executed and the index number in the Fig. 2 is the same with the following number of step:

1) **Data pre-processing:** Based on the request history from each user, a rating matrix is created. The personal detail of users such as location and the time of day, generate a user information matrix.
2) **Exploration of latent representation:** The rating matrix is the input data for autoencoder. It is used to discover the hidden features and the correlations between users and that between files. These features is combined with the user information matrix to calculate the similarity matrix of users and the similarity matrix of files. We employs the cosine similarity, which is highly effective for sparse matrices. The elements in the user's similarity matrix and the file's similarity matrix represent the distance between each user, and each file, respectively.
3) **Construction of the historical matrix:** We assume the current user is the active user. Based on the similarity matrix of the user, **K** nearest neighbour users of the active user could be determined, and then a matrix of historical watch list ($\mathbf{K}^*$) from these **K** selected neighbour users can also be constructed.
4) **Obtaining the similarity:** The matrix of historical requests of the active user is referred as $\mathbf{A}^*$. The average value of the similarity between each element in $\mathbf{A}^*$ and $\mathbf{K}^*$ is obtained through the similarity matrix of files.
5) **Aggregation:** An aggregation approach to predict the popularity of files is executed by the selecting highest to *n-th* highest similarity files to produce the recommendation list of popular files for caching. Each user uploads their recommendation list to the server. As shown in Fig. 3, the server then aggregates all the calculated results from users and selects the top N the most popular files as the cache content for the cache entity.

## C. Federated Learning

The parameters in the stacked autoencoder need to be uploaded to the central server from the user side, because the server aggregates all the results. A principal advantage of this approach is that the model is trained using the local data at users without uploading the data to the central server. It can significantly reduce security and privacy risks. To achieve this, users firstly download the global model $\mathbf{W}$ from the central server. Next, users compute the updated models $\mathbf{W}_t^1, \mathbf{W}_t^2, \mathbf{W}_t^3, ..., \mathbf{W}_t^c$ based on their local data. $t$ represents the round number. $1, 2, 3, ..., c$ means the index number of the participant user. The updates are written as $\mathbf{H}_t^c := \mathbf{W}_t^c - \mathbf{W}_t$. Lastly, the updates and a recommendation list of popular files that are estimated by hybrid filtering send to the central server. The detailed process is shown in Algorithm 1.

On the server-side, the central server aggregates all the user-side updated models to improve its global model [6] using *Federated Averaging*, see Equation (1). $\eta_t$ is the learning rate. *Federated Averaging* utilises the weighted average sum to aggregate all updates as it considers the quantity of each selected user's training dataset. Finally, the server generates a recommendation list of popular files for caching. The pseudo-code of the algorithm running in the server is provided in Algorithm 2.

**Algorithm 1** Content caching algorithm: User

```
User Updates($c, w, b$):
    $B$: Split training data into batchsize of B
    $E$: The number of local epochs
    $\eta$: The learning rate
 1: for each local epoch i from 1 to $E$ do:
 2:     for batch $b \in B$ do:
 3:         $w \leftarrow w - \eta \bigtriangledown \iota(w; b)$
 4:     end for
 5: end for
 6: UserFeatures = AutoencoderUser(user-file matrix)
 7: UserSim = similarity(UserFeatures)
 8: FileFeatures = AutoencoderFile(file-user matrix)
 9: FileSim = similarity(FileFeatures)
10: AutoencoderUser($x$)
11:     autoencoder($x$,$x$,$E$,$B$)
12: AutoencoderFile($y$)
13:     autoencoder($y$,$y$,$E$,$B$)
14: similarity($x$)
15:     $\mathrm{sim}(It^i, It^j) = \frac{It^i It^j}{|It^i||It^j|}$
16: Select $K_1, K_2, K_3, ..., K_n$ from userSim $\rightarrow K^*$
17: Average: $\frac{1}{n}\sum_{n=1}^{n} n^*$ for the similarity between requested
    content of target user $K^*$ and $A^* \rightarrow N^*$
18: Select $N_1, N_2, N_3, ..., N_n$ from $N^*$
19: return $c, w, b, N$
```

**Algorithm 2** Content caching algorithm: Server

```
Server Executes:
    $C$: users are indexed by $c$
    $t$: the number of communication round
    $w, b$: the parameter of model
    $N_c$: predict popular files from each user
 1: Initialize $w, b$
 2: for each round t = 1,2,... $t$ do:
 3:     for each user $c \in S_t$ in parallel do:
 4:         $w_{t+1}^c, b_{t+1}^c, N_{t+1}^c \leftarrow$ User Updates($c$, $w_c$, $b_c$)
 5:         $w_{t+1} \leftarrow \sum_{c=1}^{C} \frac{1}{C} w_{t+1}^c$
 6:         $b_{t+1} \leftarrow \sum_{c=1}^{C} \frac{1}{C} b_{t+1}^c$
 7:     for end
 8:     Count $A_c$
 9:     Select top-N $N_1, N_2, N_3, ..., N_n$
10: for end
```

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta_t \mathbf{H}_t, \quad \mathbf{H}_t := \frac{1}{n_t} \sum_{i \in S_t} \mathbf{H}_t^i \tag{1}$$

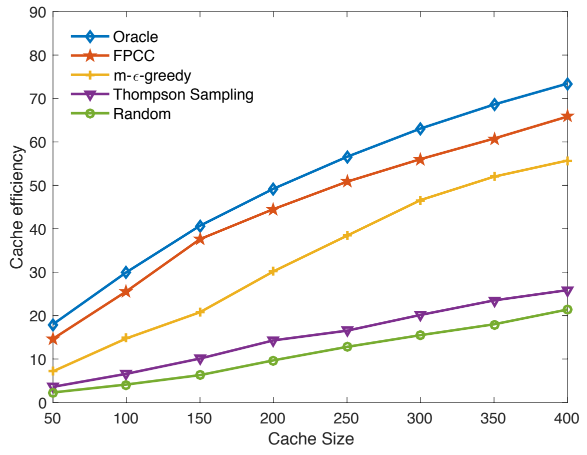

Fig. 4. Cache efficiency with different cache sizes (MovieLens 100K)

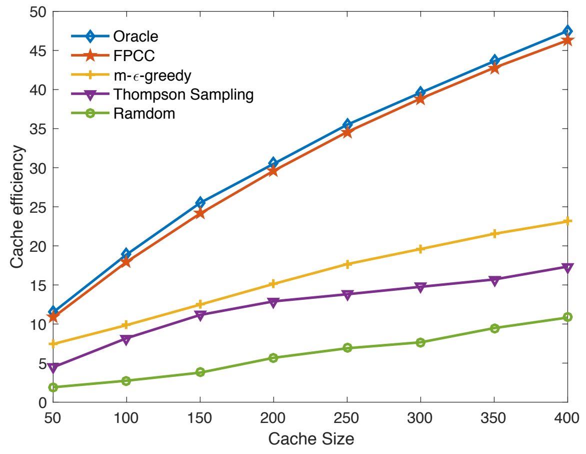

Fig. 5. Cache efficiency with different cache sizes (MovieLens 1M)

## V. Experimental Results

In this section, we present the results of the experiments conducted to evaluate the proposed content caching algorithm by comparing its performance to four reference algorithms.

### A. Datasets

We use real-world datasets – MovieLens [16] in our experiments. MovieLens 1M dataset contains 1,000,209 ratings on 3883 movies made by 6040 users, while there are 100,000 ratings from 943 users on 1682 movies in the MovieLens 100K dataset. For both datasets, the rating scale is from 0 to 5. Each user at least rates 20 movies. They also provide the demographic information of users, such as gender, age, occupation and zip-code. To simulate the content requests, we assume that the rated movies are the files requested by users. Each movie rating corresponds to a downloading or streaming request. [2] and [17] use a similar approach to simulate the process of user requests.

### B. Reference Algorithms

We compare our algorithm with four reference algorithms, which are described below:

- **Oracle**: Oracle algorithm has the perfect prior knowledge about the future demands. It provides the best possible cache efficiency.
- **Random**: Random algorithm selects $N$ files randomly to cache. It gives the lowest cache efficiency.
- **m-$\epsilon$-Greedy**: m-$\epsilon$-Greedy algorithm is one of the multi-armed bandit algorithms, which is an extension of simple $\epsilon$-Greedy algorithm. The m-$\epsilon$-Greedy picks $m$ files of the most rewards with the probability of (1 - $\epsilon$ ), but with the probability $\epsilon$ ($0 < \epsilon < 1$), to select $m$ files randomly from all the files. In our experiments, we set $\epsilon = 0.1$ based on empirical results.
- **Thompson Sampling**: Thompson Sampling is an algorithm widely used in the multi-armed bandit problem. It assumes that a value for each file is sampled from the beta distribution with two parameters: wins and losses. The file with the highest value is selected. For every trial, the beta distribution is modified based on cache hit or cache miss.

### C. Performance Evaluation

We use *cache efficiency* [2] as the performance metric to evaluate our algorithm, which is the ratio of cache hits to the number of user requests on the cache. Our experiments include two autoencoders to find latent features between users and between files to calculate similarity matrices. Both autoencoders have one hidden layer using ReLu activation. We investigate the cache efficiency for varying cache sizes between 50 and 400 files. Oracle algorithm provides an upper bound of cache efficiency, while the random algorithm gives the worst cache efficiency among the reference algorithms. As shown in Figs.4 and 5, the cache efficiency of all the algorithms rises with the increasing cache size. For both datasets, the FPCC and the m-$\epsilon$-Greedy algorithm show the better performance compared to the Thompson sampling and the random algorithm. It is due to the fact that both FPCC and m-$\epsilon$-Greedy algorithm learn from the past requests of users, while the Thompson sampling or the Random algorithm does not observe the past requests at all. Moreover, it is shown that the FPCC outperforms the m-$\epsilon$-Greedy, as the m-$\epsilon$-Greedy does not consider the context information of users. The cache efficiency of the FPCC is closer to the Oracle algorithm (optimum) with 1M dataset than with 100K dataset, because that the FPCC algorithm trains better with more data.

Fig. 6 depicts the cache efficiency against the number of federated communication rounds with different numbers of participated users. On both 100K and 1M datasets, we observe that more communication rounds are needed to achieve the sub-optimal cache efficiency (15% for 100K and 10% for 1M) with fewer users. Fig. 6 shows that when the cache size is 50, the cache efficiency reaches 15% after 20 rounds, 13 rounds and 8 rounds with 20%, 60%, and 100% participated users, respectively. For the MovieLens 1M dataset, the experiment results exhibit the same trend as MovieLens 100k. The cache efficiency reaches 10% after 20 rounds, 8 rounds and 6 rounds with 20%, 60%, and 100% participated users, respectively. The

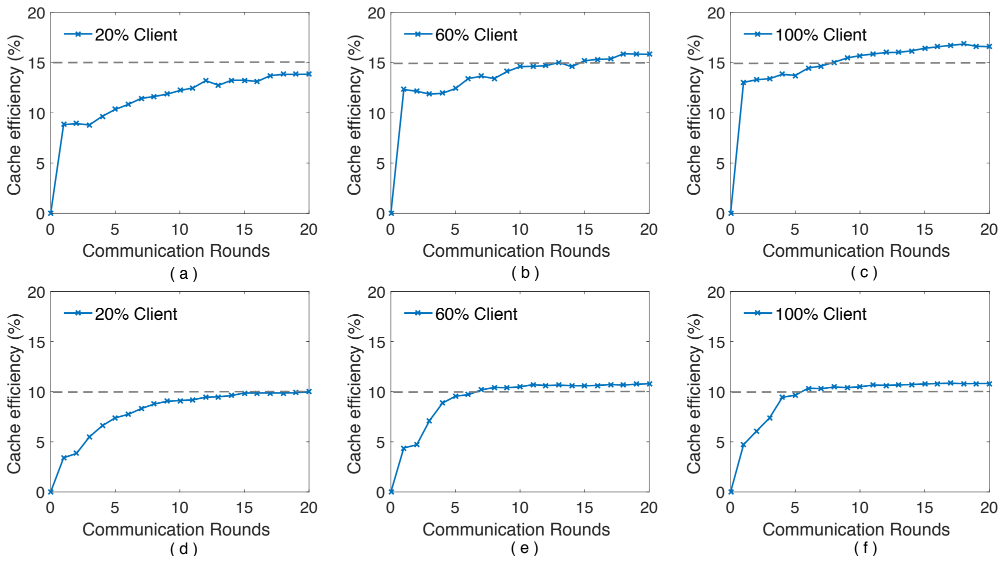

Fig. 6. Cache efficiency vs. Communication rounds with different sizes of datasets: 100k(a,b,c);1M(d,e,f)

results indicate that if the size of training data is larger for each participated user, fewer users need to achieve the same cache efficiency. In addition, fewer communication rounds are needed with more users or larger datasets.

## VI. Conclusions

In this paper, we propose a Federated learning based Proactive Content Caching algorithm (FPCC), which could achieve high cache efficiency as well as protect the privacy of users. The FPCC is based on a hierarchical architecture where each user calculates the model updates using the local data and the server aggregates user-side updates to construct a global model. Our experiments demonstrate that the FPCC outperforms other reference algorithms (Random, m-$\epsilon$-Greedy, and Thompson Sampling) on a real-world dataset (MovieLens) in terms of cache efficiency. Moreover, the results show that high-quality models can be trained in fewer rounds with more users or more data at users.

## Acknowledgements

This work was supported by UK EPSRC fund EP/M013936/2, the National Key Research and Development Program of China (2017YFB1400102), the National Natural Science Foundation of China (No. 61602095), and the EU FP7 CLIMBER project under Grant Agreement No. PIRSES-GA-2013-612652.

## References

[1] “Cisco visual networking index: forecast and methodology, 2016-2021, white paper,” *Cisco Visual,San Jose, CA, USA*, 2017.
[2] S. Müller, O. Atan, M. van der Schaar, and A. Klein, “Context-aware proactive content caching with service differentiation in wireless networks,” *IEEE Transactions on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, 2017.
[3] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, “Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution,” *IEEE Communications Magazine*, vol. 51, no. 4, pp. 142–149, 2013.
[4] Y. Shi and Q. Ling, “An adaptive popularity tracking algorithm for dynamic content caching for radio access networks,” in *Control Conference, 36th ChineseëijŇCCC*. IEEE, 2017, pp. 5690–5694.
[5] E. Bastug, M. Bennis, and M. Debbah, “Living on the edge: The role of proactive caching in 5g wireless networks,” *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.
[6] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.
[7] R. Fares, B. Romoser, Z. Zong, M. Nijim, and X. Qin, “Performance evaluation of traditional caching policies on a large system with petabytes of data,” in *IEEE 7th International Conference on Networking, Architecture and Storage, NAS*. IEEE, 2012, pp. 227–234.
[8] M. A. Maddah-Ali and U. Niesen, “Fundamental limits of caching,” *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.
[9] A. C. Güngör and D. Gündüz, “Proactive wireless caching at mobile user devices for energy efficiency,” in *International Symposium on Wireless Communication Systems, ISWCS*. IEEE, 2015, pp. 186–190.
[10] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, “Learning distributed caching strategies in small cell networks,” in *IEEE International Symposium on Wireless Communications Systems, ISWCS*. IEEE, 2014, pp. 917–921.
[11] S. Müller, O. Atan, M. van der Schaar, and A. Klein, “Smart caching in wireless small cell networks via contextual multi-armed bandits,” in *IEEE International Conference on Communications, ICC*. IEEE, 2016, pp. 1–7.
[12] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, “Communication-efficient learning of deep networks from decentralized data,” *arXiv preprint arXiv:1602.05629*, 2016.
[13] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, “Collaborative denoising auto-encoders for top-n recommender systems,” in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 2016, pp. 153–162.
[14] X. Yu, X. Wu, C. Luo, and P. Ren, “Deep learning in remote sensing scene classification: a data augmentation enhanced convolutional neural network framework,” *GIScience & Remote Sensing*, vol. 54, no. 5, pp. 741–758, 2017.
[15] A. Ng, “Sparse autoencoder,” *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.
[16] F. Harper and J. Konstan, “The movielens datasets: History and context,” *ACM Transactions on Interactive Intelligent Systems*, vol. 5, no. 4, 12 2015.
[17] S. Li, J. Xu, M. Van Der Schaar, and W. Li, “Popularity-driven content caching,” in *IEEE International Conference on Computer Communications, INFOCOM*. IEEE, 2016, pp. 1–9.