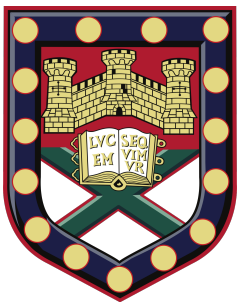
Intelligent Edge Caching based on Federated Deep Learning



Zhengxin Yu

College of Engineering, Mathematics and Physical Sciences

University of Exeter

Submitted by Zhengxin Yu to the University of Exeter

as a thesis for the degree of

Doctor of Philosophy in Computer Science

January 2021

Declaration

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

Zhengxin Yu

January 2021

Abstract

Caching contents at the edge of network is considered to be a cost-effective solution to cope with ongoing traffic growth and address the backhaul bottleneck problem in wireless networks. However, the inherent characteristics of wireless networks, including the high mobility of users and restricted storage capability of edge nodes, cause many difficulties in the design of caching schemes. Driven by the recent advancements in Machine Learning (ML), learning-based proactive caching schemes are able to accurately predict content popularity and improve cache efficiency, but they need gather and analyse users' content retrieval history and personal data, leading to privacy concerns. To address these challenges, this research mainly focuses on the design of learning-based caching schemes to improve caching efficiency and protect user privacy in various modern networks, such as Fifth Generation Mobile Networks (5G), Internet-of-Vehicles (IoV), and Fog Radio Access Networks (F-RANs).

In modern networks, mobile phones, wearable devices, and autonomous vehicles provide growing computational power and storage capability. Coupled with the increasing concern about data privacy protection, the emerging framework of federated learning has been recognised as a promising framework to efficiently build ML models while protecting user privacy by keeping data at local devices and fitting ML techniques into the network edges. In 5G, a communication Efficient Federated learning based Proactive content Caching scheme (EFPC) is proposed to mitigate the privacy risks and reduce communication consumption. Based upon the federated learning framework, each user locally trains a shared model for

content popularity prediction by using their own data, and only uploads the parameters of the model to the edge server for aggregation. To further reduce communication costs, the 3LC data compression scheme is used in EFPC to compress the upload parameters of the model. In F-RANs, a Federated Learning based Cooperative Hierarchical Caching scheme (FLCH) is designed to maximise the utilisation of available caches with edge node. FLCH exploits horizontal cooperation between neighbour F-APs and vertical cooperation between the baseband unit pool and fog access points to cache contents with different degrees of popularity.

In IoV, a Mobility-aware Proactive edge Caching scheme based on Federated learning (MPCF) is developed to support mobility of vehicles. This new scheme enables multiple vehicles to collaboratively learn a global model for predicting content popularity with the private training data distributed on local vehicles. MPCF also integrates a mobility-aware cache replacement policy, which allows the network edges to add/evict contents in response to the real-time mobility patterns and dynamic preferences of vehicles. To ease reliance on the fixed central server, eliminate the issue of hand-over between RSUs, a peer-to-peer federated deep learning based proactive caching scheme (PPFC) is proposed. A vehicle rather than a fixed edge node, acts as a central server to aggregate ML models from nearby vehicles. A dual-weighted model aggregation scheme is designed to reduce the effect of straggler vehicles and further improve the global model accuracy.

The proposed caching schemes in this thesis can greatly improve cache performance, effectively protect users' privacy and significantly reduce communication costs. The simulation experiments are conducted to evaluate the performance of these caching schemes and the accuracy of the designed prediction models using real-world datasets.

Acknowledgements

First of all, I would like to express my sincerest gratitude to my supervisor Prof. Geyong Min for all the support, patient guidance and encouragement he gave me, during my PhD study and Master study. I have learned a lot from his wisdom, way of thinking, enthusiasm for work and hard working attitude. Without his guidance and constant feedback, this PhD would not have been achievable. Apart from his academic support, Prof. Min gives me a lot of help in life, like a close family member. I really appreciate everything he has done for me and hope to be a person like him, to be kind, responsible and optimistic.

Many thanks also to my second supervisor Dr. Jia Hu for all his help to my research. His immense knowledge and insightful comments incentivise me to widen my research from various perspectives. His guidance helped me in all the time of my research and writing of this thesis. I also thank to Prof. Zhiwei Zhao for his helpful discussions and comments on my works.

I would like to thank my lab mates, Dr. Wang Miao, Dr. Haozhe Wang, Dr. Chengqiang Huang, Dr. Yuan Zuo, Jin Wang, Zheyi Chen and Yang Mi for their help to my PhD study. I also thank my friends Dr. Yang Yang, Marina Hunter, Chris Hunter, Carina Ivascu, Wanyi Cui and Luwen Zhang for always standing by my side and encouraging me.

Finally, special thanks to my parents for their love and support throughout my life. With the company of my parents, who helped me through all the ups and downs of my research. My mother and father supported me unconditionally and believed in me. I have become a better person and keep moving forward. Mommy and daddy, I love you with all my heart.

Table of contents

List of Abbreviations **x**

List of figures **xii**

List of tables **xiv**

List of Publications **xv**

1 Introduction **1**

1.1 Motivations and Challenges 3

1.2 Research Aims and Objectives 4

1.3 Contributions 5

1.4 Outline of the Thesis 7

2 Background and Literature Review **9**

2.1 Edge Caching 9

2.1.1 Multi-access Edge Computing 9

2.1.2 Fundamentals of Edge Caching 10

2.1.3 Cache Placement 11

2.1.4 Caching Policy 11

2.1.5 Cache Replacement 13

2.2 Artificial Intelligence 14

2.2.1 Machine Learning 15
2.2.2 Deep Learning 16
2.3 Federated Learning 18
2.3.1 The Framework of Federated Learning 18
2.3.2 Unique Characteristics of Federated Learning 20
2.3.3 Federated Learning and its applications in Wireless Networks . . . 20
2.4 Learning-based Edge Caching 23
2.4.1 Edge Caching with Prior Knowledge of Content Popularity 23
2.4.2 Edge Caching without Prior Knowledge of Content Popularity . . . 24
2.5 Summary 28

3 Communication-Efficient Federated Learning based Proactive Caching 30
3.1 Introduction 30
3.2 System Architecture of EFPC 32
3.3 Communication-Efficient Federated Learning for Edge Caching 34
3.3.1 Communication-Efficient Federated Deep Learning 34
3.3.2 One-Class Collaborative Variational Autoencoder 39
3.4 Experiments and Discussion 46
3.4.1 Testbed 46
3.4.2 Performance Evaluation 47
3.5 Summary 53

4 Mobility-Aware Proactive Edge Caching for Connected Vehicles 54
4.1 Introduction 54
4.2 System Architecture 57
4.3 Mobility-aware Federated Learning for Edge Caching 60
4.3.1 Mobility-aware Federated Deep Learning 61

4.3.2 Contextual-aware Adversarial Autoencoders for Prediction 67
4.3.3 Mobility-aware Cache Replacement Policy 71
4.4 Performance Results and Analysis 75
4.4.1 Simulation Settings and Dataset 75
4.4.2 Performance Evaluation 75
4.5 Summary 81

5 Peer-to-Peer Federated Learning based Edge Caching for Internet-of-Vehicles 82
5.1 Introduction 82
5.2 System Architecture 84
5.3 Peer-to-Peer Federated Learning for Edge Caching 87
5.4 Performance Evaluation 92
5.4.1 Experiment Settings 92
5.4.2 Experimental Results 94
5.5 Summary 98

6 Cooperative Hierarchical Caching in Fog Networks 99
6.1 Introduction 99
6.2 System Architecture 102
6.3 Cooperative Hierarchical Edge Caching Scheme 105
6.3.1 Federated Learning for Cooperative Hierarchical Edge Caching . . 106
6.3.2 Stacked Autoencoder with One-Class Collaborative Filtering 107
6.4 Experimental Results 109
6.5 Summary 112

7 Conclusions and Future Work 113
7.1 Conclusions 113
7.2 Future Work 116

7.2.1 Hybrid Caching Scheme 116
7.2.2 Cooperative and Hierarchical Federated Learning 117
7.2.3 Asynchronous Federated Learning 119

References **121**

List of Abbreviations

5G	Fifth generation of mobile networks
IoT	Internet-of-Things
IoV	Internet-of-Vehicles
F-RANs	Fog Radio Access Networks
MEC	Multi-access Edge Computing
BS	Base Station
SBS	Small Base Station
MBS	Macro Base Station
RSU	Roadside Unit
RRH	Radio Heads
BBU	Baseband unit
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
FL	Federated Learning
FIFO	First-In-First-Out
LRU	Least Recently Used
LFU	Least Frequently Used
MRU	Most Recently Used

UE User Equipment

List of figures

3.1 Architecture of the EFPC Caching Scheme 32
3.2 Variational Autoencoder 41
3.3 EFPC Experiment Scenario 45
3.4 Cache hit ratio vs. Different number of users (MovieLens) 46
3.5 Cache hit ratio vs. Different number of users (Netflix) 46
3.6 EFPC vs. Other reference schemes (MovieLens Dataset) 47
3.7 EFPC vs. Other reference schemes (Netflix Dataset) 48
3.8 Data Compression vs. Without Data Compression 51
3.9 Sparsity vs. Data compression ratio vs. Cache hit ratio 52

4.1 Proactive edge caching for connected vehicles 57
4.2 Mobility-aware Federated Deep Learning 62
4.3 C-AAE model architecture 68
4.4 Mobility-aware cache replacement policy: Round 1 69
4.5 Mobility-aware cache replacement policy: Round 2 70
4.6 Mobility-aware cache replacement policy: Round 3 70
4.7 Mobility-aware cache replacement policy: Summary 71
4.8 Cache hit ratio with different cache sizes 76
4.9 Cache hit ratio vs Vehicle density 77
4.10 Cache hit ratio and Training time against Communication rounds 78

4.11 FL training process (27 vehicles) 79
4.12 FL training process (5 vehicles) 80
4.13 Mobility-aware cache replacement 80

5.1 System architecture of PPFC 85
5.2 Peer-to-peer federated learning 88
5.3 Collaborative filtering based variational autoencoder 90
5.4 PPFC vs. Other reference schemes (10 vehicles) 94
5.5 PPFC vs. Other reference schemes (5 vehicles) 95
5.6 Vehicle density vs. Training time vs. Cache hit ratio 96
5.7 PPFC vs. RSU caching (10 vehicles) 96
5.8 PPFC vs. RSU caching (5 vehicles) 97

6.1 System Architecture of the Federated Learning based Cooperative Hierarchical Caching for F-RANs. 102
6.2 Federated Learning Process for Edge Caching 103
6.3 Appended Stacked AE with One-Class Collaborative Filtering 107
6.4 One-Class Collaborative Filtering 108
6.5 Cache hit ratio: N-FLCH vs Reference caching schemes 111
6.6 Cache hit ratio of the FLCH with different cooperative strategies (H-FLCH, V-FLCH, N-FLCH) 111

7.1 Hybrid Caching Scheme 117
7.2 Hierarchical Federated Learning 118

7.3 Asynchronous Federated Learning 119

List of tables

3.1 Summary of Main Notations 35

3.2 The comparison of 3LC data compression scheme and no data compression in data transmission with different sparsity 50

4.1 Notations definition in MPCF 64

List of Publications

- **Z. Yu**, J. Hu, G. Min, Z. Zhao and W. Miao, "Mobility-Aware Proactive Edge Caching for Connected Vehicles using Federated Learning", IEEE Transactions on Intelligent Transportation System, 2020, DOI:10.1109/TITS.2020.3017474.
- W. Gao, Z. Zhao, **Z. Yu**, G. Min, M. Yang and W. Huang, "Edge Computing based Channel Allocation for Deadline-driven IoT Networks", IEEE Transactions on Industrial Informatics, vol.16, no.10, pp. 6693-6702, 2020.
- **Z. Yu**, J. Hu, G. Min, H. Xu, J. Mills,"Peer to Peer Federated Deep Learning based Proactive Caching for IoV", IEEE International Conference on Parallel and Distributed Systems, to appear, 2020 (Invited paper).
- **Z. Yu**, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, N. Georgalas, "Federated Learning Based Proactive Content Caching in Edge Computing", in Proc. of GLOBECOM, 2018.
- W. Miao, C. Luo, G. Min, Y. Mi and **Z. Yu**, "Location-based Robust Beamforming Design for Cellular-enabled UAV Communications", IEEE Internet of Things Journal, 2020, DOI: 10.1109/JIOT.2020.3028853.
- H. Lin, J. Hu, J. Ma, L. Xu, **Z. Yu**, "A Secure Collaborative Spectrum Sensing Strategy in Cyber-Physical Systems", IEEE Access, vol.5, pp. 27679-27690, 2017.

- C. Huang, **Z. Yu**, G. Min, Y. Zuo, K. Pei, Z. Xiang, J. Hu, Y. Wu, "Towards Better Anomaly Interpretation of Intrusion Detection in Cloud Computing Systems", IEEE COMSOC MMTC Communications – Frontiers, 2017.

- **Z. Yu**, J. Hu, G. Min, J. Mills and J. Wang, "Proactive Content Caching in Mobile Edge Networks based on Communication-Efficient Federated Deep Learning", IEEE Journal on Selected Areas in Communications, 2020, Under review.

- **Z. Yu**, J. Hu and G. Min, "Federated Deep Learning for Cooperative Hierarchical Caching in Fog Networks", IEEE Communications Magazine, 2020, Under review.

- W. Miao, G. Min and **Z. Yu**, "A Hierarchical and Cooperative Federated Learning for Aerial 5G and Beyond Networks", IEEE Wireless Communications, 2020, Under review

- A. Qi, Z. Zhao, G. Zhao, G. Min, C. Shu and **Z. Yu**, "Mobility-aware System Deployment and Management for 5G Edge Computing: A Survey", IEEE Communications Surveys and Tutorials, 2020, Under review.

- W. Miao, G. Min, **Z. Yu** and J. Hu, "Ubiquitous Communication, Computation, Storage and Intelligence in 6G Networks", IEEE Journal on Selected Areas in Communications, 2020, Under review.

Chapter 1

Introduction

The proliferation of smart devices and the advancement of wireless communications technologies have brought us a variety of multimedia applications, including virtual reality/augmented reality, video on-demand, mobile healthcare and so on, in the Fifth Generation of mobile networks (5G), Internet-of-Things (IoT), Internet-of-Vehicles (IoV) and Fog Radio Access Networks (F-RANs). These applications continuously generate a huge amount of mobile traffic. According to the Cisco Visual Networking Index [1], mobile traffic data is expected to increase sevenfold from 2017 to 2022, reaching 77.5 exabytes per month. The steep rise of mobile traffic causes the increase of user latency and places a heavy burden on backhaul links that connect local base stations and the Internet. However, these emerging applications require higher network throughput and stricter network, which poses a significant challenge for traditional wireless networks.

To cope with the massive growth in mobile data traffic and satisfy strict performance requirements of applications, Multi-access Edge Computing (MEC) [2] has been recognised as a promising technology by bringing computing and caching capabilities to the edge of networks. Especially, the network edges are equipped with a number of edge servers to store contents that may be frequently requested by users. Users can directly fetch a variety of contents from the edge servers, instead of remote servers in the cloud. In this way, the latency

for fetching requested contents can be largely reduced and burden on network traffic can be alleviated. Caching contents at the network edges is referred to as edge caching [3]. In 5G, popular contents can be cached at Base Stations (BSs), *e.g.* Small Base Stations (SBSs), Macro Base Stations (MBSs). In IoV, the content can be placed at Roadside Units (RSUs) and vehicles. In F-RANs, both Remote Radio Heads (RRHs) and Baseband Unit (BBU) pools can store contents.

Due to the limited cache capacity of the network edges, it is crucial to design a caching scheme to effectively utilise the cache capacity. The current caching schemes can be generally classified into reactive caching and proactive caching. Reactive caching uses the observed request pattern of users to choose contents to be cached, such as First-In-First-Out (FIFO), Least Recently Used (LRU), and Least Frequently Used (LFU). They rely on static rules to decide cached contents, which can react fast to changes in recent content access patterns [4]. In contrast, proactive caching makes use of historical users' requests, content access patterns, and geographical or social information of users, etc, to predict content popularity and then places the predicted popular contents in caches before users' requests arrive. The recent breakthroughs in Machine Learning (ML) techniques have been widely used to forecast content popularity for proactive caching, such as Reinforcement Learning [5], Transfer Learning [6], and Collaborative Filtering [7], because ML techniques have powerful capability of handling large amount of data and accurate pattern recognition from these complex data. This research focuses on the learning-based caching schemes.

This chapter is organised as follows: Section 1.1 is devoted to the motivations and challenges of this research. The research aims and objectives of this thesis are presented in Section 1.2. The main contributions of this thesis are shown in Section 1.3. Finally, the outline of this thesis is provided in Section 1.4.

1.1 Motivations and Challenges

Edge caching, as an efficient approach to optimise the storage resources, brings popular contents closer to users, which can reduce service latency and avoid duplicate transmissions through backhaul links. Some progress has been made in utilising ML techniques for caching schemes in wireless networks, but learning-based caching schemes still have some challenging issues to be addressed. For example,

1) **Risks to user privacy**: Most traditional ML methods are designed for a highly controlled environment where the distributed users' data are gathered to train the learning models. The data generated by applications may involve privacy-sensitive information of users (*e.g.*, the content retrieval history and geographic information of users). Therefore, uploading and centralised processing these data may raise privacy and security concerns.

2) **Mobility**: Users frequently move from one edge node to another. This means that the cached contents at one edge node might become obsolete after users move out, while another edge node does not cache the contents for the incoming users. The lack of consideration of user mobility may lead to low cache efficiency. Additionally, in the IoV, vehicles send requests to an RSU/BS and go through its coverage area quickly, making the caching content easily to be out of date. To improve the cache performance, the caching scheme should be both context and mobility aware, making cache decisions based on the content popularity predictions and vehicles' mobility.

3) **Dynamic content popularity**: Content popularity is known to be volatile and dynamic in wireless edge networks. Different users may prefer different contents and their preferences may change frequently which is influenced by location and time. The spatial-temporal variability on popularity of contents adds substantial complexity in

content caching. It is also highly challenging for ML methods to accurately and quickly predict the content popularity given users' private data and their content request history.

4) **High communication costs**: For training these ML models, users need to upload their data to the central server that performs the training, which will cause large communication costs.

5) **Frequent hand-over**: For centralised ML model training, users or vehicles may pass several BSs/RSUs during the model training or content transmission, due to the small coverage area of BS/RSU. This may seriously affect the performance of the trained ML model and degrade the Quality of Service (QoS) and Quality of Experience (QoE) for users.

6) **Utilisation**: The redundant contents may store in the cache capacity of edge nodes, which lacks the global optimization of cache resource utilisation. It is non-trivial to decide how and where to cache, given the limited cache sizes at the different level of edge nodes.

7) **Scalability**: Scalability would be another issue for the centralised training setting. The number of users/vehicles grow, the amount of data generated by the corresponding users/vehicles increases. The centralised ML algorithms may find it difficult to handle such data due to the incurred high computation and communication costs.

1.2 Research Aims and Objectives

Federated Learning (FL) [8], as an emerging framework, fits the ML techniques into the edge of networks. It enables end nodes (*e.g.*, mobile devices, vehicles and so on) to collaboratively learn a shared model by aggregating locally-computed model updates while keeping all the training data on devices. The advantage of FL is the decoupling of the ability to train ML

model from the need for directly accessing the raw data. The privacy and security risks can be significantly reduced. Coupled with FL, the learning-based caching schemes can protect user privacy. The main objectives of this research are:

- To propose a communication-efficient federated deep learning for proactive caching schemes with the aim of reducing communication cost and improving caching efficiency.
- To investigate a mobility-aware federated learning framework and design a mobility-aware caching schemes to support high mobility of users.
- To exploit a peer-to-peer federated learning to ease reliance on the fixed central server, eliminate the issue of hand-over between RSUs/BSs and achieve lower service latency.
- To develop a cooperative and hierarchical caching scheme based on federated learning to maximise the utilisation of available caches with edge nodes.

1.3 Contributions

To achieve the above objectives, this research proposes new proactive caching schemes to improve caching performance and mitigate user privacy risks. The accuracy and effectiveness of the proposed schemes are demonstrated by extensive experimental results with real-word datasets. The major contributions of this research are summarised as follows:

- A communication-Efficient Federated deep learning based Proactive content Caching scheme is developed to improve cache hit ratio[1] and reduce communication cost. A one-class collaborative filtering based variational autoencoder model is designed and it integrates the model into our FL framework. This model learns deep latent

[1]Cache hit ratio is used to evaluate the performance of mobile edge caching, which accounts for the ratio of the requested content stored in the edge server to the number of users' requests on the cache.

representations from the recent history and contextual information of users. These learned representations will be used to accurately predict content popularity for making proactive caching decisions. This FL framework also introduces a time-driven weighted model aggregation method to improve the convergence and accuracy of the shared model by exploiting the previously trained local model. Moreover, the 3LC model compression algorithm is employed in the proposed FL framework to compress the model updates in order to further reduce communication costs.

- A mobility aware federated learning scheme for edge caching in vehicular networks is developed, which can protect users' privacy, reduce communication costs and support high mobility of vehicles. This new scheme includes four main components: content popularity prediction, vehicle selection, model aggregation and cache replacement. It utilises the context-aware adversarial autoencoder model to predict the popularity of contents, which turns an AutoEncoder into a generative model by adding the adversarial network to the AE architecture. It helps to learn deep latent representations from users' historical requests and contextual information, and obtain implicit relationships between users and contents for improving prediction accuracy. Mobility-aware vehicle selection, model aggregation and cache replacement policies are exploited with the aim of optimising the caching resource utilisation in VNs. Especially, the decision for selecting vehicles to participate in the FL training process and the value of weights for parameter aggregation are dependent on the position and resources of connected vehicles. It can guarantee that vehicles have enough time for training and the RSU can aggregate high-quality updates. Meanwhile, the cache replacement policy dynamically updates the contents at RSU in response to its connected vehicles' preferences and predictions of content popularity.

- A peer-to-peer federated learning based proactive caching scheme is proposed for IoV with vehicles of high mobility. In the proposed scheme, a vehicle rather than a

fixed edge node, acts as a central server to aggregate ML models from nearby vehicles. A location and resource-aware vehicle selection scheme is developed in the peer-to-peer FL framework to enhance the performance of the trained global model, which ensures that the participating vehicles have enough resources to train the prediction model, and the server vehicle has enough time to aggregate updated models from neighbour vehicles. Due to the heterogeneous abilities of vehicles, a dual-weighted model aggregation scheme is designed to reduce the effect of straggler vehicles, in order to further improve the accuracy of the trained global model in the designed peer-to-peer FL.

- A hierarchical cooperative caching architecture is designed for F-RANs to leverage horizontal cooperation between the Fog Access Points (F-APs) and vertical cooperation between the BBU pool and F-APs to enhance the overall caching performance and global cache resource utilisation. The proposed method integrates the appended stacked autoencoder and one-class collaborative filtering to predict the popularity of contents. The appended stacked autoencoder is used to extract the hidden representations of users and contents. Whereas, the one-class collaborative filtering is utilised to effectively process the input data for a better recommendation of popular contents.

1.4 Outline of the Thesis

The rest of this thesis is organised as follows:

- Chapter 2 introduces the background knowledge of edge caching, artificial intelligence and federated learning. A detailed literature review on learning-based caching scheme and federated learning applications in wireless networks are then presented.

- Chapter 3 presents a communication-Efficient Federated deep learning based Proactive content Caching scheme (EFPC) to improve cache hit ratio and reduce communication cost.
- Chapter 4 develops a Mobility-aware Proactive Edge Caching Scheme based on Federated Learning (MPCF) to support high mobility of vehicles and adapt to the dynamic content popularity.
- Chapter 5 exploits a peer-to-peer federated learning based proactive caching scheme (PPFC) for IoV to ease reliance on the fixed central server in RSU and eliminate the issue of hand-over between RSUs.
- Chapter 6 proposes a federated learning based cooperative hierarchical edge caching scheme (FLCH) to maximise the utilisation of available caches with edge nodes.
- Chapter 7 concludes the thesis and outlines the future works.

Chapter 2

Background and Literature Review

Caching contents at the edge of networks is considered to be a effective solution to cope with ongoing traffic growth and address the backhaul bottleneck problem in wireless networks. Recent advances in Machine Learning (ML) and Federated Learning (FL) facilitate efficient content caching. This chapter presents a general background knowledge and gives an in-depth review of the related work of Multi-access Edge Computing (MEC), edge caching, ML and FL. The rest of this chapter is organised as follows. The background knowledge of MEC, edge caching, FL and ML is introduced in Section 2.1, 2.2, 2.3 respectively. A detailed literature review on learning-based content caching scheme is presented in Section 2.4.

2.1 Edge Caching

2.1.1 Multi-access Edge Computing

Emerging applications in wireless networks require low network latency and substantial network resources (*e.g.*, caching, computation, and communication), which cannot be fulfilled by the current wireless networks. Multi-access Edge Computing (MEC) has been considered as a promising paradigm by migrating cloud computation and caching capabilities to the edge

nodes of networks to satisfy the diverse requirements of applications, alleviate the traffic on backhaul links, and reduce service latency. Edge nodes have the capabilities to process and store data.

2.1.2 Fundamentals of Edge Caching

Caching contents at edge nodes is referred to as edge caching, which has been receiving significant attentions from both industry and academia in the past few years. By caching popular contents at edge nodes, requested contents can be obtained within one transmission hop. In this way, content retrieval latency and backhaul traffic can be significantly reduced.

Contents in wireless networks have the property of asynchronous content reuse that popular contents are requested by different users for multiple times. It causes the same contents in remote servers are repeatedly sent to users. Caching contents at edge nodes enables vehicles to fetch their requested contents within one transmission hop, instead of from the remote servers in cloud. Popular contents can be cached at edge nodes during during off-peak hours, whereas the requested contents can be served to users during the peak-time.

Edge caching brings several advantages. Firstly, the service latency of obtaining requested contents can be largely reduced, because contents are stored at the network edge, closer to users. Secondly, the backhaul traffic can be significantly alleviated since edge caching avoids to use transmissions via the backhaul links. Thirdly, energy consumption for transmitting data from the Internet can also be reduced by edge caching. Fourthly, caching efficiency can be improved by utilising the information collected from edge nodes, *e.g.*, content popularity, user preferences, mobility of user and channel state.

The key research issues in edge caching include where, how and what to cache.

2.1.3 Cache Placement

Cache placement is the answer of where to cache, which involves the selection of caching locations. In wireless networks, popular contents can be cached at edge nodes, *e.g.*, Macro Base Stations (MBSs), Small Base Stations (SBSs), relays and User Equipments (UEs). They have different storage capacities. In fog radio access network (F-RAN), both Baseband Unit pools (BBU) and Remote Radio Heads (RRHs) can be used to store popular contents. BBU is the baseband processing unit of telecom systems and RRH is a radio transceiver in a radio base station. Both of them have storage capacities. Edge nodes in the Internet of Vehicles (IoV) are roadside units (RSUs) and vehicles.

Mobile devices and vehicles have larger storage capacities, which can be utilised as cache nodes. The communication among mobile devices and vehicles can through Device-to-Device (D2D) links, Vehicle-to-Vehicle (V2V) links, etc. Compared to local caching at mobile devices and vehicles, different sizes of base stations, relays and RSUs provide relatively higher latency. However, they can cover a larger area and serve more users.

2.1.4 Caching Policy

Caching policy solves the problem of how to cache, which refers to the design of caching schemes. To improve the caching performance, caching schemes are implemented on edge nodes by utilising their own storages. Reactive [9] and proactive caching [10] [11] are introduced in respect of deciding whether to cache contents after or before content demands happen. Reactive caching schemes decide caching contents after they have been requested. They use the static rules to decide caching contents, which can react fast to changes in recent content access patterns. Whereas, proactive caching makes use of historical users' demands, content access patterns and users' geographical or social information to predict content popularity and then places caching contents before users' requests arriving according to prediction. Proactive caching improves caching performance and guarantee QoS

requirements by pre-fetching predicted popular contents during off-peak times and serving predicted popular contents during the peak time.

Centralised and distributed caching [12] are classified according to position of the caching decisions. Centralised caching makes caching decisions by a central controller. Whereas, distributed caching utilises their local information to decide caching contents at local caches. Due to the limited storage at edge nodes, individual caching scheme may result in insufficient utilization of cache nodes. For example, some caches at edge nodes are overused, while other edge nodes still have many vacant spaces. To cope with this problem, cooperative caching scheme [13] [14] [15] is designed by sharing contents with other edge nodes. In this way, the utilisation rates of all edge nodes can be enhanced by fully using under-utilised caches at single edge node. The location and the storage capability determine which nodes in the cooperative caching. Also, the cache efficiency can be improved, because the cache capacity increases. Coded caching [15] uses the network coding techniques to reduce the number of transmissions. It firstly aggregates data messages. Then, data is forwarded to the same destination and separated into different messages. Therefore, network throughput can be increased and service delay can be reduced. To deal with the uncertainty and movement of users, probabilistic caching is designed [16]. In probabilistic caching, contents are cached at different caches with different probabilities.

To decide what to cache, the popularity of content is required. It is the ratio of the number of requests for a particular requested content over the total number of requests from users during a period of time in a certain region. Content popularity has a key feature, which is the asynchronous content reuse property. Only a small number of very popular contents account for the majority of data traffic. It indicates the probabilities of users requesting certain contents.

Content popularity can be generally classified into static and dynamic. For static content popularity, it is usually assumed to follow the Zipf model [17]. However, it cannot reflect the

real-time content popularity, since the popularity of contents is time-varying and unknown in advance. In contrast, dynamic content popularity is more realistic and suitable for the wireless networks. It changes according to the preferences of users. Different vehicular users may prefer different contents and their preferences may change frequently which is influenced by location and time. The spatio-temporal variability on popularity of contents adds substantial complexity in content caching. It is challenging to make caching decisions without the knowledge of the content popularity. Predicting dynamic content popularity has drawn much attention and several prediction models have been proposed, e.g., autoencoder [18], transfer learning [19] [20], LSTM [4], Bayesian [21], and multi-layer perception [10]. These learning-based forecasting models are carried out by Machine Learning (ML) techniques. The recent breakthroughs in ML show great potential for making accurate popularity predictions through learning from experience.

Additionally, the mobility of users affects the popularity of contents, since different users may prefer different contents. Moreover, various content types in wireless networks (*e.g.*, web pages, files, and videos) exhibit more dimensions and different lifetimes with varying delay requirements. Typically, the lifetime of contents in wireless networks is short. As a result, the cached content is easy to be out-of-date. Using the outdated contents in caching schemes, caching performance may degrade. Thus, it is essential to update caching contents at intervals, replacing the unpopular contents with the popular ones.

2.1.5 Cache Replacement

Conventional caching schemes include First-In-First-Out (FIFO), Least Frequently Used (LFU), Least Recently Used (LRU), Most recently used (MRU) and their variants, which are utilised for cache eviction and replacement. FIFO evicts the contents based on the order they were added, without considering how many times or how often contents were requested before. In LFU, the least frequently used content in the cache is discarded whenever the

cache capacity is full. LRU firstly removes the least recently used content in the cache, when the limit of cache capacity is reached. In contrast to LRU, MRU discards the most recently used contents firstly. These conventional caching schemes follow static rules, but they lack consideration of dynamic content popularity which varies with time.

The variants include Segmented LRU (SLRU), Least frequent recently used (LFRU), Time aware least recently used (TLRU), and so on. SLRU divides the cache into a probationary segment and a protected segment. Both segments follow LRU. New contents are firstly stored in the probationary segment. When a cached content is requested, it will move to the head of the protected segment. If the cache capacity of the protected segment is full, LFRU pushes content from the protected segment to the probationary segment. When the probationary segment overflows, it directly evicts the least recently used contents. LFRU combines the advantages of LFU and LRU caching schemes. It divides the cache into two parts: privileged part and unprivileged part. The highly popular contents are cached at the privileged part. The rule of replacement in privileged follows LRU. New contents insert into unprivileged part and popular contents are promoted from unprivileged part into privileged partition. TLRU introduces Time to Use (TTU), which is a time stamp of a content. It indicates the usability time for the content. The cache contents are updated according to the value of TTU. It ensures that the short life contents with less popularity are replaced with the new contents.

2.2 Artificial Intelligence

Recent advances in Artificial Intelligence (AI) facilitate efficient edge caching. It is used to answer what, where and how to cache at the edge/wireless networks, so that the caching performance can be improved, such as, content popularity prediction, cache decision optimization, and user clustering. In this thesis, I mainly utilise ML and DL to my proposed methods.

2.2.1 Machine Learning

The definition of ML is based on algorithms that can learn from data, without relying on rules-based programming. ML utilises algorithms to parse data, learn from it, and then make a determination or prediction about something. The aim of ML is to build learning capability in computer without human intervention or assistance. ML algorithms are categorised as supervised learning and unsupervised learning. Supervised learning aims to learn a general rule for mapping inputs to outputs based on the labelled data set. Whereas, unsupervised learning aims to learn the mapping function based on unlabelled data.

Supervised Learning

Supervised Learning is the task to learn a function which maps a training set of examples of inputs x to outputs y based on example input-output pairs. In supervised learning, the training data consists of inputs and the correct paired outputs. During the training process, the pattern in the data will be studied. After training, new unseen inputs will be put into the learned algorithm and the label for new inputs will be determined based on prior training data. The objective of a supervised learning model is to predict the correct label for new input data. A supervised learning algorithm can be written as follow:

$$y = f(x), \tag{2.1}$$

where y is the predicted output that is determined by a mapping function. A class is assigned to an input x. The mapping function created by the supervised learning model connects input features to a predicted output. Supervised learning is used to process and classify data, which can be grouped into the problems of clustering and classification. The widely used supervised learning models are K-nearest neighbours, decision trees, support vector machines, random forest and so on.

However, in some cases, the outputs y may be difficult to collect automatically, which needs to be provided by a human supervisor.

Unsupervised Learning

Unsupervised learning only has input data x (features) without the corresponding output y. There are no labels in the unsupervised learning. It leaves output y to find structure in its input x. The goal of unsupervised learning is to discover hidden patterns and learn features in data. In unsupervised learning algorithms, based on similar attributes, features in data, and naturally occurring patterns, new input can be classified. Different strategies are used to divide data into different groups.

Many applications are used in unsupervised learning, such as clustering, anomaly detection, association mining and latent variable models.

2.2.2 Deep Learning

Deep Learning (DL) is a subset of AI, which imitates the working process of the human brain to process data and discover patterns in data. DL is inspired by the structure and function of biological Neural Networks (NNs). It is able to learn from complex data that is both unlabelled and unstructured data, without human supervision. DL is widely used in making decisions, translating languages and recognising speech. NN generally consist of input, hidden, and output layers.

In DL, the neural network is the most important part. NN uses basic components, known as neurons to perform highly complex, non-linear and parallel computations. In a NN, its nodes are the equivalent components of the neurons in the human brain. These nodes use activation functions to perform non-linear computations. The most frequently used activation functions are the sigmoid and the hyperbolic tangent functions. Simulating the way neurons

are connected in the human brain, the nodes in a NN are connected to each other by variable link weights.

A NN has many layers. The first layer is the input layer and the last layer is the output layer. Layers between the input layer and the output layer are hidden layers. The output of each layer is the input of the next layer and the output of the last layer is the result. By changing the number of hidden layers and the number of nodes in each layer, complex models can be trained to improve the performance of NNs. NNs are widely used in many applications, such as pattern recognition. The most basic NN generally comprises three layers: (i) input layer, (ii) hidden layer, and (iii) output layer.

Neural networks with a single hidden layer are generally referred to as shallow NNs. In contrast, neural networks with multiple hidden layers between the input layer and the output layer are called deep NNs [17–19]. For a long time, shallow NNs are often used. To process high-dimensional data and to learn increasingly complex models, deep NNs with more hidden layers and neurons are needed. However, deep NNs increase the training difficulties and require more computing resources. In recent years, the development of hardware data processing capabilities (*e.g.*, GPU and TPU) and the evolved activation functions (e.g., ReLU) make it possible to train deep NNs [20]. In deep NNs, each layer's neurons train on a feature representation based on the previous layer's output, which is known as feature hierarchy. The feature hierarchy makes deep NNs capable of handling large high- dimensional datasets. Compared to other machine learning techniques, deep NNs generally provide much better performance.

In a feedforward neural network, a weighted and bias-corrected input value is passed through a non-linear activation function (*e.g.*, ReLu and Softmax functions) to get an output. A typical DNN contains multiple hidden layers, mapping an input to an output. The objective of training a DNN is to optimize the weights of the network such that the loss function, *i.e.*, difference between the ground truth (the expected output) and model output, is minimized.

Normally, the dataset is divided into the training dataset and test dataset. The training dataset is used as input data for weights optimization. The weights are adjusted through stochastic gradient descent (SGD) and the weights are updated by the learning rate lr and the loss function L. The SGD formula is as follows:

$$W = W - lr\frac{\partial L}{\partial W} \tag{2.2}$$

$$\frac{\partial L}{\partial W} \approx \frac{1}{m}\sum_{i\in B}\frac{\partial l^i}{\partial W} \tag{2.3}$$

Eq. 2.2 presents a mini-batch SGD and Eq. 2.3 achieves the average gradient matrix over the gradient matrices of B batches. Each batch consists of m training samples, which is a random subset. The gradient matrices are derived through backpropagation from the input gradient e. The training iterations are then repeated over many epochs.

The most popular DL models are convolutional neural networks, recurrent neural networks, multilayer perceptron, autoencoders and so on.

2.3 Federated Learning

2.3.1 The Framework of Federated Learning

Coupled with recent advancements in machine learning, learning-based proactive caching schemes are able to accurately predict content popularity and further improve cache efficiency, but they may need to centrally analyse users' content retrieval history and personal data, and model training occurs in powerful cloud servers, leading to privacy concerns. Amid growing privacy concerns, Federated learning (FL) provides a new framework for fitting ML techniques into the edge while protecting users' privacy, since the ubiquity of mobile devices are equipped with fast processors (including GPUs) and increasing computing capabilities.

FL trains a shared global ML model under the instruction of a central server from data scattered at nodes (*e.g.*, mobile devices). The participating devices are typically large in number. They upload model updates to the central server and keep their training data locally. These devices are used as the computation nodes to perform a ML model training on their local data. FL provides distinct advantages compared to learn the ML model in the cloud central server, since uploading model updates has less privacy-sensitive than uploading data itself. In FL, the server only stores model updates, not data. Thus, FL can significantly mitigate the privacy and security risks by limiting the attack surface to only the device, instead of the cloud. Compared with conventional distributed machine learning, FL adopts the large number of participating devices, non-i.i.d. data and highly unbalanced data. It leverages the computational power and data-locality of the large number of mobile devices.

FL is performed by multiple communication rounds and a typical round consists of the following steps:

1) User Selection: A subset of users are chosen to participate in a training round, according to the requirements of model training, the features of users and their data distribution, and so on. (The selection criteria can be calibrated to the need of server, *e.g.*, training efficiency and effectiveness.)

2) Model Dissemination: Once the participating users are selected, the central server sends the initial global model to each selected participating users.

3) Distributed Model Learning: Each selected user computes an updated model based on its local data.

4) User feedback: The updated models are sent from the selected users to the central sever.

5) Model Aggregation: The central server receives updated models from selected users and then constructs an improved global ML model by aggregating these updated models, according to an aggregation mechanism (*e.g.*, FedAvg algorithm).

6) Model Update: The central server updates the global ML model and then sends back to the selected users.

Above steps are repeated until a stable result achieved at the central server.

2.3.2 Unique Characteristics of Federated Learning

1. Slow and unstable communication

   Compared with the traditional distributed training in a cloud central server, the communication environment is stable. The information transmission rate is relatively high with no packet loss. By contrast, the communication environment of FL is slow and unstable since heterogeneous devices are involved in FL training. As a result, some participating users may drop out.

2. Massively distributed and heterogeneous devices

   A large number of users are involved in FL model training, which have various resource constraints, such as different computation capabilities, battery status. Additionally, devices may have different levels of willingness to participate.

3. Unbalanced Data and Non-IID

   The amount of local training data at devices can vary, since some users make heavy usage of some particular applications. Thus, the distribution of training data on any device cannot represent the whole data distribution.

2.3.3 Federated Learning and its applications in Wireless Networks

Federated Learning is firstly proposed by Google [8], which provides a new approach to fitting machine learning techniques into the edge. Mcmahan *et al.* [8] designed the primitive FL protocol. It performs synchronous optimization in federated settings. Xie et al. [22] proposed an asynchronous federated learning (FedAsync). The proposed asynchronous

federated optimization scheme regularizes local optimization and adopts the update of the global model without blocking. Similar to [22], Sprague *et al.* [23] exploited an asynchronous protocol in a geo-spatial application for training a global model asynchronously, which allows the devices to join halfway. However, the main challenge of asynchronous approaches is that the server may overwhelm, because the server receives too many local updates from active users. However, the model convergence can be improved a bit.

For model accuracy, Chen *et al.* [24] demonstrated that synchronous Stochastic Gradient Descent (SGD) outperforms asynchronous approaches in the data centre setting. A number of variants have been proposed to mitigate the deficiencies of FL from different aspects such as FL communication round efficiency [25] and communication cost [26]. Wang *et al.* [27] proposed a control algorithm that adaptively determines the interval of global aggregation under a given resource budget. To address the inefficiency of FL under poor wireless channel conditions, Nishio and Yonetani [25] designed a protocol to filter out slow users in MEC framework, which is based on the estimation of the users work time at the selection stage and consequently shorten round length. However, their scheme relies on the model accuracy and does not take the user unreliability into account.

To speed-up the convergence rate of FL, the optimisation mechanisms for traditional distributed SGD have great potential in FL. Lian *et al.* [28] proposed a gradient staleness control to guarantee convergence. Dutta *et al.* [29] presented a theoretical characterization of the speed-up offered by asynchronous methods. It analyses the trade-off between the error in the trained model and the actual training runtime (wallclock time) by considering random straggler delays. Wang *et al.* [30] designed an Asynchronous Stochastic Gradient Descent (ASGD) by adjusting the learning rate based on the staleness of incoming gradients. Smith *et al.* [31] proposed a fault and straggler tolerant multi-task learning method to address the statistical and systems challenges of FL. Chen *et al.* [24] introduced backup workers to reduce server waiting time in synchronous stochastic optimization.

Most machine learning based caching schemes are designed for a highly controlled environment by uploading data to the server. It may bring privacy risks to users. To address the above challenges, FL is an enabling technology, which allows collaborative learning in the wireless networks. Feng *et al.* [32] constructed a cooperative communication platform by adopting the relay network to support model update and trade with the Stackelberg game model. Wang *et al.* [33] used deep reinforcement learning to jointly manage the communication and computation resources. It presented the FL based In-Edge AI framework to deploy intelligent resource management in the MEC system. Similar to [33], the authors in [34] proposed a deep reinforcement learning based method to optimise computation offloading decisions in IoT systems. Qian *et al.* [35] introduced a privacy-ware service placement scheme to deploy user-preferred services on edge servers with consideration for resource constraints in edge cloud. Saputra *et al.* [36] presented a federated energy demand learning approach to manage energy resource in charging stations for electric vehicles. Samarakoon *et al.*[37] introduced a FL-based method to predict the tail distribution of the network-wide queue lengths, in order to realise the status of networks. Ye *et al.* [38] designed a selective model aggregation method to select participating vehicles by considering the computation capacity of vehicles and data quality at vehicles. Lu *et al.* [39] proposed a hybrid blockchain based asynchronous FL scheme to secure data sharing. Lu *et al.* [40] proposed an asynchronous FL scheme in IoV for resource sharing purpose, which also combines differential techniques into FL to protect the privacy of local updates. Roy *et al.* [41] proposed a peer-to-peer decentralized federated learning, without a central server. However, the above works lack consideration inherent limitations of edge caching in IoV, such as time-varying content popularity and mobility of users.

2.4 Learning-based Edge Caching

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 [42] update cache contents based on static rules such as FIFO, LRU and LFU, which are reactive caching. However, they are not adapted to the dynamically changing content popularity. Recent research has put in effort to develop proactive content caching 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.

2.4.1 Edge Caching with Prior Knowledge of Content Popularity

We start by briefly introducing related work which assumes the content popularity with prior knowledge. In some cases, the contents request from users are modelled by a Zipf distribution [17]. With knowing the demand of users, Maddah-Ali *et al.* [43] 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 [44] which supposes the user requests can be predicted. Jiang *et al.* [45] presented a cooperative content caching scheme to minimise the average downloading latency by utilising a primal-dual decomposition method. It can decouple the problem into two level optimization problems by using the subgradient method. Golrezaei *et al.* [46] proposed a distributed caching framework and D2D collaboration to address the increasing demand for video content in wireless networks. It applies an approximation technique to decide which contents are cached in which helper, in order to minimise the user's delay and increase the throughput of the network. Kang *et al.* [47] developed a framework for an mobile content delivery networks, which provides the demanded popular contents to nearby users via D2D communication links. Poularakis *et al.* [48] designed a caching paradigm to save energy through the multicast transmission of identical contents and to support the massive mobile data demand in 5G.

Liu *et al.* [49] proposed a mobility-aware coded caching scheme to optimise throughput in cellular networks. The work in [50] exploited a coded caching-and-delivery scheme to explore a trade-off between the transmission cost of base stations, the storage cost of the small cells and the cost of connecting users to multiple cells. To support seamless mobility, Siris *et al.* [51] proposed a distributed proactive caching scheme. It utilises the mobility information of users to decide the position of caching and employs a congestion pricing scheme to allocate cache storage. Shanmugam *et al.* [52] provided the FemtoCaching system to minimise the delay of user requests by placing the optimal demanded files to the caches. Chen *et al.* [53] used a Markov decision process to improve the energy efficiency of proactive caching.

2.4.2 Edge Caching without Prior Knowledge of Content Popularity

All aforementioned related works suppose that content popularity is known in advance, which is not true in real networks. Content popularity changes dynamically. Thus, some research study content popularity prediction by leveraging ML techniques. ML algorithms show the great potential for making accurate predictions based on the learning experience from the past. Bastug *et al.* [7] 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[54]. Sengupta *et al.* [55] 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 [54] is used to learn the content's popularity with considering different users' information. It is an extended work

of [56] which aggregated context information, such as user density and request file time. It proposed a contextual multi-armed bandits based algorithm to learn context-dependent popularity by considering the context information of a single user and diversity of content popularity among different users. Blasco and Gündüz [57] applied combinatorial multi-armed bandit to improve cache performance by observing the requests from users and then updating contents in the cache. Elbamby *et al.* [5] combined user clustering and content caching in wireless small cell networks and utilised the reinforcement learning algorithm to predict content popularity. An adaptive caching scheme was presented in [58] by using an extreme-learning machine neural network to predict content popularity. Then, it utilises mixed-integer linear programming to compute where to cache the content and selects the optimal physical cache sizes in the network. Chen *et al.* [59] optimised the caching policy by learning user preference for cache-enabled D2D communications. They first formulate an optimisation problem to maximise the offloading probability and then utilise a greedy algorithm to solve this problem. The user request behaviour is modelled by probabilistic latent semantic analysis and model parameters are predicted by expectation maximisation algorithm. Bacstuug *et al.* [6] and Bharath *et al.* [20] leveraged the transfer learning to estimate content popularity. Long short-term memory was applied to predict the content caching in [60]. Doan *et al.* [61] proposed a caching scheme by predicting the popularity of videos, according to the extracted features from published videos, and their similarity between published videos and new videos. An extended work [62] designed a social-aware caching scheme by using social-awareness to evaluate the frequency of D2D connections. Li *et al.* [63] presented a cache content placement and delivery for D2D networks. Echo state network is used to predict users' mobility and long short-term memory network is utilised to content popularity. Thar *et al.* [64] developed a deep learning based caching framework to predict video popularity in MEC and then pushed the popular contents to the BSs. Wang *et al.* [65] proposed a BS based distributed edge caching to minimise the transmission cost

between BSs by utilising Q-learning. Zhu *et al.* [66] applied deep reinforcement learning to mobile edge caching, in order to make caching agents adapt to the dynamic and complex environments. A distributed multi-tier caching scheme was developed in [67], which is based on deep Q-learning. Jiang *et al.* [68] exploited reinforcement learning based caching scheme to minimise download latency for D2D caching. Lei *et al.* [69] proposed a deep learning based proactive caching scheme in 5G. A stacked sparse autoEncoder is utilised to extract hidden features from historical content requests to predict content popularity. Hou *et al.* [19] investigated a proactive caching based cooperative caching scheme in MEC to improve user quality of experience and reduce transmission cost by exploiting transfer learning approach. Jiang *et al.* [70] designed a transfer learning based multi-agent reinforcement learning caching scheme, without the prior knowledge of content popularity. Chuan *et al.* [71] presented a common interests based caching scheme to maximise the probability of content delivery. Bommaraven *et al.* [72] proposed an active learning based caching scheme for predicting accurate content popularity.

Edge Caching also has been widely used in the IoV. Edge servers (*e.g.*, BSs and RSUs) and vehicular users with ample caching capacities can cache popular contents to increase the agility for service provisioning. A number of recent works [73], [74], [75], [76], [16] have been reported to investigate the content caching at RSUs in vehicular networks. Hu *et al*. [73] proposed a multi-object auction-based method to solve the competition of content providers caused by the limited storage resources of RSUs. Ding *et al*. [74] studied three methods (optimal, sub-optimal and greedy) to allocate contents on RSUs, aiming to minimise the average downloading time for requested contents. Su *et al*. [75] designed a cross-entropy based dynamic content caching scheme to optimise cache resources by utilising cooperation among RSUs and the request history of vehicles. High movement of vehicles results in unstable connectivity, and vehicles may not have enough time to download the entire requested content during the time staying in the area of one edge node. Thus, a

mobility-aware probabilistic caching scheme was applied in [76] by considering the vehicle trajectories and content service time at the edge node. Mahmood *et al*. [16] developed a probabilistic caching scheme to store content chunks at edge nodes by considering both the historical statistics of achievable data rates and the time of vehicles staying in the area of edge nodes.

Connected vehicles, equipped with storage resources, can be exploited as caching entities to cache the popular contents locally, which brings the benefits of utilising their own resources. Kumar *et al*. [77] presented a peer-to-peer cooperative caching scheme for data dissemination that leverages a Markov chain model to share information among multiple vehicles and uses a probabilistic method to update the existing data. Fang *et al*. [78] provided a cooperative caching scheme for cluster-based VNs, which considers both the caching resource-constrained vehicles and caching status of vehicular clusters in a global view. Deng *et al*. [79] presented a distributed probabilistic caching scheme to make caching decisions. In this scheme, users' content requests, the importance of vehicles and their movement characteristics are all taken into consideration. A cooperative caching scheme was designed in [80]. It is based on the mobility prediction that estimates the probability of vehicles visiting hot spot areas. In order to minimise users' delay, [81] investigated a caching placement in both vehicular and RSU layers. Zhang *et al*. [82] developed a mobility-aware cooperative caching framework, where vehicles are as caching nodes to share contents tasks with BSs. Park *et al*. [83] proposed a distributed proactive caching scheme in VNs by distributing contents for RSUs, based on the movement of vehicles. Ainagar *et al*. [84] introduced a mobility-aware proactive caching scheme by taking the effect of the vehicle velocity into account. Chen *et al.* [15] introduced a cooperative edge caching scheme for connected vehicles by considering the different requirements for location-based and popular contents. Gad *et al.* [85] designed a hierarchical proactive caching by utilising the storages at vehicles and RSUs to minimise the vehicle communication latency. Zhang *et al.* [86] proposed a proactive caching scheme for

autonomous vehicles by adopting a non-negative matrix factorization technique to estimate the preference of users. The contents at video level are stored at the core network nodes, whereas the chunk level contents are cached at edge nodes.

Ndikumana *et al*. [10] studied a deep learning based proactive caching scheme by adopting a Multi-Layer Perceptron (MLP) approach to predict the popularity of contents within the coverage area of mobile edge computing (MEC) servers, and exploiting a Conventional Neural Network (CNN) to estimate the age and gender of passengers. Comparing the MLP's outputs with CNN's outputs, the contents which need to be downloaded from MEC servers to vehicles can be decided. MEC is one of the prospective technologies to enhance the performance of 5G. It brings the cloud computing and caching capabilities to network edges (*e.g*., base stations, access points), thus various tasks can be executed at the edge rather than remote clouds. A Q-learning based proactive caching scheme was devised in [87] with the support of a long short-term memory neural network. A deep reinforcement learning based content caching scheme was exploited in [14] by optimising the content placement and content delivery to minimise content delivery latency. Zhang *et al.* [13] designed a heterogeneous information network-based content caching scheme to reduce network load and enhance the quality of experience by combining data mining techniques with the features of IoV. Zhu *et al.* [88] investigated a deep reinforcement learning based approach to solve the problem of automatic vehicle control and the selection of proactive caching action. Zhang *et al.* [89] presented a proactive caching scheme for vehicular multi-view 3D videos which utilises deep reinforcement learning to select views set and allocate cache memory.

2.5 Summary

In this chapter, the fundamental knowledge of edge caching has been investigated. Then, a description of artificial intelligence has been presented. Finally, a comprehensive survey of

federated learning has been provided. It also indicates the new challenges in edge caching and federated learning. This research will address these mentioned challenges.

Chapter 3

Communication-Efficient Federated Learning based Proactive Caching

3.1 Introduction

According to a Cisco forecast report [90], mobile data traffic would increase to 77.5 exabytes per month by the end of 2022, up from 29 exabytes per month in 2019. To cope with the massive growth in mobile data traffic, multi-access edge computing (MEC) [2] was proposed as a promising technology that brings computing and caching capabilities to the edge of networks (*e.g.* base stations, edge routers, and access points). Especially, the network edge are equipped with a number of edge servers to store different contents requested by users. In this way, the latency for fetching requested contents and network traffic can be largely reduced, because a variety of contents can be directly fetched from the edge servers. Learning-based proactive edge caching has received considerable research attention, since Machine Learning (ML) algorithms have powerful capability of handling large amount of data and accurate pattern recognition from these complex data. However, utilising ML techniques for proactive caching in edge networks faces the challenges of user privacy concerns, high communication costs, scalability and dynamic content popularity.

To address the above challenges, Federated Learning (FL) [8], as an emerging ML paradigm attracting significant interests, plays a key role. It enables mobile devices to collaboratively learn a shared model by aggregating locally-computed model updates while keeping all the training data on devices. The advantage of FL is the decoupling of the ability to train ML model from the need to directly access to the raw data. The privacy and security risks can be significantly reduced. Therefore, we propose a communication-Efficient Federated deep learning based Proactive content Caching scheme (EFPC).

The novelty of this chapter can be summarised as follows:

1. We propose a One-Class Collaborative filtering based Variational AutoEncoder (OCC-VAE) model and integrate the model into our FL framework. This model learns deep latent representations from the retrieve history and contextual information of users. These learned representations can be clustered in the latent space and will be used to accurately predict content popularity for making proactive caching decisions.

2. The FL framework we designed introduces a time-driven weighted model aggregation method to improve the convergence and accuracy of the shared model by considering the previously trained local model and currently trained local model. Moreover, the 3LC model compression algorithm is employed in the proposed FL framework to compress the model updates in order to further reduce communication costs.

3. We evaluate the performance of the EFPC scheme via a MEC-like testbed using real-world datasets (MovieLens and Netflix). The experimental results show that our scheme outperforms the reference caching schemes including LRU, LFU, m-ε-Greedy and Autoenocder based method. Moreover, our scheme can compress the transmitted model data by up to four-fold without dropping its cache hit ratio.

The rest of this chapter is organised as follows: Section 3.2 describes the system architecture of EFPC. The detailed implementation of the EFPC is presented in Section 3.3. The

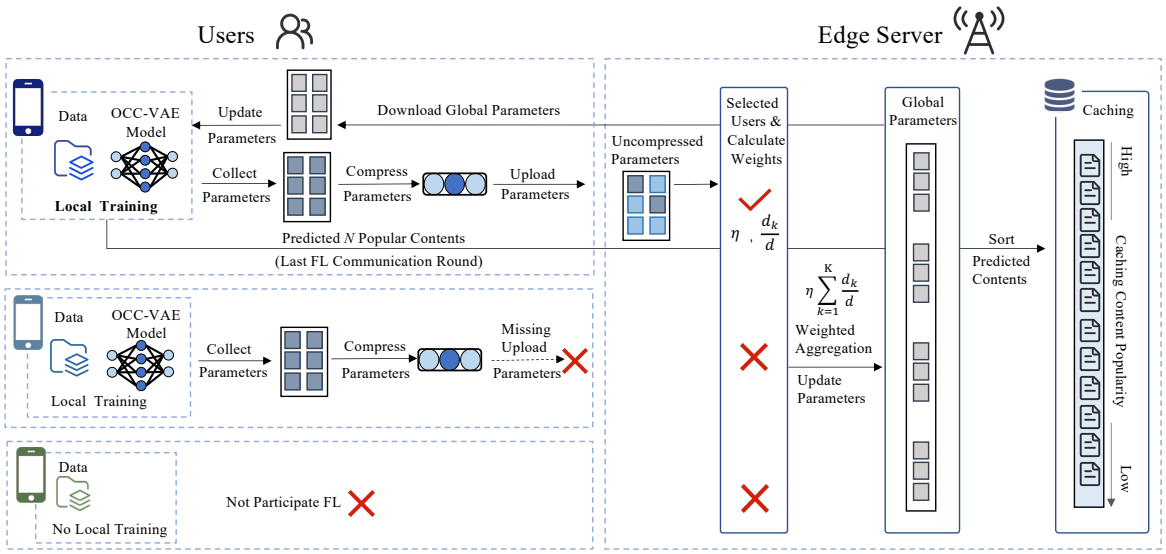


Fig. 3.1 Architecture of the EFPC Caching Scheme

performance evaluations are provided in Section 3.4. Finally, Section 3.5 concludes this chapter.

3.2 System Architecture of EFPC

We consider a MEC platform that consists of a base station (BS) and a set of U users. The BS is equipped with a cache-enabled edge server and linked to the Internet through a reliable backhaul link. The edge server in the BS has limited storage, which can store up to N contents from a content library. A caching decision module is installed at the edge server to decide how to effectively utilise the limited cache storage. Users equipped with smart devices are located in the coverage area of the BS, communicating BS via wireless links. If a user is interested in content that is stored in the BS's edge server (*i.e.* a cache hit), the BS can directly transmit the requested content to the user. In this way, the traffic load on the backhaul link can be reduced significantly. Otherwise, if the requested content is not cached in the edge server (*i.e.* a cache miss), this BS will forward the request to the Internet. The requested content is then downloaded from the Internet via the backhaul link. To alleviate

the backhaul link congestion, reduce the response time of users' services and maximise the cache hit ratio, it is essential to place the most popular contents on the edge server.

Caching popular contents on the edge server requires knowledge about content popularity distribution. However, this is not known in advance because content popularity is subject to change due to many factors. For instance, the contextual information of users influences content popularity, *e.g.* age and gender. The external factors can also affect content popularity, such as time of the day, day of the week, and locations. Hence, it is necessary to consider above information to predict future content popularity, but those information contain private and sensitive information of users. Many users are not willing to upload and share their data for privacy concerns. Thus, in practice, most existing caching schemes only can access a part contextual information of users, which affects the results of prediction and further leads to a low cache hit ratio.

Therefore, we design a new proactive caching scheme, EFPC, based on the emerging federated learning framework [8] where the training data is kept on users' own devices. In this way, it is possible to exploit all contextual information of users to learn future content popularity effectively while protecting users' privacy. Fig. 3.1 illustrates the architecture of the EFPC scheme. In the edge server, a global model is maintained that will be sent to each connected user. Users are responsible for learning the updates of the global learning model independently and locally. Then, these model updates are uploaded to the edge server where all the updates are aggregated to generate a new global model. The global model we trained is One-Class Collaborative filtering based Variational AutoEncoder (OCC-VAE), which forecasts the content popularity. One-class collaborative filtering has shown good performance for processing the input binary data [91]. VAE is able to extract useful features from data and cluster the data in the latent space that is an abstract multi-dimensional space containing feature values. Due to the limited communication bandwidth and a large number of parameters of the proposed OCC-VAE model, the upstream communication of the OCC-

VAE model from users to the server needs to be compressed. First, the OCC-VAE model is compressed at users before uploading it. Next, the server receives the compressed models and then decompresses them. The decompressed models are aggregated and conducted to a new global OCC-VAE model by using federated averaging. After that, the improved OCC-VAE model is sent to users again.

According to the predicted content popularity from each user, the N most popular contents can be downloaded from the remote cloud and then proactively cached at the edge server in advance. However, the popularity of contents fluctuates, so the cached contents need to be updated dynamically. We predict the popularity of all contents for several separate time intervals every day and update the caching contents in the edge server at the same time. The details about FL, OCC-VAE and 3LC data compression scheme will be described in the next Section.

3.3 Communication-Efficient Federated Learning for Edge Caching

In this section, we firstly introduce the communication-efficient FL framework. Next, 3LC data compression scheme is described to compress the training model in FL. The model we trained is the OCC-VAE model. Finally, we present the OCC-VAE model to predict content popularity by extracting hidden features from the information of users and clustering these information in the latent space. The N predicted popular contents are cached in the edge server. The list of notations definition used in this chapter is summarised in Table 3.1.

3.3.1 Communication-Efficient Federated Deep Learning

Federated learning (FL) is a decentralised machine learning technique which leverages distributed users' data locality and computation capacity to train a high quality shared global

Table 3.1 Summary of Main Notations

Symbol	**Definition**
U	Number of users
u	Index of users
C	Number of contents
c	Index of contents
N	Maximum number of caching contents in BS
K	Set of selected users for FL training
k	Index of selected users for FL training
w	Parameters of model
$f(w)$	Loss function
r	Number of FL communication rounds
d	Total data size among selected users
d_k	Local data size of the selected user k
D_k	Dataset of the selected user k
w_r	Parameters of model in the r^{th} round
w_{comp}	Compressed model
γ	Coefficient variable
t_r^k	Time of current round for the user k
$t_r'^k$	Time of latest update model round for the user k
B	Local minibatch size
η	Learning rate
E	Number of epoches
W_{in}	Final calculated model to compress
W_{qe}	Sparsified and quantized model
G, a_{ZR}	Variables for model compression
X	User-by-content request matrix
x	Sample from X
Z	Latent representation
z	Sampled latent variable
$\hat{a}$	Context of user
$p(x)$	Data distribution
$q(z \mid x)$	An inference neural network
$p(z \mid x)$	A generative neural network

model under the coordination of a central server. The shared global model is learnt by aggregating locally-computed updates, from a federation of selected users, while preserving all training data on users' devices. FL is free from disclosing users' data that enables a privacy-preserving training process. This training process iterates between the server and users. Each iteration is called the server-user communication round. However, the whole training process is unstable. This is because some users may lack the computational resources, have poor wireless connection or low amounts of local data. It leads to long training and loading time, and unfavourable training performance. Therefore, our proposed FL framework selects high-quality users as the participated users based on their resource conditions for efficient model training, instead of randomly choosing users.

A server-user communication round in our FL framework includes the following six steps:

1) **Training Request**: The server sends the requirement of training the OCC-VAE model to all users. Users then respond to the server with their current resource information (*e.g.* amounts of data, communication bandwidths, computation capabilities, and battery conditions).

2) **User Selection**: The server decides which users can participate in the training, according to the received information.

3) **Model Sharing**: The server distributes the global OCC-VAE model to the selected users.

4) **Local Training**: Each selected user trains the received OCC-VAE model locally by using its own data.

5) **Model Upload**: Each selected user uploads the parameters of its trained OCC-VAE model to the server.

6) **Model Aggregation**: The server aggregates all parameters of the uploaded model with a time-driven weighted aggregation method to generate a new global model and then updates the global OCC-VAE model.

The server-users communication rounds repeat the above process until the global OCC-VAE model reaches a stable cache hit ratio. At each communication round, the mobile devices use their local data, energy, and CPU resources to train the model. After the trained FL achieved a stable result, each selected user calculates the popularity of contents based on the latest obtained global OCC-VAE model and then sends a list of recommended caching content to the server. The edge server in BS sorts the recommended contents by their counts and caches the top N contents locally.

In our FL setting, we assume the whole data size of the training dataset is d which is formed by the training examples over K selected users with C contents. The training goal is to minimise the following objective of FL [8]:

$$\min_{w} f(w) = \frac{1}{d} \sum_{i=1}^{d} f_i(w), \tag{3.1}$$

where $f_i(w) = \ell(u_i, c_i; w)$ is the loss function of the prediction on the example (u_i, c_i) made with model parameters w. Based on the resource situation of each user, we select K users to participate in the FL training, where K users are indexed by k. Denote the set of indexes of training examples on user k is D_k, with $d_k = |D_k|$. Thus, the objective can be re-written as

$$\min_{w} f(w) = \sum_{k=1}^{K} \frac{d_k}{d} F_k(w), \text{ where } F_k(w) = \frac{1}{d_k} \sum_{i \in D_k} f_i(w), \tag{3.2}$$

Typically, stochastic gradient descent (SGD) is implemented to optimise the OCC-VAE model. In FL setting, with applying SGD optimisation, each communication round only calculates a single batch of gradients. It may cause large communication costs, since FL requires plenty of communication rounds to train a high-quality OCC-VAE model. Therefore,

we use the federated stochastic gradient descent method (FedSGD) [8] for optimisation. FedSGD allows each user to iterate multiple rounds and then takes the average of gradients $\bigtriangledown F_k(w_r)$ on its local data at the parameters of model w_r in the r^{th} round. These average gradients are uploaded to the server and then applied to update the global OCC-VAE model. We assumed that the data distribution of each user is independent and identically distributed. Additionally, the local dataset in each selected user k is generated from the usage of its device, such as video demands in daily life. Some users may make the heavy use of some particular services or applications, which causes the data imbalance problem for the federated training. To solve this problem, we implement the weighted aggregation method to aggregate the model in the edge server. The equations of updates of the model and aggregation are given by

$$w_{r+1} \leftarrow w_r - \eta \sum_{k=1}^{K} \frac{d_k}{d} \bigtriangledown F_k(w_r), \tag{3.3}$$

$$w_{r+1} \leftarrow \sum_{k=1}^{K} \frac{d_k}{d} w_{r+1}^k, \tag{3.4}$$

where η is the learning rate. The weighted sum is implemented in the aggregation method. Weights for parameter aggregation depend on the corresponding user's data size. More data on user k accounts for more contributions to update the shared global OCC-VAE model, as more data may train a more accurate model.

However, in each round, the selected users may be changed. Consequently, the size of the training data is diverse. The quality of their datasets cannot be guaranteed as well. Therefore, we improve the aggregation method by introducing the coefficient variable γ, for the trade-off of model updates. It is a time-driven weighted aggregation method. When aggregating the model updates from users, the server will consider the previous local model and current local

model based on their updated time. It can be presented by

$$
\begin{aligned}
w_{r+1} &\leftarrow (1-\gamma)\, w_r + \gamma \sum_{k=1}^{K} \frac{d_k}{d} w_{r+1}^k, \\
\gamma &= (e/2)^{-\left(t_r^k - t_r'^k\right)},
\end{aligned}
\tag{3.5}
$$

where e depicts the effect of time [92]. t_r^k represents the time of current round and $t_r'^k$ is the last time of the model was updated. For each FL communication round, the training data is different, because users may generate new data. The full algorithm is outlined in Algorithm 1.

3.3.2 One-Class Collaborative Variational Autoencoder

The communication cost affects the efficiency of FL training. In general, the number of users is large in mobile networks, but the communication bandwidth is limited. For example, the average upload speed of the Internet is 20Mbps [93], which cannot effectively support FL training with numerous users. The most intrinsic requirement for decreasing the communication costs of federated learning is to upload as little data as possible without deteriorating the performance of the shared model. Hence, we employ the 3LC data compression scheme [94] to compress the upload OCC-VAE model.

3LC is lossy transformation technique, which combines three successive techniques to compress a matrix of the parameters of the model, W_{in}. These three techniques are 3-value quantisation with sparsity multiplication, quartic encoding and zero-run encoding, which balances traffic reduction, accuracy and computation. The level of compression is tuned using sparsity multiplier s, which is a control knob of 3LC. First, W_{in} is sparsified and quantized to

Algorithm 1 ***EFPC***: K is the set of selected users, where $k \in K$.

Server Execution:

initialise w_0
for each round r = 1,2,... **do**:
  S_r : a set of selected users
  for each user $k \in K$ **in parallel do**:
    if k meet requirement of training model **then**
      add k to S_r
    end if
  end for
  Get the parameters of the global model w_r
  for each user $k \in S_r$ in parallel **do**:
    $w_{comp}^k \leftarrow$ **UserUpdate**(w_r, k)
    Decompress w_{comp}^k to achieve w_{r+1}^k
  end for
  $w_{r+1}^k \leftarrow \sum_{k=1}^{K} \frac{d_k}{d} w_{r+1}^k$
for end
Return w_{r+1}

User Execution:

Input: X, w_r, t_r^k, $t_r^{'k}$
UserUpdate(w,k)**:**
  for each local epoch i from 1 to E **do**
    for batch $b \in B$ **do**
      Compute parameters with gradient descent:
      $w_{r+1} \leftarrow w_r - \eta \nabla l\,(w_r; b)$
      $\gamma \leftarrow (e/2)^{-(t_r^k - t_r^{'k})}$
      $w_{r+1} \leftarrow (1-\gamma) w_r + \gamma \sum_{k=1}^{|S_r|} \frac{d_k}{d} w_{r+1}^k$
    end for
  end for
Compress w to generate w_{comp}
Return w_{comp}

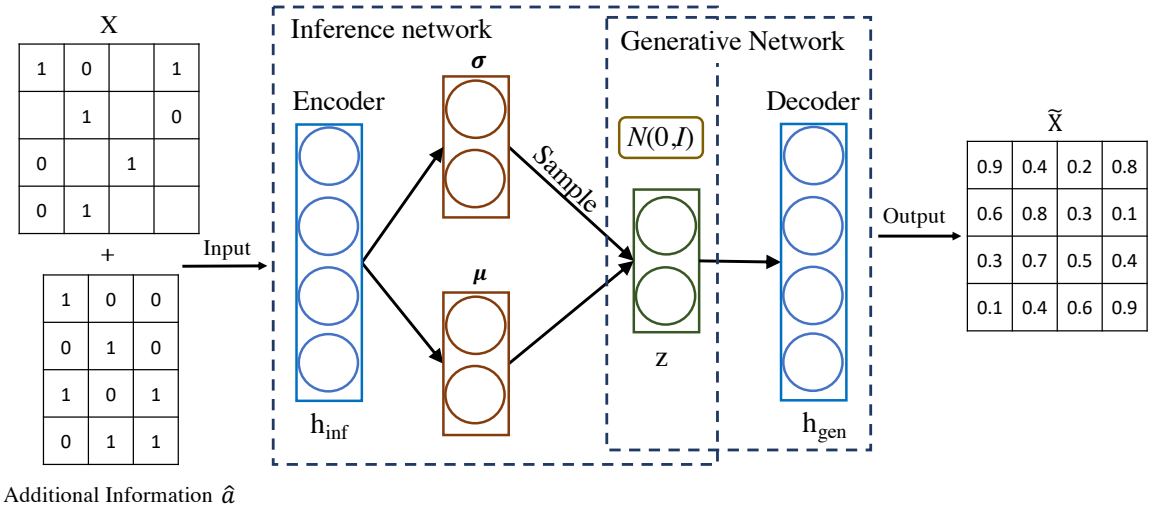


Fig. 3.2 Variational Autoencoder

the values $\{-1,0,1\}$ to produce W_{qe}:

$$\begin{aligned} G &= \max(|W_{in}|) \cdot s, \\ W_{qe} &= \text{round}(\frac{W_{in}}{G}). \end{aligned} \tag{3.6}$$

Error accumulation buffers is used to correct resulting quantisation errors. W_{qe} is then compressed using Quartic Encoding (QE), as it is a matrix of three distinct values which can be more efficiently encoded with 2 bits per value. In QE, W_{qe} is flattened and padded with zeros until its length is a multiple of five [94]. It is split into five parts of equal length. The 8-bit unsigned integer array, defined as *arr*, is computed. The input to quartic encoding is sparse, which contains a large number of zeros. Thus, the final step of 3LC is Zero Run Encoding (ZRE) that shortens consecutive runs of common bytes by using a variant of run-length encoding specialized for quartic encoded data. After ZRE, arr_{ZR} and G are sent to the server, where the reverse of the above processes are used to convert arr_{ZR} back to W_{qe}. The contribution of this method it to combine the strength of sparsification and tensor quantisation.

The OCC-VAE is proposed to predict content popularity ahead of time. VAE is a powerful unsupervised learning method, which aims to learn the data distribution $p(x)$ from the training set. Moreover, VAE can effectively cluster similar input data together in the latent space [95], which is naturally suitable for estimating content popularity. An overview of VAE is depicted in Fig. 3.2. An inference neural network $q(z \mid x)$ maps the input x to a distribution (*i.e.* Gaussian distribution) with estimating the latent variable z. A generative neural network $p(z \mid x)$ decodes the sampled latent variable z back into an observed data x.

In our setting, we employ the VAE to learn deep latent representations from user-by-content request matrix and implicit relationship between users and contents. The user-by-content request matrix X consists of samples of variable x_u^c, where $X \in \mathbb{N}^{U \times C}$, $1 < u < U$ and $1 < c < C$. u and c represent the index of users and requested contents, respectively. The $x_u = \left[x_u^1, ..., x_u^c\right]^T \in \mathbb{N}^C$ is a vector with the request number for each content from user u. Moreover, our proposed caching scheme is the context-aware content caching scheme. The users' content retrieval history and context (the personal information of users) are utilised to learn the context-specific content popularity, hence, the context of user $\hat{a}$ is appended to X.

However, the value of elements in X in our setting is only 1 or 0. The value of 1 represents the positive example of the user's interests. The value of 0 indicates an unknown positive or negative example. It is impossible for users to request all the contents. Thus, all negative examples and missing positive examples are mixed which makes it difficult to distinguish them [96]. For example, if we simply solve the issue by marking all the missing examples as negative, then the result may be inaccurate. The reason is that the negative examples may be included in the unknown positive examples. Therefore, we mark the negative examples in the input matrix X by the random sampling mechanism.

The probability of random sampling is related to the preference of users for contents. The modelling of random sampling is written as Eq. (3.7). The probability of content c as

requested by the user u is

$$Pr_{(u,c)} = r_1 \sum_{i=0}^{C} x_i^u + r_2 \frac{1}{\sum_{j=0}^{U} x_c^j}, \tag{3.7}$$

where $\sum_{i=0}^{C} x_i^u$ represents the number of contents requested by user u. $\sum_{j=0}^{U} x_c^j$ means how many times that content c has been requested by users. r_1,r_2 are coefficients.

In general, VAE assumes that for every $x_u \in X$, there are one or many settings of the latent variables $z_u \sim p(z_u)$ which causes the model to generate something very similar to x_u. Here, $p(z_u)$ is the probability distribution of z_u. Mathematically speaking the objective is to maximise the probability of each x_u in the input data under the generative process, which is formally defined as:

$$p(x_u) = \int p(x_u \mid z_u)\, p(z_u)\, dz_u. \tag{3.8}$$

In general, $p(x_u \mid z_u)$ is typically parameterised with a highly flexible function approximator such as neural networks. While both prior $p(z_u)$ and likelihood $p(x_u \mid z_u)$ can be formulated exactly, the posterior $p(z_u \mid x_u) = \frac{p(x_u,z_u)}{\int p(x_u,z_u)dz_u}$ needs an intractable integral over the latent space. Thus, instead of calculating the posterior $p(z_u \mid x_u)$, VAE takes an advantage of a parametrized variational approximation $q(z_u \mid x_u)$ to provide a distribution over the latent variables that are more likely to produce the input data x. This is done by minimizing the Kullback-Leibler (KL) divergence between $q(z_u \mid x_u)$ and $p(z_u \mid x_u)$:

$$\begin{aligned} &KL[q(z_u \mid x_u) \parallel p(z_u \mid x_u)] = \\ &\quad \mathbb{E}_{z_u \sim q(z_u|x_u)} [\log q(z_u \mid x_u) - \log p(z_u \mid x_u)]. \end{aligned} \tag{3.9}$$

By applying Bayesian inference to $p(z_u \mid x_u)$, we achieve

$$\begin{aligned} \mathrm{KL}\left[q(z_u \mid x_u) \parallel p(z_u \mid x_u)\right] &= \mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log \frac{q(z_u \mid x_u) p(x_u)}{p(x_u \mid z_u)}\right] \\ &= \mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log q\left(z_u \mid x_u\right)\right] + \mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u\right)\right] \\ &- \mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u \mid z_u\right)\right] - \mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(z_u\right)\right]. \end{aligned} \tag{3.10}$$

Then, to maximise $\mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u\right)\right]$, Eq. 5.3 can be rewritten as follow:

$$\begin{aligned} &\mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u\right)\right] - \mathrm{KL}\left[q(z_u \mid x_u) \parallel p(z_u \mid x_u)\right] = \\ &\mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u \mid z_u\right)\right] + \\ &\mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(z_u\right) - \log q\left(z_u \mid x_u\right)\right] \\ &= \mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u \mid z_u\right)\right] - \mathrm{KL}\left[q(z_u \mid x_u) \parallel p(z_u)\right]. \end{aligned} \tag{3.11}$$

The lower bound of $\mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u\right)\right]$ is as:

$$\begin{aligned} \log p\left(x_u\right) \geq & \mathbb{E}_{z_u \sim q(z_u|x_u)}\left[\log p\left(x_u \mid z_u\right)\right] \\ & - \mathrm{KL}\left[q\left(z_u \mid x_u\right) \parallel p\left(z_u\right)\right]. \end{aligned} \tag{3.12}$$

where the right hand-side is the variational lower bound of VAE. The approximate posterior $q\left(z_u \mid x_u\right)$ follows a Gaussian distribution $N\left(\mu, \mathrm{diag}\left(\sigma^2\right)\right)$ where μ is the mean and σ^2 is variance, which is represented by a neural network. The generative network $p\left(x_u \mid z_u\right)$ and inference network $q\left(z_u \mid x_u\right)$ are trained by maximising the variational lower bound with respect to their parameters. The reparameterisation trick [97] $z_u = \mu + \sigma \odot \varepsilon$ can be implemented to get the unbiased estimate of low variance bound. We suppose the mean and covariance are $\mu\left(x_u\right)$ and $\sigma\left(x_u\right)$, respectively. ε follows $N(0, I)$, the equation can be

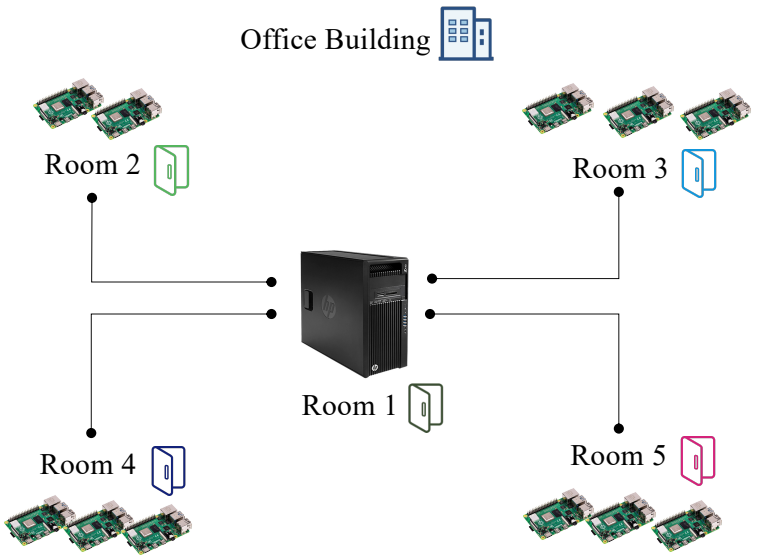


Fig. 3.3 EFPC Experiment Scenario

rewritten as follow:

$$\begin{aligned} &\mathbb{E}_{q(z_u|x_u)}\left[\log p\left(x_u \mid z_u\right)\right] = \\ &\qquad \mathbb{E}_{\varepsilon\sim N(0,I)}\left[\log p\left(x_u \mid z_u = \mu + \sigma \odot \varepsilon\right)\right], \end{aligned} \tag{3.13}$$

where ε is a vector sampled from standard Gaussian variables. With the help of the reparameterisation trick, the inference and generative networks can be trained through end-to-end backpropagation by SGD.

Therefore, we fed X with incomplete rows (resp. columns) into VAE to learn the latent representation Z. X can be recovered from Z, where the outputs are a matrix with predicting the missing entries. The highest score contents in outputs are the caching contents in the cache-enabled server.

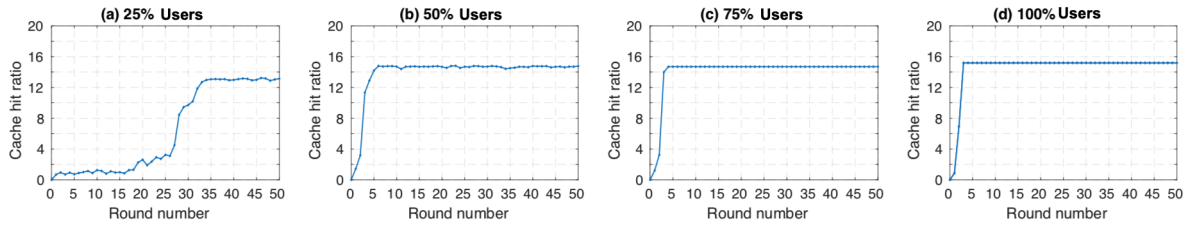


Fig. 3.4 Cache hit ratio vs. Different number of users (MovieLens)

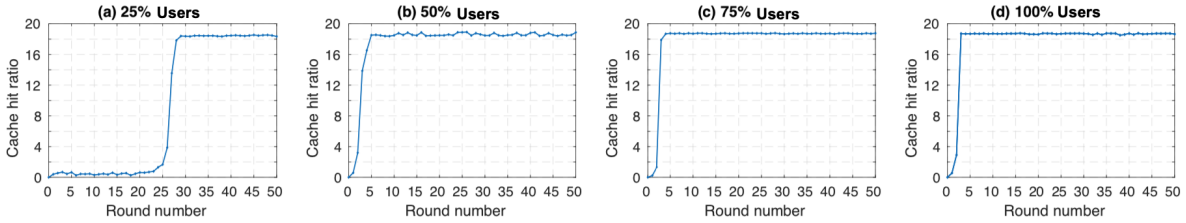


Fig. 3.5 Cache hit ratio vs. Different number of users (Netflix)

3.4 Experiments and Discussion

In this section, we evaluate our proposed FL based proactive content caching scheme using two real world datasets and compare its performance to four reference algorithms. We conduct experiments via a networking testbed with 10 user nodes.

3.4.1 Testbed

The networking testbed consists of one HP Z440 workstation with 64G memory and 10 Raspberry Pi devices (5 Raspberry Pi 3 Model B+ and 5 Raspberry Pi 2). As shown in Fig. 3.3, the workstation and 10 Raspberry Pi devices are located at five different locations in the office building. This represents a MEC environment we supposed. The workstation as the edge server is applied to aggregate the parameters of the OCC-VAE model, which is located in Room 1. All Raspberry Pi devices as mobile users are placed in other four rooms where the training of the OCC-VAE model is conducted [27]. Keras is employed as the framework of VAE with tensorflow as backend.

The datasets in our experiments are MovieLens 1M dataset and Netflix prize dataset. The data in the MovieLens dataset has been collected from the MovieLens website by

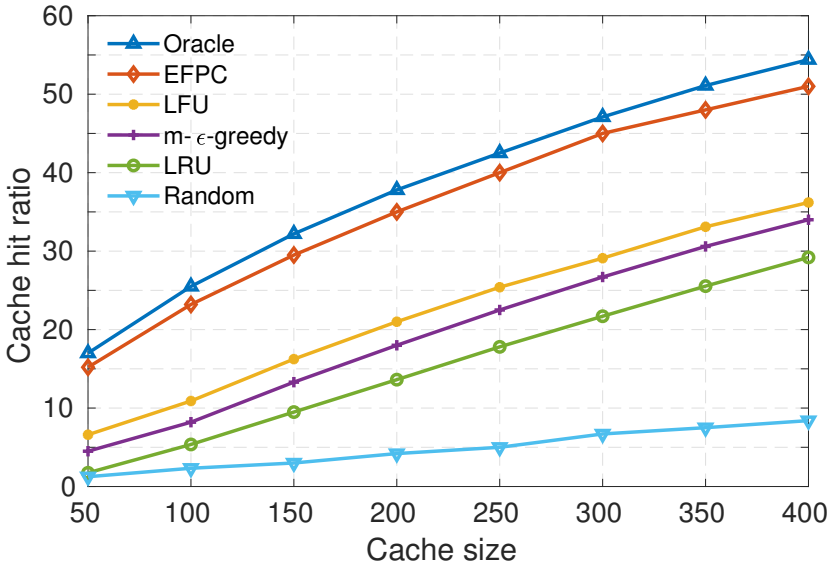


Fig. 3.6 EFPC vs. Other reference schemes (MovieLens Dataset)

groupLens research [98]. The MovieLens 1M dataset contains about 1 million ratings from 6040 anonymised users on 3883 movies. Each dataset entry consists of UserID, MovieID, Rating and Timestamp. The user information is also provided in the dataset, which includes gender, age and occupation. Netflix prize dataset [99] is released by Netflix. It consists of 17770 movies and 488000 users. Due to the limited computation capacity of Raspberry Pi and the large data size of this dataset, we only use a part of Netflix dataset (7000 movies and 4000 users) in our experiments. To simulate the process of mobile users' requests, we assume that the movie rating process in these two datasets is a content request process of users in MEC. The rated movies are the requested contents from users.

3.4.2 Performance Evaluation

As mentioned before, we applied the cache hit ratio to evaluate the performance of the caching scheme. It describes the percentage of users' requests can be directly transmitted by the edge server in BS. We compare EFPC with the following caching schemes: 1) Oracle: Oracle algorithm has prior knowledge about users' request in the future. It presents the best

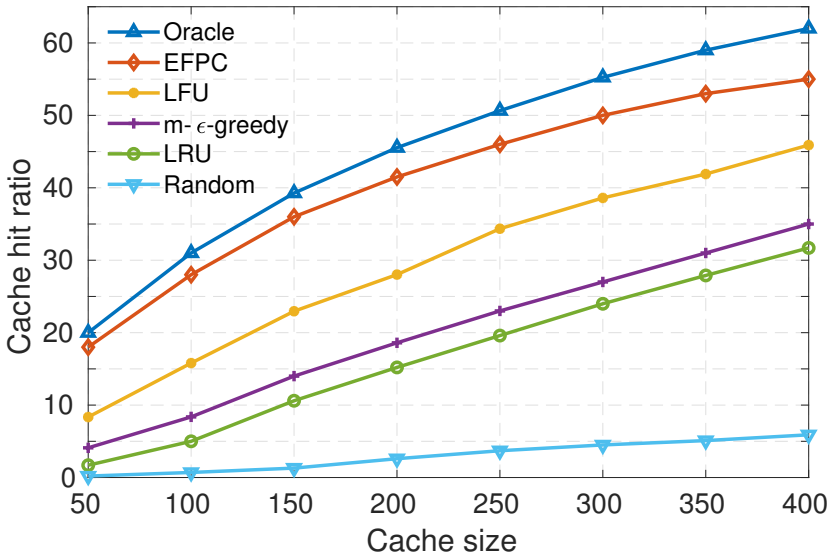


Fig. 3.7 EFPC vs. Other reference schemes (Netflix Dataset)

cache hit ratio. 2) Random: Random algorithm randomly chooses N contents from the library to store at cache, which provides the lowest cache hit ratio. 3) m-ε-Greedy: m-ε-Greedy algorithm is an extension of the simple ε-Greedy algorithm, which is one of the multi-armed bandit algorithms. With a probability of (1 - ε), m contents with most frequently accessed will be selected as the caching contents, while with the probability ε ($0 < \varepsilon < 1$), m contents will be randomly picked from the content library as the caching contents. 4) Least Frequently Used (LFU): LFU keeps tracking the number of times that a content has been requested and replaces the caching content based on the historical request frequency of cache content. If a content has been requested multiple times in the past, the frequency of this requested content may be higher in the future. LFU evicts the least popular content. 5) Least Recently Used (LRU): When the caching storage is limited, LRU removes the content based upon the time of usage. The content will be replaced if it has not been requested for a long time. If the content has not been requested in the most recent period of time, it may not be requested in the future as well. 6) AutoEncoder: AutoEncoder, as an unsupervised learning model, trains an one-hidden layer neural network to reconstruct input data from the latent representation, which can copy its input to its output.

We investigate the cache hit ratio against the number of federated communication rounds with a different number of selected users on two datasets (MovieLens and Netflix). For both datasets, they exhibits the same trend. For the MovieLens dataset, Fig. 3.4 shows that when the number of communication round is 50, the cache hit ratio reaches 12.8%, 14.7%, 14.9% and 15.2% with 25%, 50%, 75%, and 100% selected users who attend to the FL training, respectively. We observe that more communication rounds are needed to achieve the target cache hit ratio with fewer users. It is because few users can only provide less data. The target cache hit ratio 15% can be achieved after 6 rounds, 5 rounds and 3 rounds for 50%, 75%, and 100% selected users, while 25% selected users participate in the federated optimisation is unable to reach the target value. The results indicate that the more selected users in FL deep learning training, the higher quality model will be trained within fewer communication rounds. If the size of the training data is larger for each selected user, fewer users are needed to achieve the same cache hit ratio. Additionally, the number of selected users influence the cache hit ratio. As we can see from Fig. 3.4(a), if only 25% users participate in the federated optimisation, the highest cache hit ratio is 13% that is lower than the target cache hit ratio. On the other hand, Fig. 3.5 presents the target cache hit ratio 18% can be obtained after 29 rounds, 5 rounds ,4 rounds and 3 rounds on the Netflix dataset, respectively. It follows a very similar pattern to MovieLens. Thus, in order to train high quality model, maintaining a certain number of selected users in the federated optimisation is necessary.

Fig. 3.6 and Fig. 3.7 depict the cache hit ratio for varying cache sizes between 50 and 400 contents. As shown in Fig. 3.6, the overall cache hit ratios of all algorithms rise with increasing cache size. As expected, Oracle has the perfect prior knowledge about the user demands in future that gives an upper bound to the other algorithms. Whereas, random provides the worst cache hit ratio which is the lower bound. The cache performance of our proposed EFPC and AutoEncoder based caching scheme outperform LFU, m-ε-Greedy and LRU caching schemes, as they learn the latent relationship between users and contents to

Table 3.2 The comparison of 3LC data compression scheme and no data compression in data transmission with different sparsity

Sparsity	**Data Compression** (MB)		**No Data Compression** (MB)		**Ratio**
	Data per round	Tol Data	Data per round	Tol Data	
1	0.199	0.995	4	16	20.04×
1.1	0.199	0.995	4	16	20.13×
1.2	0.197	0.788	4	12	20.28×
1.3	0.193	0.965	4	16	20.72×
1.4	0.186	0.93	4	16	21.51×
1.5	0.173	0.692	4	16	23.14×
1.6	0.149	0.745	4	12	26.94×
1.7	0.124	0.496	4	12	32.26×
1.8	0.085	0.255	4	12	46.76×
1.9	0.036	0.18	4	16	111.02×
2	0.014	0.056	4	16	279.65×

predict content popularity. The EFPC shows a better performance compared to AutoEncoder, because EFPC clusters the request of users in the latent space. The results also indicate that EFPC, Autoencoder, LFU and m-ε-Greedy achieve higher cache hit ratio than LRU and Random. The reason is that all these four caching schemes make use of historical requests of users. LFU obtains better performance than m-ε-Greedy, because m-ε-Greedy randomly chooses the caching content with the probability of ε which may lead to the low cache hit. Besides, the order of users' requests influences the cache hit ratio of LRU and LFU caching scheme as they make caching decision by observing local recent user request patterns. We evaluate all caching schemes on two datasets (MoviesLens and Netflix). For the Netflix prize dataset, it exhibits the same trend as MovieLens, which shows an upward trend towards cache hit ratio with the increasing of cache sizes. The cache hit ratio of EFPC is higher than LFU, m-ε-Greedy, LRU and Random, but lower than the Oracle. The cache hit ratio for Netflix dataset is different to MovieLens due to the different content number, user number and the sparsity of dataset.

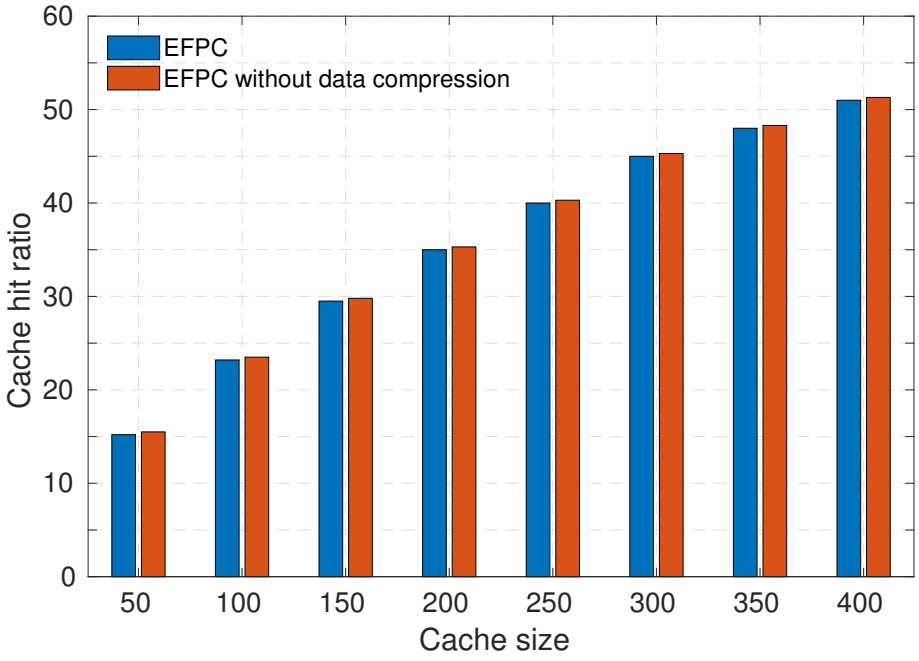


Fig. 3.8 Data Compression vs. Without Data Compression

Fig. 3.8 indicates the influence of the data compression method to the cache hit ratio, since 3LC is a lossy transformation technique. The experiment results compare the impact of varying cache size on cache hit ratio with data compression method and without data compression. As we can see from Fig. 3.8, the achieved cache hit ratio for EFPC and EFPC without data compression are similar. EFPC without data compression slightly outperforms EFPC. When the cache size is 50, the average cache hit ratio of 10 users with EFPC achieves 15.3%, while EFPC without data compression get the cache hit ratio of 15.6%. For the other cache sizes, the same trend has been shown. This experiment results demonstrate the 3LC data compression scheme keeps high accuracy during the OCC-VAE model training.

Fig. 3.9 investigates the relationship between data compression ratio, sparsity multiplier and cache hit ratio. Sparsity multiplier s represents the compression level of 3LC data compression scheme. It shows the data compression ratio rises with the increasing sparsity multiplier value. In the beginning, the data compression ratio grows slowly. When the sparsity multiplier varies between 1 and 1.6, the data compression ratios are similar, around

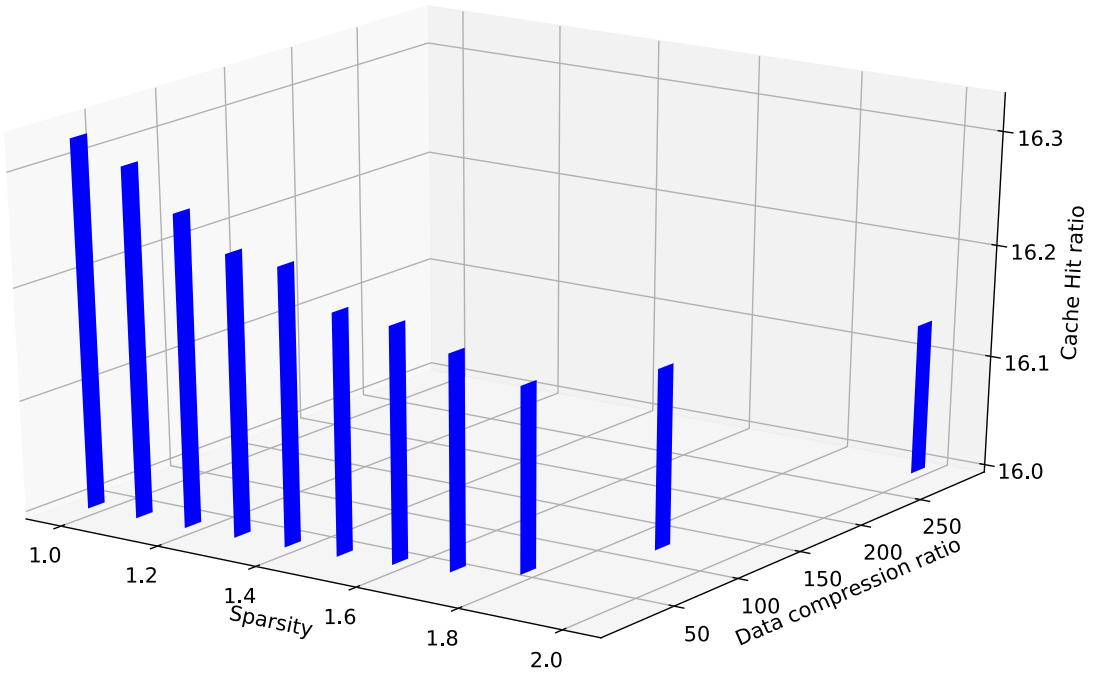


Fig. 3.9 Sparsity vs. Data compression ratio vs. Cache hit ratio

25×. It grows faster and faster when the value of sparsity multiplier is larger than 1.6. It reveals that a high sparsity multiplier makes a high data compression ratio. Fig. 3.9 also compares trade-offs between data compression ratio and cache hit ratio. With the increase of data compression level, the data compression ratio is grown. Meanwhile, the cache hit ratios are almost the same, but it has a trend of slightly decreasing. When the value of sparsity multiplier is equal to 2, the cache hit ratio is 16.1% that is lower than the corresponding result of sparsity multiplier is 1, but the cache performance of the proposed scheme still outperforms other reference caching schemes.

Table 3.2 further illustrates the 3LC data compression scheme in our proposed caching scheme reduces the transmission data by reducing the size of transmission data while preserving a high cache hit ratio. As the results are shown in Table 3.2, without data compression, the original transmission data is 4 Mbps for each communication round. After data compression, the size can be reduced from 4 Mbps to 0.19-0.01 Mbps for per communication round. The total transmission data size decreases from 16 Mbps to 0.099-

0.056 Mbps without increasing the number of communication round. At the same time, the data compression ratio can be achieved from 20.04 $\times$ to 279.65 $\times$, but the accuracy of FL model may decrease.

3.5 Summary

In this chapter, a proactive caching scheme named EFPC based on federated learning framework for multi-access edge networks is proposed, aiming to improve cache performance, protect users' privacy as well as reducing communication costs. We utilise a federated learning based variational autoencoder approach to estimating content popularity in the future and then placing the most predicted popular contents to the edge server, to increase cache hit ratio and reduce the risk of privacy disclosure. Moreover, we employ a 3LC data compression scheme to effectively decrease the amount of transmission data in federated learning to further reduce communication costs. We have carried out the experiments via a networking testbed to verify the effectiveness of our proposed EPAC caching scheme. Numerical results indicate that EPAC outperforms LRU, LFU and m-ε-Greedy in terms of the cache hit ratio. The 3LC data compression scheme in our proposed model achieves a data compression ratio of up to 279$\times$ with almost the same cache hit ratio of no data compression.

Chapter 4

Mobility-Aware Proactive Edge Caching for Connected Vehicles

4.1 Introduction

With the advancement in wireless communications and Internet-of-Things (IoT), self-driving has been considered as a key enabling technology in Intelligent Transportation Systems (ITS) to decrease traffic congestion, improve traffic efficiency and enhance road safety [100]. Self-driving vehicles enable a wide range of applications, from infotainment applications to safety-related applications [101]. These applications may require large computation, communication and storage resources, and have strict performance requirements on network bandwidth and response time. Thus, supporting these applications imposes high pressure on the resource-constrained Vehicular Networks (VNs). Vehicular Edge Computing (VEC) is recognised as a promising paradigm to satisfy the increasing demands by integrating edge computing into VNs [101]. VEC allows data to be processed and stored at edge nodes, such as Roadside Units (RSUs) and Base Stations (BSs).

Caching content at edge nodes enables vehicles to fetch their requested contents within one transmission hop [102]. It is capable of reducing service latency and alleviating backhaul

network burden. Due to the limited storage at edge nodes, the caching schemes need to identify and cache the popular contents that are interesting to most vehicular users. However, the high mobility of vehicles and complex vehicular environments cause highly dynamic content popularity. In this case, the previously requested contents may become obsolete soon, so the reactive caching scheme cannot satisfy strict performance requirements of users. Proactive caching predicts content popularity and caches predicted popular contents before the arrival of user requests. It can pre-fetch the popular contents, even these contents may have never been requested before. Thus, proactive caching is considered to be more suitable for the VEC scenarios. In proactive caching, Machine Learning (ML) is a powerful approach to predict content popularity for efficient caching. Some works focus on learning-based caching schemes in VNs by utilising reinforcement learning [103], [87], multilayer perceptron and convolutional neural networks [10], etc.

Although some progresses have been achieved in learning-based proactive caching, utilising ML techniques for edge caching in VNs still faces the following three challenges: 1) High mobility: Vehicles send requests to an RSU and go through its coverage area quickly, making the caching content easily to be out of date. To improve the cache performance, the caching scheme should be both context and mobility aware, making cache decisions based on the content popularity predictions and vehicles' mobility. 2) Privacy: Most ML algorithms train models in a centralised manner where the data generated by multiple vehicles must be sent to an edge server in RSU for analysis. These generated data may involve personal sensitive information used for various vehicular applications. Therefore, uploading and processing these data centrally may raise privacy and security concerns. 3) Scalability: As the number of connected vehicles grows, data generated by the vehicles increase. The centralised ML algorithms may find it difficult to handle such data due to the incurred high computation and communication costs.

A Mobility-aware Proactive Edge Caching Scheme based on Federated Learning (MPCF) is proposed. MPCF utilises Context-aware Adversarial AutoEncoder (C-AAE) to predict the content popularity and then caches the predicted popular contents in RSUs. Our proposed proactive caching scheme is based on the Federated Learning (FL) framework [26]. In the designed scheme, vehicles collect and store data for local training. A global model (*i.e.*, C-AAE) is updated at RSU by aggregating the locally trained models. Moreover, a mobility-aware cache replacement policy is developed to dynamically update cached contents according to the mobility and position information of vehicles.

The main contributions of this chapter are summarized as follows.

1. We propose a mobility-aware federated learning scheme for edge caching in VNs, which can protect users' privacy, reduce communication costs, and support high mobility of vehicles. This new scheme includes four main components: content popularity prediction, vehicle selection, model aggregation, and cache replacement.

2. We utilise the C-AAE model to predict the popularity of contents, which adds the adversarial network to the AutoEnocoder (AE) architecture by turning an AE into a generative model. It helps to learn deep latent representations from users' historical requests and contextual information, and obtain implicit relationships between users and contents for improving prediction accuracy.

3. We design mobility-aware vehicle selection, model aggregation, and cache replacement policies with the aim of optimising the caching resource utilisation in VNs. Especially, the decision for selecting vehicles to participate in the FL training process and the value of weights for parameter aggregation depend on the current positions and local resources of connected vehicles. It can ensure that vehicles have enough time for training and the RSU can aggregate high-quality updated models. Meanwhile, the cache replacement policy dynamically updates the contents at RSUs in response to the content requests from their connected vehicles and predictions of content popularity.

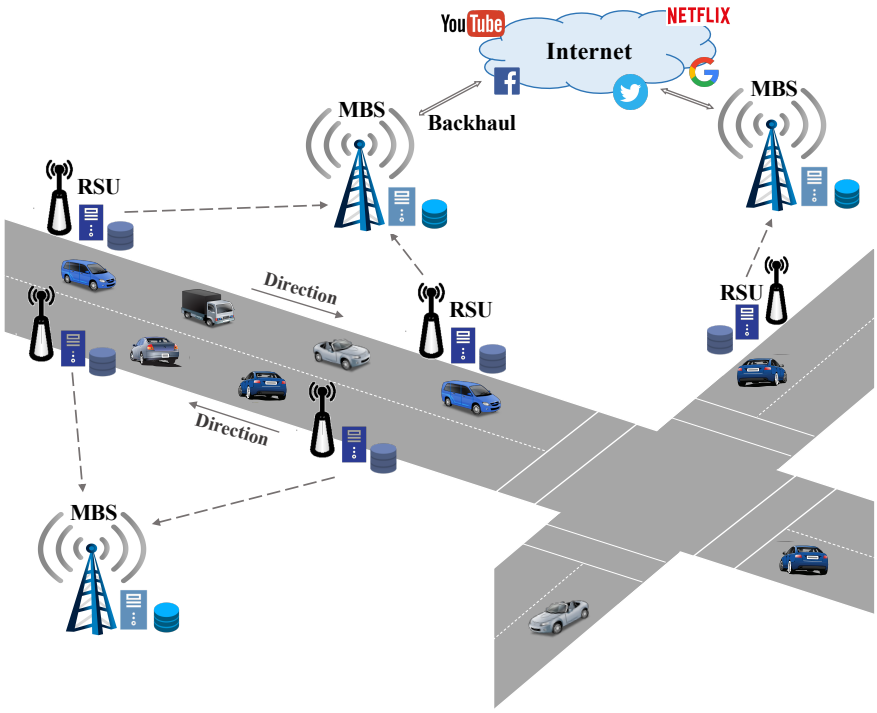


Fig. 4.1 Proactive edge caching for connected vehicles

4. We numerically evaluate our proposed caching scheme by using real-world datasets (MovieLens). The experimental results show that our scheme outperforms reference algorithms for connected vehicles, such as LRU and LFU.

The rest of this chapter is organised as follows: The system architecture of the proposed cache scheme is presented in Section 4.2. Section 4.3 describes the detailed implementation of the MPCF. The performance evaluation and analysis of MPCF are provided in Section 4.4. Section 4.5 concludes this chapter.

4.2 System Architecture

We consider a vehicular network in an urban scenario, consisting of several MBSs, RSUs and vehicles, as shown in Fig. 6.1. The MBSs are located in different locations at the edge of VNs. Within the coverage area of an MBS, a set of RSUs $S = \{S_1, S_2, S_3, ..., S_n\}$ are

placed equidistantly with distance $\mathfrak{D}$ over both sides of the road, where n is the number of RSUs. Each RSU serves its connected vehicles. These vehicles are denoted by a set $V = \{V_1, V_2, V_3, ..., V_m\}$, where m is the number of connected vehicles. Vehicles traverse the coverage areas of several MBSs. The communication among vehicles, RSUs and MBSs are through wireless links, while MBSs connect to the Internet via a reliable backhaul link. Both MBSs and RSUs are equipped with cache-enabled edge servers. RSUs are used to cache contents likely to be requested by the vehicles nearby. MBSs store the lists of cached contents in connected RSUs and manage their cache resources.

The speeds of vehicles are assumed to be independent and identically distributed, forming a set $U = \{U_1, U_2, U_3, ..., U_m\}$. They are generated by a truncated Gaussian distribution. Compared to the normal Gaussian distribution or a fixed speed, the truncated Gaussian distribution is more feasible for modelling vehicles' speed because it limits the scope of vehicles' speed to a certain range. This assumption has also been widely used in many state-of-the-art works of vehicular networks [84], [104], [105]. Vehicles keep their assigned speeds invariable during each experiment. There is an entrance RSU on each side of the road. The number of arrived vehicles for entering each entrance during the period t is defined as $V_q(t)$. It follows a Poisson process with the parameter λ:

$$P\left(V_q(t) = g\right) = \frac{(\lambda t)^g}{g!} e^{-\lambda t}, \tag{4.1}$$

where g equals the number of vehicles generated in a period t.

These entered vehicles are interested in a set of popular contents. In the concerned scenario, each RSU can prefetch up to N contents from the Internet and cache these contents locally. The moving vehicles can connect to an RSU and send content requests to it, when vehicles are located within the area of the RSU. If the requested content is available in the current connected RSU (*i.e.*, a cache hit), the RSU can directly transmit this content to the

vehicle. Otherwise, the RSU has to obtain the requested content from the Internet (*i.e.*, a cache miss).

Placing the popular caching contents at RSUs can effectively improve cache performance, which primarily depends on the knowledge of content popularity. The popularity of contents is influenced by many factors, including the contextual information of vehicular users (*e.g.*, age and gender) and the mobility pattern of vehicles. Thus, to enhance the cache performance, predicting the content popularity and deciding which contents to be cached in RSUs need to consider the above information.

To address the above challenge, we design a mobility-aware proactive content caching scheme for connected vehicles using FL. As shown in Fig. 6.1, it is a three-layered architecture. The bottom layer contains vehicles requesting for contents. The middle layer includes several RSUs equipped with cache-enabled edge servers. The top layer has a cache-enabled MBS. The multiple connected vehicles in an RSU collaboratively train a shared global learning model. The RSU firstly disseminates an initial global model to the connected vehicles. Based on the received model, vehicles utilise their local data to compute an updated model. Next, each vehicle sends the updates of global model back to the RSU. Finally, the RSU aggregates the updates from vehicles and builds an updated global model. The above steps are repeated until a satisfying global model is achieved, which the outputs remain stable. The learning model in this work is specially developed to predict content popularity by learning data representation from the data of local vehicles. We rank all contents by their predicted popularity and select the top N popular contents as caching contents in the RSU. Meanwhile, the name list of cached contents in each RSU is stored at the MBS.

The federated deep learning model in RSU uses the data from current connected vehicles to predict the content popularity and prefetches their predicted results in the cache. However, the high mobility characteristic of vehicles may result in the following situation: Vehicles send content requests to the current RSU, but trying to fetch the requested contents from

another RSU. Furthermore, due to the small coverage area of the RSU, vehicles may not have enough time to download the whole requested content. It may pass several RSUs to obtain the full content. Therefore, a mobility-aware cache replacement policy is developed to address these issues. Based on the prior knowledge of vehicle trajectories, predicted content popularity and lists of cached contents in MBS, the MBS dynamically updates the caching contents of each RSU. This policy enables the predicted contents of vehicles to serve themselves. In other words, the predicted popular contents can follow their movement. When the vehicle is going to leave the current RSU and enter to the next RSU, the MBS will cache the popular contents for the vehicle in the neighbour RSU that it will enter. Due to the similar locations of these vehicles, cached contents in RSUs have less geographical features. Thus, the predicted popular contents for each RSU may not vary much, since the similar places have similar content popularity. In this case, only a small number of contents need to be replaced during this cache replacement.

4.3 Mobility-aware Federated Learning for Edge Caching

This section elaborates on our proposed caching scheme. We first describe the mobility-aware federated deep learning framework which includes the connected vehicle selection, federated training process and weighted aggregation method. Then, we introduce the context-aware adversarial autoencoders based method to predict the popularity of contents. Finally, with the prediction of the content popularity and a coarse knowledge of vehicle trajectories for self-driving vehicles, we explore a mobility-aware replacement cache policy. Table 4.1 lists the definition of notations in the MPCF.

4.3.1 Mobility-aware Federated Deep Learning

FL facilitates collaborative training of a deep neural network model among vehicles under the orchestration of a server in RSU by keeping the training data on vehicles. It significantly mitigates the privacy risk of vehicles and largely reduces communication costs, resulting from centralised ML [26]. FL is performed by multiple communication rounds (iterations). Based on the speed and position of vehicles, K vehicles are selected at each communication round to conduct model training, as shown in Fig. 4.2. The K vehicles are indexed by k. Then, each vehicle receives a global model from the RSU and trains this model from its local data. Following the local training at vehicles, the updated weights and gradients are sent back to the RSU. The RSU aggregates the collected models from vehicles to construct an updated global model. Finally, according to the predicted content popularity by the updated global model and the coarse knowledge of vehicle trajectories, the cache replacement strategy will decide where and which contents to be cached.

The details of our designed FL communication round consists of the following steps:

Vehicle Selection

Due to the small coverage area of an RSU, some vehicles with high-mobility may go though quickly and cannot finish the FL training. It leads to train an inefficient model and deteriorates the cache performance [106]. Aggregating high-quality updated models of vehicles on the RSU server can construct a more accurate global model. Thus, we design a mobility-aware vehicle selection method to cope with high-mobility training environment. Only the RSU located at the road entrance is chosen to execute the FL training. A set of its connected vehicles are selected as nodes performing computation on their local data to update the global model. The vehicle selecting process will consider the factors of good channel condition, unmetered wi-fi stable connectivity, sufficient local training data and a long standing time in the current RSU's coverage area [25]. Sufficient local training data guarantees to train a

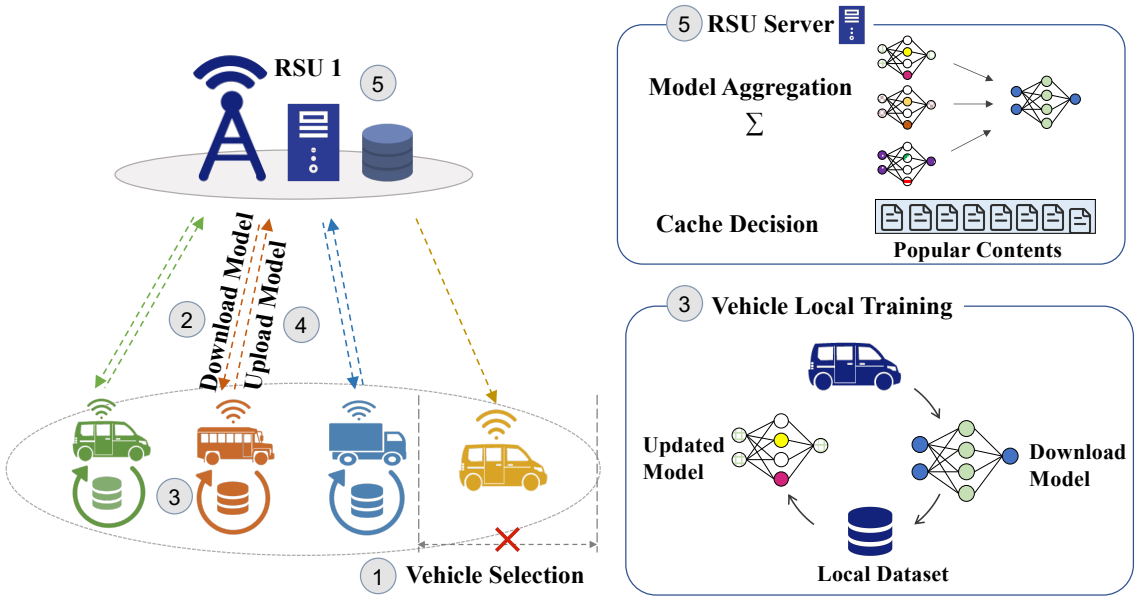


Fig. 4.2 Mobility-aware Federated Deep Learning

high-quality model. The standing time is the driving time of the vehicle staying in the area of RSU. It largely depends on the position and speed of connected vehicles. The long standing time in the coverage area promises that the training process can be completed and its results can be delivered.

U_k denotes the speed of the k-th connected vehicle, which is a constant with the minimum and maximum speed $(U_{min} \leq U_k \leq U_{max})$ in the urban area. We suppose that U_k follows a truncated Gaussian distribution [84]:

$$f(U_k) = \begin{cases} \frac{e^{-\frac{1}{2\sigma^2}(U_k-\mu)^2}}{\sqrt{2\pi\sigma^2}\left(erf\left(\frac{U_{max}-\mu}{\sigma\sqrt{2}}\right)-erf\left(\frac{U_{min}-\mu}{\sigma\sqrt{2}}\right)\right)}, \\ \qquad\qquad U_{min} \leq U_k \leq U_{max}, \\ 0, \qquad\qquad \text{otherwise}, \end{cases} \tag{4.2}$$

where σ^2 is the variance and μ $(-\infty < \mu < \infty)$ is the mean, $erf()$ is the Gauss error function. P_k is the position of the k-th vehicle when the FL training starts, which represents the distance to the entrance. The diameter of coverage area of RSU s is $\mathfrak{D}$. Thus, for each vehicle, the

standing time in the coverage area of current RSU is:

$$T_{standing}^{k} = (\mathfrak{D} - P_k) / U_k \tag{4.3}$$

We assume that the average training time for each communication round is T_{round} and the inference time is T_{inf}. (They depend on the size of dataset and the deep learning model. In this work, they are sampled from our experiments, instead of estimating inference time.) As shown in Fig. 4.2, step (1), if the $T_{standing}^{k} > T_{round} + T_{inf}$, the vehicle will meet the requirement for standing time and is chosen for FL training.

Model Download

A set of vehicles K_r are selected to participate in FL training for the r-th communication round. The next step in the typical FL training process is that selected vehicles download the global model from the RSU and train this model over their own local data, see Fig. 4.2, step (2). In self-driving scenario, the MBS has a coarse knowledge of vehicle trajectories. Thus, the MBS allows some vehicles to directly use their own models downloaded from the previous RSU to participate in the currently FL training. The previous connected RSU of these vehicles is located next to the current connected RSU with the same driving direction. Using the previous model brings the benefits to accelerate the training of a global shared model, as it trains based on a high-quality model and the training time can be greatly saved. In this way, it also considers preferences of the vehicles connected to the previous RSU. They may enter the current connected RSU with high probability. Considering the preferences of future coming vehicles that calculated by the previous RSU can help to train a mobility-aware model. For other selected vehicles, they perform the typical FL training process. They download the parameters w_r of the global model from the RSU.

Table 4.1 Notations definition in MPCF

Notation	Definition
S	Set of RSUs
n	Number of RSUs
V	Set of connected vehicles for the RSU
m	Number of connected vehicles
U	Set of vehicles' speed
N	Maximum number of caching contents in RSU
λ	Rate parameter of Poisson process
K	Set of selected vehicles for FL training
k	Index of selected vehicles for FL training
H	Set of datasets stored in selected vehicles
d_k	Size of the local dataset in the connected vehicle k
d	Size of datasets among connected vehicles
P_k	Position of the connected vehicle k
$\mathfrak{D}$	Diameter of the coverage area of RSU (Distance between RSUs)
$T^k_{standing}$	Standing time of the connected vehicle k
r	Number of FL communication rounds
T_{round}	Average training time for each communication round
T_{inf}	Inference time
w	Parameters of model
w_r	Parameters of model in the r^{th} round
$L_k(w)$	FL loss function of the selected vehicle k
Q	Position of RSU
c	Number of contents
b	Local minibatch size
η	Learning rate
D	Discriminative model
G	Generative model
X	User-by-content request matrix
x	Sample from X
$\hat{a}$	Users' context matrix
$\tilde{X}$	Reconstructed user-by-content request matrix
z	Latent code to reconstruct input to output
z'	Real input with the required distribution
$p(x)$	Model distribution
$p_d(x)$	Data distribution
$p(z)$	Prior distribution
$q(z \mid x)$	Encoding distribution
$p(x \mid z)$	Decoding distribution

FL Model Training

The third step in our proposed FL is to train the model by utilising local data at vehicles, as shown in Fig. 4.2, step (3). Let $H = \{H_1, H_2, H_3, ..., H_k\}$ represent the datasets stored in selected vehicles. H_k represents the local dataset of the k-th vehicle with the length d_k, $d_k = |H_k|$. d is the size of the whole data among the selected vehicles, which can be calculated by the connected RSU. Each selected vehicle will report their data size to the RSU during the vehicle selection process. Similar to the typical FL, the goal of our proposed FL is to minimise the loss function $\ell(w)$:

$$\min_{w} \ell(w) = \sum_{k=1}^{K} \frac{d_k}{d} L_k(w) \quad \text{where} \quad L_k(w) = \frac{1}{n_k} \sum_{j \in H_k} \ell_j(w), \tag{4.4}$$

where $\ell_j(w)$ is the loss of the prediction on the j-th dataset in H with the parameters of model w. k is the index of total selected vehicles K. $L_k(w)$ represents the local loss function of vehicle k. Minimising the weighted average of local loss function $L_k(w)$ is equal to optimise the loss function $\ell(w)$ of FL.

Upload Updated Model

As shown in Fig. 4.2, step (4), the fourth step is to upload the local model w_{t+1}^k from vehicles to the RSU server. Compared to the computation costs, communication costs dominate in FL [8]. In order to reduce communication costs and save the upload time, the model can be compressed before being uploaded to the RSU, as the uplink speed is slower than the download speed [107].

Weighted Aggregation

After vehicles upload their models, the fifth step is to generate the new global model w_{r+1} by computing a weighted average of all received local models w_{r+1}^k, as shown in Fig. 4.2,

step (5). The new constructed global model is used for the next training round. r denotes the communication rounds in FL. Federated Averaging (FedAVG) algorithm is widely applied in FL. Compared to the typical federated stochastic gradient descent algorithm (FedSGD) [8], it increases local training epochs as well as decreases mini-batch sizes. In FedSGD, each vehicle k utilises its own data to locally compute the average gradient $\triangledown L_k(w)$ on its global model w_r which is downloaded from the RSU. The RSU server then aggregates these computed gradients by taking a weighted average sum and applies the update gradients:

$$w_{r+1} \leftarrow w_r - \eta \sum_{k=1}^{K} \frac{d_k}{d} w_{r+1}^k, \tag{4.5}$$

where η is the fixed learning rate. In FedAVG, each vehicle adds more computation by iterating the local updates ($w_r^k \leftarrow w_r^k - \eta \triangledown L_k\left(w_r^k\right)$) for multiple times before the averaging step in the RSU server. The optimization strategy will be updated in each iteration. During the iteration progresses, the optimal strategy will gradually be obtained. The weighted averaging algorithm is implemented to aggregate the model. Weights for parameter aggregation are dependent on the position of the connected vehicle, which is $\gamma_k = P_k/\mathfrak{D}$, and the size of local training data. Then, we can re-write the aggregate method as

$$w_{r+1} \leftarrow w_r - \eta \sum_{k=1}^{K} \gamma_k \frac{d_k}{d} w_{r+1}^k. \tag{4.6}$$

Selected vehicles with a longer available training time account for more contributions and are given greater weight in model aggregation.

Additionally, after training a shared global model, each RSU predicts its popular contents and then sends a list to the MBS. The MBS stores all lists of cached contents for replacing contents in the future. The above process is repeated. The full algorithm is outlined in Algorithm 2 and 3.

4.3.2 Contextual-aware Adversarial Autoencoders for Prediction

The model we trained in the above FL framework is the Contextual-aware Adversarial Autoencoders (C-AAE) model. It is used to predict the popularity of contents for proactive edge caching purpose. Adversarial Autoencoders (AAE) is a probabilistic AutoEncoder (AE), combining Generative Adversarial Networks(GANs) and Variational Autoencoders (VAE) [108]. The architecture of a C-AAE is shown in Fig. 4.3. The top row is an AE. It is able to learn a latent code z to replicate its input X to its output $\tilde{X}$ in an unsupervised learning manner [109]. The bottom row is an adversarial network that utilises two neural networks by pitting one against the other. It is mainly used to distinguish whether the sample draws from the specified distribution of the user or the sample comes from the latent code z of the AE. The input matrix consists of content retrieval history by vehicular users, named as the user-by-content request matrix X. It consists of samples of variable x, where $X \in \mathbb{N}^{m \times c}$. m and c stand for the number of connected vehicles and contents, respectively. Moreover, which contents will be requested in the future may depend on the vehicular users' context. The contextual information of connected vehicular users $\hat{a}$ is used in our proposed method, in order to predict the popularity of context-specific contents. $\hat{a}$ is appended to X. The output of C-AAE is $\tilde{X}$, which is the reconstructed inputs filling in prediction values.

C-AAE adds the GAN to the AE architecture by turning an AE into a generative model. It trains an AE with an adversarial loss, which can adapt the distribution of latent space to an arbitrary prior. GANs build two neural networks: the generative model G and the discriminative model D. G uses a vector of random numbers as the inputs and generates the outputs. D is utilised to differentiate between a sample generated from the G and a sample is taken from the input data. The min-max game of GAN between a generative model G and a discriminative model D can be expressed as follows [108]:

$$\min_{G} \max_{D} \mathrm{E}_{x \sim p_d(x)} \left[\log D(x) \right] + \mathrm{E}_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right]. \tag{4.7}$$

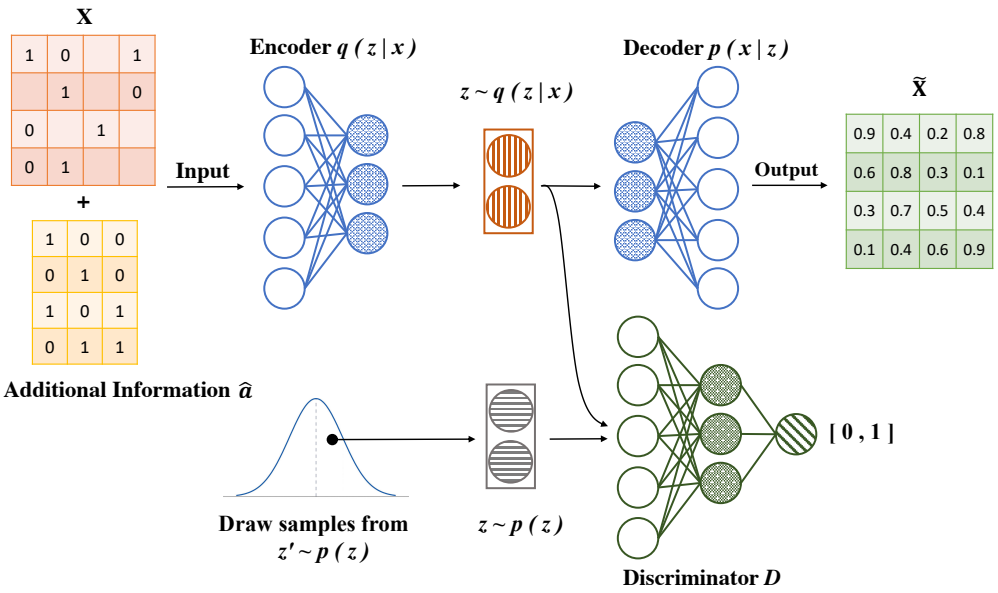


Fig. 4.3 C-AAE model architecture

The reconstruction phase and regularization phase are two phases in the training process of C-AAE. In the phase of reconstruction, an AE is to update the encoder and encoder is to minimise reconstruction error of input x. Firstly, z is generated by the generator network $q(z \mid x)$, an encoding distribution. Next, z is feed to the decoder and the output $\tilde{x}$ is reconstructed from z. The loss of reconstruction is calculated between x and $\tilde{x}$. In the phase of regularization, discriminator D is firstly updated by the adversarial network to distinguish true prior samples from generated samples. Then, in order to fool the discriminator D, the generator G is updated. The discriminative network considers the hidden code, which is distributed as the true prior distribution $p(z)$.

The output is imposed to the encoder by an adversarial network to follow the distribution of $p(z)$. z can be obtained by the discriminator, while z' is sampled by $p(z)$. Backpropagation is applied to adjust the weights of discriminator and the parameters of generator are updated at the same time. This process is repeated. The generative model is defined by the decoder of the AE. The imposed prior of $p(z)$ is mapped to the data distribution $p_d(x)$. Thus, the regularisation of C-AAE can be achieved by matching $q(z)$ to $p(z)$, where $q(z)$ is the

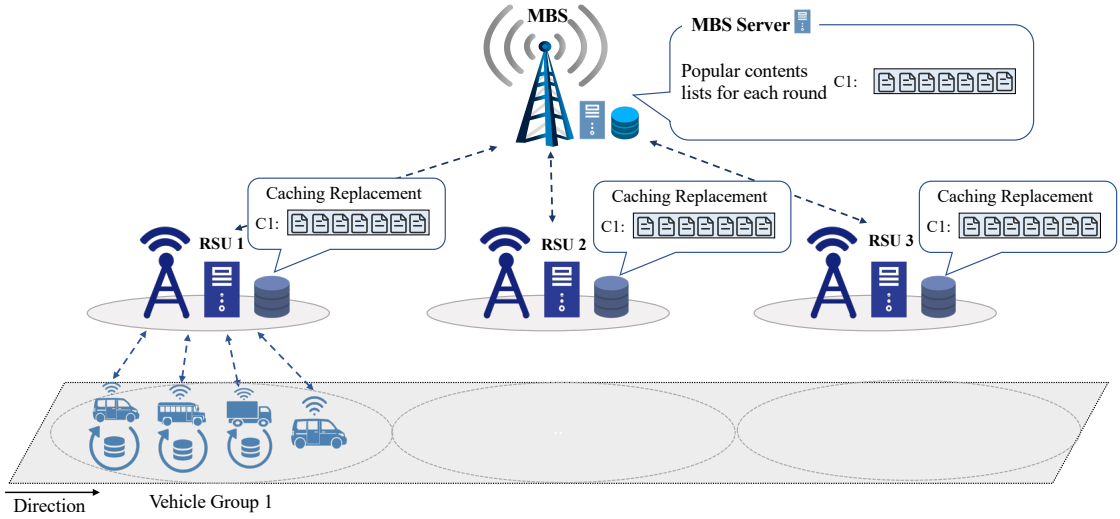


Fig. 4.4 Mobility-aware cache replacement policy: Round 1

aggregated posterior and $p(z)$ is the arbitrary prior. The $q(z)$ is given by [108]:

$$q(z) = \int_x q(z \mid x)\, p_d(x)\, dx, \tag{4.8}$$

where $p(x)$ is the model distribution. The loss of the discriminator is

$$L_D = -\frac{1}{b}\sum_{a=1}^{b} log\left(D\left(z'\right)\right) + log\left(1 - D(z)\right), \tag{4.9}$$

where b is the minibatch size. The adversarial generator we used is

$$L_G = -\frac{1}{b}\sum_{a=1}^{b} log\left(D(z)\right). \tag{4.10}$$

Additionally, C-AAE is inspired in VAE. VAE is a generative autoencoder, aiming to learn the data distribution $p(x)$. It attempts to minimize the KL-Divergence between latent codes distribution and the desired distribution (*i.e.*, Gaussian). A sample from the desired distribution is feed to the decoder, for reconstructing inputs. VAE can effectively cluster similar input data in the latent space [95]. However, one drawback of utilising the KL-

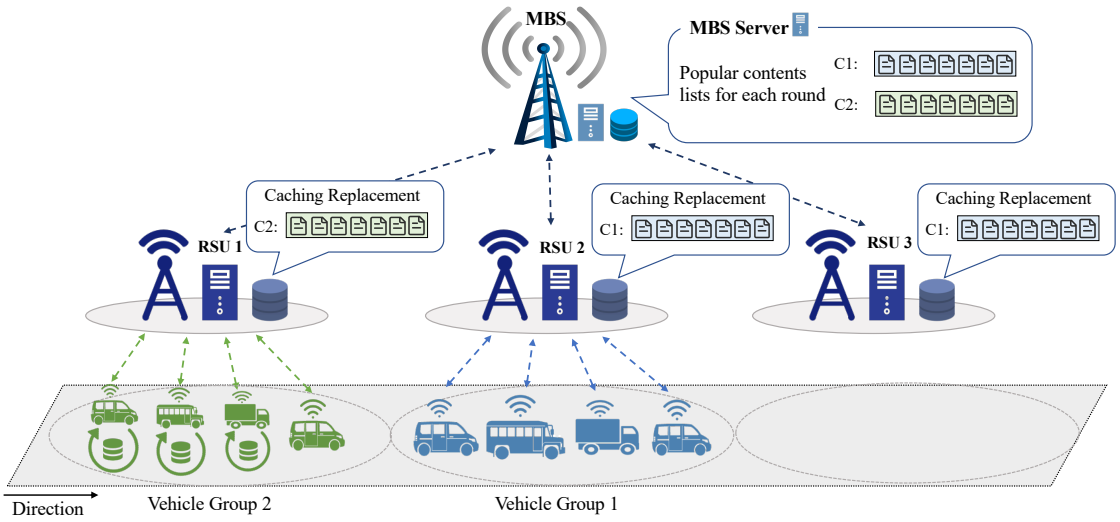


Fig. 4.5 Mobility-aware cache replacement policy: Round 2

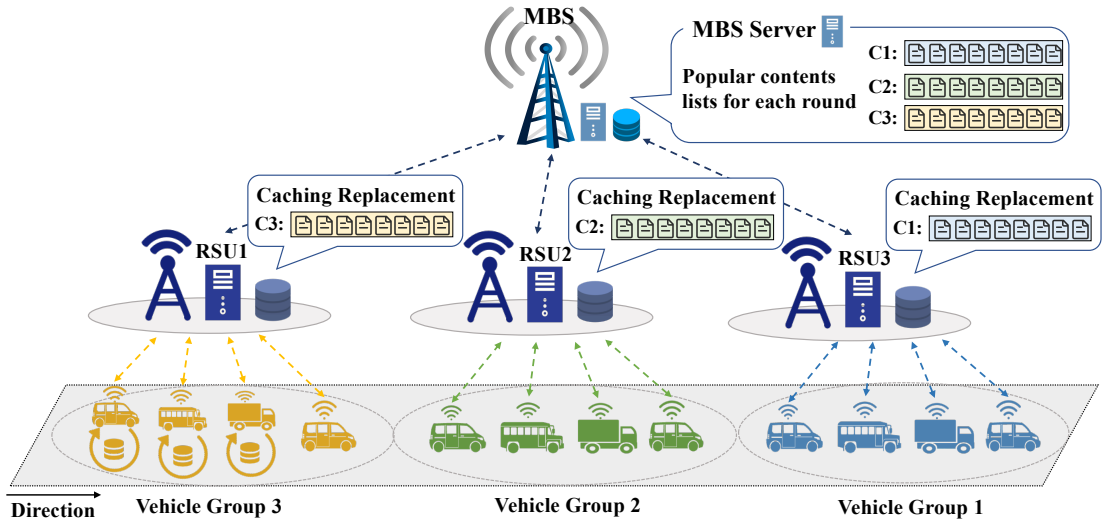


Fig. 4.6 Mobility-aware cache replacement policy: Round 3

divergence in VAE is to handle the functional form of $p(z)$. Instead, AAE provides a more flexibility way that is to sample from $p(z)$, in order to match latent codes distribution with the prior. As a result, more distributions can be used as prior for the latent code.

In our design, the input matrix is the user-by-content request matrix X, which is large and binary. 1 and 0 represent the interested contents for users and uninterested contents, respectively. We mark the contents that users requested before as the interested contents. In fact, it is hard to identify the uninterested contents, since unknown contents and uninterested

	RSU 1	***RSU 2***	***RSU 3***
Round 1	**Contents 1**	**Contents 1**	**Contents 1**
	Vehicle Group 1		
Round 2	**Contents 2**	**Contents 1**	**Contents 1**
	Vehicle Group 2	**Vehicle Group 1**	
Round 3	**Contents 3**	**Contents 2**	**Contents 1**
	Vehicle Group 3	**Vehicle Group 2**	**Vehicle Group 1**

Fig. 4.7 Mobility-aware cache replacement policy: Summary

contents are mixed in the unrequested contents. Marking all unrequested contents as uninterested contents is a bias prediction. Thus, we marked some unknown contents (missing entries) as 1 by a random sampling mechanism. The probability of random sampling is related to the preference of users for contents. We utilise the proposed C-AAE model to predict these missing values. The C-AAE learns the latent code Z from the input matrix X by clustering data in the latent space. Then, X can be recovered from Z to generate $\tilde{X}$. X has incomplete rows and columns. $\tilde{X}$ is a matrix with predicting the missing entries. We rank the predicted contents and the highest score contents in outputs will be chosen as the caching contents in the RSU. The complexity of the MPCF algorithm depends on the C-AAE model's complexity, which is $O(h)$. h is the size of a hidden layer in the C-AAE model.

4.3.3 Mobility-aware Cache Replacement Policy

As the vehicles move from one RSU to another, the requested contents cached at one RSU might become obsolete, whereas the other RSUs do not cache these previously requested contents for the coming vehicles. Such ineffective utilization of cache resources motivates us

to design a mobility-aware cache replacement policy. This policy enables the RSU to replace its caching contents, in response to the mobility pattern of vehicles.

Fig. 4.4 shows the details of how the process of replacement is carried between RSUs and vehicles. To unpack the dynamic process, three rounds of the replacement scenarios are shown in Figs. 4.4 (a), (b) and (c). Fig. 4.4 (d) summarises the activities of each RSU in the three rounds. Figs. 4.4 (a), (b) and (c) illustrate the interactions amongst three RSUs, groups of vehicles and one MBS in our proposed design. All RSUs are located within the coverage area of MBS. RSU1 is different from other RSUs, because it is the only RSU that executes the training of C-AAE model to predict popularity of contents for its connected vehicles. In self-driving scenario, MBS holds prior knowledge of its connected vehicles, including speed, position and destination. MBS has the capacity to calculate the arrival times of vehicles at each RSU. If the connected vehicles changed their behaviour, they will reset their destination and MBS will know immediately.

In round 1, Vehicle Group 1 is moving through the area covered by RSU 1. At the same time, popularity of contents is predicted and a list of predicted popular contents (C1) for Vehicle Group 1 is instantly sent to MBS. MBS then sends the predicted contents (C1) to all RSUs to cache. In round 2, new vehicles (*i.e.*, Vehicle Group 2) move into the coverage area of RSU 1 and Vehicle Group 1 moves to the area covered by RSU 2. MBS then updates the caching contents for all RSUs. RSU 1 will cache the second round prediction (C2), RSU 2 and RSU 3 still store C1. In round 3, new vehicles (*i.e.*, Vehicle Group 3) enter the RSU1's coverage area. Similar to round 2, RSU 1 replaces caching contents to newly updated prediction (C3), depending on the requests made by the new vehicles. C2 is cached in RSU 2, as Vehicle Group 2 moves into the coverage area of RSU 2. RSU 3 still stores C1. Vehicles continue to move over time. In the fourth round, Vehicle Group 1 will leave the area covered by the current MBS and enter to the next MBS. Therefore, the current MBS shall remove C1 from the cache. Due to the small coverage area of each RSU and high speed of vehicles,

the preferences of vehicles may not change from one RSU to another. If it changes, the FL model in RSU will predict new popular contents quickly according to their preferences.

To sum up, the proposed cache replacement policy aims to effectively update the contents on RSU in response to its connected vehicles' prediction results. It is worth noting that the predict contents (*e.g.*, C1, C2, C3 *etc.*) can be similar and MBS only needs to replace the different contents at each round, which effectively reduces the processing time.

Algorithm 2 : ***Mobility-aware Proactive Edge Caching Scheme based on Federated Learning (MPCF)***, M is the set of vehicles connected to the RSU, where $m \in M$. S is the set of RSUs, where $s \in S$; Q is the position of RSU.

RSU Server Execution:

Initialise w_0
for each round r = 1, 2, ... **do**:
  K_r: A set of selected vehicles in r^{th} round
  for each vehicle $m \in M$ **in parallel do**:
    $T^m_{standing} = \left(\mathfrak{D} - P^i_m\right)/U_m$
    if $T^m_{standing} > T_{round} + T_{inf}$ **then**
      add m to K_r
    end if
  end for
  C_r: A set of caching contents in r^{th} round
  C_k: A set of predicted popular contents from vehicle k
  for each vehicle $k \in K_r$ in parallel **do**:
    if $Q^k_{s-1} == Q^k_s$:
      Use the previous model $w_r = w_{r-1}$
    else
      Download the current global model w_r
    end if else
    $w^k_{r+1}, C_k \leftarrow$**VehicleUpdate**$(w_r, k)$
    Add C_k to C_r
  end for
  $w^k_{r+1} \leftarrow \sum_{k=1}^{K} \frac{d_k}{d} w^k_{r+1}$
  Count C_r
  Cache Top N contents from C_r
  CacheReplace(C_r)**:**
end for
Return w_{r+1}

Algorithm 3 : ***Mobility-aware Proactive Edge Caching Scheme based on Federated Learning (MPCF)***

Vehicle Execution:

Input: X, w_r, P_k, $\mathfrak{D}$
VehicleUpdate(w,k)**:**
  for each local epoch e = 1, 2, ...**do**
    for each batch b **do**
      Compute parameters with gradient descent:
      $w_{r+1}^k \leftarrow w_r - \eta \nabla l\,(w_r; b)$
      $\gamma_k = P_k/\mathfrak{D}$
      $w_{r+1}^k \leftarrow \gamma_k w_{r+1}^k$
    end for
  end for
  Rank predicted contents C_k
Return w_{r+1}^k, C_k

Cache Replacement Execution:

CacheReplace(C)**:**
  for each round r = 1,2,... **do**:
    Compare C_{r-1} and C_r
    Update new contents from C_r
  end for

4.4 Performance Results and Analysis

The performance of the proposed MPCF under various environments in VEC is evaluated in this section.

4.4.1 Simulation Settings and Dataset

We simulate a VEC environment in an urban area consisting of an MBS, 3 RSUs and several vehicles located within the coverage areas of RSUs. A HP Z440 workstation with 64G memory is exploited as the MBS to store the lists of cached contents and manage cache resources. Two HP Z440 workstations, working as RSUs, aggregate the parameters of C-AAE model and cache the predicted popular contents. The number of vehicles under each RSU varies from 1 to 100. All vehicles have datasets to conduct local model training. Keras is employed as the Deep learning framework to implement C-AAE and FL, with TensorFlow as backend. The dataset we used in our experiments is MovieLens 1M dataset collected from the MovieLens website [98]. About 1 million ratings are contained in this dataset, which came from 6040 anonymized users on 3883 movies. The contextual information of users, *e.g.* gender, age, address and occupation, is also provided in the dataset. To simulate the process of vehicular users' requests, the rated movies are assumed to request contents from vehicles.

4.4.2 Performance Evaluation

Cache hit ratio is the performance metric we used to evaluate the proposed MPCF, which measures the effectiveness of a cache in fulfilling content requests. Cache hit ratio is calculated as follows: Cache hit ratio = cache hits/ (cache hits + cache misses). One cache hit is captured when the requested content is delivered by the cache, whereas a cache miss is captured when the requested content is not stored in the cache.

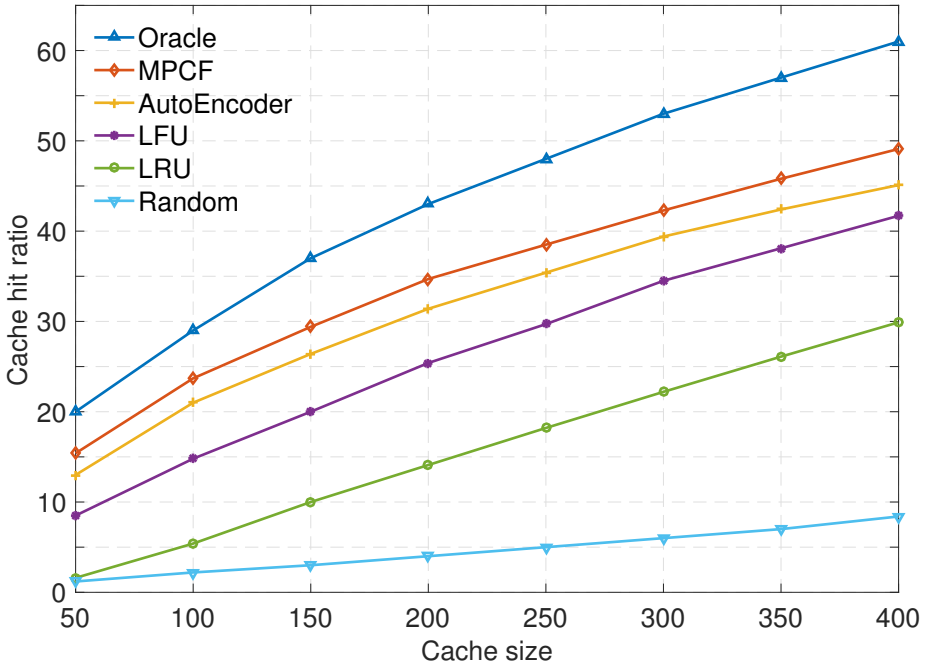


Fig. 4.8 Cache hit ratio with different cache sizes

We compare our proposed scheme to the following four baseline caching schemes, as shown in Figure 4.8.

- Oracle: It has prior knowledge of the exact future content requests from vehicles and provides the maximum of cache hit ratio.
- Random: The contents stored at cache are randomly selected by the RSU.
- Least Recently Used (LRU): When the limit of cache capacity is reached, it firstly removes the least recently used content in the cache [9].
- Least Frequently Used (LFU): In LFU, the least frequently used content in the cache is discarded whenever the cache capacity is full [110].
- AutoEncoder: It is a learning based caching scheme, using AutoEncoder model [109].

Fig. 4.8 depicts the cache hit ratio for varying cache sizes from 50 to 400 contents. The results demonstrate that our proposed MPCF outperforms other reference caching schemes.

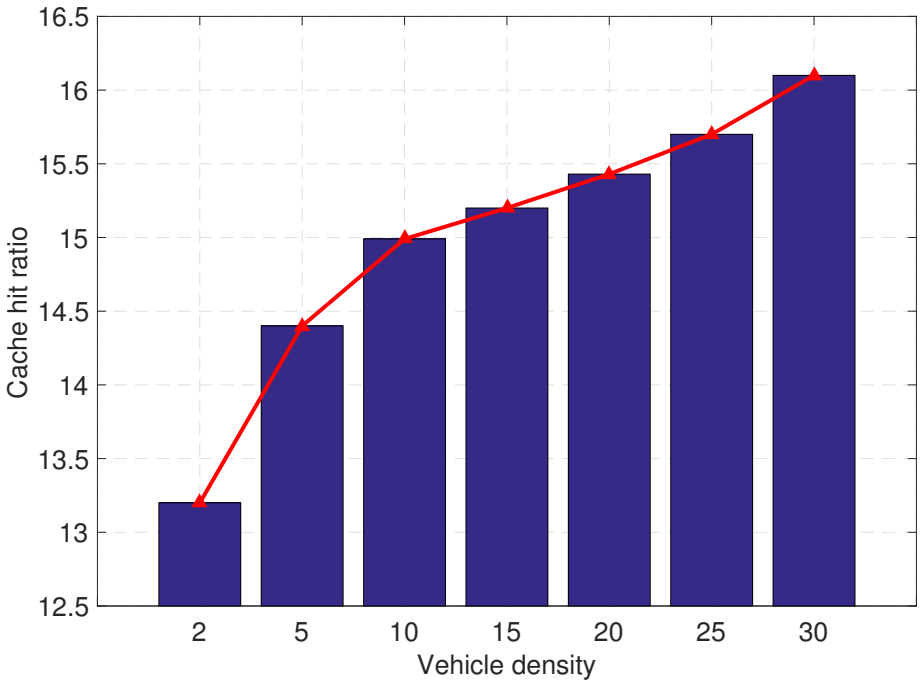


Fig. 4.9 Cache hit ratio vs Vehicle density

With the increase of cache size, the cache hit ratios of all caching schemes rise. As expected, the lowest cache hit ratio is presented by Random. The proposed MPCF and AutoEncoder outperform LFU, LRU and Random because they learn the latent representations and extract features from the content request history of connected vehicles to predict precise content popularity. LFU and LRU follow static rules, but dynamically changing content popularity is not considered. The MPCF shows a better performance compared to AutoEncoder. It is due to the fact that MPCF captures useful features from data and clusters the data in the latent space. Oracle provides the best cache hit ratio, because it has the prior knowledge of content requests from vehicles in the future.

Fig. 4.9 presents the influence of the vehicle density on the cache hit ratio. The cache size of RSU is fixed at 50 in this experiment and the density of vehicles varies from 2 to 30 vehicles/km. The results show that the cache hit ratio increases with a grown in vehicle density. When only two vehicles are moving in the coverage area of RSU, the cache hit ratio is 13.2%. However, when five vehicles connect to the RSU, the cache hit ratio increases to

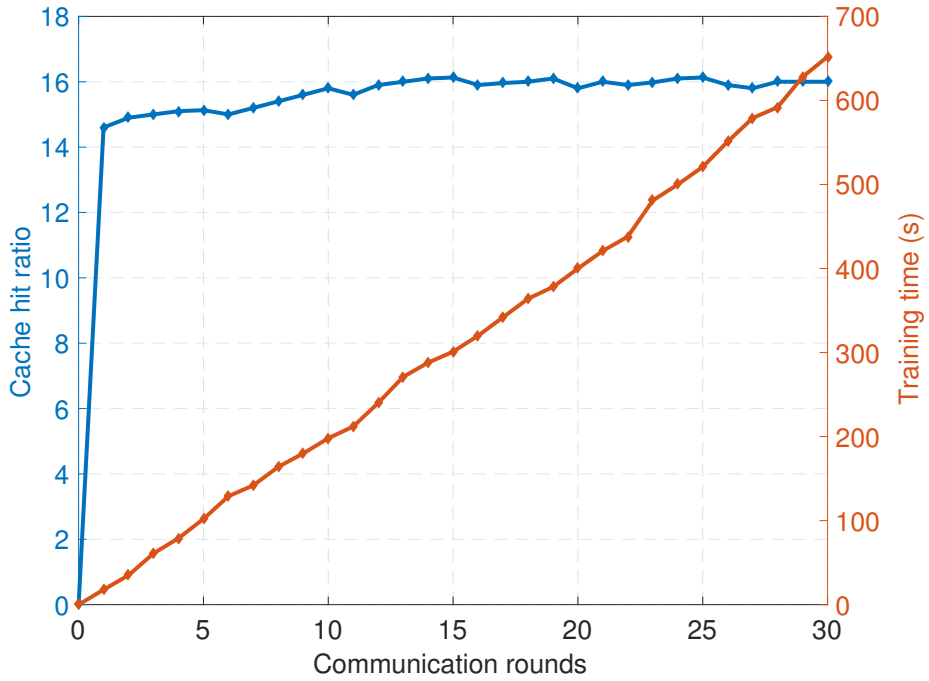


Fig. 4.10 Cache hit ratio and Training time against Communication rounds

14.4%. Along with more vehicles in the RSU's coverage area, more computation capacity and training data are offered by these vehicles. Correspondingly, more accurate prediction of content popularity can be obtained.

Fig. 4.10 shows the results of cache hit ratio, communication rounds versus training time. In this experiment, 10 vehicles collaboratively participate in global model training. In the first round, the cache hit ratio is 14.6 %, while it increases to 14.9% in the second round. When the communication round is 30, the cache hit ratio achieves above 16%. These show that the cache hit ratio rises with the increasing communication rounds. It comes with a price of training time. The training time for one round is about 18 seconds, compared to more than 600 seconds for 30 rounds. As can be seen in Fig. 4.10, the cache hit ratio changes not significant after 15 rounds. Thus, considering a trade-off between communication rounds, training time and cache hit ratio, training FL model for 15 rounds is the best choice to achieve the optimal cache hit ratio. The optimal cache hit ratio changes based on the context of users and their requests.

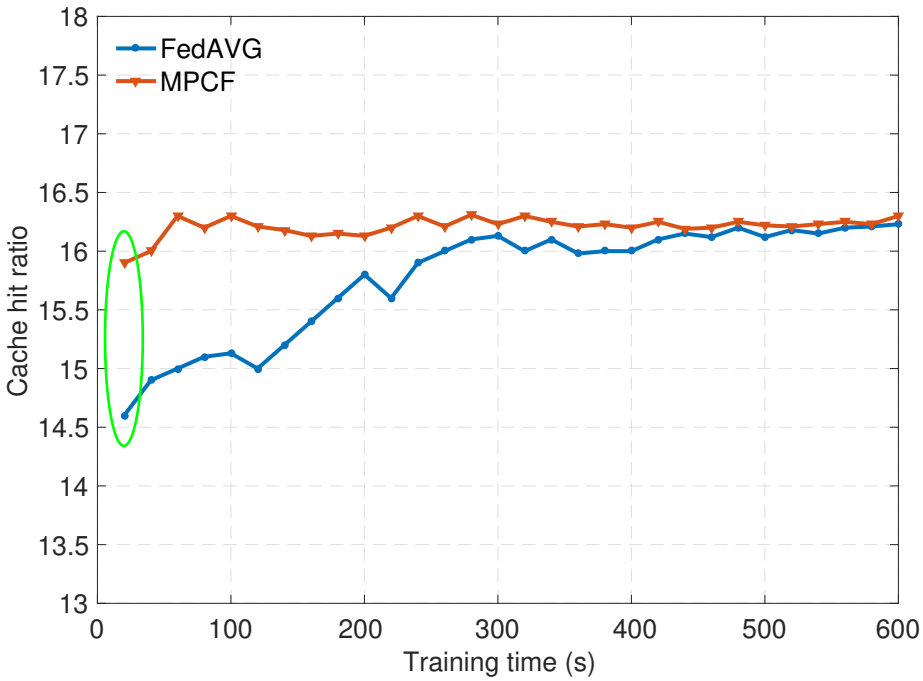


Fig. 4.11 FL training process (27 vehicles)

Fig. 4.11 and Fig. 4.12 show the difference between typical federated learning training process (FedAVG) [8] and our proposed federated learning method (MPCF), in respect of the training time and cache hit ratio. Both methods exhibit the same trend. With longer training time, more accurate results can be achieved. In particular, after training for a while, both methods reach a similar cache hit ratio. However, compared with FedAVG in the first round, MPCF can obtain a higher cache hit ratio. As depicted in Fig. 4.11, when 27 vehicles participate in the FL training, the cache hit ratio of FedAVG is 14.3%. In comparison, the cache hit ratio of MPCF is 15.7%. Fig. 4.12 (5 vehicles) also shows that the MPCF outperforms FedAVG at the beginning in terms of cache hit ratio, while the final results are similar. In VEC, vehicles are driving at high speed. They go through the coverage area of RSU quickly and result in the short training time. The results exhibit that the proposed MPCF can gain a desirable cache hit ratio in a shorter time than FedAVG. For example, Fig. 4.11 depicts the MPCF achieves the target cache hit ratio of 16% within 3 rounds, less than

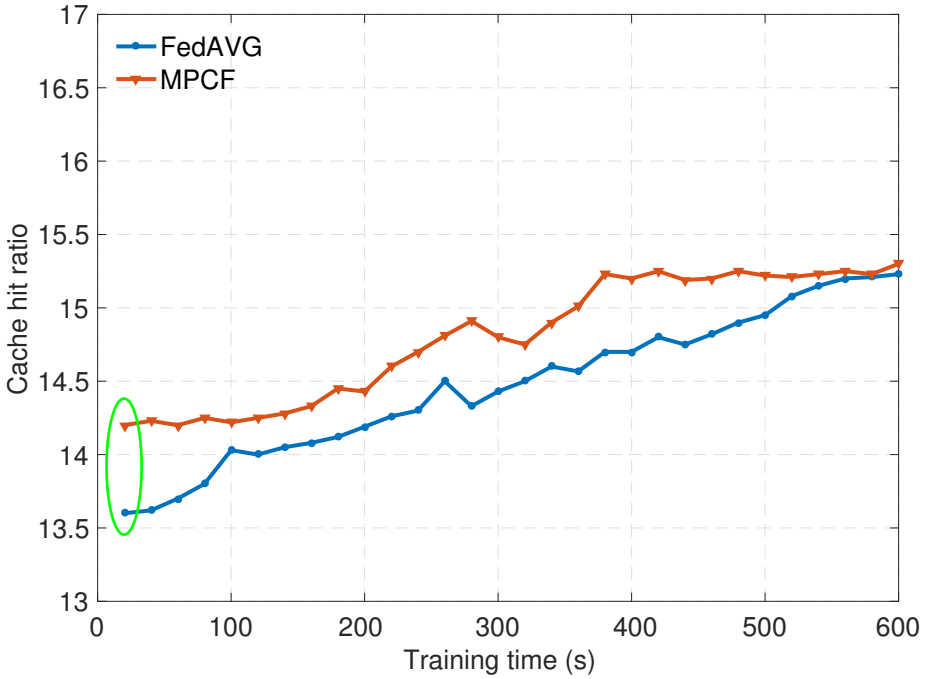


Fig. 4.12 FL training process (5 vehicles)

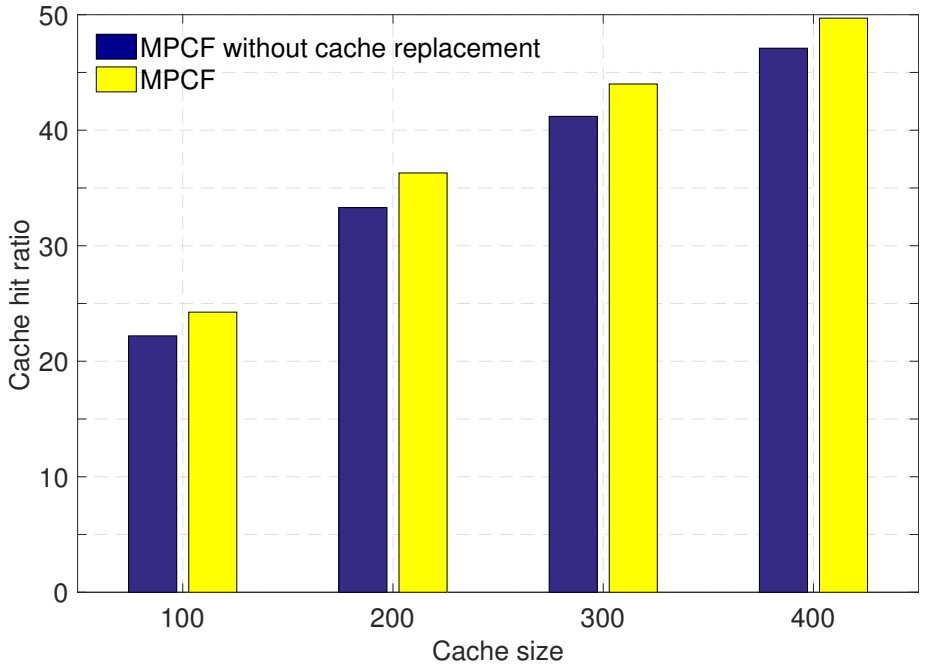


Fig. 4.13 Mobility-aware cache replacement

60 seconds, while FedAVG needs more than 600 seconds. The reason is that the MPCF is mobility-aware and thus more suitable to VEC scenarios.

Fig. 4.13 demonstrates the effectiveness of the mobility-aware cache replacement policy. We compare the cache hit ratio of MPCF and MPCF without cache replacement. As shown in Fig. 4.13, the cache hit ratio for MPCF outperforms MPCF without cache replacement policy. For example, the cache hit ratio of MPCF is around 24.25% when the cache size is 100, while MPCF without cache replacement policy achieves the cache hit ratio of 22.2%. These experiment results demonstrate that the proposed mobility-aware cache replacement policy is able to further enhance the cache performance of FL learning-based caching schemes in VEC. It can update contents dynamically according to the changes in content requests from vehicles.

4.5 Summary

In this chapter, we have proposed a new Mobility-aware Proactive edge Caching scheme on Federated learning, termed MPCF, to improve cache performance and protect vehicles' privacy. It utilises a context-aware adversarial autoencoder model to estimate content popularity and then places predicted popular contents at the edge of vehicular networks to reduce latency. To maximise the cache hit ratio, a mobility-aware cache replacement policy is designed to dynamically update cache contents at RSUs, according to the mobility pattern of moving vehicles and their predictions of content popularity. Numerical results demonstrate that MPCF outperforms other baseline caching schemes on cache hit ratio. The FL training process of MPCF accelerates the training of a global shared model and achieves the optimal cache hit ratio in a shorter time. Implementing the mobility-aware cache replacement policy further improves the cache hit ratio.

Chapter 5

Peer-to-Peer Federated Learning based Edge Caching for Internet-of-Vehicles

5.1 Introduction

To improve road safety and travel comfort, the Internet-of-Vehicles (IoV) has emerged as a new paradigm for intelligent transportation systems [111]. It supports a wide range of emerging vehicular applications, such as smart navigation and infotainment [112]. These applications require low network latency and substantial network resources (*e.g.*, caching, computation, and communication), which places huge challenges to the IoV. Shifting cloud computing and storage capabilities to the edge nodes of IoV has been considered as a promising approach to satisfy the diverse requirements of vehicular applications. Especially, caching popular contents at edge nodes (*e.g.*, Base Station (BS), Road side unit (RSU), vehicles) can alleviate the data traffic on backhaul links and reduce service latency.

Due to the limited caching storage at edge nodes, efficient caching schemes that manages the caching resources is necessary. Recent breakthroughs in Machine Learning (ML) facilitate many learning-based content caching schemes [13] [10] [15]. ML techniques can effectively extract hidden features and representations from users' data to accurately predict content

popularity. However, conventional caching schemes cannot be directly applied in IoV, due to the inherent characteristics of IoV, *e.g.*, the high mobility of vehicles and dynamic network environment. Additionally, most of the existing learning-based caching schemes need to centrally analyse users' data to make caching decisions. This process may cause the disclosure of users' privacy. Thus, it is of paramount importance to design a learning-based caching scheme for IoV that can achieve high caching performance while protecting users' privacy.

Federated learning (FL) [8] provides a new framework for fitting ML techniques into the edge while mitigating user privacy risks. It allows a central server to cooperate with multiple vehicles to jointly train an ML model in the IoV. Vehicles upload parameters of the trained model to the central server and keep their training data locally. However, if an RSU is chosen as a central server, vehicles with high speed may pass several RSUs during the FL training process, since the coverage area of RSU is small. This may seriously affect the performance of the trained model in FL. To address this challenge, we propose a Peer-to-Peer Federated learning based proactive Caching scheme (PPFC) that is well suited to the highly dynamic IoV environments. In PPFC, a vehicle with enough computation, caching and communication resources can be selected as a central server to aggregate a global model from peers. Nearby vehicles with the same direction can then connect to this server vehicle to participate in the FL training. Compared with traditional FL, peer-to-peer FL can eliminate the issue of hand-over between RSUs, achieve lower latency and adapt to the mobility of vehicles. PPFC utilises a Collaborative Filtering based Variational AutoEncoder (CF-VAE) model to predict content popularity based on the contextual information of users for making smart caching decisions.

The main contributions of the chapter are summarised as follows:

1. A peer-to-peer federated learning based proactive caching scheme is proposed to adapt to high mobility of vehicles in IoV. In the proposed scheme, a vehicle rather than a fixed edge node, acts as a central server to aggregate ML models from nearby vehicles.

2. Due to the heterogeneous abilities of vehicles, a dual-weighted model aggregation scheme is designed by considering data size and staleness of vehicles to reduce the effect of straggler vehicles, in order to further improve the accuracy of the trained global model in the designed peer-to-peer FL.

3. A collaborative filtering based variational autoencoder model is proposed to predict the popularity of contents by using users' historical requests and contextual information, which can learn deep latent representations of users' characteristics, while preserving data privacy through the use of FL.

The rest of this chapter is organised as follows. The system architecture of the proposed cache scheme is presented in Section 5.2. Section 5.3 describes the detailed implementation of PPFC. The performance evaluation and analysis of PPFC are provided in Section 5.4. Section 5.5 concludes this chapter.

5.2 System Architecture

The system architecture of the proposed PPFC is shown in Fig. 5.1. A vehicular network is considered, which consists of a BS, RSUs and vehicles. It is a hierarchical structure. The top layer is a BS, which links to the Internet through a reliable backhaul link. In the middle layer, several RSUs are placed equidistantly at the coverage area of the BS. Each RSU connects to several vehicles that are distributed at the bottom layer. The communication between BS, RSUs and vehicles are via wireless links. In our design, both RSUs and vehicles have cache capability, because vehicles are equipped with OBUs and RSUs have cache-able servers. Users can fetch their requested contents from RSUs and vehicles, instead of the internet

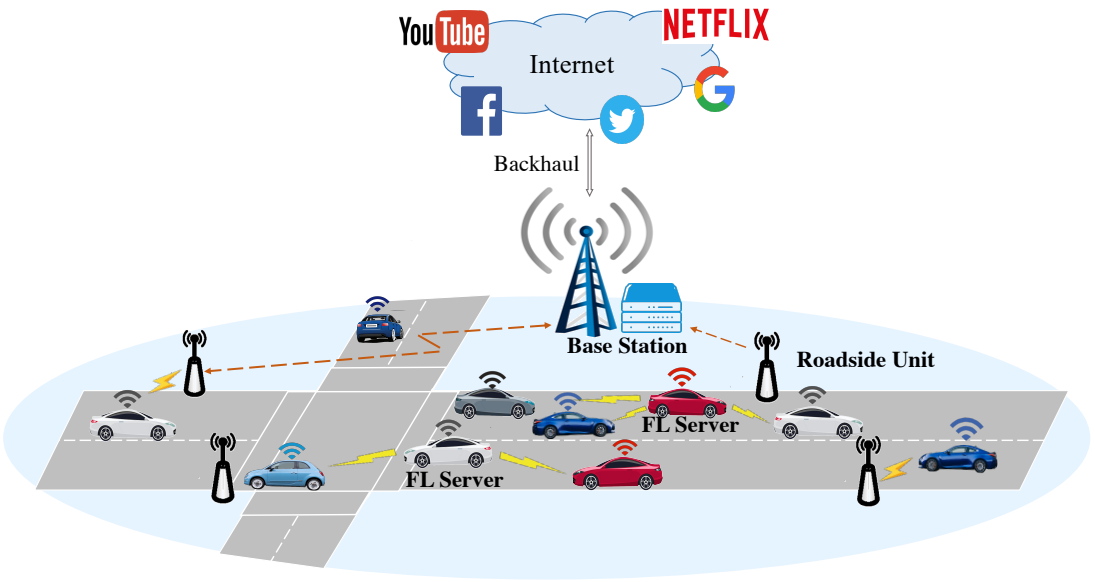


Fig. 5.1 System architecture of PPFC

only. When a vehicular user requests a content, it will firstly check its own cache. If the requested content is stored locally, the vehicle can directly obtain it without any transmission. If not, this request will broadcast to neighbour vehicles. If the broadcast is responded, nearby vehicles will send the requested content to the vehicle which requested the content. Otherwise, the request will be forward to the current connected RSU. If the requested content is available in the RSU, the RSU can delivery this content to the requested vehicle. If the requested content is still missing, the vehicle has to request this content from the Internet.

Thus, vehicles fetch contents mainly in the following three ways: Vehicle-to-Vehicle (V2V) content delivery, Vehicle-to-RSU (V2R) content delivery and Vehicle-to-Internet (V2I) content delivery. V2V content delivery links two vehicles within distance γ. Vehicle k_j will broadcast its content request firstly. If vehicle k_i stores the requested content and responses for the nearby vehicle k_j, vehicle k_i can directly deliver the requested content to vehicle k_j through V2V link. RSU is another cache place in our designed PPFC. Vehicles which are located at the same coverage area of the RSU use the same frequency band. Vehicles fetch the requested contents from RSU via V2R links. V2I content delivery is the transmission

between the Internet and vehicles. If the requested contents are missing in both vehicles and RSUs, it will get from the Internet with V2I transmission. Obtaining contents from vehicles and RSUs, without asking for the Internet, can significantly ease the network load. Moreover, users fetch their requested contents from near vehicles which can largely reduce latency.

Due to the limited storage resource of RSUs and vehicles, we assume that a vehicle can only store up to m contents and an RSU can cache n contents at most. To make full use of caching storage at RSUs and vehicles, designing a smart caching scheme is essential. The gain from the caching scheme highly depends on the accuracy of content popularity. However, content popularity is dynamic and hard to predict. Different vehicular users may prefer different contents and their preferences may change frequently which are influenced by location and time. The spatio-temporal variability on the popularity of contents adds substantial complexity in content caching of IoV. Moreover, the lifetime of contents in IoV is short. As a result, the cached content can easily become out of date. Thus, according to the estimated content popularity and the lifetime of contents, updating cached contents regularly is necessary to the edge caching in IoV.

We design a proactive caching scheme to make caching decision by predicting content popularity, based on the peer-to-peer federated learning. In typical FL, multiple vehicles collaboratively train a global model from their site-specific datasets under the instruction of a central server in the RSU. Instead of sending raw data to the central server in RSU for model training, vehicles only send parameters of the model to the central server. With the help of distributed training at vehicles, user privacy can be largely protected. The global model that trained in FL is a content popularity prediction model, which is utilised to make the smart caching decision. However, due to the high mobility of the vehicle and short coverage area of an RSU, vehicles with high-speed cannot complete FL training process within one RSU's coverage area. Switching between RSUs happens frequently. If the fixed RSU is chosen as the central server in FL, computation and communication costs are increased,

and the accuracy of prediction is degraded. To adapt the FL framework to the IoV scenario and address the limitations and conundrums caused by the high mobility of vehicles, the peer-to-peer FL is proposed, without depending on a central server in an RSU. A vehicle can be selected as a moving central server of FL. The same direction of vehicles within one transmission hop are clustered into one group and then execute FL training within this group.

5.3 Peer-to-Peer Federated Learning for Edge Caching

This section describes the details of our proposed proactive content caching scheme. In light of growing privacy concerns, FL is designed to collaboratively train a global ML model by using the local data at distributed vehicles. However, in the complex and dynamic IoV environments, the typical FL faces the challenge of frequently switching connected central servers and heterogeneous abilities of vehicles. To better fit FL to IoV, we proposed a peer-to-peer FL, as shown in Fig 5.2. Training a model in peer-to-peer FL is performed by multiple communication rounds and each communication round r consists of the following six steps:

1) **Location based vehicle selection**: To avoid frequently switching connected central servers for vehicles during the FL training process, the vehicle with sufficient computation and caching capacity can be selected as a central server. Unlike the server in the fixed RSU for typical FL, the vehicle server is a moving central server to aggregate models as well as providing caching contents to other vehicles. The same direction of vehicles with one transmission hop neighbours are chosen as participating vehicles to be involved in the FL model training. In self-driving scenario, vehicles will set their destinations before the journey. Thus, the prior knowledge of vehicle trajectories can be achieved.

2) **Model dissemination**: Once the server vehicle and participating vehicles K are selected, the server vehicle initialises the global ML model w_r and sends it to the participating vehicles with the aim of distributed model training at these vehicles.

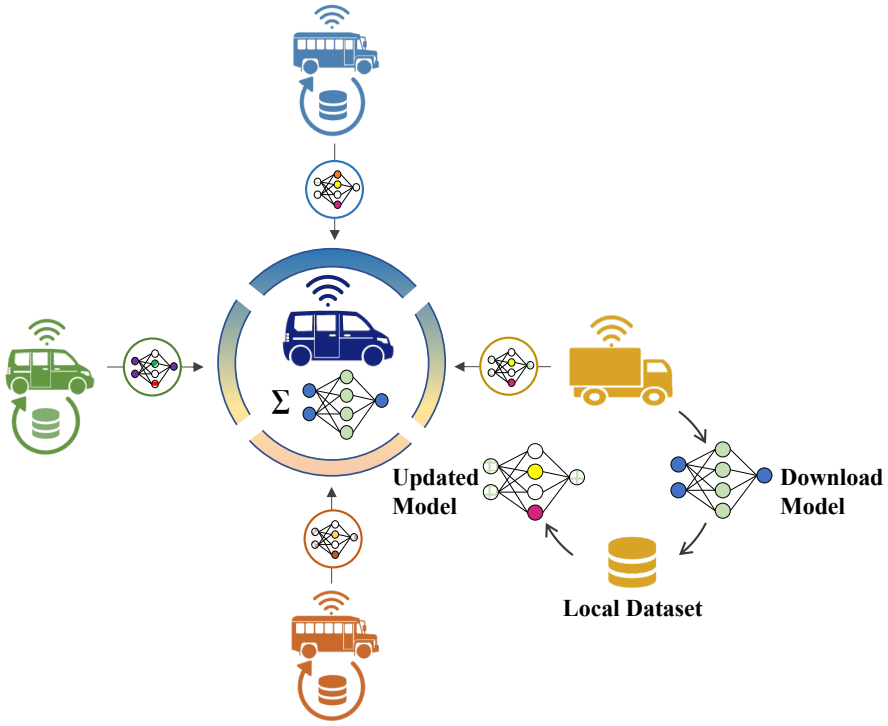


Fig. 5.2 Peer-to-peer federated learning

3) **Learning of distributed prediction model**: Each participating vehicle utilises its own data to train the ML model, which is a Collaborative Filtering based Variational Autoencoder (CF-VAE) model to predict the content popularity. It is an unsupervised learning algorithm to copy its input X to its output $\tilde{X}$, as shown in Fig. 5.3. The X is a user-content rating matrix, which consists of vehicular users' historical requests. The $\hat{a}$ is the matrix of the vehicular users' context information. We fed X and $\hat{a}$ into CF-VAE to learn the hidden representations Z, respectively. Then, these obtained representations are combined to reconstruct the input X. X samples variable x. The encoder $q(z \mid x)$ which is an inference neural network maps x to a Gaussian distribution and the latent variable z is estimated. The decoder, a generative neural network $p(z \mid x)$, decodes z back into x. In the generative process, our objective is to maximise the probability of each x. It can be defined as:

$$p(x) = \int p(x \mid z)\, p(z)\, dz. \tag{5.1}$$

$p(x \mid z)$ is parameterised with a function approximator. The likelihood $p(x \mid z)$ and the prior $p(z)$ can be formulated, while the posterior $p(z \mid x)$ requires an intractable integral over the latent space. The posterior $q(z \mid x)$ generates a distribution over the latent variables. Kullback-Leibler divergence can be used to minimise difference between $p(z \mid x)$ and $q(z \mid x)$. It can force the latent space distribution to be Gaussian.

$$\begin{aligned} KL[q(z \mid x) \parallel p(z \mid x)] = \\ \mathbb{E}_{z \sim q(z|x)} [\log q(z \mid x) - \log p(z \mid x)]. \end{aligned} \tag{5.2}$$

Applying Bayesian inference we have

$$\begin{aligned} \mathrm{KL}[q(z \mid x) \parallel p(z \mid x)] = \\ \mathbb{E}_{z \sim q(z|x)} [\log q(z \mid x) - \log p(z \mid x)] + \log p(x). \end{aligned} \tag{5.3}$$

Then, to minimise $\mathrm{KL}[q(z \mid x) \parallel p(z \mid x)]$, the Eq. (5.3) can be simplified as the following form:

$$\begin{aligned} \log p(x) \geq & \mathbb{E}_{z \sim q(z|x)} [\log p(x \mid z)] \\ & - \mathrm{KL}[q(z \mid x) \parallel p(z)]. \end{aligned} \tag{5.4}$$

where the right hand-side is the variational lower bound of VAE. The approximate posterior $q(z \mid x)$ follows a Gaussian distribution $N(\mu, \mathrm{diag}(\sigma^2))$ where μ is the mean and σ^2 is variance. The generative network $p(x \mid z)$ and inference network $q(z \mid x)$ are trained by maximising the variational lower bound with respect to their parameters. The reparameterisation trick $z = \mu + \sigma \odot \varepsilon$ can be implemented to get the unbiased estimate of low variance bound. We suppose the mean and covariance are $\mu(x)$ and $\sigma(x)$, respectively. ε follows $N(0, I)$, the

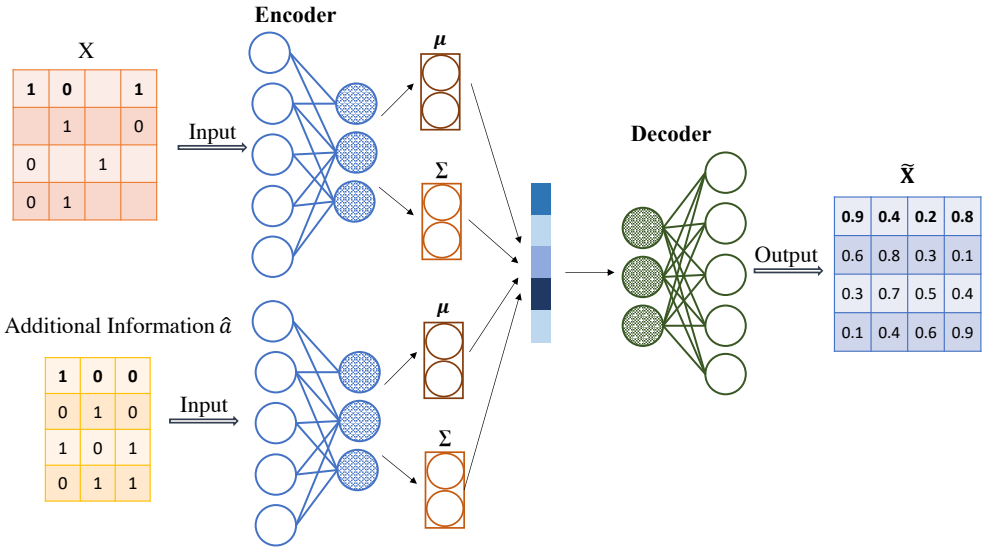


Fig. 5.3 Collaborative filtering based variational autoencoder

equation can be rewritten as follow:

$$\begin{aligned} &\mathbb{E}_{q(z|x)}\left[\log p\left(x \mid z\right)\right] = \\ &\qquad \mathbb{E}_{\varepsilon \sim N(0,I)}\left[\log p\left(x \mid z = \mu + \sigma \odot \varepsilon\right)\right], \end{aligned} \tag{5.5}$$

where ε is a vector sampled from standard Gaussian variables. With the help of the reparameterisation trick, the inference and generative networks can be trained through end-to-end backpropagation by SGD.

Once the local training of CF-VAE at vehicles is completed, the parameters of CF-VAE are sent back to the server vehicle for model aggregation.

4) **Dual-weighted model aggregation**: To improve the quality of the global model, the server vehicle constructs a new version of the global ML model by aggregating updated models from nearby vehicles with a dual-weighted method. Due to the heterogeneous abilities of vehicles, vehicles contain different amount of local data and have different learning status. Vehicles cannot equally contribute to the global model with such large differences. The effect of straggler vehicles needs to be reduced for the current FL communication round. Therefore, we introduce a dual-weighted aggregation scheme to solve this problem, which is divided

into two parts: data weight and staleness weight [113]. The data weight λ_D is decided by the proportion of the local data size d_k at a vehicle k to the total data size D of all participating vehicles. The server vehicle will know the value of d_k and D during the vehicle selection. The data weight of vehicle k is $\lambda_D^k = \frac{d_k}{D}$, where $D = \sum_{k=0}^{K} d_k$, $d_k = |D_k|$ and D_k is the set of training samples on vehicle k. The staleness weight λ_S is influenced by the uploading time T_{up} and downloading time T_{down}, which reflects the staleness for the model. The staleness is calculated as $\varphi = T_{up} - T_{down}$. It also indicates the computing power of the vehicle. The stronger computation capability of vehicle spends less time to train the model with smaller staleness. A smaller weight is given to the vehicle with a larger staleness. The staleness weight is calculated using the following exponential function [113][92]:

$$\lambda_S^k = (e/2)^{-\varphi}. \tag{5.6}$$

where e is Euler's number. Thus, the updated model is conducted with the weighted average sum:

$$w_{r+1} \leftarrow w_r - \sum_{k=1}^{K} \lambda_D^k \lambda_S^k w_{r+1}^k, \tag{5.7}$$

After one round of FL training, vehicles overwrite their local parameters to the latest downloaded parameters and refresh their dual-weights to prepare the next round FL training without any communication costs.

5) **Model optimisation**: To improve the convergence of FL, an adam-based optimisation [114] is exploited in the server vehicle, which is an extension to stochastic gradient descent. Based on the local data of server vehicle, the model will be evaluated. The aim of our

proposed peer-to-peer FL is to minimise the loss function $\ell(w)$:

$$\begin{aligned} \min_{w} \ell(w) &= \sum_{k=1}^{K} L_k(w), \\ \text{where } L_k(w) &= \frac{1}{d_k} \sum_{i \in D_k} \ell_i(w). \end{aligned} \tag{5.8}$$

6) **Caching decision and model update**: Based on the output of the updated model, the highest m predicted rating scores in $\tilde{X}$ are selected as the caching contents in the vehicle. The less n popular contents will cache in RSUs. Meanwhile, the server vehicle updates the global model and this model will be disseminated to all one-hop neighbour vehicles who will participate in the next FL communication round.

Above steps are repeated until an optimal model is achieved at the server vehicle. The pseudo-code of PPFC is outlined in Algorithm 4.

5.4 Performance Evaluation

In this section, comprehensive experiments are conducted to evaluate the performance of PPFC under various IoV environments and compare the PPFC with four baseline caching schemes with respect to the cache hit ratio.

5.4.1 Experiment Settings

We set up a networking testbed, consisting of 10 Raspberry Pi devices. Each Raspberry Pi represents a vehicle and has a local dataset to conduct learning-based prediction model training. The dataset is MovieLens 1M which contains about 1 million ratings from 6000 anonymized users on 3883 contents [98]. This dataset also involves the contextual information of users, such as, age, gender and address. Keras and TensorFlow are used to implement the CF-VAE and FL. The evaluation metric we used to measure the proposed PPFC is cache

Algorithm 4 : The peer-to-peer federated learning based proactive caching scheme, K is the set of participating vehicles, where $k \in K$. η is the learning rate; E is the number of epoches. B is the local minibatch size;

Select A Server Vehicle and Participating Vehicles
Server Vehicle Execution:

initialise w_0
for each round r = 1,2,... **do**:
  K_r : a set of participating vehicles
  Get the parameters of the global model w_r
  for each vehicle $k \in K_r$ in parallel **do**:
    $w_r^k \leftarrow$ **VehicleUpdate**(w_r, k)
  end for
  $w_{r+1}^k \leftarrow w_r - \sum_{k=1}^{K} \lambda_D^k \lambda_S^k w_{r+1}^k$
for end
Return w_{r+1}

Participating Vehicle Execution:

Input: X, w_r
VehicleUpdate(w,k)**:**
  for each local epoch i from 1 to E **do**
    for batch $b \in B$ **do**
      Compute parameters with gradient descent:
      $w_{r+1} \leftarrow w_r - \eta \nabla l\,(w_r; b)$
    end for
  end for
Return w_{r+1}

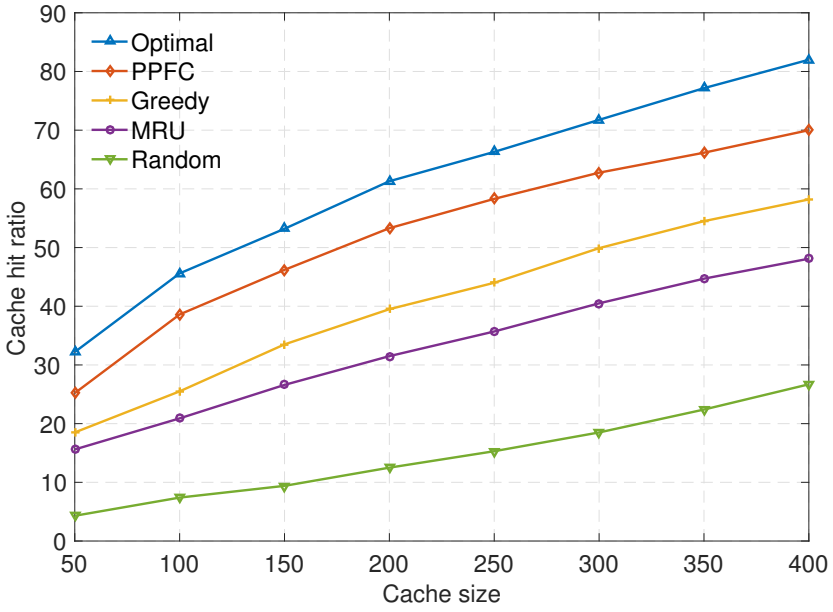


Fig. 5.4 PPFC vs. Other reference schemes (10 vehicles)

hit ratio, which represents the percentage of requested contents from users that enable to serve from vehicles and RSUs.

5.4.2 Experimental Results

Fig. 5.4 and Fig. 5.5 compare the performance of PPFC with other four reference caching schemes (Optimal, Greedy, MRU and Random) and show the cache hit ratio for varying cache sizes from 50 to 400 contents. They also demonstrate the impact of 10 vehicles and 5 vehicles participating in peer-to-peer FL training on cache hit ratio, respectively. Both figures exhibit the same trend. The cache hit ratios of all caching schemes increase, with the growth of cache size. The Optimal reference caching scheme presents the highest cache hit ratio, because it has a perfect knowledge of future vehicular user demands. Our proposed PPFC outperforms the other three reference caching schemes, since PPFC predicts the future popular contents for users by learning hidden features from the request of users and clustering these requests in the latent space, in order to realise the relationship between users and contents. Greedy is a simple learning algorithm, caching the m highest previous demanded contents, but it

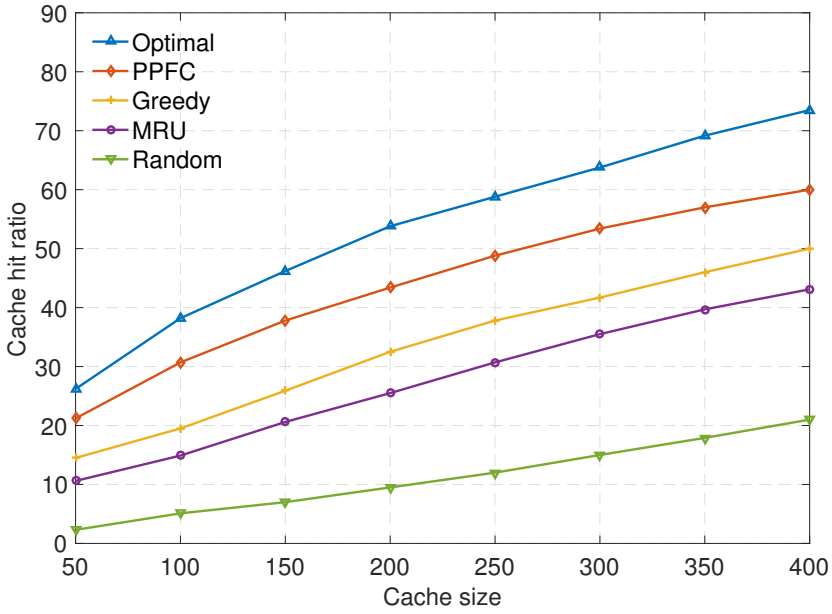


Fig. 5.5 PPFC vs. Other reference schemes (5 vehicles)

does not consider future content popularity. MRU is the third reference caching scheme. It follows a static rule that firstly discards the most recently used contents. However, it lacks consideration of dynamically changing content popularity. The random algorithm shows the lowest cache hit ratio, which randomly selects the m contents to the cache. Comparing Fig. 5.4 with Fig. 5.5, the cache hit ratio of 10 participating vehicles in the peer-to-peer FL training is higher than 5 participating vehicles. When the cache size is 50, the cache hit ratio of 10 participating vehicles is 25.2%, while 5 participating vehicles can only achieve 21%.

Fig. 5.6 investigates the relationship between vehicle density, training time and cache hit ratio. The vehicle density is varied from 2 to 10 vehicles per/km. The results demonstrate that when the cache size is 50, the cache hit ratio rises with the increase in vehicle density. When 2 vehicles participate in the peer-to-peer FL, the cache hit ratio is 16.2%. Whereas, the cache hit ratio will rise to 25.3%, if 10 vehicles attend in FL training. Meanwhile, when the number of participating vehicles changes from 2 to 10, the training time for per communication round increases from 3.98 seconds to 16 seconds. It indicates that more accurate prediction can be achieved if more vehicles participate in the FL training. It is because that more participating

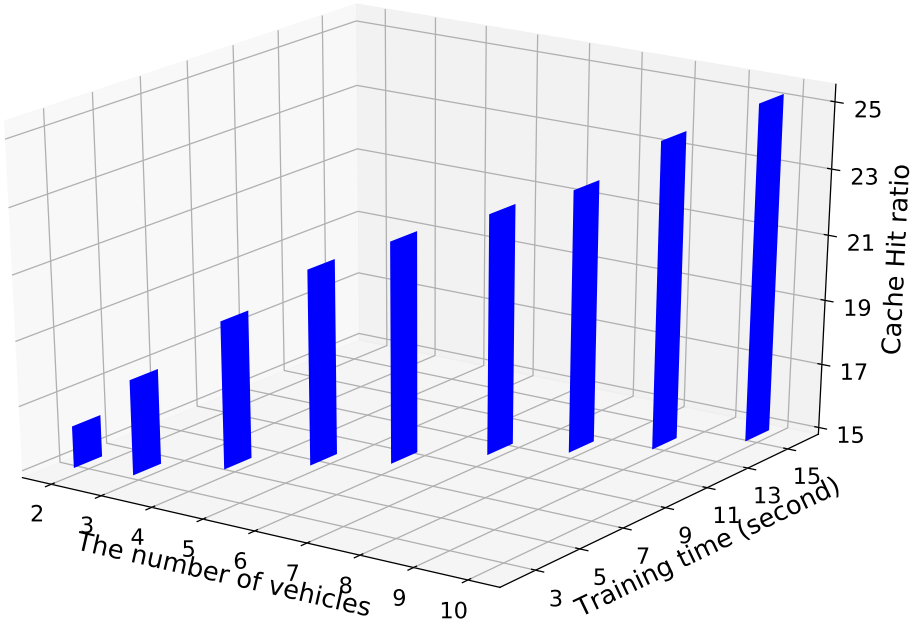


Fig. 5.6 Vehicle density vs. Training time vs. Cache hit ratio

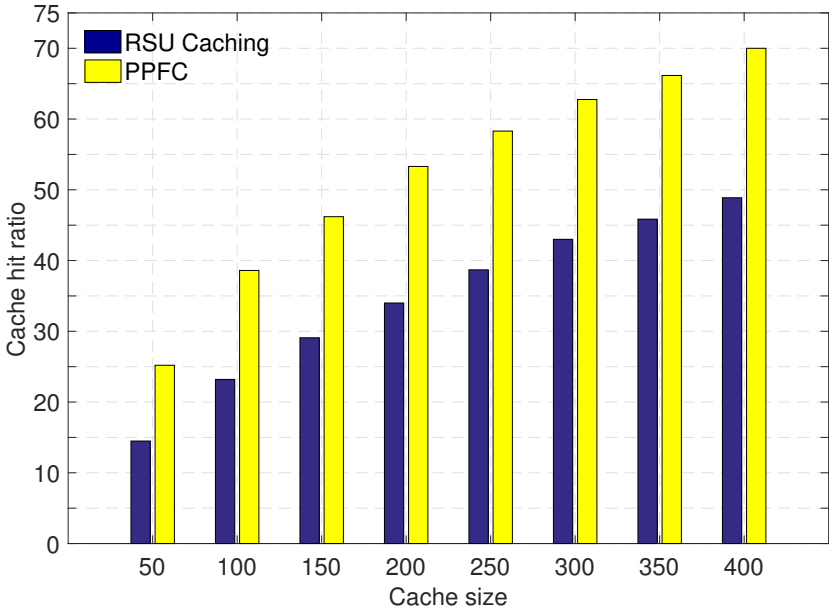


Fig. 5.7 PPFC vs. RSU caching (10 vehicles)

vehicles provide more training data and computation capacity. However, it is a trade-off between the training time, vehicle density and cache hit ratio. As more vehicles attend to the FL training, the cache hit ratio improves, but the training time takes longer.

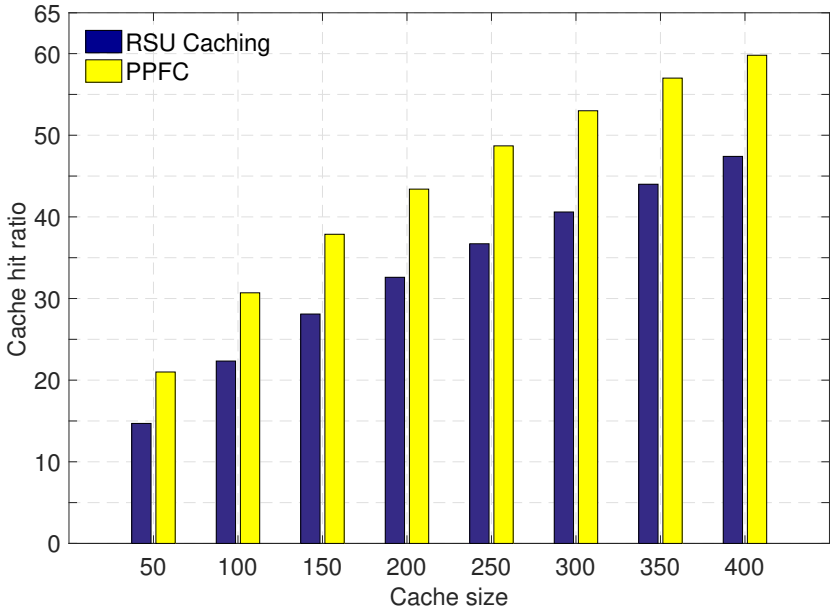


Fig. 5.8 PPFC vs. RSU caching (5 vehicles)

Fig. 5.7 and Fig. 5.8 depict the effectiveness of vehicle-to-vehicle caching. These experiments compare the caching performance of traditional RSU caching with the proposed PPFC that combines vehicle-to-vehicle caching and RSU caching. Fig. 5.7 shows the cache hit ratio for various cache sizes between 50 and 400 contents, when the number of participating vehicles in peer-to-peer FL training is set to 10. By contrast, Fig. 5.8 describes the results of 5 vehicles attending to the FL training. It can be seen from both Fig. 5.7 and Fig. 5.8, the PPFC demonstrates a better caching performance compared to RSU caching. When cache size is 50 and 10 vehicles participate in the FL training, the cache hit ratio of PPFC is 25.3%, while RSU caching can only reach 14.49%. Maintaining the same cache size, when the number of participating vehicles reduces to 5 vehicles, the cache hit ratio of PPFC can obtain 21%, but a similar cache hit ratio for RSU caching is achieved. For the other cache sizes, the same trend has been observed. It indicates vehicle-to-vehicle caching can improve the cache hit ratio. As the number of participating vehicles increases, more caching capacity from vehicles are brought, and therefore, the cache hit ratio rises. However, the caching performance of RSU caching is not affected by the number of participating vehicles.

5.5 Summary

In this chapter, we have proposed a new Proactive content Caching scheme on Peer-to-Peer Federated learning (PPFC) to protect vehicles' privacy, enhance caching performance and reduce latency. Due to the mobility of vehicles, a vehicle is selected as a moving central server, to ease reliance on the fixed central server in RSU and eliminate the issue of frequently hand-over between RSUs. PPFC utilises a collaborative filtering based variational autoencoder model by leveraging the user-content interactions in the form of a content request matrix from vehicles to predict content popularity in the future and pre-fetch predicted popular contents at vehicles and RSUs to improve caching performance. Numerical results show that PPFC outperforms other reference caching schemes in terms of cache hit ratio. More vehicles participating in the peer-to-peer FL training can achieve better caching performance.

Chapter 6

Cooperative Hierarchical Caching in Fog Networks

6.1 Introduction

Fog radio access networks (F-RANs) have been proposed as a promising network architecture that is able to support the growing traffic, reduce the service latency and promote content delivery rate. In the F-RANs, edge nodes, *e.g.*, remote radio heads (RRHs) and Fog Access Points (F-APs), are capable of local signal processing, distributed caching and cooperative radio resource management [115]. These distributed caching among edge nodes provide a great opportunity for content caching in F-RANs. Since some contents are often requested by different users at different times, placing these popular contents as close as possible to users at the edge nodes can greatly reduce duplicate data transmission and service latency. As the cache capacity of edge nodes is limited compared to a huge number of contents, the effective utilization of the available cache capacity has a profound effect on caching performance.

Cooperative caching is an effective way to optimise the management of cache capacity for edge nodes as well as maximise the satisfaction of users' request. For example, Taghizadeh *et al.* [116] proposed a cooperative caching strategy, where neighbour mobile users cooperate

with each other for caching contents. A cooperative caching framework in mobile cellular networks was proposed in [117]. In this framework, evolved NodeBs (eNBs) in 4G and base stations collaborate to decide the location of caching contents and find out which contents need to be cached. All the above caching schemes assume that the content requests are followed by Zipf distribution. In reality, the popularity of contents is highly dynamic and hard to be accurately modelled. Therefore, it is required to design a learning-based cooperative caching scheme to learn dynamic content popularity trends so as to adapt caching schemes to the dynamic environment of F-RANs [118].

Despite many efforts, some key challenges are still faced by the existing learning-based cooperative caching schemes. 1) Privacy: The majority of the prior works on cooperative caching schemes are designed for a highly controlled environment, where users need to upload their data (*e.g.*, content retrieval history and geographic information of users) to a central server for processing. It brings the risk of privacy for users. 2) Mobility: Users frequently move from one edge node to another. This means that the cached contents at one edge node might become obsolete after users move out, while another edge node does not cache the contents for the incoming users. The lack of consideration of users mobility may lead to low cache efficiency. 3) Utilisation: The redundant contents may be stored in edge nodes, which lacks the global optimization of cache resource utilisation. It is non-trivial to decide how and where to cache, given the limited cache sizes at the different level of edge nodes.

To tackle the above challenges, this chapter proposes a federated learning based cooperative hierarchical edge caching scheme (FLCH). It enables to make the intelligent decision for caching contents at the edge while protecting users' privacy. The FLCH scheme trains a shared global learning model under the coordination of a central server with training data distributed over a number of users. Only the parameters of the model are uploaded to the central server, instead of users' private data [8]. The FLCH scheme consists of three tiers.

The bottom tier contains users equipped with smart devices. The middle tier includes F-APs with small cache storages. The top tier has a BBU pool with large cache storage that BBUs are located at the network centre. BBU is a unit to process baseband. The proposed hierarchical caching architecture achieves the better utilisation of available caches with mobile users, since the requested contents can be fetched from the local F-AP, neighbour F-APs and BBU.

The major contributions of this work are as follows:

1) A federated learning based caching scheme is proposed to make caching decisions by analysing the content retrieval history and context information of users. In this caching scheme, the training data is left locally on the user (*e.g.*, smart devices) and a shared global model is learnt by aggregating locally-computed updates from users. This approach can reduce privacy risk significantly.

2) A hierarchical cooperative caching architecture is designed to leverage horizontal cooperation between the F-APs and vertical cooperation between the BBU pool and F-APs to enhance the overall caching performance and global cache resource utilisation.

3) The proposed method integrates the appended stacked autoencoder and one-class collaborative filtering to predict the popularity of contents. The appended stacked autoencoder is used to extract the hidden representations of users and contents. Whereas, the one-class collaborative filtering is utilised to effectively process the input data for a better recommendation of popular contents.

4) The experimental results show that the proposed method outperforms other four reference algorithms (*e.g.*, m-ε-greedy and thompson sampling) on cache hit ratio by using real-world dataset. Moreover, the FLCH-based hierarchical caching scheme further improves caching hit ratio while preserving user privacy.

The remainder of this chapter is structured as follows: Section 6.2 presents the system architecture of the proposed FLCH caching scheme. The detailed implementation of the

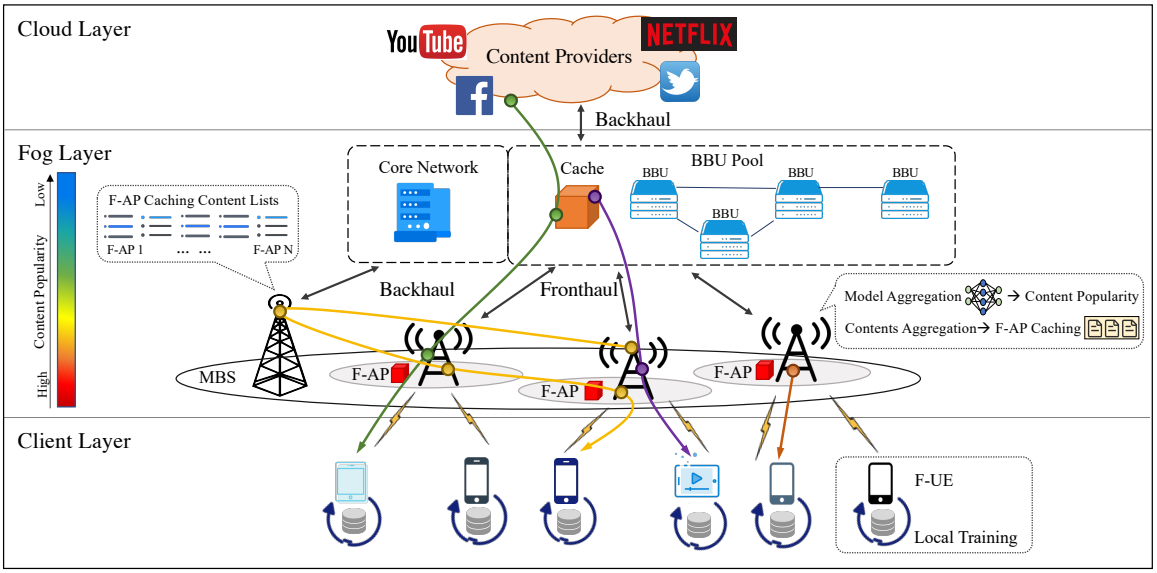


Fig. 6.1 System Architecture of the Federated Learning based Cooperative Hierarchical Caching for F-RANs.

FLCH is presented in Section 6.3. Section 6.4 provides the performance evaluation of the FLCH. Finally, Section 6.5 concludes this chapter.

6.2 System Architecture

Fig. 6.1 illustrates the system architecture of our proposed federated learning based edge caching scheme in F-RANs. The F-RANs integrate fog computing with radio access networks (RANs), which evolved from cloud radio access networks (C-RANs) [119]. In C-RANs, a crowd of RRHs are deployed distributively within a particular region and are connected to a centralised BBU pool via high-bandwidth fronthaul links. RRH is a remote radio transceiver to connect an operator radio control panel. A BBU pool greatly reduces power consumption and largely improves resource utilisation, but large data transmission also places a heavy burden on the fronthaul at the same time, resulting in the high service latency for users. F-RANs address the above issues of C-RANs through extending caching and computing

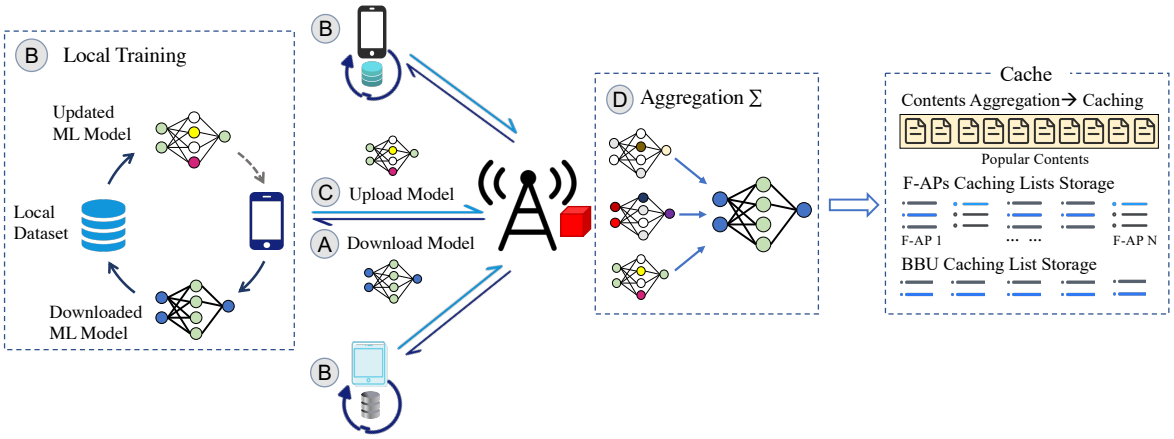


Fig. 6.2 Federated Learning Process for Edge Caching

functions to the edge of the network. For example, F-APs, which evolved from traditional RRH, are equipped with caches in F-RANs and are able to manage the caches flexibly.

As shown in Fig. 6.1, we consider a three-level User-Fog-Cloud hierarchical structure. The User level distributes users who are equipped with smart devices (F-UE). The Fog level consists of two tiers. The upper tier contains a BBU pool, while a Macro Base Station (MBS) and some F-APs are located in the lower tier. The MBS and F-APs all connect to the BBU pool. More specifically, the F-APs are connected to the BBU pool via high-speed fronthaul links. The MBS connects to the BBU pool with a reliable backhaul link. The MBS is responsible for delivering the overall control signalling and providing seamless coverage for F-UEs. The Cloud level is the Internet. Content providers are normally located at Cloud to provide contents to users. In F-RANs, the BBU pool and F-APs are equipped with caches. We assume that the storage capacity for the BBU and F-APs are up to N and M contents, respectively, where $N \geqslant M$.

The primary goal of our work is to take full advantage of distributed caching storages at the F-RANs. It requires to know content popularity distribution. However, content popularity is subject to change dynamically due to the mobility of users. Different users prefer different contents since the preference of users are various. The preference of users may link to many factors, such as age, gender and occupation. Even the location of connected users,

the connected time of day and the type of connected equipment device also influence the preference of users and further affect the content popularity. Hence, the popularity of contents changes according to the fluctuating users and their context. In our designed caching scheme, the content popularity decides what and where to cache at edge nodes.

Federated learning (FL) is an alternative distributed machine learning approach, which proposes to train a high-quality model distributively without gathering the data from users. In our proposed FL based caching scheme, each user associated with an F-AP performs local training using its own data. The parameters of the training model from these users are then aggregated at the F-AP to jointly learn a model that is used to predict the local content popularity, as shown in Fig. 6.1. Based on the predicted content popularity, each user uploads a list of popular contents to the local F-AP. The F-AP aggregates the lists from each connected user and constructs an aggregated list of local popular contents that will be uploaded to the MBS. The M highly popular contents are selected to cache at F-APs, to provide local caching services to their associated users. By contrast, the BBU pool caches the N less popular contents. Based on the knowledge of the content popularity from F-APs, the place of caching (at F-APs or BBU pool) can be decided, as described in the next subsection.

The cache hit ratio is used to evaluate the performance of caching schemes, which describes the percentage of content requests that can be served by cache. The proposed FLCH aims to maximise the cache hit ratio by leveraging the vertical cooperation (V-FLCH) between the BBU pool and F-APs, and the horizontal cooperation (H-FLCH) between the local F-AP and neighbour F-APs. Users can fetch the requested content not only from the cache of the local F-AP but also from the neighbour F-APs and the BBU pool. Specifically, as shown in Fig. 6.1, a user who is located in the coverage area of the F-AP can send the content request to the local F-AP. The requested content will be firstly searched whether it is stored in its cache [120]. If so, the requested content would be directly delivered to the user. Otherwise, the request will be searched in a caching content list for all neighbour F-APs

maintained. The caching content list of each F-AP is stored at the cache manager in the MBS. According to the list, if the requested content is cached at any other F-APs, the found F-AP would transmit the requested content to the user. In this way, the user does not need to connect the BBU pool to fetch the requested content, which relieves the burden on fronthaul between the BBU pool and F-APs. If the requested content is not stored in any F-APs, the local F-AP will send the request to the BBU pool. Upon receiving a request from the F-AP, it conducts a search to look for the content. If the content can be found in the BBU pool, the BBU pool delivers the corresponding content to the requesting F-AP via fronthaul link. The requested content can be provided by caching at either F-APs or the BBU pool, therefore, it greatly reduces the traffic between the core network and the cloud. However, if the requested content is neither cached at F-APs nor BBU pool, the request would be forwarded to the Internet to obtain the content from the source (*i.e.*, content provider) in the cloud.

6.3 Cooperative Hierarchical Edge Caching Scheme

In this section, the details of the proposed federated deep learning models for edge caching are presented. It consists of three parts: FL, appended stacked autoencoder (aSAE) and one-class collaborative filtering. FL framework is utilised to train the content popularity model while protecting users' privacy. The content popularity model we applied in FL is aSAE, which can obtain the relationship between users and contents, and find its hidden representations. The historical requests of users together with their contextual information (*e.g.*, time, location, age), as the training data for the aSAE, are exploited to learn the popularity of content in order to make smart caching decisions. The training data is binary where 1 (positive examples) and 0 (negative examples) represent interested contents and uninterested contents for users, respectively, but the negative examples are often absent. Marking all missing examples as the negative examples is a bias prediction. One-class collaborative filtering with a random sampling mechanism is used to correct this bias.

6.3.1 Federated Learning for Cooperative Hierarchical Edge Caching

For most of the existing learning-based caching schemes, a central server gathers users' data for the purpose of training. However, some of the users' data include sensitive and private information, such as age, gender, locations. Uploading these information to a central server presents a risk of breaching users' privacy. FL leverages the local data of users and computation capacity of their devices to perform distributed training. In the FL setting, multiple communication rounds are run to achieve a high-quality model. In each communication round, the F-AP firstly distributes the current model to participating mobile users. These users compute an update to the current model independently (*e.g.*, taking several iterations of gradient descent) by using their local data. Next, the model updates from each participating user is uploaded to the associated F-AP where all updates are aggregated to generate a new shared global model by the federated average method [8]. The federated average method we used is a weighted average, where the weight of a user depends on the data size of user. The value of data size will be sent to the server from users at the beginning FL communication round. That means a user with more data accounts for more contribution to the shared global model.

As shown in Fig. 6.2, a FL communication round executes at an F-AP and its associated users that consists of the following four steps [26]:

Step A: The users who are plugged-in and have good network connection are selected to participate in the federated communication round. Each of them downloads the current global model from the associated F-AP.

Step B: Using the local dataset, each participating user computes the updates of the downloaded global model.

Step C: The calculated model updates are uploaded to the F-AP from participating users.

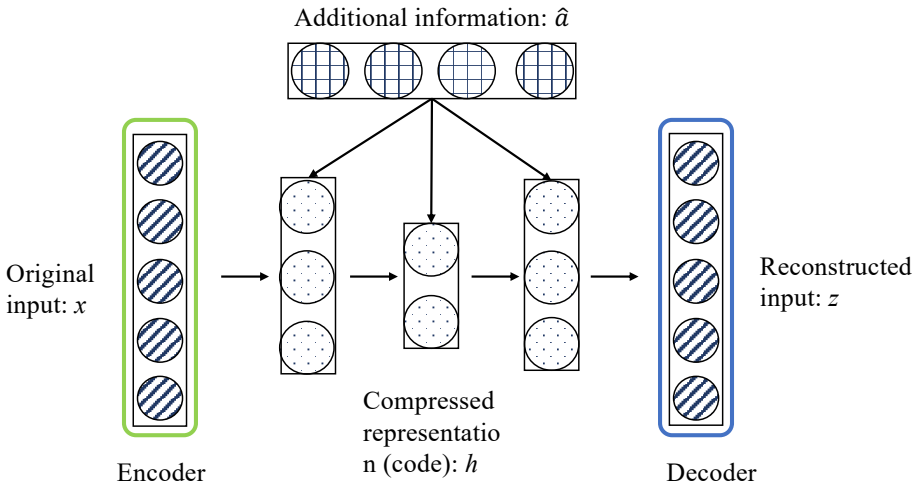


Fig. 6.3 Appended Stacked AE with One-Class Collaborative Filtering

Step D: The F-AP aggregates all model updates to construct an improved global model.

Compared to the traditional centralised learning approach, the distinct advantage of FL is to reduce privacy and security risks, since uploading model updates is much more secure than uploading the data of users. The integrated model of the appended stacked autoencoder and one-class collaborative filtering (aSAE-CF) is the model we trained in FL. The next section will provide the details of this proposed aSAE-CF learning model.

6.3.2 Stacked Autoencoder with One-Class Collaborative Filtering

The appended stacked autoencoder (aSAE) [121] is a neural network that attempts to copy its input to its output, aiming to learn a compressed hidden representation from the input. Fig. 6.3 shows the structure of the aSAE, mapping the input x to the output z which can be called as reconstructed input through the compressed representation (code) h. The aSAE consists of an encoder, a decoder and an additional information part. The function of encoder is $h = f(x)$, mapping x to h. A decoder describes a reconstruction $z = g(h)$, mapping h to z. As our proposed cache scheme is a context-aware scheme, it not only utilises the content

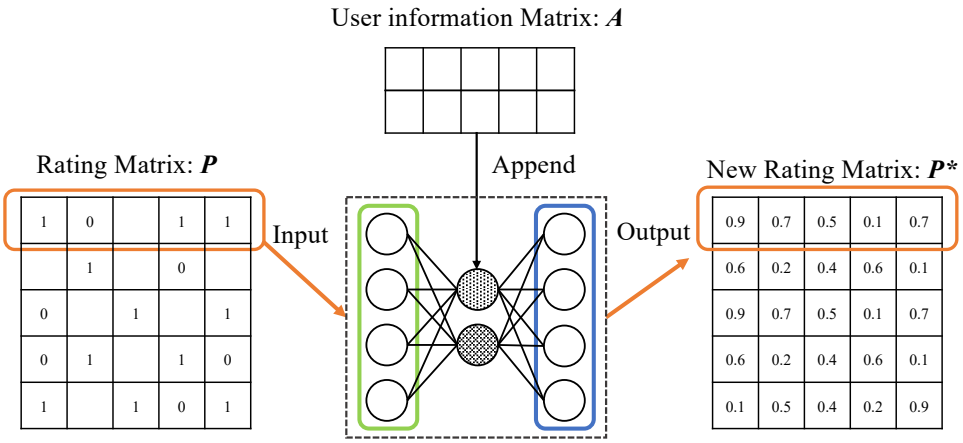


Fig. 6.4 One-Class Collaborative Filtering

retrieval history of users but also considers the context of users to learn the content popularity. Furthermore, the context information of users $\hat{a}$ (*e.g.*, gender, age and occupation), as an additional information, appends to each layer of the stacked autoencoder to form the aSAE [121], as shown in Fig. 6.3. Additionally, we apply Adam stochastic optimisation method [122] to optimise aSAE.

As shown in Fig. 6.4, the user-by-content rating matrix P, as the input matrix, is composed of the requested content history for a user. The rating matrix consists of binary data (1 and 0), reflecting the request behaviour of the user. The value of 1 indicates that the user requested this content before (positive example). The value of 0 represents that the user is not interested in this content (negative example) or the content is unknown to this user (missing example). However, the missing examples may include some unmarked positive examples, as the user may be interested in some content without knowing it. Marking the rest of contents as 0 leads to that all negative examples and missing examples are mixed, which makes it difficult to distinguish them [96].

We use one-class collaborative filtering based random sampling mechanism to mark the negative examples, instead of marking the rest of the contents as 0. In our cache scheme, we mark the negative examples in the rating matrix based on a probability related to the request

behaviour of the user. If the user requested a lot of contents before, but he/she never asks for one content, then this content may be unpopular and has a high probability of being a negative example (marked). Additionally, if one content has not been requested by other users often, it also has a high probability to be marked as a negative example. After the sampling is completed, the marked negative examples are considered as unpopular contents for users, while the unmarked examples are regarded as the missing examples. Thus, the input matrix has 1, 0 values and missing values, while the output matrix P^* is filled with predicted ratings for those missing values. Moreover, all context information of users construct a user information matrix A. Appending the matrix of user information A to each hidden layer can reduce the sparsity of the input to achieve better accuracy of the content popularity. The contents with the highest ratings in the output matrix are the contents with high popularity, which are recommended as caching contents. All the recommended content lists are stored in the MBS and the F-AP caches the most popular contents. The less popular contents are stored at the BBU pool. The designed aSAE minimises the reconstruction error between the input matrix and the output matrix via one-class collaborative filtering, in order to improve the accuracy of content popularity prediction.

6.4 Experimental Results

In this section, we describe the simulation experiment environment and investigate the performance of the proposed FLCH caching scheme using real-world datasets. The dataset we used to evaluate FLCH is MovieLens 1M [98], which is collected by GroupLens Research. It contains about 1 million ratings of 3883 movies from 6040 users. UserID, MovieID, Rating, Timestamp and user information (such as gender, age, Zip-code and occupation) are included in the dataset. In our experiments, MovieLens dataset is divided into 10 users. To be specific, the dataset for each user consists of 604 users' data in the MovieLens' dataset.

We firstly compared FLCH with four reference algorithms which are Oracle, Random, m-ε-Greedy and Thompson Sampling. We then evaluate the performance of hierarchical caching with cooperation between the BBU pool and F-APs. The four reference algorithms are described as following:

- **Oracle**: Oracle algorithm gets the best cache hit ratio, as it has the perfect knowledge of users' requests.
- **Random**: Random algorithm randomly selects the contents to cache.
- **m-ε-Greedy**: m-ε-Greedy algorithm [123] is one of the multi-armed bandit algorithms. The contents with the highest m requests will be cached with the probability of (1 - ε), while the algorithm will randomly choose contents to cache, with the probability ε (0 $< \varepsilon <$ 1).
- **Thompson Sampling**: Thompson Sampling [124] is another multi-armed bandit algorithm. The contents with the highest reward will be cached, while the reward for each content follows the beta distribution.

A real scenario is considered in this paper that users have unbalanced data due to their different activities and behaviours.

We investigate the impact of cache size on the cache hit ratio for an F-AP without the hierarchical cooperative caching (N-FLCH). The cache size of each F-AP varies from 50 to 400 contents. Fig. 6.5 shows that the cache hit ratios of all algorithms rise with the cache size increasing. Oracle algorithm knows the perfect prior knowledge of users' requests, so it gets the best cache hit ratio. By contrast, Random algorithm gives the lowest cache hit ratio. m-ε-Greedy achieves better performance than Thompson Sampling. It is due to the fact that m-ε-Greedy studies the past user requests while Thompson Sampling follows the beta distribution without considering the request history from users. N-FLCH outperforms

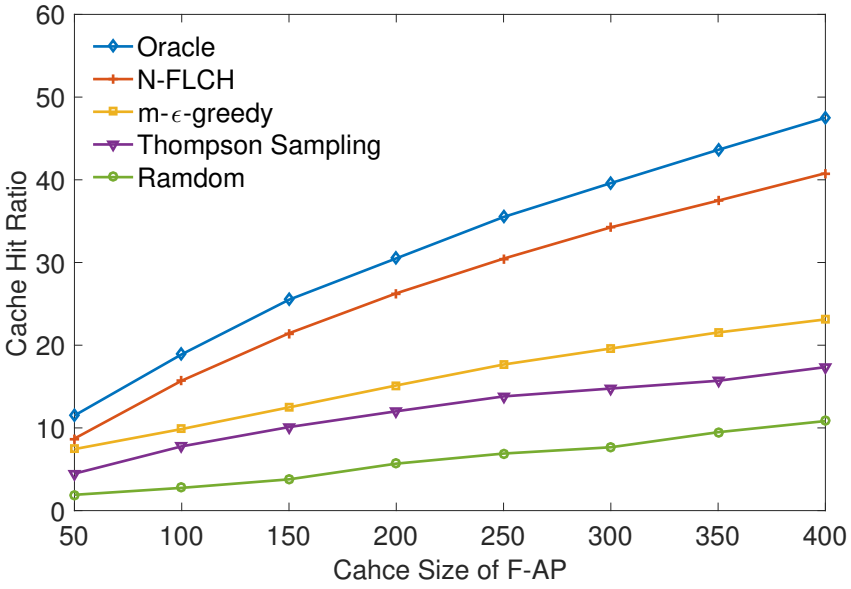


Fig. 6.5 Cache hit ratio: N-FLCH vs Reference caching schemes

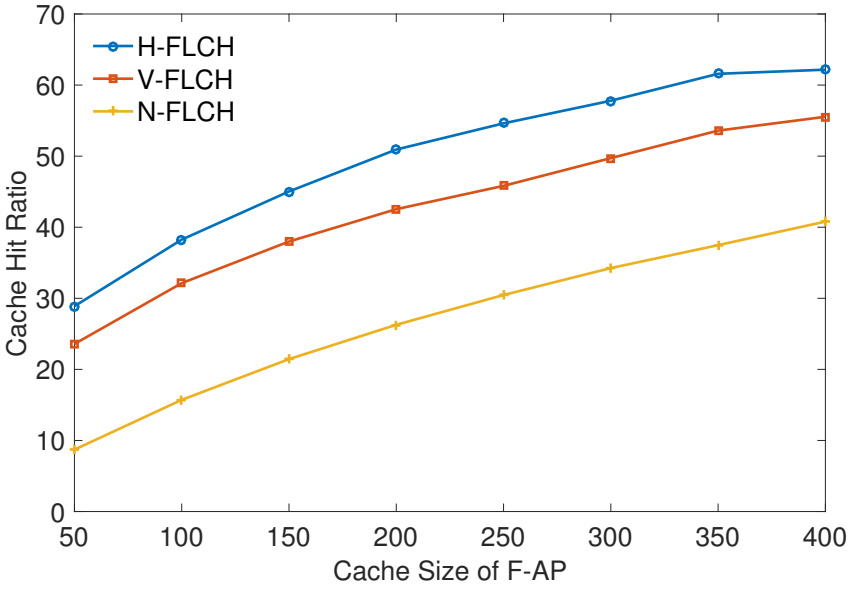


Fig. 6.6 Cache hit ratio of the FLCH with different cooperative strategies (H-FLCH, V-FLCH, N-FLCH)

m-ε-Greedy and Thompson Sampling. The reason is that N-FLCH considers both the request history of users and their context information.

Fig. 6.6 reveals that the higher cache hit ratio can be achieved by using the hierarchical cooperative caching mechanism. N-FLCH only caches contents at F-APs, which gives the

lowest cache hit ratio compared to the hierarchical cooperative caching schemes V-FLCH and H-FLCH. As shown in Fig. 6.6, the cache hit ratio of H-FLCH is higher than V-FLCH. In H-FLCH, if the requested content is not stored in the local F-AP, the request will be sent to neighbour F-APs, which will deliver the content to the requester if found in their caches. V-FLCH allows a content to be fetched from the BBU cache through cooperation between BBU and F-APs, which obtains the highest cache hit ratio as expected. When the cache size of F-APs is 50, the cache hit ratios for N-FLCH, H-FLCH and V-FLCH reach 9%, 23% and 29%, respectively. With the cache size of F-AP increasing to 400, the cache hit ratios of N-FLCH, H-FLCH and V-FLCH are 41%, 55% and 62%, respectively.

6.5 Summary

In this chapter, a novel federated learning based cooperative hierarchical caching scheme (FLCH) for F-RANs is proposed. FLCH protects users' privacy by utilising the emerging federated learning framework. Through integrating the appended stacked autoencoder and one-class collaborative filtering, FLCH is able to predict the context-specific content popularity by utilising users' request history and their context information. To further enhance the cache hit ratio and reduce users' latency, FLCH leverages the vertical and horizontal cooperations between the BBU pool and F-APs. The experimental results demonstrate that FLCH achieves higher cache hit ratio than other learning-based caching schemes, including m-ε-Greedy and Thompson Sampling. Moreover, the proposed hierarchical cooperative caching mechanism can further improve caching performance. Our work is a valuable step towards intelligent and secure future networks.

Chapter 7

Conclusions and Future Work

With the increasing complexity of traffic, the proliferation of smart devices, and the diversification of user requests, the load accumulation has aggravated the service pressures of the current wireless networks. Edge caching has been considered as a promising approach to reduce service latency, alleviate backhaul link congestion and satisfy the strict performance requirements (*e.g.*, low-latency) of emerging applications. Due to the limited cache capability, the caching scheme need to be identified. Recent breakthroughs in ML facilitate the learning-based caching scheme, which can effectively extract hidden features and representations from users' data. These can be used to accurately predict content popularity. FL is utilised to fitting ML into the edge of networks and protect user privacy. In the following, a summary of the research in this thesis is provided and several directions of future work are presented.

7.1 Conclusions

This thesis has presented new learning-based proactive caching schemes and federated learning frameworks in 5G, IoV and F-RANs. To be specific, a communication-efficient federated deep learning for proactive caching scheme is designed to reduce communication cost in Chapter 3. To support high mobility of users, a mobility-aware caching scheme based

on federated learning is exploited in Chapter 4. To ease reliance on the fixed central server, eliminate the issue of hand-over between RSUs, a peer-to-peer federated learning based caching scheme is described in Chapter 5. To maximise the utilisation of available caches with edge nodes, a cooperative caching scheme based on federated learning is developed in Chapter 6. The major achievements in this thesis are summarised as follows:

- A communication Efficient Federated learning based Proactive content Caching scheme (EFPC) is designed. Based upon the federated learning framework, each user locally trains a shared model by using their own data, and only uploads the parameters of the model to the edge server for aggregation. This process can greatly protect users' privacy and largely reduce communication costs. In EFPC, we exploit the one-class collaborative filtering based variational autoencoder model to predict content popularity and then utilise it to make caching decisions to improve the cache hit ratio. To further reduce communication costs, the 3LC data compression scheme is used in EFPC to compress the upload parameters of the model. The performance of EFPC is evaluated via experiments on a networking testbed with two real-world datasets. The performance evaluation demonstrates that EFPC outperforms other caching schemes such as LRU and m-ε greedy on cache hit ratio, and achieves a data compression ratio up to 20-280$\times$.

- A Mobility-aware Proactive edge Caching scheme based on Federated learning (MPCF) is exploited. This new scheme enables multiple vehicles to collaboratively learn a global model for predicting content popularity with the private training data distributed on local vehicles. MPCF also employs a Context-aware Adversarial AutoEncoder to predict the highly dynamic content popularity. Besides, MPCF integrates a mobility-aware cache replacement policy, which allows the network edges to add/evict contents in response to the mobility patterns and preferences of vehicles. MPCF can greatly improve cache performance, effectively protect users' privacy and significantly reduce

communication costs. Experimental results demonstrate that MPCF outperforms other baseline caching schemes in terms of the cache hit ratio in vehicular edge networks.

- A peer-to-peer federated learning based proactive caching scheme (PPFC) is proposed, where the global prediction model is trained from data scattered at vehicles to mitigate the privacy risks. In our proposed scheme, a vehicle acts as a parameter server to aggregate the updated global model from peers, instead of an edge node. A dual-weighted aggregation scheme is designed to achieve high global model accuracy. Moreover, to enhance the caching performance, a Collaborative Filtering based Variational AutoEncoder model is developed to predict the content popularity. The experimental results demonstrate that our proposed caching scheme largely outperforms typical baselines, such as Greedy and Most Recently Used caching.

- A Federated Learning based Cooperative Hierarchical Caching scheme (FLCH) is developed, which keeps users' data locally and employs users' devices to train a shared learning model for content popularity prediction. FLCH exploits horizontal cooperation between neighbour F-APs and vertical cooperation between the baseband unit (BBU) pool and F-APs to cache contents with different degrees of popularity. The simulation experiments are conducted to evaluate the performance of FLCH using real-world datasets. The results demonstrate that FLCH outperforms certain learning-based caching schemes in terms of the cache hit ratio, without aggregating users' data centrally. Moreover, the results show the effectiveness of the cooperative hierarchical caching mechanism for FLCH.

7.2 Future Work

7.2.1 Hybrid Caching Scheme

The gain from proactive caching highly depends on the accurate estimation of the content popularity. Unwanted variation always exists during the training process. Moreover, both noisy data quality and rapid changing circumstance can reduce the accuracy of prediction. Thus, reducing the prediction errors and limiting the impact of unavoidable ones deserve researching efforts.

Federated learning based method to predict future request patterns only expect to implement for limited times per day at a non-peak time, in order to provide convenience to users and reduce communication costs. Due to the popularity of contents continuously changes and new popular contents generate, the assumption of content request patterns becomes invalid over time. Thus, the caching content at the cache should be updated dynamically to response the sudden changing request patterns during the peak time. Reactive caching is based on the recent locally observed request pattern to decide which contents need to be cached, which can react the sudden changing quickly.

An adaptive cache replacement policy should be designed that is interoperable with traditional reactive cache policy, such as LRU. The designed cache replacement policy is demonstrated in Fig. 7.1. The caching storage can be separated into two parts. The first part is the fixed caching area where the cached contents are predicted by the ML based caching scheme. Knowing content popularity in advance can proactively make decisions for storing the N popular contents in order. The adaptive replacement policy will not implement in this area, which avoids to evict the future popular contents. The second part is the flexible caching area that the replacement policy adapts to update the content, in order to discover the new popular contents. The priority queue is implemented in flexible area, which follows the LRU policy to cache and evict the content effectively at set intervals. We consider the

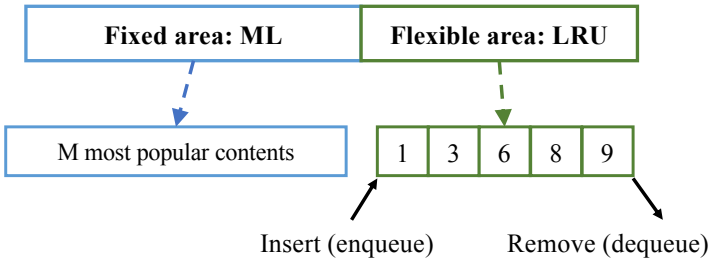


Fig. 7.1 Hybrid Caching Scheme

capacity of cache is N contents and a tunable parameter Q is to decide the size of these two parts. We assume that Q and $N-Q$ represent the fixed area size and flexible area size, respectively. Using this adaptive method to determine the caching contents would adapt new popular contents and leverage the predicted contents to increase cache hit ratio. It conducts an adaptive trade-off between predicted results and recency of users' requests.

7.2.2 Cooperative and Hierarchical Federated Learning

A hierarchical FL scheme in 5G and Beyond (5GB) system should be developed for conducting machine learning in resource-constraint UAV devices and preserving data privacy. The traditional machine learning algorithms in 5GB AEC system require the uploading of data in access devices to the UAV, which may cause privacy issues as the collected data usually contain the personal behaviour or information. In addition, it is not efficient or even feasible to store all the data and perform model training on UAV devices as the storage and computation resources are limited. To address these challenges, we proposed a cooperative and hierarchical FL scheme as shown in Fig. 7.2, which has the following four stages,

- The first stage: A baseline model is trained in the cloud based on the common dataset, and then the model compression technologies are used to reduce the complexity of baseline model to make it deployable on resource constrained end devices, UAV and edge servers.

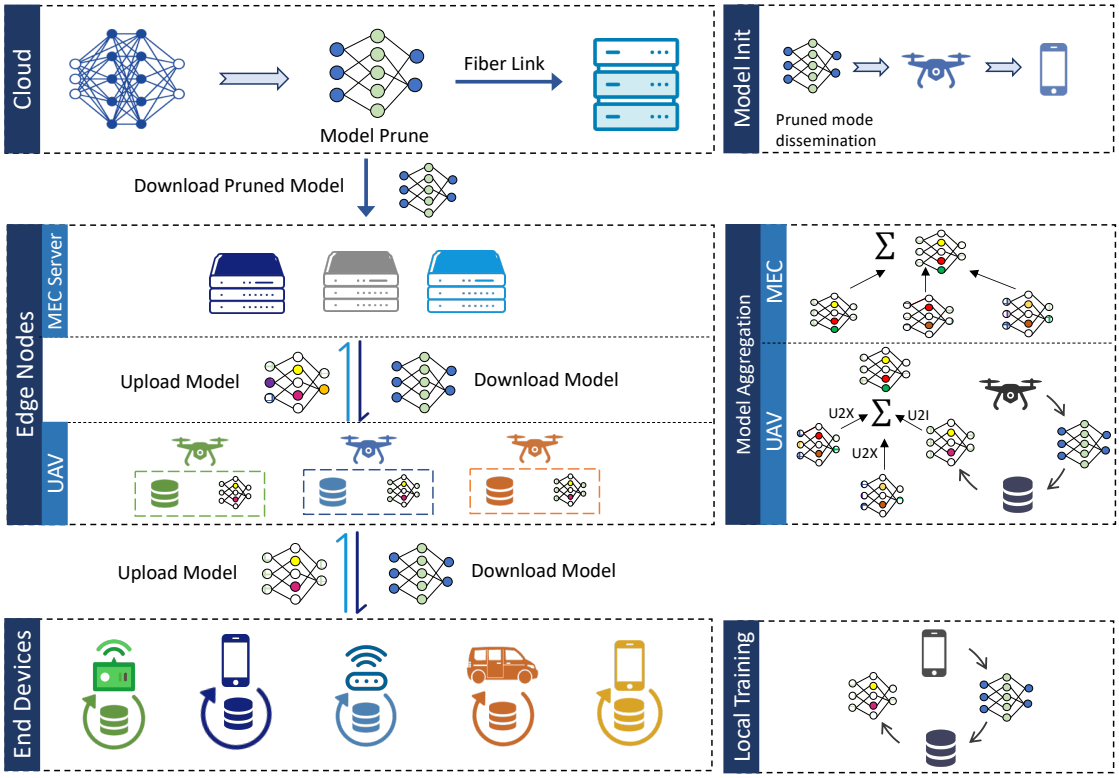


Fig. 7.2 Hierarchical Federated Learning

- The second stage: After receiving the pruned model, an updated model will be trained at the end device based on the local dataset, which will be uploaded to UAV through U2X wireless channel.

- The third stage: With the prune model, similar to end devices, UAV trains a local model based on the local dataset. Once receiving the models from end devices, UAV aggregates the received models and the local model and send the aggregated model to MEC server through U2I link for the global aggregation.

- The fourth step: The MEC server firstly pushes the pruned model to UAV and end devices through V2I and V2X links. When receiving the updated models from UAVs, MEC server will aggregates them to generated a global model, which will be sent back to UAVs through U2I link for the next round model learning.

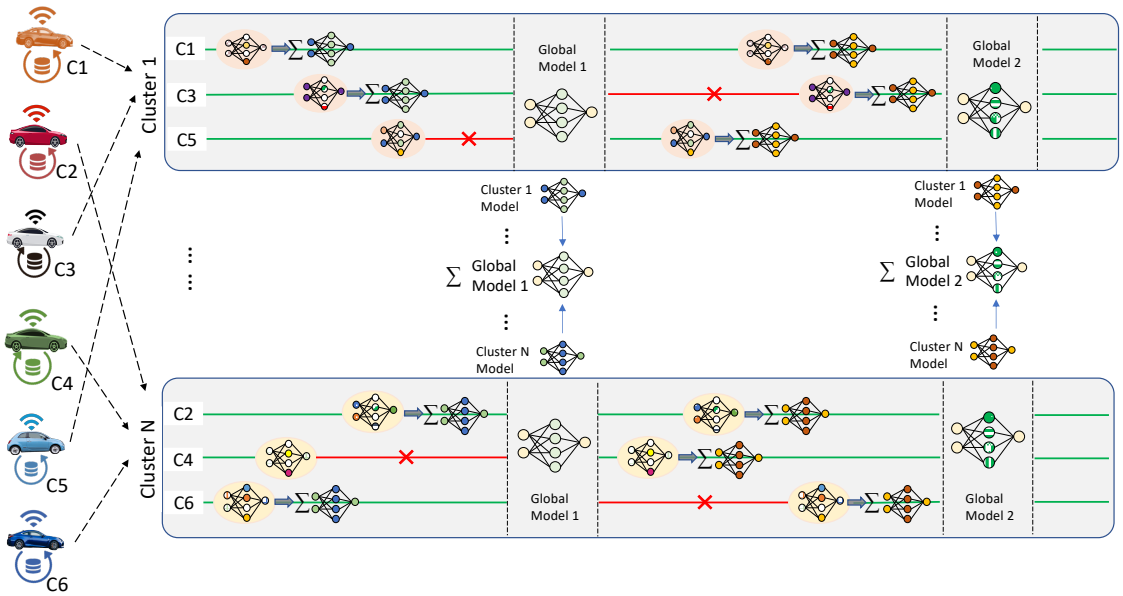


Fig. 7.3 Asynchronous Federated Learning

7.2.3 Asynchronous Federated Learning

Mobile devices or vehicles may be offline or lag behind or dropout, due to the heterogeneous communication and computing capabilities. As a result, the synchronized federated learning frameworks may be extremely slow. To cope with this challenge, an asynchronous federated learning framework needs to be designed.

As shown in Fig. 7.3, the asynchronous federated learning consists of three parts. The first part is hierarchical clustering of local updates. Based on the similarity of users' local updates, users will be separated to few clusters. Once separated, the updates in each cluster will be aggregated firstly in an Asynchronous manner. Each cluster starts to update model parameters after receiving one to several users' updates, without waiting for all users in this cluster to finish. Next, the global model are constructed by aggregating models from all clusters synchronously. In this way, the influence of users who dropout or lag behind during the training process can be reduced. Because updates in each cluster are similar. Although some users fail to upload their updates, others in this cluster upload successfully. The expected updates still can be achieved. However, if in a cluster, no one uploaded successfully.

In this situation, this cluster will continue to use the previous aggregated updates to attend the global aggregation and a hyperparameter will be introduced. It is used to give the lower weights for global aggregation. The method of aggregation I used is weighted averaging. The third part in the designed FL is the residual-based parameter transmission. It is like a method of model compression to reduce the communication costs. I will calculate the similarity between new calculated updates and the previous one. If they are very close, the new updates will not upload to the associated cluster. Otherwise, only different part will be uploaded to the cluster.

References

[1] GMDT Forecast. Cisco visual networking index: global mobile data traffic forecast update, 2017–2022. *Update*, 2017:2022, 2019.

[2] Quoc-Viet Pham, Fang Fang, Vu Nguyen Ha, Md Jalil Piran, Mai Le, Long Bao Le, Won-Joo Hwang, and Zhiguo Ding. A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access*, 2020.

[3] Jingjing Yao, Tao Han, and Nirwan Ansari. On mobile edge caching. *IEEE Communications Surveys & Tutorials*, 21(3):2525–2553, 2019.

[4] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*, pages 48–53. ACM, 2018.

[5] Mohammed S ElBamby, Mehdi Bennis, Walid Saad, and Matti Latva-Aho. Content-aware user clustering and caching in wireless small cell networks. In *IEEE International Symposium on Wireless Communications Systems, ISWCS*, pages 945–949. IEEE, 2014.

[6] Ejder Baştuğ, Mehdi Bennis, and Mérouane Debbah. A transfer learning approach for cache-enabled wireless networks. In *2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 161–166. IEEE, 2015.

[7] Ejder Bastug, Mehdi Bennis, and Mérouane Debbah. Living on the edge: The role of proactive caching in 5g wireless networks. *IEEE Communications Magazine*, 52(8):82–89, 2014.

[8] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *Proc. of 2016 AISTATS*, pages 1–10.

[9] Valentin Touzeau, Claire Maïza, David Monniaux, and Jan Reineke. Fast and exact analysis for lru caches. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019.

[10] Anselme Ndikumana, Nguyen H Tran, Ki Tae Kim, Choong Seon Hong, et al. Deep learning based caching for self-driving cars in multi-access edge computing. *IEEE Transactions on Intelligent Transportation Systems*. doi:10.1109/TITS.2020.2976572, 2020.

[11] Haoye Chai, Supeng Leng, Ming Zeng, and Haoyang Liang. A hierarchical blockchain aided proactive caching scheme for internet of vehicles. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[12] Binbin Hu, Luoyang Fang, Xiang Cheng, and Liuqing Yang. In-vehicle caching (iv-cache) via dynamic distributed storage relay (d2sr) in vehicular networks. *IEEE Transactions on Vehicular Technology*, 68(1):843–855, 2018.

[13] Yin Zhang, Ranran Wang, M Shamim Hossain, Mohammed F Alhamid, and Mohsen Guizani. Heterogeneous information network-based content caching in the internet of vehicles. *IEEE Transactions on Vehicular Technology*, 68(10):10216–10226, 2019.

[14] Guanhua Qiao, Supeng Leng, Sabita Maharjan, Yan Zhang, and Nirwan Ansari. Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks. *IEEE Internet of Things Journal*, 7(1):246–257, 2019.

[15] Jiayin Chen, Peng Yang, Feng Lyu, Xuemin Shen, et al. Cooperative edge caching with location-based and popular contents for vehicular networks. *IEEE Transactions on Vehicular Technology*, 2020.

[16] Ahsan Mahmood, Claudio Ettore Casetti, Carla Fabiana Chiasserini, Paolo Giaccone, and Jérôme Härri. The rich prefetching in edge caches for in-order delivery to connected cars. *IEEE Transactions on Vehicular Technology*, 68(1):4–18, 2018.

[17] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM*, volume 1, pages 126–134. IEEE, 1999.

[18] Shailendra Rathore, Jung Hyun Ryu, Pradip Kumar Sharma, and Jong Hyuk Park. Deepcachnet: A proactive caching framework based on deep learning in cellular networks. *IEEE Network*, 33(3):130–138, 2019.

[19] Tingting Hou, Gang Feng, Shuang Qin, and Wei Jiang. Proactive content caching by exploiting transfer learning for mobile edge computing. *International Journal of Communication Systems*, 31(11):e3706, 2018.

[20] BN Bharath, Kyatsandra G Nagananda, and H Vincent Poor. A learning-based approach to caching in heterogenous small cell networks. *IEEE Transactions on Communications*, 64(4):1674–1686, 2016.

[21] Peng Cheng, Chuan Ma, Ming Ding, Yongjun Hu, Zihuai Lin, Yonghui Li, and Branka Vucetic. Localized small cell caching: A machine learning approach based on rating data. *IEEE Transactions on Communications*, 67(2):1663–1676, 2018.

[22] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[23] Michael R Sprague, Amir Jalalirad, Marco Scavuzzo, Catalin Capota, Moritz Neun, Lyman Do, and Michael Kopp. Asynchronous federated learning for geospatial applications. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 21–28. Springer, 2018.

[24] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[25] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *Proc. of 2019 ICC*, pages 1–7. IEEE, 2019.

[26] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[27] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[28] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[29] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. *arXiv preprint arXiv:1803.01113*, 2018.

[30] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. Staleness-aware async-sgd for distributed deep learning. *arXiv preprint arXiv:1511.05950*, 2015.

[31] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.

[32] Shaohan Feng, Dusit Niyato, Ping Wang, Dong In Kim, and Ying-Chang Liang. Joint service pricing and cooperative relay communication for federated learning. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 815–820. IEEE, 2019.

[33] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165, 2019.

[34] Jianji Ren, Haichao Wang, Tingting Hou, Shuai Zheng, and Chaosheng Tang. Federated learning-based computation offloading optimization in edge computing-supported internet of things. *IEEE Access*, 7:69194–69201, 2019.

[35] Yongfeng Qian, Long Hu, Jing Chen, Xin Guan, Mohammad Mehedi Hassan, and Abdulhameed Alelaiwi. Privacy-aware service placement for mobile edge computing via federated learning. *Information Sciences*, 505:562–570, 2019.

[36] Yuris Mulya Saputra, Dinh Thai Hoang, Diep N Nguyen, Eryk Dutkiewicz, Markus Dominik Mueck, and Srikathyayani Srikanteswara. Energy demand prediction with federated learning for electric vehicle networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[37] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Mérouane Debbah. Distributed federated learning for ultra-reliable low-latency vehicular communications. *IEEE Transactions on Communications*, 68(2):1146–1159, 2019.

[38] Dongdong Ye, Rong Yu, Miao Pan, and Zhu Han. Federated learning in vehicular edge computing: A selective model aggregation approach. *IEEE Access*, 8:23920–23935, 2020.

[39] Yunlong Lu, Xiaohong Huang, Ke Zhang, Sabita Maharjan, and Yan Zhang. Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. *IEEE Transactions on Vehicular Technology*, 69(4):4298–4311, 2020.

[40] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. Differentially private asynchronous federated learning for mobile edge computing in urban informatics. *IEEE Transactions on Industrial Informatics*, 16(3):2134–2143, 2019.

[41] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv preprint arXiv:1905.06731*, 2019.

[42] Ribel Fares, Brian Romoser, Ziliang Zong, Mais Nijim, and Xiao 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*, pages 227–234. IEEE, 2012.

[43] Mohammad Ali Maddah-Ali and Urs Niesen. Fundamental limits of caching. *IEEE Transactions on Information Theory*, 60(5):2856–2867, 2014.

[44] Arif Can Güngör and Deniz Gündüz. Proactive wireless caching at mobile user devices for energy efficiency. In *International Symposium on Wireless Communication Systems, ISWCS*, pages 186–190. IEEE, 2015.

[45] Wei Jiang, Gang Feng, and Shuang Qin. Optimal cooperative content caching and delivery policy for heterogeneous cellular networks. *IEEE Transactions on Mobile Computing*, 16(5):1382–1393, 2016.

[46] Negin Golrezaei, Andreas F Molisch, Alexandros G Dimakis, and Giuseppe Caire. Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution. *IEEE Communications Magazine*, 51(4):142–149, 2013.

[47] Hye J Kang, Kown Y Park, Kumin Cho, and Chung G Kang. Mobile caching policies for device-to-device (d2d) content delivery networking. In *2014 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 299–304. IEEE, 2014.

[48] Konstantinos Poularakis, George Iosifidis, Vasilis Sourlas, and Leandros Tassiulas. Exploiting caching and multicast for 5g wireless networks. *IEEE Transactions on Wireless Communications*, 15(4):2995–3007, 2016.

[49] Xinwei Liu, Jiaxin Zhang, Xing Zhang, and Wenbo Wang. Mobility-aware coded probabilistic caching scheme for mec-enabled small cell networks. *IEEE Access*, 5:17824–17833, 2017.

[50] Jad Hachem, Nikhil Karamchandani, and Suhas Diggavi. Content caching and delivery over heterogeneous wireless networks. In *IEEE International Conference on Computer Communications, INFOCOM*, pages 756–764. IEEE, 2015.

[51] Vasilios A Siris, Xenofon Vasilakos, and George C Polyzos. Efficient proactive caching for supporting seamless mobility. *arXiv preprint arXiv:1404.4754*, 2014.

[52] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. Femtocaching: Wireless content delivery through distributed caching helpers. *IEEE Transactions on Information Theory*, 59(12):8402–8413, 2013.

[53] Zhijie Chen, Hoshyar Mohammed, and Wei Chen. Proactive caching for energy-efficiency in wireless networks: A markov decision process approach. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[54] Sabrina Müller, Onur Atan, Mihaela van der Schaar, and Anja Klein. Context-aware proactive content caching with service differentiation in wireless networks. *IEEE Transactions on Wireless Communications*, 16(2):1024–1036, 2017.

[55] Avik Sengupta, SaiDhiraj Amuru, Ravi Tandon, R Michael Buehrer, and T Charles Clancy. Learning distributed caching strategies in small cell networks. In *IEEE International Symposium on Wireless Communications Systems, ISWCS*, pages 917–921. IEEE, 2014.

[56] Sabrina Müller, Onur Atan, Mihaela van der Schaar, and Anja Klein. Smart caching in wireless small cell networks via contextual multi-armed bandits. In *IEEE International Conference on Communications, ICC*, pages 1–7. IEEE, 2016.

[57] Pol Blasco and Deniz Gunduz. Learning-based optimization of cache content in a small cell base station. In *IEEE International Conference on Communications, ICC*, pages 1897–1903. IEEE, 2014.

[58] SM Shahrear Tanzil, William Hoiles, and Vikram Krishnamurthy. Adaptive scheme for caching youtube content in a cellular network: Machine learning approach. *Ieee Access*, 5:5870–5881, 2017.

[59] Binqiang Chen and Chenyang Yang. Caching policy optimization for d2d communications by learning user preference. In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pages 1–6. IEEE, 2017.

[60] Milad Hashemi, Kevin Swersky, Jamie A Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. *arXiv preprint arXiv:1803.02329*, 2018.

[61] Khai Nguyen Doan, Thang Van Nguyen, Tony QS Quek, and Hyundong Shin. Content-aware proactive caching for backhaul offloading in cellular network. *IEEE Transactions on Wireless Communications*, 17(5):3128–3140, 2018.

[62] Khai Nguyen Doan, Thang Van Nguyen, Hyundong Shin, and Tony QS Quek. Socially-aware caching in wireless networks with random d2d communications. *IEEE Access*, 7:58394–58406, 2019.

[63] Lixin Li, Yang Xu, Jiaying Yin, Wei Liang, Xu Li, Wei Chen, and Zhu Han. Deep reinforcement learning approaches for content caching in cache-enabled d2d networks. *IEEE Internet of Things Journal*, 7(1):544–557, 2019.

[64] Kyi Thar, Nguyen H Tran, Thant Zin Oo, and Choong Seon Hong. Deepmec: Mobile edge caching using deep learning. *IEEE Access*, 6:78260–78275, 2018.

[65] Wei Wang, Ruining Lan, Jingxiong Gu, Aiping Huang, Hangguan Shan, and Zhaoyang Zhang. Edge caching at base stations with device-to-device offloading. *IEEE Access*, 5:6399–6410, 2017.

[66] Hao Zhu, Yang Cao, Wei Wang, Tao Jiang, and Shi Jin. Deep reinforcement learning for mobile edge caching: Review, new features, and open issues. *IEEE Network*, 32(6):50–57, 2018.

[67] Fan Jiang, Zeng Yuan, Changyin Sun, and Junxuan Wang. Deep q-learning-based content caching with update strategy for fog radio access networks. *IEEE Access*, 7:97505–97514, 2019.

[68] Wei Jiang, Gang Feng, Shuang Qin, Tak Shing Peter Yum, and Guohong Cao. Multi-agent reinforcement learning for efficient content caching in mobile d2d networks. *IEEE Transactions on Wireless Communications*, 18(3):1610–1622, 2019.

[69] FangYuan Lei, QinYun Dai, Jun Cai, HuiMin Zhao, Xun Liu, and Yan Liu. A proactive caching strategy based on deep learning in epc of 5g. In *International Conference on Brain Inspired Cognitive Systems*, pages 738–747. Springer, 2018.

[70] Wei Jiang, Gang Feng, Shuang Qin, and Ying-Chang Liang. Learning-based cooperative content caching policy for mobile edge computing. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[71] Jianbin Chuan, Li Wang, and Ruqiu Ma. Machine learning based popularity regeneration in caching-enabled wireless networks. In *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2019.

[72] Srikanth Bommaraveni, Thang X Vu, Satyanarayana Vuppala, Symeon Chatzinotas, and Björn Ottersten. Active content popularity learning via query-by-committee for edge caching. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 301–305. IEEE, 2019.

[73] Zhiwen Hu, Zijie Zheng, Tao Wang, Lingyang Song, and Xiaoming Li. Roadside unit caching: Auction-based storage allocation for multiple content providers. *IEEE Transactions on Wireless Communications*, 16(10):6321–6334, 2017.

[74] Ruizhou Ding, Tianyu Wang, Lingyang Song, Zhu Han, and Jianjun Wu. Roadside-unit caching in vehicular ad hoc networks for efficient popular content delivery. In *Proc. of 2015 WCNC*, pages 1207–1212. IEEE, 2015.

[75] Zhou Su, Yilong Hui, Qichao Xu, Tingting Yang, Jianyi Liu, and Yunjian Jia. An edge caching scheme to distribute content in vehicular networks. *IEEE Transactions on Vehicular Technology*, 67(6):5346–5356, 2018.

[76] Ahsan Mahmood, Claudio Casetti, Carla-Fabiana Chiasserini, Paolo Giaccone, and J Harri. Mobility-aware edge caching for connected cars. In *Proc. of 2016 WONS*, pages 1–8. IEEE, 2016.

[77] Neeraj Kumar and Jong-Hyouk Lee. Peer-to-peer cooperative caching for data dissemination in urban vehicular communications. *IEEE Systems Journal*, 8(4):1136–1144, 2013.

[78] Sangsha Fang and Pingzhi Fan. A cooperative caching algorithm for cluster-based vehicular content networks with vehicular caches. In *Proc. of 2017 Globecom GC Wkshps*, pages 1–6. IEEE, 2017.

[79] Gang Deng, Liwei Wang, Fengchao Li, and Rere Li. Distributed probabilistic caching strategy in vanets through named data networking. In *Proc. of 2016 INFOCOM WKSHPS*, pages 314–319. IEEE, 2016.

[80] Lin Yao, Ailun Chen, Jing Deng, Jianbang Wang, and Guowei Wu. A cooperative caching scheme based on mobility prediction in vehicular content centric networks. *IEEE Transactions on Vehicular Technology*, 67(6):5435–5444, 2017.

[81] Junchao Ma, Jiahuan Wang, Gang Liu, and Pingzhi Fan. Low latency caching placement policy for cloud-based vanet with both vehicle caches and rsu caches. In *Proc. of 2017 IEEE Globecom GC Wkshps*, pages 1–6. IEEE, 2017.

[82] Ke Zhang, Supeng Leng, Yejun He, Sabita Maharjan, and Yan Zhang. Cooperative content caching in 5g networks with mobile edge computing. *IEEE Wireless Communications*, 25(3):80–87, 2018.

[83] Sungjin Park, Seungmin Oh, Youngju Nam, Jaejeong Bang, and Euisin Lee. Mobility-aware distributed proactive caching in content-centric vehicular networks. In *Proc. of 2019 WMNC*, pages 175–180. IEEE, 2019.

[84] Yousef AlNagar, Sameh Hosny, and Amr A El-Sherif. Towards mobility-aware proactive caching for vehicular ad hoc networks. In *Proc. of 2019 WCNCW*, pages 1–6. IEEE, 2019.

[85] Mohamed E Gad, Sameh Hosny, Bassem Mokhtar, and Amr A El-Sherif. Hierarchical proactive caching for vehicular ad hoc networks. In *2019 Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, volume 1, pages 150–153. IEEE, 2019.

[86] Zhe Zhang, Chung-Horng Lung, Marc St-Hilaire, and Ioannis Lambadaris. Smart proactive caching: Empower the video delivery for autonomous vehicles in icn-based networks. *IEEE Transactions on Vehicular Technology*, 2020.

[87] Lu Hou, Lei Lei, Kan Zheng, and Xianbin Wang. A Q-learning-based proactive caching strategy for non-safety related services in vehicular networks. *IEEE Internet of Things Journal*, 6(3):4512–4520, 2018.

[88] Zihui Zhu, Zhengming Zhang, Wen Yan, Yongming Huang, and Luxi Yang. Proactive caching in auto driving scene via deep reinforcement learning. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE, 2019.

[89] Zhengming Zhang, Yaoqing Yang, Meng Hua, Chunguo Li, Yongming Huang, and Luxi Yang. Proactive caching for vehicular multi-view 3d video streaming via deep reinforcement learning. *IEEE Transactions on Wireless Communications*, 18(5):2693–2706, 2019.

[90] Cisco Visual Networking Index. Global mobile data traffic forecast update, 2017–2022 white paper. *Cisco: San Jose, CA, USA*, 2019.

[91] Gai Li, Zhiqiang Zhang, Liyang Wang, Qiang Chen, and Jincai Pan. One-class collaborative filtering based on rating prediction and ranking prediction. *Knowledge-Based Systems*, 124:46–54, 2017.

[92] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation. *arXiv preprint arXiv:1903.07424*, 2019.

[93] Aaron Howdle. Best broadband deals by upload speed.

[94] Hyeontaek Lim, David G Andersen, and Michael Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning. *arXiv preprint arXiv:1802.07389*, 2018.

[95] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

[96] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Proceedings of the 8th IEEE International Conference on Data Mining, ICDM*, pages 502–511. IEEE, 2008.

[97] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[98] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):19, 2016.

[99] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA., 2007.

[100] Sahil Garg, Kuljeet Kaur, Georges Kaddoum, Syed Hassan Ahmed, and Dushantha Nalin K Jayakody. SDN-based secure and privacy-preserving scheme for vehicular networks: A 5G perspective. *IEEE Transactions on Vehicular Technology*, 68(9):8421–8434, 2019.

[101] Lei Liu, Chen Chen, Qingqi Pei, Sabita Maharjan, and Yan Zhang. Vehicular edge computing and networking: A survey. *arXiv preprint arXiv:1908.06849*, 2019.

[102] Siming Wang, Zehang Zhang, Rong Yu, and Yan Zhang. Low-latency caching with auction game in vehicular edge computing. In *Proc. of 2017 ICCC*, pages 1–6. IEEE, 2017.

[103] Zhaolong Ning, Kaiyuan Zhang, Xiaojie Wang, Mohammad S Obaidat, Lei Guo, Xiping Hu, Bin Hu, Yi Guo, Balqies Sadoun, and Ricky YK Kwok. Joint computing and caching in 5G-envisioned internet of vehicles: A deep reinforcement learning-based traffic control system. *IEEE Transactions on Intelligent Transportation Systems*. doi:10.1109/TITS.2020.2970276, 2020.

[104] Sherif M Abuelenin and Adel Y Abul-Magd. Empirical study of traffic velocity distribution and its effect on vanets connectivity. In *Proc. of 2014 ICCVE*, pages 391–395. IEEE, 2014.

[105] Saleh Yousefi, Eitan Altman, Rachid El-Azouzi, and Mahmood Fathy. Analytical model for connectivity in vehicular ad hoc networks. *IEEE Transactions on Vehicular Technology*, 57(6):3341–3356, 2008.

[106] Sahil Garg, Kuljeet Kaur, Syed Hassan Ahmed, Abbas Bradai, Georges Kaddoum, and Mohammed Atiquzzaman. Mobqos: Mobility-aware and QoS-driven SDN framework for autonomous vehicles. *IEEE Wireless Communications*, 26(4):12–20, 2019.

[107] Tom Leighton. Improving performance on the internet. *Communications of the ACM*, 52(2):44–51, 2009.

[108] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *Proc. of ICLR*, pages 1–10, 2016.

[109] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[110] Gerhard Hasslinger, Juho Heikkinen, Konstantinos Ntougias, Frank Hasslinger, and Oliver Hohlfeld. Optimum caching versus lru and lfu: Comparison and combined limited look-ahead strategies. In *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 1–6. IEEE, 2018.

[111] Chen Chen, Cong Wang, Tie Qiu, Mohammed Atiquzzaman, and Dapeng Oliver Wu. Caching in vehicular named data networking: Architecture, schemes and future directions. *IEEE Communications Surveys & Tutorials*, 2020.

[112] Yueyue Dai, Du Xu, Yunlong Lu, Sabita Maharjan, and Yan Zhang. Deep reinforcement learning for edge caching and content delivery in internet of vehicles. In *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 134–139. IEEE, 2019.

[113] Xiaofeng Lu, Yuying Liao, Pietro Lio, and Pan Hui. Privacy-preserving asynchronous federated learning mechanism for edge network computing. *IEEE Access*, 8:48970–48981, 2020.

[114] Ameer Hamza Khan, Xinwei Cao, Shuai Li, Vasilios N Katsikis, and Liefa Liao. Bas-adam: an adam based approach to improve the performance of beetle antennae search optimizer. *IEEE/CAA Journal of Automatica Sinica*, 7(2):461–471, 2020.

[115] Haijun Zhang, Yu Qiu, Xiaoli Chu, Keping Long, and Victor CM Leung. Fog radio access networks: Mobility management, interference mitigation, and resource optimization. *IEEE Wireless Communications*, 24(6):120–127, 2017.

[116] Mahmoud Taghizadeh, Kristopher Micinski, Subir Biswas, Charles Ofria, and Eric Torng. Distributed cooperative caching in social wireless networks. *IEEE Transactions on Mobile Computing*, 12(6):1037–1053, 2013.

[117] Xiaofei Wang, Xiuhua Li, Victor CM Leung, and Panos Nasiopoulos. A framework of cooperative cell caching for the future mobile networks. In *Proceedings of the 48th Hawaii International Conference on System Sciences, HICSS*, pages 5404–5413. IEEE, 2015.

[118] Yuxia Niu, Xiaoqi Qin, and Zhi Zhang. A learning-based cooperative caching strategy in d2d assisted cellular networks. In *2018 24th Asia-Pacific Conference on Communications (APCC)*, pages 269–274. IEEE, 2018.

[119] Haijun Zhang, Yu Qiu, Keping Long, George K Karagiannidis, Xianbin Wang, and Arumugam Nallanathan. Resource allocation in noma-based fog radio access networks. *IEEE Wireless Communications*, 25(3):110–115, 2018.

[120] Qiang Li, Wennian Shi, Xiaohu Ge, and Zhisheng Niu. Cooperative edge caching in software-defined hyper-cellular networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2596–2605, 2017.

[121] Xin Dong, Lei Yu, Zhonghuo Wu, Yuxia Sun, Lingfeng Yuan, and Fangxi Zhang. A hybrid collaborative filtering model with deep structure for recommender systems. In *Proceedings of the Association for the Advancement of Artificial Intelligence, AAAI*, pages 1309–1315, 2017.

[122] Diederik P Kingma and Lei Ba. J. adam: a method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations, ICLR*, 2015.

[123] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[124] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Proceedings of Conference on Learning Theory, COLT*, pages 39–1, 2012.