UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Review of Deep Learning Schemes for Caching Optimization

Diploma Thesis

Bekiarian Christos

Supervisor: Katsaros Dimitrios

March 2024

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Review of Deep Learning Schemes for Caching Optimization

Diploma Thesis

Bekiarian Christos

Supervisor: Katsaros Dimitrios

March 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αξιολόγηση Μεθόδων που Χρησιμοποιούν Βαθιά Μάθηση για την Βελτιστοποίηση του Caching

Διπλωματική Εργασία

Μπεκιαριάν Χρήστος

Επιβλέπων: Κατσαρός Δημήτριος

Μάρτιος 2024

Approved by the Examination Committee:

Supervisor **Katsaros Dimitrios**

Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Tsompanopoulou Panagiota**

Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Georgios Thanos**

Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually, and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant

Bekiarian Christos

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών

Μπεκιαριάν Χρήστος

Acknowledgements

I extend my deepest gratitude to those whose unwavering support and encouragement have made the completion of this thesis possible. First and foremost, I would like to express my sincere appreciation to my thesis advisor, Mr. Katsaros whose guidance and expertise, have been invaluable throughout this process. A special thanks goes to my family for their boundless love, understanding, and encouragement. Your unwavering support provided the foundation upon which I built the resilience necessary to navigate the challenges of this academic journey. To my friends and peers who have shared in both the triumphs and tribulations of this process, your assistance has been a constant source of inspiration. My time in the university is a testament to a proverb I heard long ago: 'If you want to go fast, go alone. If you want to go far, go together.' This sentiment captures the collaborative spirit that has defined my academic journey, emphasizing the profound impact of shared efforts and collective progress. Thank you all for being an integral part of this significant chapter in my academic life.

Diploma Thesis

Evaluating methods that use deep learning to optimize caching in edge networks

Bekiarian Christos

Abstract

The rapid increase in internet traffic creates challenges in the current network infrastructure. An emerging solution to the traffic congestion problem is edge caching, by storing popular content near the edge of the network it reduces the traffic in the core network. A good caching strategy is crucial for increasing system performance. Deep learning emerges as a suitable solution, with its ability to learn user preferences and predict what content will be frequently requested. This work presents an extensive literature review focused on the exploration of various deep learning architectures for caching optimization in edge networks. Initially, an overview of edge computing and edge caching is provided, highlighting their benefits and various design possibilities, along with their respective impact on caching performance. Then, a collection of machine learning algorithms is presented including supervised, unsupervised, reinforcement and federated learning. Moreover, this paper conducts an analysis of recent research publications, with a focus on factors such as deep learning architectures and the objectives guiding their use. Closing with, exploring implementation challenges and future research paths of utilizing deep learning in edge caching,

Keywords: Edge network, machine learning, deep learning, content caching, content popularity, traffic offloading

Διπλωματική Εργασία

Αξιολόγηση μεθόδων που χρησιμοποιούν βαθιά μάθηση για την βελτιστοποίηση του caching σε edge networks

Μπεκιαριάν Χρήστος

Περίληψη

Η ταχεία αύξηση της κίνησης στο διαδίκτυο δημιουργεί προκλήσεις στην τρέχουσα υποδομή δικτύου. Μία υποσχόμενη λύση στο πρόβλημα της κυκλοφοριακής συμφόρησης είναι η προσωρινή αποθήκευση περιεχομένου στην άκρη του δικτύου, με την αποθήκευση δημοφιλούς περιεχομένου κοντά στην άκρη του δικτύου, μειώνεται η κίνηση στο κεντρικό δίκτυο. Μια καλή στρατηγική αποθήκευσης είναι ζωτικής σημασίας για την βελτίωση της απόδοσής του δικτύου. Η βαθιά μάθηση προκύπτει ως αποτελεσματική λύση, με την ικανότητά της να μαθαίνει τις προτιμήσεις των χρηστών και να προβλέπει το περιεχόμενο που θα ζητηθεί συχνά. Αυτή η εργασία παρουσιάζει μια εκτενή βιβλιογραφική ανασκόπηση που επικεντρώνεται στην εξερεύνηση διαφόρων αρχιτεκτονικών βαθιάς μάθησης για τη βελτιστοποίηση προσωρινής αποθήκευσης σε δίκτυα άκρης. Αρχικά, παρέχεται μια επισκόπηση των δικτύων άκρης και της προσωρινής αποθήκευσης σε αυτά, επισημαίνοντας τα πλεονεκτήματά τους και τις διάφορες δυνατότητες σχεδίασης, μαζί με τις επιπτώσεις τους στην απόδοση της προσωρινής αποθήκευσης. Στη συνέχεια, παρέχεται μια συλλογή αλγορίθμων μηχανικής μάθησης, συμπεριλαμβανομένης της επιβλεπόμενης και μη επιβλεπόμενης μάθησης, της μάθησης με ενίσχυση και της ομοσπονδιακής μάθησης. Επιπλέον, η παρούσα εργασία διεξάγει μια ανάλυση πρόσφατων ερευνητικών δημοσιεύσεων, με έμφαση σε παράγοντες όπως οι αρχιτεκτονικές βαθιάς μάθησης και οι στόχοι που καθοδηγούν τη χρήση τους. Κλείνοντας, με τη διερεύνηση των εμπόδιων της πρακτικής εφαρμογής, και των μελλοντικών ερευνητικών μονοπατιών στη χρήση βαθιάς μάθησης για την προσωρινή αποθήκευση περιεχομένου στην άκρη.

Λέξεις-κλειδιά: Δίκτυα άκρης, μηχανική μάθηση, βαθιά μάθηση, αποθήκευση περιεχομένου, δημοτικότητα περιεχομένου, εκφόρτωση κυκλοφορίας

Table of Contents

Acknowledgements ..... *xiii*

Abstract ..... *xv*

Περίληψη ..... *xvii*

Table of Contents ..... *xix*

List of figures ..... *xxiii*

List of tables ..... *xxv*

Abbreviations ..... *xxvii*

Chapter 1 Introduction *1*

1.1 Scope 2

1.2 Relevant work 3

1.3 Structure 4

Chapter 2 Edge Caching Overview *6*

2.1 Edge computing introduction 6

2.1.1 Web caching 6

2.1.2 Cloud computing 7

2.1.3 Edge computing 8

2.1.4 Cloudlet and micro data centers 10

2.1.5 Fog computing 11

2.1.6 Mobile edge computing 12

2.1.7 Edge caching 13

2.2 Caching locations 14

2.2.1 Base stations 15

2.2.2 Cloud/Fog radio access networks and baseband units 16

2.2.3 User equipment 17

2.3 Caching criteria 18

2.3.1 Cache hit rate and probability 19
2.3.2 Response time 20
2.3.3 Energy efficiency 20
2.3.4 Spectral/spectrum efficiency 21
2.3.5 Throughput 21
2.3.6 Quality of experience 22
2.4 Caching scheme 22
2.4.1 Policy vs learning based. 23
2.4.2 Proactive vs reactive caching 24
2.4.3 Cooperative vs uncooperative caching 25
2.4.4 Coded vs uncoded caching 26
2.4.5 Centralized vs decentralized training 27
2.5 Underlying assumptions and challenges 29
2.5.1 User mobility awareness 29
2.5.2 Content popularity and request distribution 30
2.5.3 Security and privacy 31
2.5.4 Time sensitive data 32
2.5.5 Cache dimensioning 32
Chapter 3 Deep Learning Outline 33
3.1 Supervised learning outline 34
3.1.1 Fully connected neural networks 35
3.1.2 Convolutional neural networks 35
3.1.3 Recurrent neural networks 36
3.1.4 Long short-term memory and gated recurrent unit 37
3.1.5 Seq2seq and pointer networks 38
3.1.6 Echo state networks 39
3.2 Unsupervised learning outline 40
3.2.1 Autoencoders 40
3.3 Reinforcement learning outline 42

3.3.1 Deep Q-learning 43
3.3.2 Actor critic networks 44
3.4 Federated learning outline 46
Chapter 4 Deep Learning for Data Caching 48
4.1 Supervised learning schemes 48
4.2 Unsupervised learning schemes 56
4.3 Reinforcement learning schemes 58
4.4 Federated learning schemes 66
Chapter 5 Implementation Challenges and Research Directions 71
5.1 Implementation challenges 71
5.1.1 The Cost of deep learning 71
5.1.2 Unlabeled data 71
5.1.3 Dropped participants 72
5.1.4 Privacy concerns 72
5.1.5 Cost and monetization models 73
5.2 Research Directions 74
5.2.1 Software defined networking 74
5.2.2 Transfer Learning 74
5.2.3 Deep learning for cache dimensioning 74
5.2.4 Standard data set 75
Chapter 6 Conclusion 76
Bibliography 77

List of figures

Figure 2.1 Caching Procedure 6

Figure 2.2 Architecture of Cloud computing [20] 7

Figure 2.3 Architecture of Edge Caching [20] 9

Figure 2.4 Architecture of Cloudlets [20] 10

Figure 2.5 Architecture of Fog Computing [20] 11

Figure 2.6 Architecture of Mobile Edge Computing [20] 12

Figure 2.7 Architecture of a mobile edge network [14] 13

Figure 2.8 Classification structure of edge caching , modified from [18] 14

Figure 2.9 Caching locations , modified from [7] 15

Figure 2.10 A general C-RAN system, [31] 16

Figure 2.11 Caching criteria, modified from [7] 19

Figure 2.12 Caching schemes, modified from [7] 23

Figure 2.13: a)Centralized training, b) Distributed training [49] 27

Figure 2.14 Distributed training in terms of data and model parallelism.[15] 28

Figure 3.1 Basic structures of typical supervised DNNs, modified from [15] 35

Figure 3.2 Recurrent Neural Network Variants [18] 37

Figure 3.3 Structure of an auto encoder [59] 41

Figure 3.4 Actor-critic logic [61] 45

Figure 3.5 Federated Learning over mobile edge, modified from [49] 46

List of tables

Table 1.1 Relevant Surveys 4

Table 2.1 Comparison of various computing paradigms, modified from [22] 9

Table 4.1 Summary of supervised learning research for data caching 49

Table 4.2 Summary of unsupervised learning research for data caching 56

Table 4.3 Summary of reinforcement learning research for data caching 59

Table 4.4: Summary of federated learning research for data caching 67

Abbreviations

AI	Artificial Intelligence
BBU	Baseband Unit
BS	Base Stations
CDN	Content Delivery Network
CHR	Cache Hit Rate
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRAN	Cloud Radio Area Network
D2D	Device to Device
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DNN	Deep Neural Networks
DQL	Deep Q Learning
DRL	Deep Reinforcement Learning
EE	Energy Efficiency
ESN	Echo State Networks
FL	Federated Learning
FRAN	Fog Radio Area Network
GRU	Gated Recurrent Unit
HetNet	Heterogeneous Networks
IoT	Internet of Things
LSTM	Long Short-Term Memory
MEC	Mobile Edge Computing
ML	Machine Learning
QoE	Quality of Experience

QoS	Quality of Service
RNN	Recurrent Neural Network
RRH	Remote Radio Head
SDN	Software Defined Networks
UE	User Equipment
VoD	Video on Demand

Chapter 1 Introduction

The rapid growth in data traffic and the surge in the number of users accessing online services pose significant challenges to traditional computing paradigms. According to the latest data analysis from DemandSage [1] on average 900 thousand new users, connected to the internet for the first time every day from 2018 until today. Additionally, Data Reportal [2] in their October 2023 report, estimated the number of current internet users being 5.3 billion which equates to about 65.4 percent of the world's population and is expected to reach 6.54 billion by 2025, while the projected number of networked devices is expected to reach 27.1 billion by the conclusion of 2023. These results are in line with Cisco's prediction for 2023 [3], that also predicts an increase in IoT devices with the number of Machine-to-Machine connections (M2M) rising to 14.7 billion and an estimate of 82.5 percent for all web traffic to be video. According to a report by Sandvine [4] in 2022, video traffic jumped by 24 percent and comprised 65 percent of all internet traffic. While Statista's report for 2022 [5] found the total mobile video traffic at 60,889 Petabytes per month, about 15 gigabytes per smartphone user and forecasts a four time increase by 2028 [6].

It is evident that the rapid rise in online traffic, fueled by the expansion of Internet of Things (IoT) devices and the increasing popularity of Video on Demand (VoD) services, creates unprecedented challenges for digital infrastructure. IoT devices, with their diverse applications ranging from smart homes to industrial sensors, generate vast amounts of time-sensitive data, placing demands on networks for real-time processing. Concurrently, the surge in VoD services contributes to heavy and unpredictable traffic, leading to bandwidth issues and latency concerns, especially during peak demand periods. As the digital landscape continues to evolve, there is a pressing need for innovative solutions to handle this surge in online activity. Robust and scalable approaches, such as edge computing, smart caching strategies, and optimized network configurations, are becoming essential to ensure the efficient processing and delivery of time-sensitive data, meeting the requirements of IoT devices and VoD services while providing users with seamless and high-quality experiences.

Web caching in edge networks is a crucial strategy to enhance the efficiency and responsiveness of content delivery. Unlike traditional caching methods that rely on centralized servers, edge caching brings the caching infrastructure closer to the end-users, reducing latency and optimizing the utilization of network resources. In edge networks, caches are strategically deployed at the network's edge, in proximity to users. This allows frequently accessed content to be stored locally, minimizing the need for repeated requests to the origin server. By utilizing edge caches, web applications can deliver faster response times, reduce server load, and mitigate the impact of latency, resulting in an overall improved user experience.

A good caching strategy is essential to improving network performance. Conventional methods may rely on static rules or predefined algorithms, which might fall short in accurately predicting the most effective caching decisions for the current spatial and temporal conditions. This is where deep learning emerges as a powerful solution. Deep learning algorithms excel in learning intricate patterns from historical data and adapting to evolving conditions. By harnessing the capabilities of deep learning algorithms at the edge, these caching methods dynamically analyze patterns in user behavior, content popularity, and network conditions, to make real-time predictions to intelligently anticipate and store content locally at the edge. This adaptive approach ensures that frequently requested content is readily available, minimizing latency and enhancing overall system performance and enables a more responsive context-aware approach, addressing the limitations of traditional caching methods and enhancing overall system performance in edge networks.

The aim of this work is to explore different methods for addressing the edge caching issue using deep learning. By functioning as an index and guide, it helps clarify the implications and complexities associated with edge caching using deep learning while also examining the trends in problem selection and solution approaches.

1.1 Scope

While Video on Demand and (VoD) content and IoT data are the areas benefiting most from edge caching due to the big amount of data and their time sensitive nature. VoD takes up the majority of all internet traffic and has a lot to gain from an efficient caching

strategy, based on content popularity prediction, an area where deep learning methods excel. IoT data benefit from edge caching in a different way, by utilizing the computational capabilities of the node, sensor data is processed and evaluated closer to the point of generation, increasing the responsiveness, and decreasing network traffic. Specifically, Vehicle to any device (V2X) communication has seen a lot of interest in the research field.

Due to the different utilization of edge node resources by VoD an IoT, attempting to analyze these two areas of research together would be overly ambitious. Therefore, this work directs its focus away from IoT data and focuses more closely on VoD and general content-related topics.

The primary objective of this work is to present research concerning the application of deep learning in edge caching. Given the dynamic nature of this field, verifying the methodology, assumptions, or conclusions of such research is a formidable challenge. Hence, it lies beyond the boundary of this work to conclusively verify or challenge the accuracy of the research findings presented.

1.2 Relevant work

A number of surveys have covered the subject of optimizing caching in edge caching with deep learning, below together with Table 1.1 the most relevant surveys and their aim is presented.

In [7] the authors provide a comprehensive overview of mobile edge parameters, primarily focusing on non-learning-based approaches. In contrast, [8] adopts a social-aware approach, emphasizing the user's characteristics, while [9] concentrates on the implementation of reinforcement learning techniques. Additionally, [10] delves into the use of transfer learning, where [11] focuses on the challenges associated with deep learning implementation, while [12] explores how deep learning can enhance applications. Furthermore, [13] extensively examines the implications of federated learning, [14] shifts the focus to proactive caching strategies, whereas [15] investigates various application and implementation scenarios for deep learning in edge networks. Finally, [16], [17], and [18] examine papers using various machine learning models.

While the subject of deep learning in edge caching has been visited many times, this work adds value by reviewing different and latter papers than the ones visited before.

Table 1.1 Relevant Surveys

Reference	Main Contributions
[7]	Conducts a comprehensive survey of mobile edge caching and informs on the advantages and challenges of mobile edge caching
[8]	Provides a comprehensive review of the existing social-aware caching approaches in edge computing
[9]	Discusses the intricacies of the different networking architectures and targeted performance metrics on the operation of RL-aided edge caching
[10]	Explores the use of artificial intelligence for data caching in wireless networks to optimize performance
[11]	Discuses deep learning applications for 5G, and addresses new applications standardization efforts, and challenges in mobile edge computing
[12]	Explores how advancements in deep learning can improve applications at the edge of the network, focusing on smart multimedia, transportation, city, and industry
[13]	Explores the implementation of federated learning in edge caching as a solution to privacy concerns
[14]	Evaluates machine learning methods for proactive caching
[15]	A survey on applications of machine learning for in-network edge caching
[16]	Discusses AI's significance in edge devices, presents optimization methods, and addresses security concerns
[17]	Investigates scenarios, implementation methods, technologies, and challenges to foster the use of deep learning on the edge
[18]	Explores the potential of edge caching in 5G networks to alleviate core network congestion and reduce latency. Focusing on deep learning for data caching

1.3 Structure

The rest of this work is organized as follows. Chapter 2 offers a comprehensive overview of edge computing and caching, covering the potential locations for the cache storage, caching criteria, various caching schemes, as well as the assumptions and challenges in this

domain. Chapter 3 presents the typical neural network structures, highlighting their strengths and weaknesses, including architectures from supervised, unsupervised, reinforcement and federated learning. Chapter 4 presents a variety of research papers, providing summaries and descriptions for the proposed schemes and points out trends in the problem selection and solution approaches. Chapter 5 discusses the challenges while researching edge caching with deep learning as well as potential future research directions, while Chapter 6 provides a summary of this work.

Chapter 2 Edge Caching Overview

2.1 Edge computing introduction

2.1.1 Web caching

Caching is a mechanism employed in computer systems to enhance the efficiency and speed of data retrieval by storing frequently accessed information in a temporary storage location, known as the cache. This practice is particularly crucial in scenarios where accessing data from the primary storage, such as a remote server, incurs significant time and resource overhead. By storing a copy of frequently requested data closer to the point of use, subsequent requests can be fulfilled more swiftly. This method significantly reduces latency, improves response times, and optimizes overall system performance. The main operation of a cache is shown in Figure 2.1, if the user requested content is stored in the cache it is downloaded from that local memory, if it is not, then the content is requested from the server, then it is decided if the content will be cached and if so, what content will be evicted, depending on the replacement/eviction strategy. Having a good caching policy can significantly improve system performance by improving response times helping save bandwidth and minimize the server load.

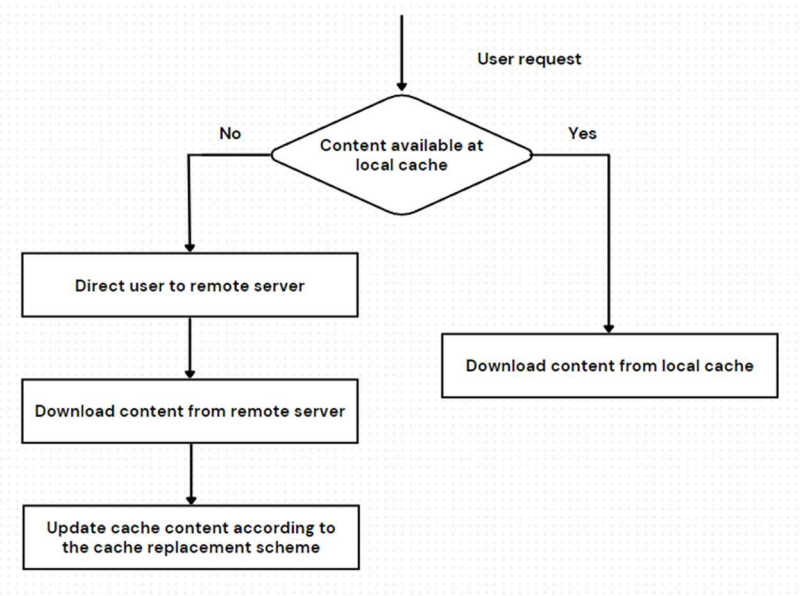

Figure 2.1 Caching Procedure

2.1.2 Cloud computing

Cloud computing is on-demand access to computing resources, applications, servers (physical and virtual), data storage, development tools, networking capabilities, and more via the internet. It replaces the need for local servers and traditional infrastructure and is hosted at a remote data center under the management of a Cloud Services Provider (CSP). This model provides users with on-demand access to a diverse range of computing services, including storage, processing power, and applications, without the need for significant upfront investments in hardware and maintenance. Figure 2.2 illustrates the general architecture of mobile cloud computing which contains 2 tiers, the cloud, and the mobile network. All the requests from the user devices are processed and served by the cloud server [19], [20].

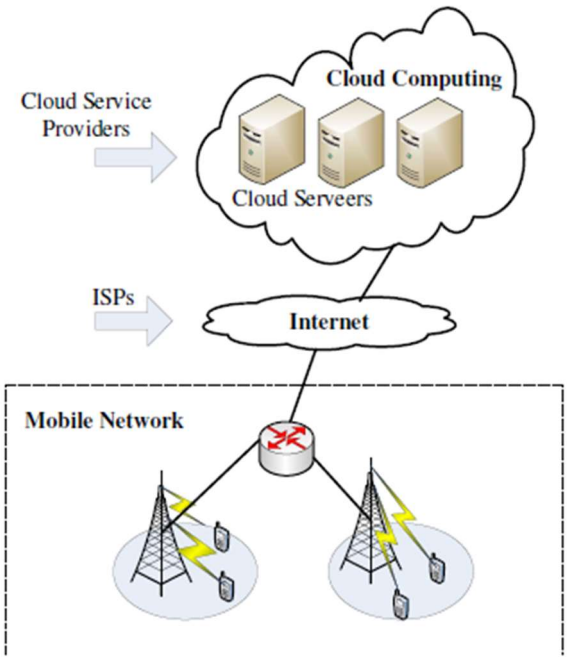

Figure 2.2 Architecture of Cloud computing [20]

The rapid rise web traffic data generated by IoT devices and demand for VoD services poses significant challenges for cloud computing. As the number of IoT devices continues to surge, the volume of data they produce grows exponentially, straining cloud infrastructure's capacity to store, process, and manage such vast datasets, leading to increased network congestion as they communicate with cloud servers for data processing and storage, affecting the timely processing of critical information. Simultaneously, the popularity of VoD services places a substantial burden on cloud networks due to the large-

scale streaming of video content. This can result in bandwidth issues, latency, and degraded user experiences, particularly during peak demand periods. To combat these challenges edge computing has emerged as a potential countermeasure, offering a distributed and decentralized approach to data processing [19], [20].

2.1.3 Edge computing

Edge computing or edge networks refers to a decentralized computing paradigm where data is processed closer to its source or "edge", such as the end-users or the devices generating the data. Unlike traditional cloud computing, where data processing occurs in centralized data centers, edge computing involves processing data near the location where it is generated, reducing the need to send vast amounts of data to distant data centers for processing. As shown in Figure 2.3 this typically implies the addition of a backhaul node with storage and computational capabilities. This approach brings computation and data storage closer to where it's needed. Edge computing is often used in scenarios where real-time data processing is crucial, such as in IoT devices, autonomous vehicles, industrial automation, and various other applications requiring immediate processing or decision-making. This concept has gained prominence due to the increasing demand for faster data processing, reduced latency, and the growth of applications that rely on real-time data analysis and decision-making. In the world of edge computing, various different implementations proposals exist [7], [20], [21].

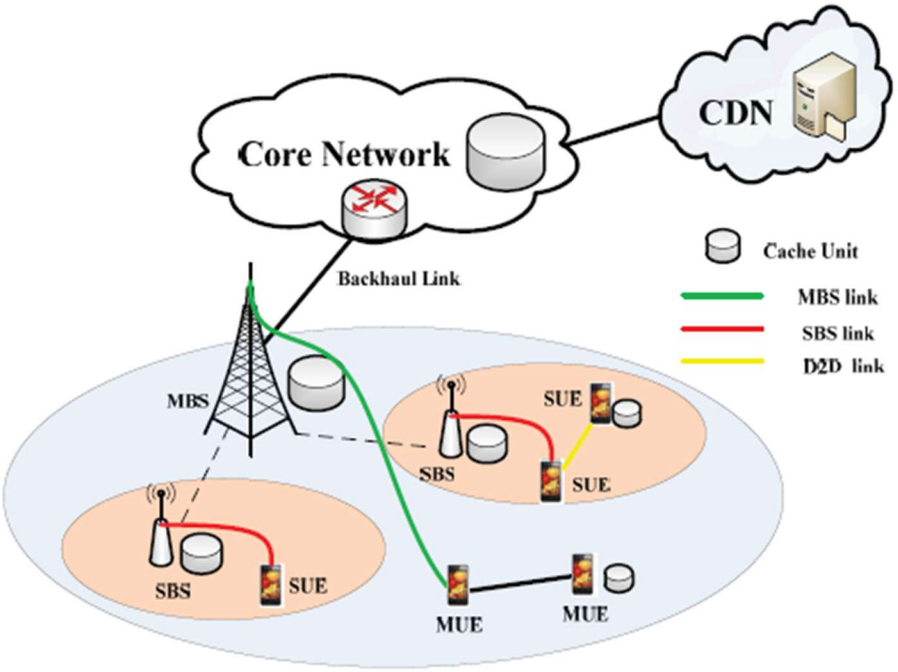

Figure 2.3 Architecture of Edge Caching [20]

These include Cloudlet, Micro Data Centers (MDCs), Fog Computing, and Mobile Edge Computing (also known as Multi-access Edge Computing). However, the edge computing community hasn't agreed on the standard's definitions, architectures, and protocols yet. To simplify, all these technologies are collectively referred to as "edge computing." In the section bellow as well as table 2.1 a quick overview of the different concepts and the characteristics of each proposal are presented.

Table 2.1 Comparison of various computing paradigms, modified from [22]

Paradigms	Location for Computing	Virtualization	Research Focuses
MEC	Base stations and nearby devices	VM and container	Uplink computation offloading downlink caching, and so on
Fog	Devices along the routing path	VM and container	Uplink computation offloading, downlink caching, and so on
Cloudlet	Nearby cloudlets	VM	Computation offloading, VM management, and so on
MDC	Nearby data centers	VM and container	Computation offloading, VM management, and so on
Cloud	Central Cloud	VM	Workflow scheduling, VM management, and so on

2.1.4 Cloudlet and micro data centers

Cloudlet, introduced by Carnegie Mellon University [23], is a network element merging mobile and cloud computing and its architecture is presented in Figure 2.4. Positioned as the intermediary layer among mobile devices, and the primary cloud infrastructure these small clusters are placed near mobile devices in places like buildings or shopping centers, serving as a micro cloud. These clusters aid in processing, offloading, and caching tasks. Cloudlet employs virtualization management technologies to better assist mobile applications. Its primary goal is to bring cloud benefits to mobile users, aiming for faster, more efficient processing with lower latency [12], [15], [20], [23].

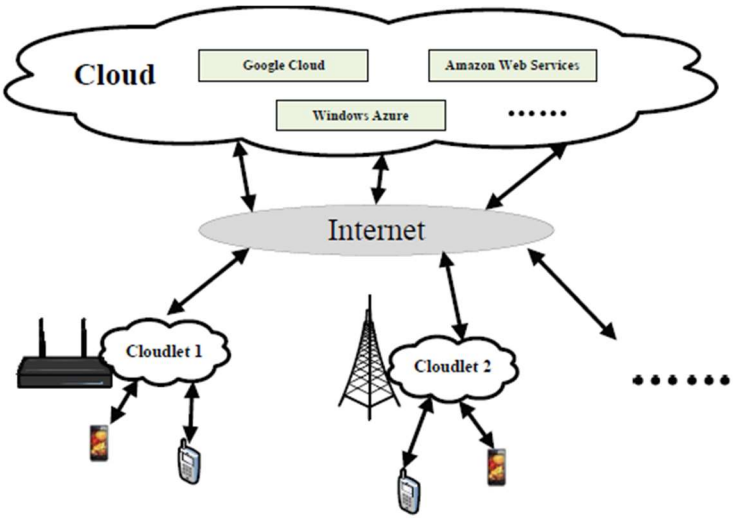

Figure 2.4 Architecture of Cloudlets [20]

Micro Data Centers (MDCs), initiated by Microsoft [24], share a resemblance to the concept of cloudlet as they're also intended to complement the existing cloud infrastructure. They function as smaller-scale versions of data centers, aiming to expand the capabilities of larger cloud data centers. The core concept involves bundling all necessary computing, storage, and networking equipment into a single enclosure, creating a self-contained and secure computing environment. This setup caters to applications demanding lower latency or devices with limited battery life or computing capabilities. Various MDCs are interlinked via a backbone network, enabling more efficient and intelligent computation, caching, and management [12], [15], [20], [24], [25].

2.1.5 Fog computing

Fog computing, initially introduced by Cisco [26], represents a computing framework aiming to extend cloud computing services to the farthest edges of enterprise networks. A key aspect of Fog computing is its reliance on a fully distributed multi-tier cloud architecture that integrates numerous devices and large-scale cloud data centers as presented in Figure 2.5. Within the Fog computing model, data processing takes place at Fog nodes, typically positioned at the network gateway [12], [15], [20].

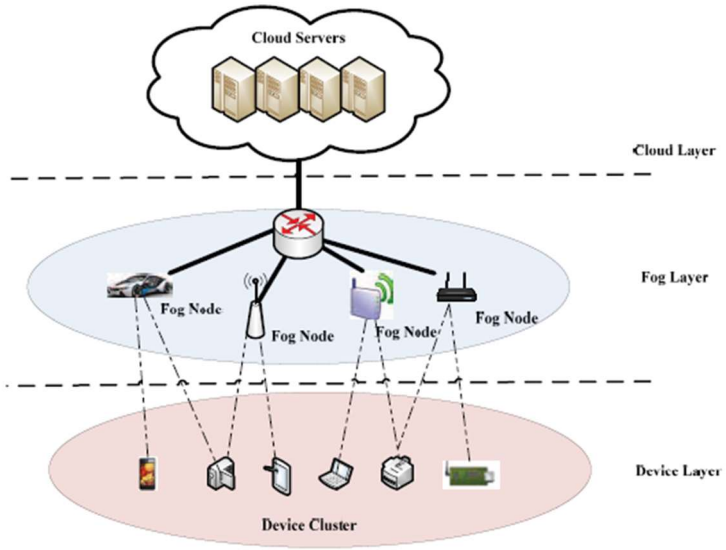

Figure 2.5 Architecture of Fog Computing [20]

While cloud and Fog paradigms offer a similar range of services like computing, storage, and networking, the distinct deployment strategy of Fog computing targets specific geographic regions. This structure serves as a high-level platform enabling numerous IoT devices to interconnect through distributed Fog nodes, facilitating collaborative services. These Fog nodes are primarily engineered to offer enhanced support for IoT devices, especially in applications requiring real-time responses with minimal latency, such as interactive and IoT applications. Unlike Cloudlet and MDCs, Fog computing places a stronger emphasis on IoT applications, with a specific focus on real-time responses and less latency [12], [15], [20].

2.1.6 Mobile edge computing

The concept of Mobile Edge Computing (MEC) was initially standardized by the European Telecommunications Standards Institute (ETSI) [27]. It centers on positioning computing capabilities and service environments at the cellular network's edge. It is important to note that while cloudlets primarily utilize virtual machines for virtualization, MEC and Fog prioritize the use of containers. The main goal is to offer faster response times, location awareness, and better data capacity. By setting up edge servers on cellular base stations, users can swiftly introduce new applications and services as shown in Figure 2.6. This setup is particularly beneficial for IoT devices as it allows direct service to these devices in a single hop, streamlining data processing and enhancing convenience [12], [15], [20], [22], [27].

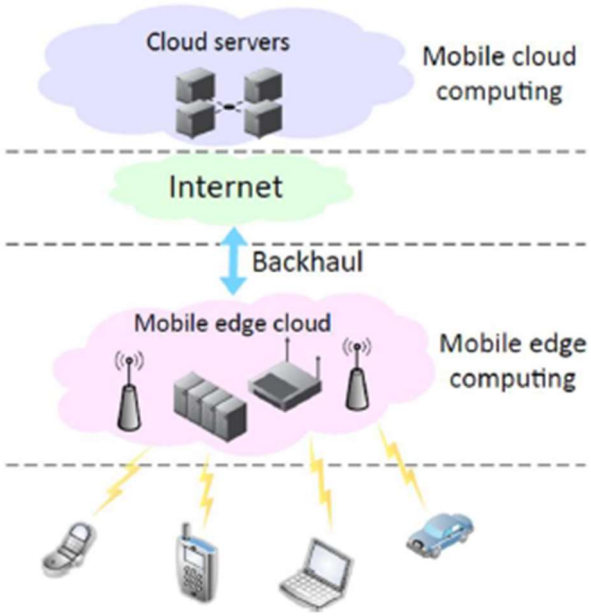

Figure 2.6 Architecture of Mobile Edge Computing [20]

2.1.7 Edge caching

Edge caching involves utilizing the storage and computational capabilities inherent in edge computing to store frequently accessed content in node storage. This strategy aims to minimize redundant data traffic and enhance overall system performance. Figure 2.7 illustrates a network architecture implementing the previously mentioned proposals, while Figure 2.8 presents the classification framework for various aspects of edge caching decisions. This framework encompasses considerations such as caching location, criteria, schemes, as well as the assumptions and challenges in research are presented. These aspects will be explored in depth below.

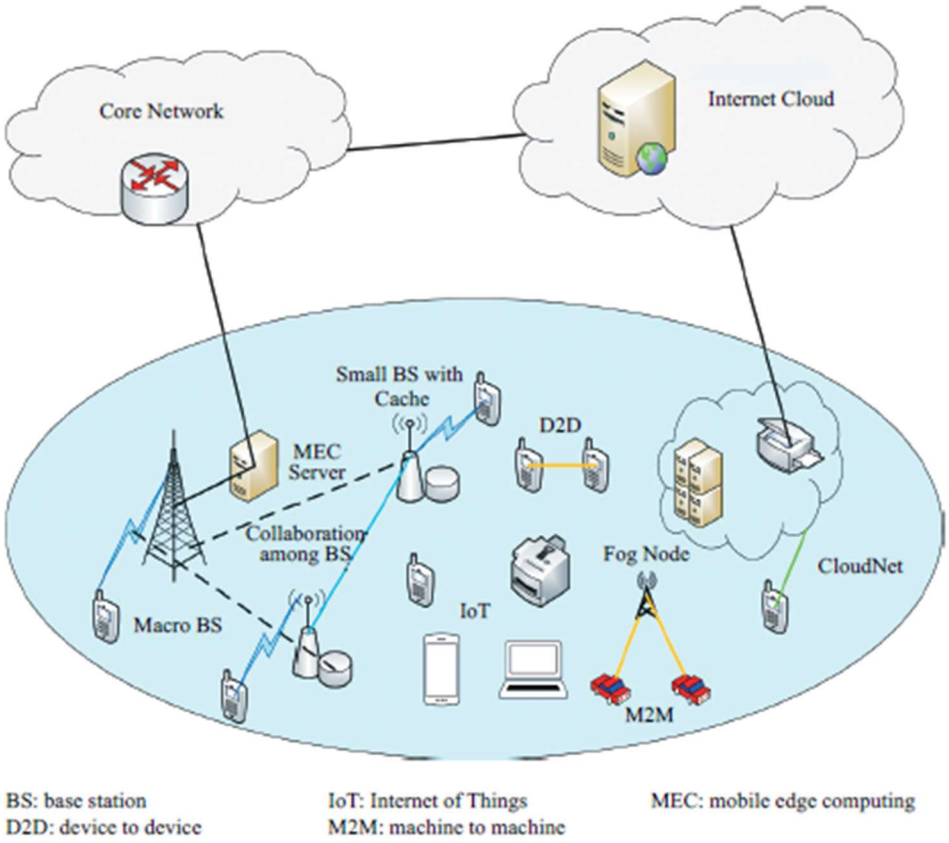

Figure 2.7 Architecture of a mobile edge network [14]

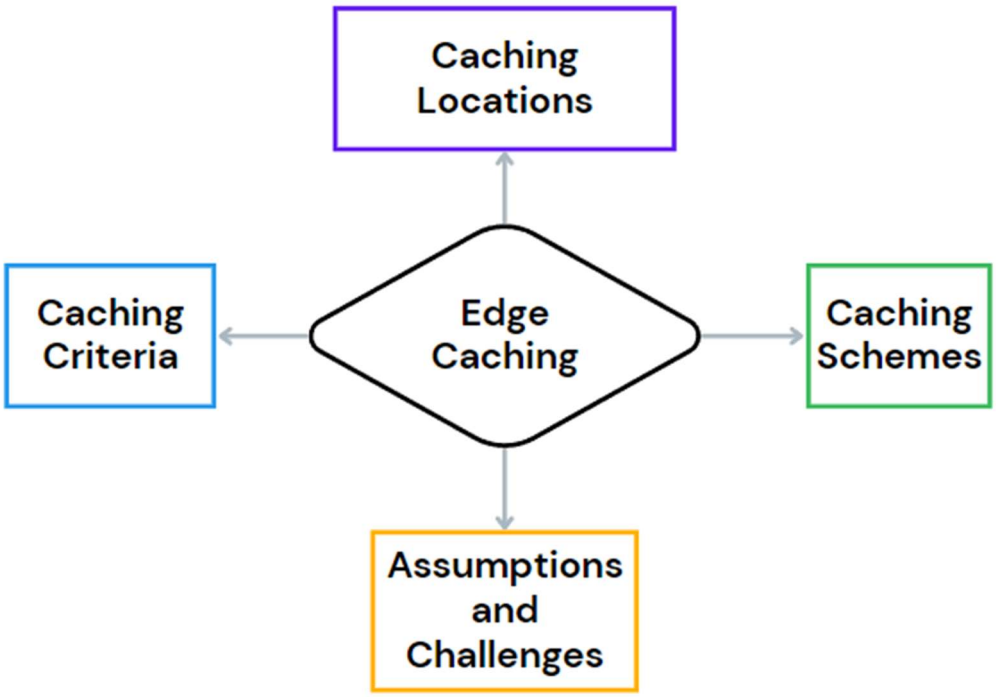

Figure 2.8 Classification structure of edge caching , modified from [18]

2.2 Caching locations

Even though different standards are proposed the idea of bringing the processing and storage closer to the user and adding a node of some sort remains the same across all proposals. While most works explored are using the Mobile Edge Computing standards and the Fog computing standards the similarity in how these approaches operate and can be implemented makes addressing edge computing as a whole, rather than every proposed standard separately, more comprehensible, and efficient [14].

The standard, this work will use going forward is the one depicted in Figure 2.7. Caching units can be strategically placed across various components of a mobile network, including the core network, radio area networks, and user devices, to meet the needs of mobile users. The mobile caching places will be discussed below since the challenges the core network faces in caching are the same with the mobile edge caching but lesser [14].

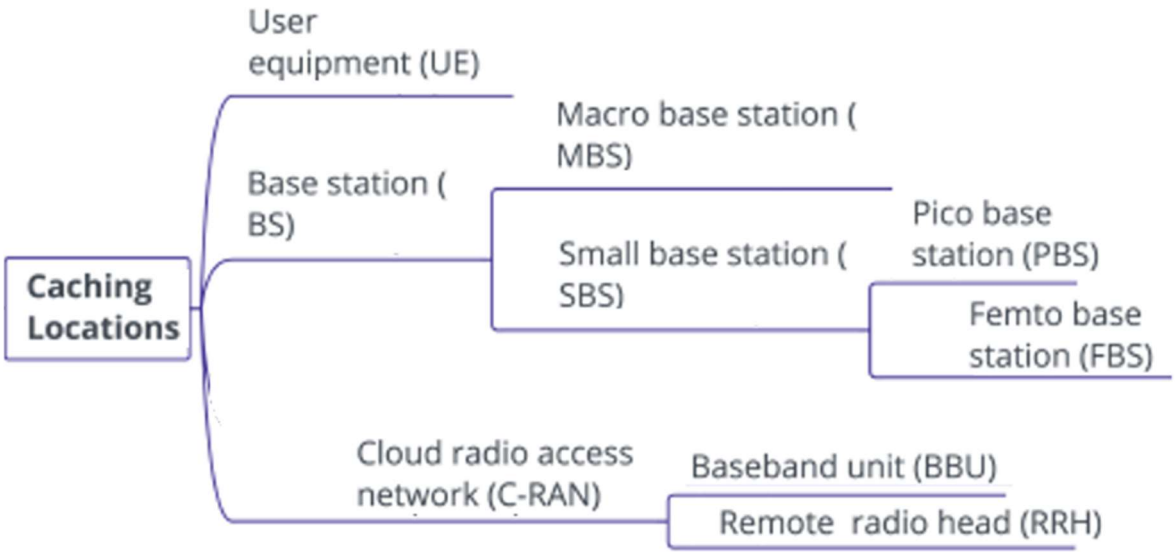

Figure 2.9 Caching locations , modified from [7]

2.2.1 Base stations

In edge networks, Base Stations (BSs) play a pivotal role as caching locations, by bringing content closer to mobile users working to enhance content delivery and reduce latency. The deployment of base stations in Heterogeneous Networks (HetNets) encompasses various types, each contributing uniquely to the caching architecture. Macro Base Stations (MBS), with their extensive coverage, act as central hubs for caching popular and frequently accessed content. Small Base Stations (SBS), characterized by their compact footprint and improved capacity, efficiently cache data in denser urban areas where network demands are higher. Femto Base Stations (FBS) cater to localized and indoor environments, ensuring content proximity for better user experiences in residential or enterprise settings. Pico Base Stations (PBS), being the smallest in scale, address ultra-dense scenarios by optimizing caching for micro-cells. In HetNets, these diverse base stations collaboratively form a seamless network fabric, ultimately fostering a dynamic and responsive edge environment. However, caching at BSs faces challenges such as limited coverage, uncertainty in wireless connections, and inter-cell interference [7], [28], [29], [30].

2.2.2 Cloud/Fog radio access networks and baseband units

The Cloud Radio Access Network (C-RAN) or often called Fog Radio Access Network (F-RAN) represents a transformative advancement in mobile network architecture, integrating technologies from both the wireless and information technology sectors to enhance spectral efficiency and energy efficiency. At the core of C-RAN lies the decentralization of traditional base stations into distributed Remote Radio Heads (RRHs) and a centralized Baseband Unit (BBU) pool, a general C-RAN architecture/system is presented in Figure 2.10. This separation allows RRHs to focus on radio frequency functions, catering to high capacity demands in densely populated areas, while the virtualized BBU pool facilitates large-scale collaborative processing, cooperative radio resource allocation. and intelligent networking. The communication between RRHs and the BBU pool is facilitated by the common public radio interface protocol, ensuring constant bit rate [7], [22], [31].

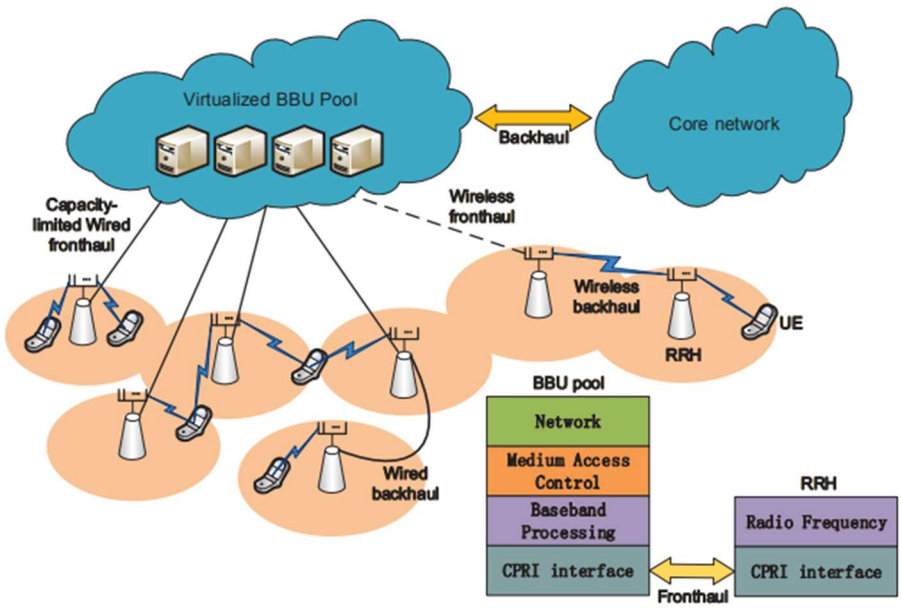

Figure 2.10 A general C-RAN system, [31]

In C-RAN, baseband processing and control functions are moved from individual base stations to a centralized data center. This centralization enables more efficient resource allocation, dynamic load balancing, and centralized coordination of multiple base stations. By pooling resources and virtualizing network functions, C-RAN offers scalability, flexibility, and cost savings for mobile operators. Additionally, C-RAN facilitates the deployment of advanced technologies such as network slicing, and Software-Defined

Networking (SDN), paving the way for future-proof mobile network architectures capable of supporting diverse services and applications with varying requirements. C-RAN's centralized approach not only addresses capacity fluctuations but also enhances energy efficiency in mobile networks by leveraging shared storage and computing resources through virtualization. However, realizing the full potential of C-RAN requires efficient fronthaul connections with stringent throughput and latency requirements. Moreover, while MEC complements C-RANs by reducing latency and improving local user experience, it possesses lesser processing power and storage compared to the centralized cloud infrastructure [7], [22], [31].

By integrating edge caching with C-RANs, neighbor edge devices are now able to provide the data that User Equipment (UEs) request. Next, a more effective data fetching strategy should be created to assist in determining where to retrieve the data from and the related route if the data requested by a UE is not cached in these devices. While edge caching at UEs and RRHs has several advantages, there are a number of technical difficulties when implementing it in C-RANs. Firstly, determining the edge cache strategy involves balancing the frequency of data updates to optimize user experience while minimizing fronthaul resource consumption. Additionally, designing an effective edge cache strategy relies on understanding the popularity of data, which can be challenging when dealing with incomplete information. Secondly, devising a data fetching strategy becomes crucial with edge caching in C-RANs. If the requested data isn't cached in nearby devices, a robust strategy must determine where to fetch the data and the corresponding route efficiently. Lastly, optimizing RRH association presents a challenge, especially considering that one UE is often served by multiple RRHs simultaneously in C-RANs [7], [22], [31].

2.2.3 User equipment

Modern smartphones are evolving to possess advanced computing and storage capabilities, enabling them to function as local caches for storing and sharing content directly with other UEs through Device-to-Device (D2D) links utilizing licensed-band (e.g., LTE) or unlicensed-band protocols (e.g., Bluetooth and Wi-Fi) [7], [32], [33].

This caching strategy, known as D2D caching, involves the base station monitoring caching status in each UE, directing requests to nearby devices with the desired content if

available, and resorting to downlink transmission from the BS if needed. D2D caching enhances spectrum utilization, energy efficiency, and throughput while enabling novel peer-to-peer and location-based applications. Additionally, caching at the UE level fosters a more resilient network architecture by alleviating the burden on central servers, particularly during periods of peak demand, thus enhancing the scalability and reliability of edge networks [7], [32], [33].

However, caching in user equipment within edge networks also presents certain disadvantages. One primary drawback is the limited storage capacity and processing power inherent in many consumer-grade devices, which can constrain the amount and types of data that can be effectively cached. This limitation may result in frequent cache evictions or reduced cache hit rates, leading to increased latency and decreased performance for users. Additionally, interference from neighboring D2D pairs, poses a tradeoff between cache hit probability and interference, influenced by collaboration distances. Social relationships also influence D2D caching, affecting content delivery routing path selection and predicting future requests, thereby contributing to a more resilient network architecture during peak demand periods. Moreover, caching at the UE level may raise privacy and security concerns, as sensitive or personal data stored locally could be vulnerable to unauthorized access or exploitation. Additionally, managing and synchronizing cached content across a multitude of diverse devices within the network can be complex and resource-intensive, posing challenges in maintaining consistency and coherence in cached data[7], [32], [33].

2.3 Caching criteria

When evaluating a caching scheme, several criteria can be used to determine performance, every scheme targets one or more of these metrics depending on the constraints of the problem they are trying to solve or the target they aim to achieve. The different caching criteria are presented in Figure 2.11. The most common metric to improve upon in cache performance is cache hit rate because several of the other metrics are related to it.

For example, a cache miss means the content will have to be fetched from the server, thereby negatively impacting every other metric such as energy consumption, response time and throughput [34].

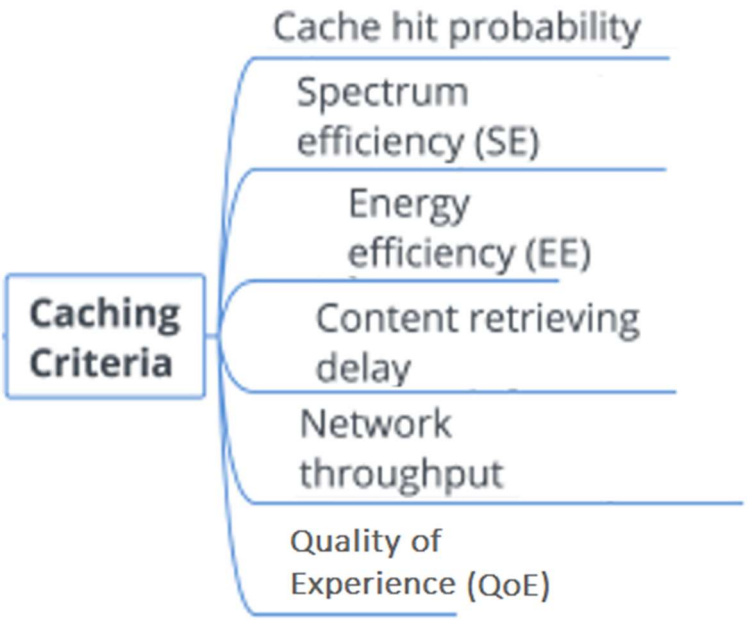

Figure 2.11 Caching criteria, modified from [7]

2.3.1 Cache hit rate and probability

The cache hit rate reflects the historical efficiency of a cache by measuring the percentage of times a requested piece of data is found in the cache as opposed to having to fetch it from a slower, larger memory or storage. It is calculated as the ratio of cache hits to the total number of memory accesses. A high cache hit rate means the cache is storing frequently accessed data resulting in better system performance [34].

On the other hand, cache hit probability is a predictive measure that estimates the likelihood of a cache hit for a specific memory access scenario. It considers various factors such as cache size, block size, and access patterns to determine the probability that a given memory access will find the required data in the cache [34].

While cache hit rate provides retrospective insights into past performance, cache hit probability offers a forward-looking perspective, aiding in the theoretical analysis and optimization of caching systems [34].

2.3.2 Response time

Response time also known as end-to-end delay or latency is defined as the total time it takes from the content request generation by the user to the delivery of that content and is influenced by many factors like network condition, routing efficiency, network congestion and many more. The single-hop network delay typically involves three elements: data transmission delay, queueing delay, and propagation delay. A low response time is critical for time-sensitive application like online gaming, real time video streaming, smart city applications and industrial IoT devices, where delivery of the content after the deadline constraint makes the contents value obsolete [7], [34].

In the reviewed work the authors try to minimize the response time with different methods, creating a good proactive caching policy is a common approach so that frequently accessed content can be retrieved from the local cache reducing the average response time. In coded policies, meaning the content can be broken into pieces and saved across many base stations, like [35] storing the content pieces close to each other reduces the fetching time. Content placement is also very important to reducing time delay, caching the content in the area of the base station they user will be rather than the one request was made from is pivotal [36]. In reference [37], the authors propose a deadline-aware policy designed to predict the deadline of content by estimating both soft and hard deadlines, indicating when the content becomes less useful or obsolete. The policy aims to prioritize meeting as many deadlines as possible, ensuring timely delivery of content before it loses relevance or utility.

2.3.3 Energy efficiency

Energy efficiency refers to the optimization of energy consumption. By storing frequently requested content closer to end-users, edge caching minimizes the need for data transmission over longer distances, thereby reducing energy consumption associated with backhaul communication. This optimization is crucial in heterogeneous network environments, where various factors such as cache size, caching cost (power consumption per bit stored), and the number of requests influence energy efficiency. By employing energy-efficient caching strategies at the edge, such as proactive caching policies, and intelligent cache replacement algorithms, energy consumption can be minimized while

ensuring high-quality service delivery. Energy-efficient edge caching not only improves the user experience by decreasing latency and enhancing content accessibility but also increases cost-effectiveness, by lowering the energy demands associated with data processing and delivery. This aligns with the strategic objectives of network businesses aiming to reduce operational expenses while maintaining high-quality service delivery and customer satisfaction leading to a broader adoption of edge computing [34], [38], [39].

2.3.4 Spectral/spectrum efficiency

Spectral efficiency is a metric in telecommunications, measuring the amount of information transmitted within a given bandwidth. It signifies a communication system's capacity to optimize data rate or throughput within a specific frequency spectrum, crucial for maximizing radio frequency resources in wireless networks. With the escalating demand for data transmission in modern wireless communication systems, ranging from voice calls to high-definition video streaming, enhancing spectral efficiency becomes imperative [34], [39].

Higher spectral efficiency allows for more data transmission within the available bandwidth, vital for efficient spectrum utilization as network densities increase with more users and devices. To enhance spectral efficiency, strategies like network densification by deploying more small cell base stations in a base stations range and implementing caching techniques have been proposed. Storing frequently accessed content closer to end-users, reduces the need for data transmission over longer distances, thereby improving network throughput and reducing congestion [34], [39].

2.3.5 Throughput

Throughput in a wireless network refers to the rate of successful data transmission over a certain period of time. It represents the efficiency of data transmission and reception processes, indicating the amount of data successfully transferred over a certain period. Throughput is influenced by several key factors, including the number of antennas at the base station, user cache size, transmit power, and the chosen caching strategy (uncoded or coded). Higher throughput signifies better network performance and faster data transfer

rates, which are crucial for ensuring seamless communication services to users. Additionally, factors such as bandwidth availability, signal strength, interference, and network congestion also impact throughput, ultimately determining the overall quality and reliability of the wireless network's data transmission capabilities [34].

2.3.6 Quality of experience

Quality of Experience (QoE) refers to the subjective assessment of an individual's overall satisfaction or perception of a service, product, or system, typically in the context of digital technology and communication services. Unlike traditional Quality of Service (QoS) metrics, which focus on objective technical parameters such as latency, throughput, and error rates, QoE considers the user's perception, preferences, and expectations. It encompasses various factors including usability, responsiveness, reliability, and emotional response, all of which contribute to the user's holistic experience. QoE in networks is dependent on several key elements of network performance metrics which directly influence the responsiveness, reliability, and smoothness of network communication. Additionally, the reliability and availability of the network play a critical role, as users expect uninterrupted service and minimal downtime. Congestion and traffic management are also significant determinants of QoE, as network congestion can lead to delays and reduced throughput, especially during peak usage times. Moreover, the efficient delivery of content, particularly for multimedia streaming services, is crucial for ensuring a positive user experience, with factors such as buffering, resolution, and playback consistency directly impacting QoE. Overall, achieving high QoE in networks requires a holistic approach that considers both technical and user-centric aspects to deliver satisfactory experiences in networked environments [34], [40].

2.4 Caching scheme

The choice and implementation of caching strategies significantly impact system behavior, including resource utilization, scalability, and data consistency. Thus, understanding the intricacies of caching strategies and their implications is essential for designing robust and high-performing systems in modern computing environments. This introductory paragraph

sets the stage for exploring the mechanics, significance, and implications of caching strategies in system design and operation.

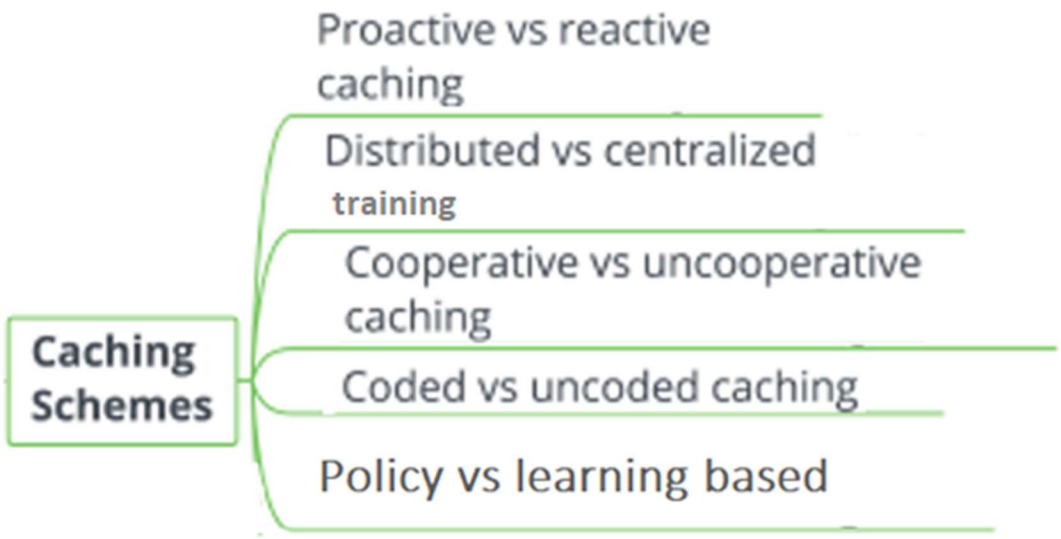

Figure 2.12 Caching schemes, modified from [7]

2.4.1 Policy vs learning based.

Policy-based caching policies involve predetermined rules and strategies for deciding what content to cache and when. These policies are established based on fixed guidelines and specific criteria such as content popularity, file size, or the available cache space. Policy-based caching relies on predefined rules and does not adapt dynamically to changing network conditions or evolving content.

Least Recently Used or LRU, is a popular content replacement algorithm employed in computer systems for efficient memory management. The fundamental principle behind LRU is to prioritize retaining the most recently accessed content in memory and evicting the least recently visited when the cache is full. This algorithm assumes that recently used content is more likely to be requested again in the near future, improving the utilization of available memory. Its implementation involves tracking and managing the usage history of content, ensuring that the system retains the content most relevant to ongoing processes.

Least Frequently Used or LFU is a page replacement algorithm used in computer systems to manage memory efficiently. LFU removes the least frequently accessed content from

memory when new content needs to be loaded. The key criterion for eviction is the frequency of content access, and LFU aims to retain content that is requested less frequently. If multiple contents have the same access frequency, LFU may consider other factors for replacement. This algorithm proves valuable in scenarios where understanding the historical usage pattern of pages is important for optimizing memory utilization. LFU enhances the overall system performance by intelligently swapping out less relevant data.

Learning-based content caching policies utilize machine learning, or algorithms that adapt and evolve over time. These approaches analyze historical data, user behavior, and network conditions to predict and make caching decisions. Learning-based caching continuously refines its strategies, considering changing content popularity, user preferences, and network dynamics. By learning from past experiences and adapting to current trends, learning-based caching policies aim to optimize content placement and retrieval dynamically.

The studies examined in this work propose learning-based content caching policies and mostly compare them to policy-based techniques to assess their effectiveness. This is because, policy-based techniques are the primary methods used in the industry and are considered the gold standard. However, with the recent surge in machine learning algorithms, researchers are exploring learning-based techniques as more efficient potential alternatives.

2.4.2 Proactive vs reactive caching

Proactive caching policies rely on accurate predictions of content popularity to foresee if a file will gain popularity in the future. This method utilizes historical user preferences and past content request data to make these predictions. By pre-fetching files and distributing them to cache-enabled edge nodes, proactive caching aims to alleviate the impact of network load fluctuations. Based on the anticipation of user requests, proactive caching policy chooses which contents should be cached before they are requested. To increase caching performance and ensure QoS requirements, proactive caching typically makes advantage of the estimations of request patterns (such as user mobility patterns, user preferences, and social interactions). The advancement of machine learning and big data analytics makes it beneficial to cache popular information locally before requests come in.

Proactive caching improves the caching efficiency by pre-downloading popular contents during off peak times and serving predictable peak-hour demands [9], [28].

Unlike proactive caching policies, which make predictions about future content requests based on historical data, reactive policies reactively adjust caching decisions in real-time based on the current system states or user interactions. Since policy-based schemes inherently react to the system and cannot predict future popularities, all policy-based strategies are reactive. However, there are policies that integrate machine learning algorithms to analyze real-time data and user interactions, enabling them to adaptively adjust caching decisions. By harnessing the power of machine learning, these policies can dynamically learn and respond to evolving access patterns and system conditions, optimizing cache hit rates and resource utilization more effectively than traditional reactive policies. This approach often involves the utilization of sophisticated feedback mechanisms and heuristics to continuously refine caching decisions based on the most up-to-date information available [17].

In recent research endeavors, there has been an appearance of schemes that combine both proactive and reactive caching strategies. These hybrid schemes aim to capitalize on the strengths of each approach, combining proactive techniques, which anticipate future content demands based on historical data, with reactive methods that dynamically adjust caching decisions in real-time. By blending these approaches, these schemes strive to strike a balance between the benefits of proactive caching, such as improved hit rates through anticipation of popular content, and the adaptability of reactive caching to handle volatile access patterns and system conditions [41], [42].

2.4.3 Cooperative vs uncooperative caching

Cooperative or collaborative caching involves multiple cache servers working together to improve the overall efficiency of the caching system. When a request for data is made, the cache servers collaborate to determine the most suitable server to fulfill the request based on factors such as proximity and current load. If the requested data is not available in the local cache, the servers communicate with each other to locate a copy of the data in another cache within the network. By sharing cached data among multiple servers,

cooperative caching minimizes the need to access distant data sources, thereby reducing network traffic and improving response times for users. This approach is particularly beneficial in distributed environments where resources are geographically dispersed, and network latency is a concern [7], [43], [44].

In contrast, uncooperative caching operates independently, with each cache entity making caching decisions based solely on its local data access patterns and resource availability, without actively collaborating with other caches in the system. Without shared information about cached data and eviction policies, uncooperative caching may lead to increased cache misses and redundant data transfers across the distributed system, ultimately impacting the overall performance and responsiveness of the system. While uncooperative caching simplifies cache management by decentralizing decision-making processes, it often falls short in maximizing cache efficiency and exploiting the collective caching capabilities available in distributed environments [7], [45].

2.4.4 Coded vs uncoded caching

Coded caching is a sophisticated technique employed in modern communication systems that exploits the inherent redundancy in content demand patterns. By dividing content into smaller, coded segments stored across various caching nodes within the network, coded caching enables simultaneous transmission of multiple content segments to multiple users. It capitalizes on the benefits of network coding to satisfy diverse user requests with minimal bandwidth consumption. During the retrieval process, users request specific content segments, and the caching nodes cleverly combine these requests to transmit coded information, thereby minimizing the overall data transmission overhead. Leveraging coded caching allows network operators to significantly reduce bandwidth usage and latency, especially in scenarios where multiple users access the same content simultaneously [46], [47].

Uncoded caching, on the other hand, represents a simpler approach to content caching where the content is stored in its original form at caching nodes within the network. When users request specific content, the caching nodes retrieve and transmit the requested content directly to the users without any additional coding or processing. While uncoded caching lacks the optimization benefits of coded caching in terms of bandwidth efficiency

and latency reduction, it still offers advantages in scenarios where the computational complexity associated with coding and decoding operations is a concern [46], [47].

In scenarios where the user cache size is small and the base station cache is sufficiently large, uncoded caching tends to outperform coded caching, leading to higher throughput. However, as the user cache size increases or the base station cache becomes relatively smaller, coded caching becomes more efficient, resulting in improved throughput due to optimized data delivery mechanisms [34].

2.4.5 Centralized vs decentralized training

In centralized training data from multiple sources is aggregated, collected, and processed in a single location, typically a centralized server or data center, where machine learning models are trained using the combined dataset. Centralizing the data simplifies the training

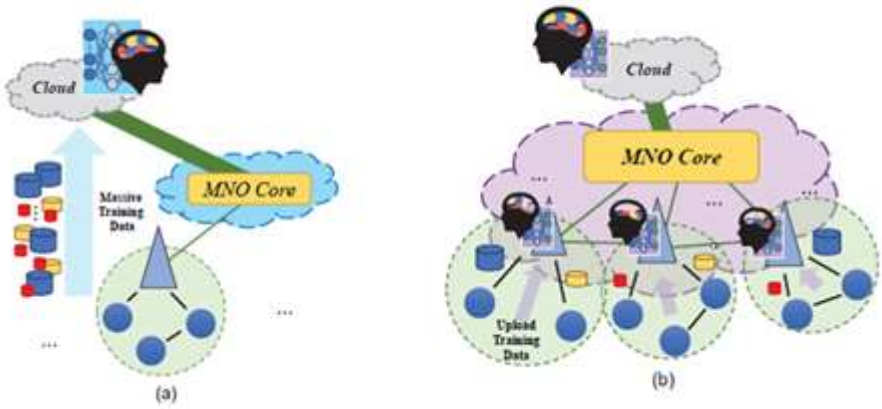

Figure 2.13: a)Centralized training, b) Distributed training [49]

process by consolidating information into a single, accessible location. Centralized caching relies on a central controller with a comprehensive overview of network status to establish caching strategies. This controller typically monitors user mobility patterns and channel state information by analyzing received requests and is shown in Figure 2.13.a. Consequently, centralized caching aims to achieve optimal caching performance through effective caching decisions, such as content placement. The central controller must handle significant traffic volumes, placing a substantial burden on both the controller and the links

between it and network entities. This situation can potentially lead to the central controller becoming a bottleneck in the mobile caching system [7], [15], [48].

Distributed or decentralized training harnesses the computational power of edge nodes in edge computing networks. By spreading model training across multiple edge devices, the workload on any single device is minimized, enabling efficient resource usage and scalability. Additionally, decentralized learning encourages collaboration among edge devices, allowing them to learn collectively from diverse data sources while safeguarding individual data privacy [7], [15], [48].

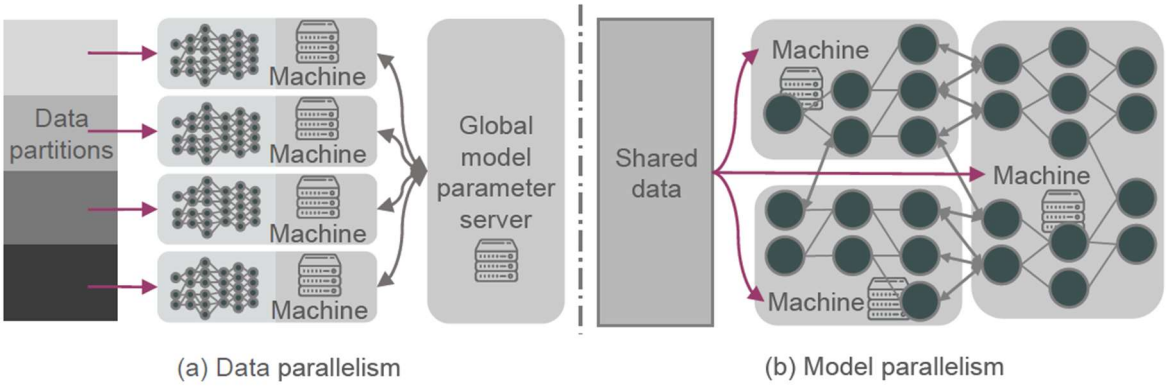

Figure 2.14 Distributed training in terms of data and model parallelism.[15]

There are two primary methods for conducting distributed training: data parallelism and model parallelism as shown in Figure 2.13.b. Model parallelism involves dividing a large DL model into multiple segments, then concurrently training these segmented models with data samples. This approach not only enhances training speed but also addresses scenarios where the model exceeds device memory capacity. Thus, avoiding the challenges of training a large DL model that typically demands significant computational resources. On the other hand, data parallelism involves splitting data into multiple partitions and training separate model copies in parallel, each with its own assigned data samples. This method enhances the efficiency of model training [7], [15], [48].

In distributed caching cache nodes make decisions, such as content placement and updates, solely based on their local information and information from neighboring nodes. This approach is particularly useful in scenarios where neighboring base stations are jointly optimized to enhance cache hit probability. By accessing content from multiple nearby caches, the apparent cache size for users can be expanded. Distributed caching is

inherently less complex compared to centralized methods. However, algorithms based on distributed caching may not always achieve globally optimal solutions since they only have a view of the local networks [7], [15], [48].

2.5 Underlying assumptions and challenges

2.5.1 User mobility awareness

User mobility is a crucial consideration in the design and implementation of edge caching systems, where user's movements within the network dynamically impact their proximity to cache servers and caching decisions [7], [36].

In scenarios where users are assumed to be static, edge caching offers benefits such as reduced latency, alleviated network congestion, and enhanced content availability and reliability. However, static user assumptions may lead to inefficiencies in cache utilization and resource allocation due to shifting user demand patterns. It creates challenges such as cache invalidation, where content cached based on users' preferences at one location may become irrelevant as the users move, necessitating efficient cache management strategies to seamlessly adapt to user movements [7], [36].

In contrast, schemes that consider user mobility present unique advantages and challenges. By dynamically adjusting cache placement based on user movements, these schemes can adapt to changing user demand patterns more effectively, potentially reducing cache misses and improving content delivery efficiency. These approaches usually involve using the historical trajectory of the user to predict their future one. However, accommodating user mobility introduces complexities in cache management and content replication, requiring constant updates to cache content based on user locations and robust caching algorithms to maintain session continuity and data consistency during user handovers [7], [36].

2.5.2 Content popularity and request distribution

Content request distributions are commonly analyzed to anticipate future request patterns, providing data for preemptive caching. Many studies presume that the arrivals of requests adhere to either the Poisson process or Markov process. Nevertheless, these mathematical models may not always suit actual traffic situations. By examining past requests, machine learning or big data analysis methods could be employed instead [7], [50], [51].

Content popularity is a crucial aspect of the content placement decisions, as it estimates the probability of user requesting certain contents. Many studies rely on the Zipf distribution to describe content popularity. Zipf's law suggests that a small number of items will be highly popular, while a large number of items will have very low popularity. However, the Zipf model assumes that content popularity remains constant, and that each user's request is independent. This approach overlooks the temporal and spatial correlations of content popularity, where temporal correlation reflects fluctuations in popularity over time and spatial correlation indicates varying content preferences across geographical regions and social-cultural contexts. Additionally, while Zipf has been shown to be accurate in modeling video downloads, it may not be suitable for characterizing other types of data [7], [50], [51].

Some researchers assume the content popularity dynamic and unknown doing so offers many different advantages. Firstly, incorporating dynamic content popularity allows for a more accurate representation of real-world scenarios where content catalogues are continuously evolving, with new content being introduced regularly. By considering dynamic popularity, caching systems can better adapt to changing user preferences and content consumption patterns, leading to improved overall performance. Additionally, dynamic content popularity enables a more efficient allocation of caching resources, as it allows systems to prioritize caching resources for content that is currently trending or in high demand, thereby maximizing cache hit rates and reducing cache miss rates. Moreover, by embracing dynamic content popularity, caching systems can stay relevant and competitive in dynamic environments where user interests and content trends evolve rapidly [7], [50], [51].

For training the deep learning models, datasets from MovieLens [52] Kaggle and IQIYI were used. These contain user ratings for movies, YouTube videos and celebrity

information. The most popular dataset to be used was the MovieLens dataset by far, both when assuming a Zipf content popularity distribution and an unknown and dynamic popularity [7], [50], [51].

2.5.3 Security and privacy

Privacy and security concerns in wireless networks encompass various threats and vulnerabilities that can compromise the confidentiality and integrity of data transmissions. The broadcast nature of wireless networks renders them susceptible to passive eavesdropping and active jamming, enabling malicious actors to intercept data or disrupt legitimate transmissions. Cryptographic techniques are commonly employed to mitigate these threats by securing data against interception. However, privacy concerns extend beyond transmission security, encompassing issues such as unauthorized access to cached content and the potential for data modification or injection of malicious code, which can compromise user information and the integrity of delivered content. In response to increasing privacy sensitivities among consumers, policymakers have enacted regulations like the European Commission's General Data Protection Regulation (GDPR) and Consumer Privacy Bill of Rights in the US to limit data collection and storage to what is necessary and consented by consumers. Several privacy concerns have been identified, including the potential for adversaries to compromise or control edge caches, extract sensitive information from stored data, and perform inference attacks to identify user information like trajectory and personal identifiable information. Despite efforts to anonymize data transmissions, advances in big data technology have empowered attackers to re-identify individuals by combining anonymized data with non-anonymized public data users (e.g., shopping and reading preferences, locations, and photos). Privacy concerns are a big adversary with machine learning methods that require information such as user mobility patterns, preferences and social and threatens their integration by discouraging privacy-sensitive consumers from taking part in model training [7], [8], [13], [53].

2.5.4 Time sensitive data

Time-sensitive data, by definition, pertains to information that holds a limited window of relevance, necessitating immediate action or analysis to extract its full value. This temporal characteristic inherently means that its significance diminishes over time [37], [54], [55].

While traditional cached contents predominantly comprise files and videos, it's important to note that document files typically exhibit a higher tolerance for time, whereas videos are categorized as time-sensitive data. The surge in video content, fueled notably by applications like remote education, video conferencing, and on-demand video and impose specific time constraints, known as deadlines, to uphold QoS and user satisfaction. The failure to meet these deadlines risks compromising QoS and dissatisfy users [37], [54], [55].

2.5.5 Cache dimensioning

Cache dimensioning involves determining the appropriate amount of storage space to allocate, addressing both the challenge of how much memory each edge node should have and how to configure it effectively within the network infrastructure. In edge networks, this process is crucial for optimizing cache size and configuration at various edge locations to enhance network performance and reduce costs, making it a more appealing option. Increasing cache memory can improve the cache hit ratio, thus reducing traffic congestion in the core network in contrast, smaller cache sizes may lead to more frequent cache misses and higher latency but offer advantages such as lower hardware costs and simplified management. Striking the right balance between the benefits of larger cache sizes in enhancing user experience and the drawbacks of increased infrastructure complexity and costs is crucial. It’s essential to consider specific use cases, traffic patterns, and content dynamics when determining the optimal cache size. Leveraging technologies like software-defined and virtualization can add flexibility to cache dimensioning decisions in edge environments by dynamically adjusting cache sizes based on the needs of content delivery network providers [7], [18], [56].

Chapter 3 Deep Learning Outline

Deep learning, a subset of machine learning, harnesses artificial neural networks to emulate human learning patterns and decision-making processes. Its foundation lies in deep neural networks, composed of interconnected nodes or artificial neurons across multiple layers, enabling data processing and learning. These networks autonomously unveil data representations through hierarchical learning [10], [14], [15], [18], [57].

The term "deep" signifies the multitude of layers transforming data, facilitating the system to grasp data representations with increased intricacy and abstraction in each layer. A deep neural network encompasses the input, hidden, and output layers, where the output of each layer, is processed by mathematical functions called activation functions, and fed into the subsequent layer. Typically, a network is regarded as deep if it comprises more than three layers including the input and output layer. Ultimately, the final output embodies the model's prediction [10], [14], [15], [18], [57].

For a deep learning model to make useful decisions in needs to undergo training. Deep learning training involves the iterative process of feeding large amounts of data into a neural network model to enable it to learn and improve its performance on a specific task. This training typically starts with data preprocessing to clean, normalize, and prepare the input data for the model. During training, the model adjusts its internal weights and biases, usually by trying to reduce the error between the predicted and actual output. This process is repeated multiple times using optimization algorithms to minimize the loss function and enhance the model's predictive accuracy. Deep learning training requires significant computational resources and can take a long time depending on the complexity of the task and the size of the dataset [10], [14], [15], [18], [57].

In edge caching this lengthy process would be prohibitory for the systems performance so most approaches follow a two-phase plan when deployed, separating prediction from training. In the prediction phase the model can quickly decide what content will be popular and precache them in proactive caching or decide whether to cache the requested content and what to evict, in reactive caching. While in the training phase, typically executed every few hours the model can use the information observed about content requests since the

previous training phase, to update the model and adapt to improve its accuracy [10], [14], [15], [18], [57].

The classification of deep neural networks encompasses three main training categories: supervised learning, unsupervised learning, and reinforcement learning and will be analyzed below.

3.1 Supervised learning outline

In Supervised learning the network learns from labeled training data, using input and corresponding output labels to discern patterns and relationships. The primary goal is for the model to extend its understanding from the training data to new, unseen data by minimizing the gap between predicted and actual outputs. Techniques such as classification and regression aid in assigning inputs to predefined categories or predicting continuous outcomes. While this method significantly enhances decision-making, the requirement for labeled data can be a practical challenge [10], [14], [15], [18], [57].

Obtaining labeled data can be expensive, time-consuming, or even infeasible in some cases. Additionally, the quality of the labels can significantly impact the performance of the model. This task is complicated by the constantly changing nature of network traffic and the diverse behaviors of users, making it difficult to clearly label which content should be cached at the edge. Additionally, scalability is a significant issue due to the large-scale distributed nature of edge caching systems and the vast amount of content to manage. This requires centralized coordination for training models, which goes against the decentralized principle of edge computing. Furthermore, the dynamic environment of edge caching, where content popularity and user access patterns constantly change, makes it hard for supervised learning models trained on past data to adapt effectively. The cold start problem adds to these challenges, new added content may not have enough historical data for accurate predictions. The basic structures of supervised learning are shown in 3.1 and are analyzed below [10], [14], [15], [18], [57].

3.1.1 Fully connected neural networks

Fully connected Neural Networks (FNNs), often known as dense neural networks, form a type of artificial neural network design where every neuron in a layer links to each neuron in the following layer as is presented in Figure 3.1. As every neuron in a FNN is interconnected, each element of the input vector plays a role in determining the final output. This means that FNN possesses the capability to capture global patterns. Although without additional data processing techniques, they might struggle to focus on local patterns. According to the universal approximation theorem, FNN has the ability to approximate any closed and bounded function with enough neurons in the hidden layer. However, the complete interconnection of neurons leads to a large increase in complexity, moderate performance, and slower convergence rates making them unfit for real-world tasks [10], [14], [15], [18], [57].

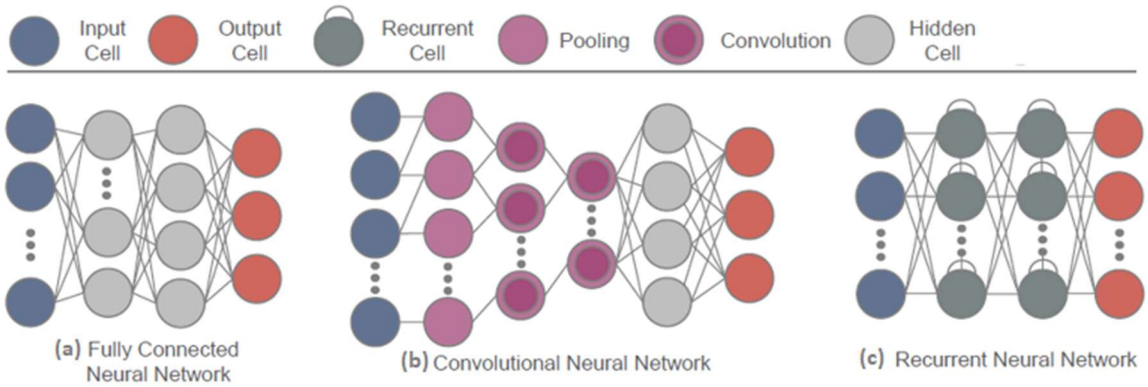

Figure 3.1 Basic structures of typical supervised DNNs, modified from [15]

3.1.2 Convolutional neural networks

Unlike fully connected networks, Convolutional Neural Networks (CNNs) significantly reduce the number of parameters, making them more efficient for learning hierarchical representations. CNN introduces a structure with stacked convolutional and pooling layers, followed by fully connected layers as shown in Figure 3.1. The convolutional layer uses sliding convolutional filters with weight sharing, ensuring constant filter weights during processing. Meanwhile, the pooling layer conducts down sampling, reducing the number of training parameters, though this process risks losing information. However, challenges such as susceptibility to overfitting, particularly with limited data, and the computational

complexity in extremely deep networks are prevalent limiting their popularity in edge caching [10], [14], [15], [18], [57].

3.1.3 Recurrent neural networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks specialized for handling sequential data by preserving memory of prior inputs. Differing from the hidden neurons in a feedforward neural network, the output of a recurrent neuron relies on both the current output of the preceding layer and its last hidden state as shown in Figure 3.1. This enables RNNs to make use of sequential data and dynamic temporal patterns, like those found in user mobility. For instance, an RNN can forecast the movement patterns of mobile devices and wireless users by analyzing their historical location. While FNNs can approximate continuous functions, RNNs, can emulate a universal Turing machine, having the capability to solve all computational problems. RNNs are able to discern and capture temporal patterns, such as recent shifts in content popularity, the evolving trends of popularity over time, and the correlation between content popularity and the current moment [10], [14], [15], [18], [57].

However, RNNs face challenges, such as vanishing or exploding gradients, which hinder their capacity to retain prolonged dependencies within the data. Vanishing and exploding gradients refer to issues during training where the gradients become extremely small or large, respectively, as they propagate through many layers. Vanishing gradients occur when the gradients become very small as they backpropagate through many time steps in the network during training. This leads to slow or stalled learning because the weights receive very small updates, causing the model to have difficulty capturing long-term dependencies in the data. Exploding gradients, on the other hand, occur when the gradients become very large during training, causing the weights to be updated by excessively large amounts. This can lead to numerical instability and make the training process unstable or divergent [10], [14], [15], [18], [57].

Below and in Figure 3.2 different advanced variants of RNN are introduced like LSTMs and GRUs specialized to solve the mentioned gradient problem, as well as Seq2Seq, pointer networks and ESNs specialized in sequence modeling and generation.

3.1.4 Long short-term memory and gated recurrent unit

The Long Short-Term Memory (LSTM) is a more complicated architecture of RNN designed to avoid the exploding/vanishing gradient problem. The term "Long Short-Term Memory" stems from the fact that network stores long and short-term dependencies based on historical information, akin to a form of memory. Unlike traditional RNNs that use a single feedback loop that utilizes information from events that happened long ago, and events that happened recently to make predictions, LSTMs uses two different memory paths, for short and long-term memory. The long-term memory path is called the cell state, while the short-term memory path is called the hidden state, also the combination of the cell state with the hidden state and the current input is called the inner state [14], [15], [18], [57].

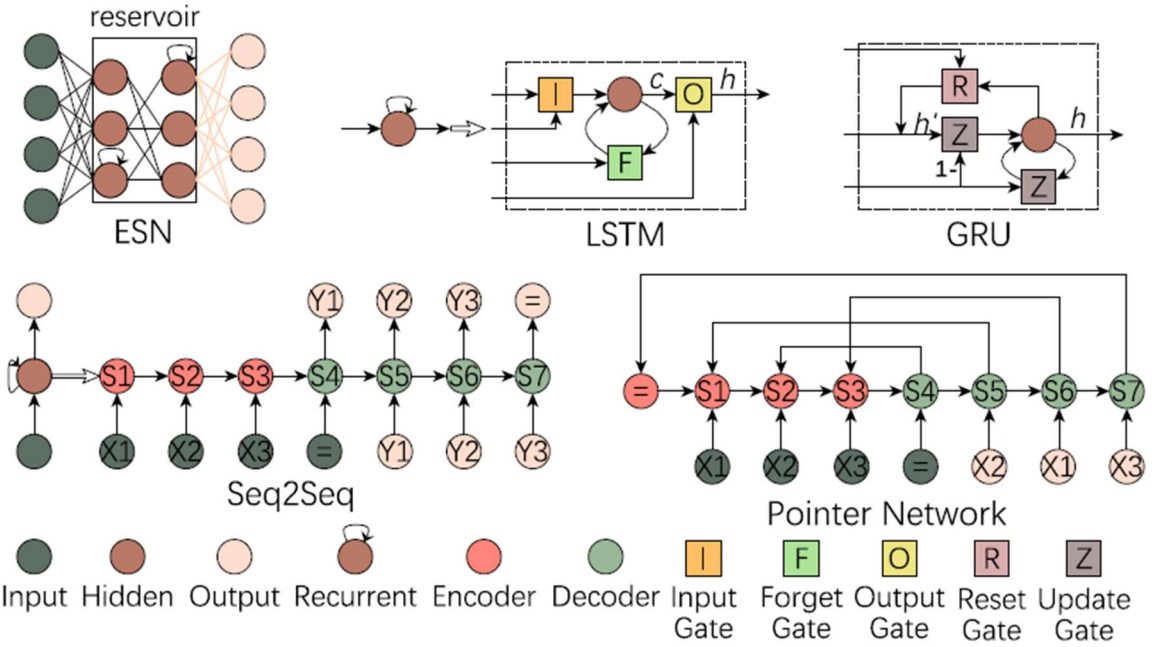

Figure 3.2 Recurrent Neural Network Variants [18]

Internally, LSTM introduces three gates (input gate, forget gate, and output gate) to control signal propagation and is presented in Figure 3.2. These gates control whether information is passed into the memory cell, more specifically the forget gate determines how much of the current long-term memory to remember going forward, the input gate determines how the long-term memory is influenced by the new input, while the output gate determines how much the short-term memory is affected by the input. These gates combine the inner state with their weights to produce the next hidden and cell state. This arrangement helps

alleviate the problem of gradient vanishing by allowing old memories to influence current decisions [14], [15], [18], [57].

Through the learning process, computations stored in the memory cells remain intact over time, resulting in improved performance, especially when dealing with data characterized by long dependencies. However, the complexity of implementing gate mechanisms in LSTM poses challenges, leading to slower training and testing processes. Additionally, due to the temporal causal nature of LSTM and common RNNs, parallel training like CNNs is not supported increasing the training time [10], [14], [15], [18], [57].

Utilized in [42], and [58], Bidirectional Long Short-Term Memory (Bi-LSTM) is a type of recurrent neural network architecture designed to capture dependencies in sequential data both in forward and backward directions. Unlike traditional LSTMs that process sequences in one direction, Bi-LSTMs employ two separate hidden layers, one processing the input sequence in the forward direction and the other in the reverse direction. This allows the model to effectively capture both past and future context for each time step in the sequence, enabling more comprehensive understanding of temporal data.

A more straightforward alternative involves using a GRU. Unlike LSTM, which relies on separate input and forget gates, GRU consolidates these functions into a single gate, called the update gate, to manage the balance between retaining previous memory and integrating new information. Additionally, GRU eliminates the output gate found in LSTM, as both the hidden state and inner state of LSTM already encapsulate the information from the previous state. Instead, it uses a reset gate that is responsible for how much of the past information should be forgotten or reset when computing the current state of the network [10], [14], [15], [18], [57].

GRUs feature a simpler architecture due to their reduced number of gates, resulting in faster training times but have a slightly reduced capacity in capturing complex long-term dependencies compared to LSTMs [10], [14], [15], [18], [57].

3.1.5 Seq2seq and pointer networks

Sequence-to-Sequence (Seq2Seq) models and Pointer Networks are specialized architectures using an encoder-decoder model for sequence modeling and generation.

In Seq2Seq models the encoder component processes historical data, such as past content requests and user interactions, to generate a fixed-length context vector representing the relevant features of the input sequence. This context vector is then fed into the decoder component, which generates predictions of future content requests. These predictions guide the caching decisions at the edge servers, enabling them to proactively cache content that is likely to be requested soon. Nonetheless, it encounters difficulties when the size of the output is determined by the length of the input, as it relies on a fixed output dictionary [10], [14], [15], [18], [57].

Pointer Networks, a variant of Seq2Seq models, aim to overcome the limitations of Seq2Seq by allowing the model to dynamically utilize parts of the input sequence as part of the output dictionary, thus reducing the fixed vocabulary problem. This allows pointer networks to address the challenge of handling variable-length input sequences, such as user request patterns or content popularity trends. Pointer networks are employed to dynamically choose which content items to cache, considering factors like content popularity, user preferences, and network conditions. Additionally, although Seq2Seq models traditionally utilize RNN structures, particularly LSTM and ESN, Pointer Networks deviate in their application modes [10], [14], [15], [18], [57].

3.1.6 Echo state networks

Echo State Networks (ESNs) constitute a distinct variant of RNNs characterized by their unique architecture and operational efficiency. Their structure comprises a fixed input layer, a significantly large recurrent hidden layer with randomly initialized connections, and a trainable output layer. The hidden layer, often termed the "reservoir," maintains sparse connections among neurons, preserving information from previous instances akin to an echo. To extract diverse features, a sufficiently large reservoir becomes necessary, posing a design challenge in ESN implementation. Their advantage primarily stems from the simplified training process, as only the output layer requires adjustment, while the recurrent connections in the hidden layer remains fixed. This design choice contributes to their computational efficiency and reduces the risk of overfitting. The principal strength of ESNs lies in their exceptional capability to model and forecast temporal and sequential data, particularly in tasks necessitating the capture of temporal dependencies. In wireless

networks, ESNs have been applied for various natural applications, such as content prediction, resource management, and mobility pattern estimation [10], [14], [15], [18], [57].

They exhibit strength in tasks where the output directly depends on input sequence elements. However, challenges may arise in scenarios where the output space is vast and less directly related to the input sequence. Furthermore, the performance of ESNs heavily relies on the random initialization of the hidden layer, which can be a limitation in certain cases [10], [14], [15], [18], [57].

3.2 Unsupervised learning outline

Unsupervised learning operates solely on input data, without the need for corresponding output labels. Its fundamental objective involves understanding data by modeling the underlying structure or distribution within the information. These architectures operate independently, exploring inherent patterns or structures in the data. Techniques like clustering, grouping similar data points based on features, and dimensionality reduction for simplifying complex data, while retaining important characteristics, are commonly employed in unsupervised learning. The primary aim is to unveil hidden patterns and relationships, aiding in exploratory data analysis and enhancing comprehension of the intrinsic nature of data. Often, unsupervised learning acts to complement or as a precursor to sophisticated supervised learning tasks, facilitating a comprehensive understanding of datasets.

3.2.1 Autoencoders

Autoencoder has two parts, an encoder and a decoder as seen in Figure. 3.3, each with its own learnable parameters. The encoder takes the input and maps it to a lower-dimensional latent space, and then the decoder reconstructs the input from this representation. Their main strength is in learning data representations efficiently, helping with tasks like reducing dimensions and extracting features. Autoencoders are quite handy for predicting content popularity by learning and pulling out important features from the data. This is

useful in areas like content recommendation systems, trend forecasting, such as in social media, streaming platforms, and online content distribution [10], [12], [15], [59].

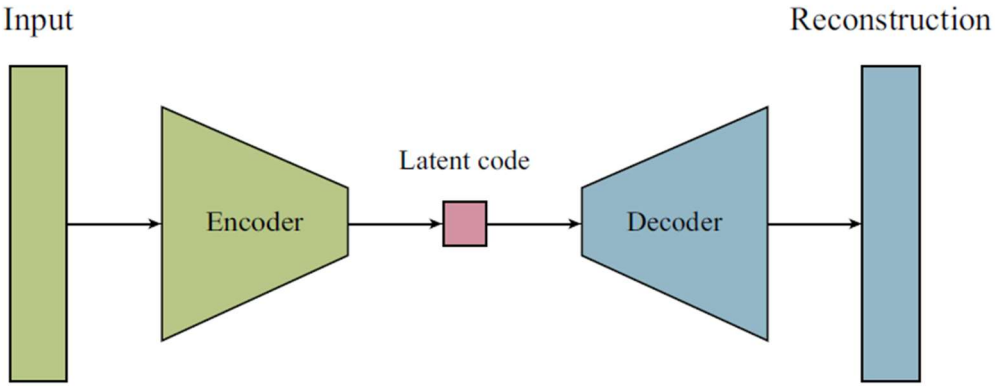

Figure 3.3 Structure of an auto encoder [59]

However, they face challenges, like overfitting in complicated datasets and struggle to work well with new, unseen data. This shows their reliance on having diverse and high-quality training data. To measure how well they work, the reconstruction error is typically used, which checks how similar the input and output are [10], [12], [15], [59].

Different variations of auto encoders have been developed to cater to different data characteristics and specific tasks: [59], [60].

1) **Denoising autoencoder:** In the training phase, denoising autoencoders intentionally introduce noise or corruption to the input, compelling the network to learn a more resilient and generalized representation. The objective is for the model to reconstruct a clean and accurate version of the original input despite the introduced noise. This process not only aids in mitigating overfitting but also fosters the development of a latent space that captures essential features of the data. The deliberate introduction of noise during training enhances the robustness and generalization capabilities of the model in the face of real-world data imperfections.
2) **Sparse autoencoders:** This variance introduces a regularization mechanism that encourages the activation of only a subset of neurons in the network, promoting the creation of sparse and selective features. This sparsity constraint is often achieved through penalties in the loss function or by explicitly limiting the number of active neurons. By encouraging sparsity, these autoencoders offer a way to capture the most relevant information in the input data, contributing to more effective and concise feature learning.

3) **Stacked autoencoders:** This variance chains multiple autoencoders, making the output of one autoencoder the input of the next. This hierarchical architecture allows stacked autoencoders to learn increasingly abstract and complex representations of the input data through successive encoding and decoding layers. Each layer captures different levels of abstraction, enabling the model to discern intricate patterns and features in the data. The depth and complexity of these models contribute to their capacity to automatically extract meaningful features and patterns from diverse and high-dimensional datasets.

3.3 Reinforcement learning outline

Reinforcement Learning (RL) constitutes a machine learning framework where an agent learns to make decisions through interactions with an environment, aiming to accomplish specific objectives without relying on labeled datasets. The agent undergoes a trial-and-error process, receiving feedback in the form of rewards or penalties based on its actions. The primary goal is to derive the best strategy or policy that maximizes cumulative rewards over time. Through a balance of exploration (trying new actions) and exploitation (taking actions that have provided high rewards previously), the agent takes actions, observes the environment's state, receives feedback, and adjusts its strategy to make better decisions in future interactions [10], [14], [15], [18], [57], [61].

RL can be categorized into 3 main categories, value-based models, policy-based models, and hybrid models. Value-based models typically estimate the value of each action in a given state and then select the action with the highest estimated value. Policy-based models directly parameterize the policy, i.e., the strategy or behavior that the agent follows to select actions in different states of the environment [10], [14], [15], [18], [57], [61].

Unlike value-based methods that estimate the value of taking actions in states, policy-based methods learn a policy directly without explicitly computing value functions. When comparing the two the value-based method tends to be less stable and can face challenges with convergence. However, it usually requires fewer samples to learn effectively. On the other hand, the policy-based approach is more susceptible to converge in local optima due to the larger search space it explores [10], [14], [15], [18], [57], [61].

Hybrid based models combine elements of both value-based and policy-based approaches. It usually takes the form of an actor critic network. The actor network learns a policy that selects actions, while the critic network learns to evaluate the value of those actions. By using both value and policy information, actor-critic architectures can achieve more stable and efficient learning compared to using either approach alone. However, this increased complexity comes at a higher computational cost and training time [10], [14], [15], [18], [57], [61].

Only value based and hybrid models are used in the examined work, they are expanded upon bellow.

3.3.1 Deep Q-learning

Deep Q-Learning (DQL) integrates the Q-learning strategy with deep neural networks to address intricate decision-making challenges within diverse states. Q-learning is a model-free reinforcement learning algorithm used to learn an optimal policy for decision-making in an environment. It is a value-based method that learns to estimate the value of taking a particular action in a given state, commonly referred to as the Q-value. The Q-value represents the expected cumulative reward obtained by taking a specific action from a particular state and following the optimal policy thereafter [10], [14], [15], [18], [57], [61].

DQL is employed in edge caching problems due to its ability to effectively handle raw, high-dimensional inputs, allowing agents to make informed decisions while maximizing cumulative rewards. Several optimization techniques are employed to enhance DQL algorithms, ensuring their efficiency and stability [10], [14], [15], [18], [57], [61]:

1. **Experience Replay:** Experience replay stores past experiences in a replay buffer, allowing the algorithm to sample and learn from a batch of experiences rather than learning from one state transition at a time. This technique helps in breaking the temporal correlation among consecutive samples and improves the learning efficiency [10], [14], [15], [18], [57], [61].
2. **Prioritized Experience Replay:** Deep Q-Learning with prioritized experience replay enhances the experience replay technique by introducing a prioritized approach to choosing replay transitions. It prioritizes the sampling of experiences stored in the

replay buffer based on their significance in the learning process. By focusing on experiences that lead to higher learning progress, it enhances the efficiency of learning [10], [14], [15], [18], [57], [61].

3. **Fixed Target Network:** It utilizes a separate target network for calculating the target Q-values, which changes less frequently compared to the primary Q-network. This approach stabilizes learning by reducing Q-value estimation fluctuations during training [10], [14], [15], [18], [57], [61].
4. **Double DQL:** In traditional DQN, when an action is selected and evaluated, it often involves using a maximum value that surpasses the actual Q value, leading to an overly optimistic estimation. To address this issue, double DQN mitigates the overestimation bias in Q-learning by using two separate estimators to evaluate the value of an action. It achieves this by decoupling the action selection from the action evaluation process. One estimator determines the best action, while the other estimates the value of that action. This straightforward adjustment significantly decreases overestimation, resulting in a faster and more dependable training process [62].
5. **Dueling DQL:** The Q-learning estimates both the state-value and action-value functions. The dueling networks architecture represents the Q-function with two distinct estimators. One estimator is dedicated to the state value function, while the other focuses on the state-dependent action advantage function. Therefore, this architectural design has the potential to improve policy evaluation in scenarios with multiple actions of similar value [63].

3.3.2 Actor critic networks

In Actor-critic models, there are two main components: the actor network and the critic network. The actor is responsible for determining the policy, i.e., the actions to take in a given state, while the critic evaluates the benefit of the actions taken by the actor by estimating the expected cumulative reward as shown in Figure 3.4. This feedback from the critic is then used to update the actor's policy, allowing it to learn and improve over time. By implementing both policy and value estimation, actor-critic networks can achieve better sample efficiency and stability compared to other reinforcement learning approaches at the cost of a higher computational cost [18], [35], [61], [64], [65].

Since the critic uses value-based methods all the enchantments available for value-based models can be utilized. Several architectural choices appear when designing an actor-critic network [18], [35], [61], [64], [65].

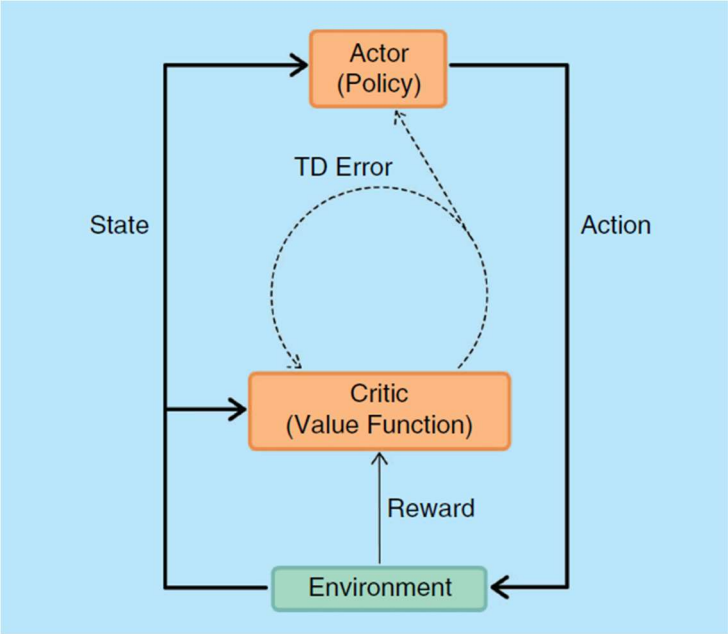

Figure 3.4 Actor-critic logic [61]

1. **Deep Deterministic Policy Gradient (DDPG):** A type of actor-critic method, that stands out from others primarily due to its focus on continuous action spaces, employing a deterministic policy rather than stochastic ones commonly found in discrete action spaces. Which means that given a particular state, the policy network outputs a precise action rather than a probability distribution over possible actions [66], [67].
2. **The Wolpertinger architecture** implements the K-Nearest Neighbors (KNN) between the actor and the critic. In training the actor network picks a best action based on the current situation. Then, the K-nearest neighbors algorithm finds similar actions to the one selected. Next, the critic network evaluates at all these actions and selects the one with the best expected outcome. This architecture helps the actor network avoid dealing with a huge number of actions by narrowing down choices while the KNN helps to broaden the range of actions to prevent bad decisions [68], [69].
3. **Multi-Agent Actor-Critic (MAAC)**: In many real-world scenarios, such as cooperative caching, multiple agents need to coordinate their actions to achieve a common goal or interact with each other. MAAC networks can be multi-actor

multi-critic or multi-actor single-critic. In edge caching multi actor-single critic is preferred, where actor networks are deployed in the edge nodes for selecting the best caching action based on the current parameters and caching state, then the critic network takes as input the caching states of all edge nodes and the action of each actor and evaluates their value. This design is based on the idea that the effectiveness of an edge nodes action depends on the combined state of the entire system and the individual actions taken by each node[65], [68].

3.4 Federated learning outline

Unlike traditional methods where data is centralized on a server for analysis, Federated Learning (FL) distributes the learning process across multiple devices or nodes, allowing computations to be performed locally without sharing raw data. These sources perform training on their respective local data, sending only the model updates back to the central server.

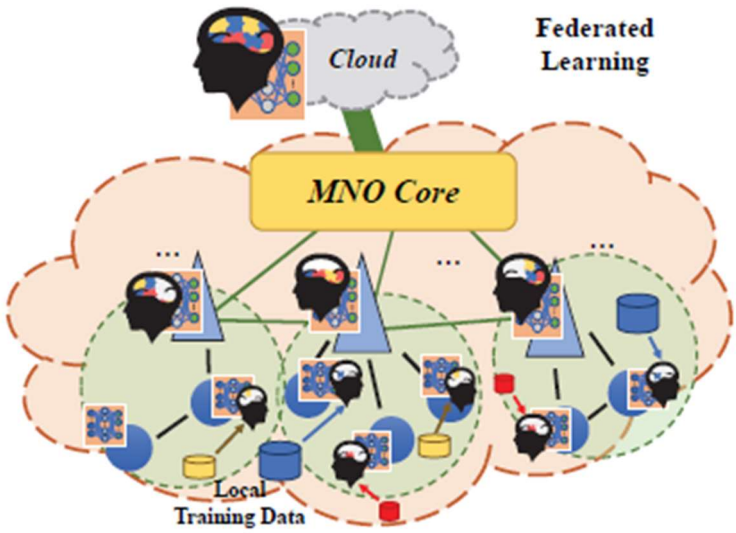

Figure 3.5 Federated Learning over mobile edge, modified from [49]

This decentralized approach enables organizations to collaborate on training machine learning models while ensuring that sensitive information remains secure and private. The server aggregates these updates from various locations, thus enhancing the global model. FL has wide-ranging applications, from improving predictive models edge caching from predicting content popularity and enhancing personalized recommendations in online services without exposing user data. FL describes the mentioned distributed training model but does not determine how each of those models are trained. For training those models supervised, unsupervised and reinforcement learning can be employed [13], [49].

However federated learning is not completely free from the potential inference of sensitive information, even without direct data exchange. Malicious participants can deduce private details like gender, occupation, or location from shared models. Solutions like differential privacy-based mechanisms and collaborative training frameworks aim to mitigate these threats by adding noise to parameters and limiting information sharing to preserve participant privacy[70].

Chapter 4 Deep Learning for Data Caching

In the reviewed of studies employing deep learning methodologies, patterns emerge concerning the selection of research problems, performance metrics, network architectures, caching scheme location and choices.

Bellow the summaries of different studies using deep learning for edge caching is presented, presenting their various architectures and methodologies. They are divided into four categories depending on the learning method used into supervised, unsupervised, reinforcement and federated.

4.1 Supervised learning schemes

The prevalent optimization objective, with supervised learning, in the examined research is content popularity prediction, followed by user mobility prediction. This choice can be explained due to the availability of labeled data in those fields, necessary for supervised learning. Labeled data for video content can be obtained through feature extraction methods, to enable prediction of future popularity of content, while historical data and patterns of the users' mobility can be used as labels to predict their future position.

The majority of studies, using supervised learning, choose cache hit rate as their primary performance metric. This is influenced by the fact that, most researchers using supervised learning choose the use of uncooperative caching in their schemes, meaning the focus is on the performance of individual nodes. Therefore, the cache hit rate emerges as a suitable solution for analyzing a single cache's strategy performance. A departure from conventional metrics is presented in the [71], where the focus shifts towards minimizing the monetary expenses from the perspective of the application vendor.

When it comes to network architecture LSTM are the most popular choice, employed by more than half of the examined research. Also, LSTM appears to be the most effective approach as evidence by the fact that when similar neuron topologies (depth and number of neurons) are compared LSTM outperforms ESN, GRU and CNN [58], [71], [72], [73], the only time GRU outperforms LSTM is in [74] when used in a cascading topology.

Table 3.1 Summary of supervised learning research for data caching

Study	Metric Target	Dl Architecture	Dl objective	Main Conclusions
[75]	QoE	CNN	Predict and cache future popular content to improve QoE and traffic load	Model for sequential data outperformed the one for general data
[48]	Cache hit rate	CNN	Proposes both a centralized and a distributed model for content prediction popularity	The distributed model outperforms the centralized model, without revealing private user information
[71]	Cache hit rate Latency	LSTM	Minimize the user access delay of video content by storing popular video contents at base stations	LSTM models outperform CNN and CRNN with LSTM models on all key metrics
[36]	Cache hit rate	LSTM	Predicts the user's mobility based on their historical trajectory	Content prediction without location prediction has similar hit rate but consumes more storage
[76]	Cache hit rate	LSTM	Cache replacement policy by learning the pattern of content popularity	The proposal requires no data preprocessing, preprogrammed model, or additional information
[42]	Cache hit rate	Bi-directional LSTM	Propose a combined proactive and a reactive caching policy based on predicted popularity to maximize hit rate	Can quickly monitor different trends in popularity, including spatial-temporal popularity, and user dynamics
[58]	Cache hit rate	Bi-directional LSTM	Proactive caching, by learning the temporal patterns in user requests	Outperforms Bi-GRU and Bi-RNN, deeper LSTM layers with longer time steps improve performance
[72]	Cache hit rate	LSTM	Maximize MVNO cost reduction by leasing optimal cache capacity and storing popular video contents at base stations	LSTM outperforms both CNN and CRNN architectures on all metrics both for cache demand prediction and content popularity prediction

[73]	Cache hit rate Latency	ESN LSTM	Proposes content caching algorithm for D2D networks, focusing on the content placement, and delivery problems	LSTM has better cache hit rate than ESN when solving the content placement problem
[37]	Cache hit rate Latency	ESN	Maximize saved delay while considering the cache capacity and deadline constraints	Outperforms other cooperative and non-cooperative prediction based caching schemes
[77]	System cost	Pointer network with LSTM cells	Caching method to minimize the overall system cost with lower service latency	Outperforms other prediction-based caching schemes that don't use deep learning
[74]	Cache hit rate	Cascading GRU	Content popularity prediction and cache placement optimization in D2D networks	In testing cascading GRU outperformed cascading LSTM and CNN models
[78]	Cache hit rate	Evolving GRU	Cache eviction policy that learns the time-varying content popularity	Starts as a shallow network with fast convergence and a and evolves into a deeper network as requests come

In Reference [75], the authors propose two proactive caching models aimed at enhancing QoE and alleviating traffic load. One model is designed for general content, while the other is tailored for content featuring sequential characteristics, with the objective of predicting user interests. The general model relies solely on historical requests and is relevant only to current users. It employs a combination of CNN and self-attention mechanisms to extract semantic features from the general content. Content popularity dictates storage allocation, with popular items being stored in high-capacity nodes such as MBS, while less popular ones are stored in smaller stations like small, femto, or pico base stations. In contrast, the sequential model employs a CNN with self-attention to predict subsequent content pieces. This model anticipates the next content item and stores it on edge nodes proximate to the user device. Both models exhibit a minimal runtime of one training epoch. The sequential model demonstrates slightly superior performance, which aligns with expectations given that the MovieLens dataset, used for training has sequential characteristics.

In their study documented in [48], the authors present two proactive caching methodologies employing deep learning techniques to anticipate user content preferences. The first approach adopts a centralized training strategy, where the content server gathers data from MENs and utilizes a deep learning algorithm to forecast content demand across the entire network. This prediction is utilized to proactively cache content at MEC locations. However, this centralized training method raises privacy concerns as MENs must share their local user data with the CS. In contrast, the second approach employs a distributed training methodology. Each MEC independently utilizes deep learning to generate local output matrices. These MECs then transmit their local gradients to a central CS. The CS consolidates these gradients into a global gradient, updating the global model, which is subsequently communicated back to the MENs. This information exchange facilitated by the CS fosters collaborative learning among MENs, thereby improving prediction accuracy. Although the specific type of DNN is not explicitly mentioned, the authors' descriptions suggest characteristics of a feedforward network, resembling a CNN. Overall, the distributed approach demonstrates superior performance in terms of prediction accuracy, communication efficiency, and cache optimization compared to the centralized approach.

In Reference [71], the authors address the challenge of minimizing user access delay for video content by employing a caching strategy at BS. They separate the content popularity problem into two steps: first, predicting the future popularity class labels of video contents, and second, forecasting the future request counts of video contents. Their proposed model involves a central controller, located at the cloud data center, which is responsible for training a deep learning model using data collected from the base stations. Subsequently, the controller sends the list of popular contents to the base stations for caching purposes. To determine the optimal model configuration, the authors introduce a random search method that explores various configurations from models such as CNN, LSTM, and CNN with LSTM cells. The best-performing model is then selected for training. Among the evaluated models, LSTM consistently outperforms both CNN and CNN with LSTM cells across key metrics including training accuracy, validation accuracy, and training time. The results demonstrate that the proposed scheme exhibits superior performance across all key performance indicators such as cache hit probability, backhaul usage, and average delay.

In Reference [36], propose a scheme to predict the user's mobility using their historical trajectory based on a LSTM network, and predict the users interest using gradient boosting decision tree. Testing shows the LSTM approach outperforms different prediction algorithms such as RNN and GRU, in terms of prediction accuracy. By experimenting with various numbers of cached replicas, which involves storing a copy of the data at the predicted location and different neighboring positions, the researchers determined that the most effective combination is caching at the predicted location along with the two nearest neighboring positions. Their conclusion is based on the observation that the hit rate, or the rate at which the requested data is successfully retrieved from the cache, does not exhibit a significant increase beyond this combination. While location prediction does not affect the hit rate significantly, it reduces the storage space consumed.

In Reference [76], the authors propose a cache replacement strategy that employs a deep LSTM network to discern content popularity trends across both short and long time frames. This network is constructed by stacking multiple fully connected LSTM layers. The effectiveness of their approach is evaluated across three distinct scenarios: dataset management within a datacenter, at a base station, and under the control of a CDN service provider. Their approach circumvents the need for data preprocessing, pre-programmed models, or additional information to inform cache replacement decisions. Instead, the network autonomously learns to make such decisions by capturing past request sequences through its internal memory cells. The authors argue that implementing their approach in a real-world caching system is straightforward, as it does not necessitate changes to the caching server itself. Merely integrating the prediction and training modules into the existing cache server is enough. Furthermore, the approach does not demand a powerful GPU, a standard CPU is deemed sufficient to handle real-world request loads.

The authors of [42], introduce a proactive and reactive caching policy centered on predicted popularity. This approach is divided into two phases. In the offline phase, user location prediction and content popularity prediction are performed to optimize caching efficiency. A Bi-LSTM network, is utilized to learn popularity/location prediction models, considering the dynamic nature of popularity trends and user mobility patterns. The popularity series is divided into trend, seasonal, and irregular parts, with preprocessing to remove irregularities. Different prediction models are assigned to various popularity trend classes to accurately predict content popularity. The offline phase includes user location

prediction, popularity prediction, and pre-caching tasks, which can be executed centrally by the cloud server or distributedly by the F-APs. The F-APs determine potential users, based on location predictions and cache popular content accordingly, while the UEs adjust location prediction models and train popularity prediction models iteratively to improve accuracy. The computational complexity of the offline phase can be managed efficiently due to the separation of responsibilities between UEs and F-APs and the flexibility in implementation. During the online phase of the proposed edge caching policy, the F-AP implements a reactive caching scheme to track content popularity after requests. If the content feature is not in the content feature database, KNN algorithm is used to determine the trend class, then the model specific to that trend class is employed to forecast its popularity and determine if it will be cached by comparing it to the least popular content in the cache. They support that considering changes in spatial-temporal popularity, evolving trends, user behavior, and random data, their suggested strategy accurately and quickly monitors different popularity patterns.

In reference [58], the authors propose an online cache scheme for proactive cache management, aiming to predict future requests by learning the time-series association of user requests. The neural network architecture is structured into three main components: a CNN block for feature extraction and dimension reduction of input data, a Bi-LSTM block for capturing temporal associations in user requests and generating time-series predictions, and a FNN block to enhance overall prediction performance. Tests conducted by replacing the Bi-LSTM block with Bi-GRU and Bi-RNN alternatives revealed that the Bi-LSTM block consistently outperformed the others in terms of validation loss. Additionally, testing confirmed that the Bi-LSTM's ability to learn long-term dependencies is further enhanced by utilizing longer time steps, resulting in improved accuracy in predicting requested files.

In Reference [72], the authors propose a comprehensive strategy aimed at maximizing cost reduction for mobile virtual network operators by optimizing cache capacity leasing and strategically storing popular video content at BSs, thus minimizing backhaul usage. The suggested approach comprises three interconnected steps. Utilizing RL to determine the most suitable prediction model. Leasing the optimal cache space based on predictions derived from the selected model. Storing popular video content at BSs based on predictions generated by the same model. The authors, assume that tasks such as model search and computational operations will be conducted at the cloud datacenter, so computational

complexity is not considered. Results obtained through DQL indicate that the LSTM architecture consistently outperforms CNNs and CRNNs across all key metrics including training accuracy, validation accuracy, and training time, for both cache demand prediction and content popularity prediction.

In [73], the authors introduce an algorithm designed for content caching in D2D networks, with a focus on addressing content placement and delivery challenges. The algorithm uses two RNNs, each tasked with predicting distinct aspects: one predicts user locations, while the other predicts content popularity. Users expected to visit the same location are grouped together, and their collective content request distribution is forecasted based on various user attributes. Once both RNNs are trained, the authors integrate the content request distribution with mobility patterns, enabling them to organize requested content and make informed decisions regarding caching locations. Two RNN prediction models, the ESN and LSTM, are considered and evaluated. Results indicate that LSTM outperforms ESN in terms of cache hit rate when addressing the content placement problem. For dynamic content delivery decision-making, the authors employ a DRL algorithm. This approach effectively reduces overall time delay and transmission consumption associated with requested content.

In Reference [37], the authors tackle the cache placement problem within mobile edge networks with the objective of maximizing saved delay. Their approach involves cooperative caching and takes into account the provision of service, to deadline-sensitive content, along with the available cache capacity of each BS. Their proposed algorithm operates within a framework known as the Network Evaluation Function (NEF) , which computes and allocates contents to individual MEC nodes. The NEF acts as a coordinator, maintaining indexes of the content cached at individual MECs and monitoring content requests by users at each MEC. To predict content request distribution, an echo state network is employed, establishing a relationship between requested content and user context information. Subsequently, a fuzzy logic algorithm evaluates the content to be cached in the base station based on the content request prediction results, as well as considering the benefits and deadlines associated with the content request.

In Reference [77], the authors present a caching method aimed at minimizing the overall system cost while reducing service latency from the perspective of the app vendor. The system cost encompasses several factors: data caching cost, which includes the expense of

storing data and purchasing edge resources from a provider. Latency cost, where content cached farther away from the edge user incurs higher latency costs. Constraints are set by the app vendor to ensure smooth operation. Penalty cost, representing the loss of benefit resulting from the failure to satisfy the user's data request from the edge server. To optimize edge caching, the authors propose a method based on deep learning. Their approach utilizes a combination of LSTM cells, and a pointer network. The LSTM cells are employed to address challenges related to sequential data, preventing issues such as vanishing gradients. The pointer network is capable of processing input sequences of varying lengths and producing corresponding variable-length output and it enhances the model's ability to identify crucial information during decoding. Furthermore, the attention mechanism, incorporated into the pointer network, improves the extraction of relevant information.

In Reference [78], the authors introduce a cache eviction policy designed to adaptively learn the time-varying content popularity. To mitigate the computational overhead associated with DNN, they propose a network architecture that starts as a shallow network for quick convergence and progressively deepens as requests accumulate over time. Their approach employs a multi-layer GRU architecture, where each hidden layer representation is linked to an output regression, facilitating evolving learning through hedge backpropagation. This mechanism enables the network to autonomously determine the optimal depth and adapt its architecture dynamically. For cache replacement, the authors aim to emulate the MIN algorithm, which prioritizes content with the longest expected time until the next visit. This is achieved with the content popularity prediction. The algorithm operates in two phases: an offline phase where it learns to predict popularity using the network, and an online phase where, if a requested content is not cached, its features are extracted, and its predicted popularity is assessed to determine caching. To ensure the prediction of content popularity remains accurate and up to date, the algorithm periodically retrains with new information, continually refining its predictions.

4.2 Unsupervised learning schemes

In the examination of unsupervised learning schemes some familiar patterns emerge with, a heavy emphasis on predicting content popularity as the objective, and with the cache hit rate serving as the main performance metric alongside prediction accuracy. Despite the availability of various unsupervised learning frameworks, the auto-encoder architecture consistently emerges as the preferred approach.

The quantity of studies utilizing unsupervised techniques in edge caching, serves as an indicator for their perceived effectiveness by researchers. The limited availability of literature using unsupervised learning-based methods, in comparison to supervised, reinforcement, and federated methods suggests a prevailing academic notion that this type of deep learning may not be particularly effective in addressing the challenges associated with edge caching.

Table 3.2 Summary of unsupervised learning research for data caching

Study	**Metric Target**	**Dl Architecture**	**Dl objective**	**Main Conclusions**
[60]	Cache hit rate	Auto encoder Stacked denoising auto encoder	Predict content popularity by using extracted user and content features	The feature-based setting in the framework helps overcomes the data sparsity and cold-start problems
[79]	Prediction accuracy	Collaborative denoising autoencoder	Predict most popular video content.	Has high prediction accuracy but suffers from the cold start problem
[80]	Cache hit rate	Stacked auto encoder	Content popularity prediction in software defined networks	The prediction model is distributed and reconfigurable

In Reference[60], the authors propose a method to extract features from users and content, to predict content popularity and decide what to cache at base stations. User features include personal characteristics such as demographic information, explicit context meaning information where the details are clearly described such as weather conditions, and latent context meaning, finding hidden patterns from mobile sensors to understand and represent the user's context. An auto-encoder is used to determine the user features and extract them in low-dimensional representation. Then a stacked denoising autoencoder is utilized to the extract content features. A content popularity matrix is used, that maps a popularity rating between user and content, to achieve that a feature based collaborative filtering method is used that estimates the content popularity matrix. It enables the prediction of unknown entries in the content popularity matrix by incorporating side information through feature engineering. Testing indicate that incorporating a feature-based approach in the framework offers more insights into user-content interaction mitigating the challenge of cold start and data sparseness.

In Reference [79], the authors propose a caching method by predicting video popularity. A collaborative denoising auto encoder is first utilized to extract user features, then similar users are grouped into the same cluster and each cluster fine-tune an autoencoder model separately. Each model makes a list of the top-K popular videos and eventually all these lists are combined by an ensemble method to formulate the final top-K popular videos to cache. To predict the top-K popular videos the model uses the users' viewing history data which includes userIDs, item IDs and timestamps. Testing shows the model seems to suffer from the first appearance problem, meaning some videos may become very popular right away, but it's difficult to forecast them in advance since it has no information about the future. So, it performs best when implemented on a platform that don't frequently introduce new material such as Netflix but struggles in platforms like YouTube where there is a lot of new content.

In Reference [80], the authors propose a method to predict content popularity in SDN. The natural connections between massive compute nodes in the SDN make it a natural fit for use of deep learning by addressing the problem of the slow model training. To achieve content prediction multiple switches in the distributed SDN collect the features of the content being requested, which contains inherent spatial-temporal and social correlation. Then a stacked auto-encoder network is used to understand these features in a global view

and find the distribution pattern of content popularity. The network is pre-trained by bottom-up unsupervised learning and fine-tuned by top-down supervised learning. With SDN's programmability and the SDN controller's capability to oversee the network globally, the hidden layers and neurons in each layer can be easily modified. This programmable network allows for the straightforward deployment, adjustment, and removal of their method across the network.

4.3 Reinforcement learning schemes

In reinforcement learning schemes, the objective that receives the most attention continues being predicting popular content, While the previous method mainly focused on the behavior of a single base station, the proposed reinforcement learning techniques place a lot of attention on collaborative approaches, where the edge devices communicate to reduce the caching of duplicate content on nearby nodes.

When it comes to performance metrics cache hit ratio shares the focus with latency. Latency is often the metric of choice in collaborative approaches where the cache hit rate is not completely representative of the performance of the system since the content could be strategically stored in nearby nodes. So, latency serves as a measurement to gauge the effectiveness of the technique both in the content popularity prediction accuracy but also the placement on the content on the appropriate node.

Hybrid model approaches seem to be on par in popularity with value-based model approaches. When considering value-based approaches, DQL dominates as the implementation choice, indicating their effectiveness on dealing with the high state space. A combination of optimization techniques such as, experience replay or prioritized experience replay, fixed target networks, and DQL variants are always utilized to make the training more efficient and stable, highlighting their performance impactfulness.

In hybrid approaches, the DQL component consistently assumes the role of the critic, while the critic itself adopts either a DDPG framework or a specific architectural design is not mentioned. The same optimization is employed for the DQL part of the network, while architectures such as the Wolpertinger architecture and MAAC network are employed to boost the overall performance of the network.

Table 3.3 Summary of reinforcement learning research for data caching

Study	Metric Target	Dl Architecture	Dl objective	Main Conclusions
[81]	Cache hit rate	DQN with experience replay and fixed target network	Proposes a caching strategy for UE, FAP, and BS based on content popularity	Can handle variations of content popularity in a timely manner
[82]	CSP cost	DQN with experience replay and fixed target network	Content caching strategy to minimize CSP cost in D2D	A reverse auction model is employed to reward user in partaking in D2D offloading
[83]	Cache hit rate	DQN with experience replay and fixed target network	A temporal-spatial recommendation strategy to guide users' requests	Adding recommendations reduces cache misses
[84]	QoE	DQN with experience learning, fixed target network and adaptive learning	Popularity prediction algorithm to maximize QoE	Strikes a balance between the accuracy of Q- values and the stability accelerated by DRL
[85]	Trans-mission cost	DQN with experience replay	Proactive caching policy for user device and base station	Reduces the state space dimensions by applying a divide and conquer strategy
[63]	Energy efficiency	Recurrent dueling DQN with prioritized experience learning	Content popularity prediction algorithm aiming to maximize energy efficiency	Can be applied to real systems with power sensitive data retrieval settings
[86]	Latency	Double DQN with experience replay	Cooperative replacement caching strategy to reduce access delay and traffic backhaul	Does not require any prior knowledge of content popularity distribution

[62]	Latency	Double DQN with fixed target network.	Collaborative caching strategy aimed to optimize latency	Outperforms similar DQN and dueling DQN architectures. Has long training period
[67]	Latency	Actor critic with fixed target network and experience replay	Joint optimization of caching, computing, and radio resources	Latency is depended on the Fog node to UE ratio
[64]	Cache hit rate	DRL with Wolpertinger architecture and experience replay	Content prediction to maximize long term cache hit in a centralized scenario	Proposed framework is efficient in handling large-scale data
[68]	Cache hit rate Latency	Multi agent DRL with Wolpertinger architecture	Predicting content for better long-term cache hits in a decentralized setting.	Despite its high computational cost, it achieves high performance
[65]	Cache hit rate	Multi agent actor-critic	Maximize cache hit rate while considering inconsistent IoT data item size	Supports both online work and offline training, allowing for efficient updates and cooperation among distributed edge servers
[35]	Network cost	DDPG with target network and experience replay	Coded content popularity and placement optimization	Pretraining heavily improves the per-slot optimization

In Reference [81], a distributed DQL content caching scheme is proposed for F-RAN, focusing on user preference prediction and content popularity prediction. For user preference prediction, an unsupervised learning method called Probabilistic Latent Semantic Analysis (PLSA) is employed. PLSA utilizes a matrix representing historical user behaviors to capture relationships between users, files, and topics, resulting in a matrix reflecting the individual interests of each Fog user edge. This matrix, along with a network topological relationship matrix, is used to derive a content popularity matrix that

represents the common content preferences across different Fog access points and base stations. To determine the optimal caching strategy, a DQL algorithm is utilized, integrating the content popularity matrix with the properties of CNNs and Q learning, along with experience replay. To enhance stability a fixed target network is used meaning, two CNNs with the same structure but different parameters are used. The first network predicts target Q values during training and updates less frequently, periodically synchronizing with the second network, which evaluates actions in the current state and updates more frequently to predict Q values more accurately. Regarding the content update algorithm for FAPs and BSs, an update timer is utilized, while for Fog user edges, the status is immediately updated upon requests, eliminating the need for a synchronized timer.

The authors of [82] propose a joint caching mechanism, which considers both the content caching strategy and an incentive mechanism to enhance the performance of D2D offloading from the perspective of the CSP. The idea is that D2D caching offers significant benefits to the CSP by reducing the overall cellular traffic and associated costs. To encourage user participation in D2D offloading, an incentive mechanism is introduced. Given the resource-intensive nature of offloading, a reverse auction mechanism is suggested as part of the incentive scheme. This reverse auction aims to maximize cost savings for the content service provider, thereby motivating users to engage in D2D offloading. To optimize the content caching method and select appropriate caching nodes for each content, a DQN method is proposed. Optimization techniques like fixed target network with experience replay are employed to enhance the efficiency and effectiveness of the caching mechanism by leveraging experience replay to improve the training process and optimize the selection of caching nodes for content distribution in D2D offloading scenarios.

In [85] , the paper addresses the intricate challenges surrounding joint pushing and caching within a MEC network characterized by multiuser and multicast data. The authors present a solution to this optimization problem using hierarchical RL tailored for systems with large state and action dimensions. The complexity of the problem leads to its division into two subproblems: one centered on optimizing user-side policies and the other on optimizing base station-side policies in the context of caching and pushing. By breaking down the problem, the aim is to reduce its dimensions and manage the optimization task effectively through a divide-and-conquer strategy. The proposed hierarchical RL-based

policy approach involves iteratively fixing the policy on one side (user or base station) while optimizing the other side. This iterative process entails utilizing a Q-learning algorithm with value approximation for user-side policy optimization and a DQN for base station-side policy optimization

In Reference [86] a cooperative replacement caching strategy is introduced with the aim of minimizing end-to-end delay within the system. To achieve this objective and effectively reduce the average delay experienced by users, the authors propose the utilization of Double DQL with experience replay. This approach allows the system to dynamically adapt to changing environments without the need for prior knowledge of content popularity distribution.

In Reference [62], a collaborative caching strategy is introduced to optimize the user experience in terms of delay. Upon completion of a user's computing request by the node, the user compensates the node based on the delay of computing completion. If the computation falls within the specified time limit, the payment is determined by the actual delay, otherwise, no payment is made if the edge node exceeds the time limit. The objective is to maximize the total payment while considering computing and storage resource constraints. To address this challenge, a double DQN is proposed to make decisions and maximize long-term gains. To assess the performance of the Double DQN, it is compared against a dueling DQN and a natural DQN with identical replay memory size, training steps, and learning rate. Test results indicate that the double DQN outperforms the other reinforcement learning agents in terms of task success rates, training loss, and overall system performance. However, it requires a longer training period before making better decisions, thus necessitating a while to achieve optimal performance. Despite this requirement for extended training, the double DQN demonstrates superior effectiveness in optimizing the collaborative caching strategy and enhancing the user experience in mitigating delay.

In reference [84], the authors aim to increase QoE by proposing a popularity prediction caching algorithm optimizing storage cost and latency. To achieve that a DQL approach is proposed utilizing different techniques such as fixed target network (F) experience replay(E), and adaptive learning(L) calling it the FEL approach. Each technique contributes to the improvement of the algorithm in a different way, fixed target network benefits stability by using a separate target network with fixed parameters for a certain

duration. Experience replay increases temporal relativity by delivering batches of the batches of the dataset in a random way during training, and adaptive learning rate increasing efficient learning by optimally adjusting the learning rate based on the age of the sample, preventing large mistakes in transition probability and average reward from negatively affecting the training process. A plethora of different approaches were used in testing to compare the results such as DQL, FE-DQL and Actor-Critic DRL, and results show that FEL-DQL demonstrated superior performance across various metrics such as cache hit ratio, latency, and storage cost, making it the most promising for QoE optimization.

In Reference [63], a content popularity predictions algorithm is presented to maximize energy efficiency. To optimize the problem a deep RNN is proposed that utilized prioritized experience learning and a dueling Q function. The recurrent nature of the network allows it to capture non-stationary patterns and is well suited for adapting to changing conditions, while priories experience learning, and the dueling architecture contribute to faster learning. In testing the proposed algorithm demonstrates its superior learning rate and scalability as well as energy efficiency relevant to the compared DRL methods. They conclude that their proposed DRL based cache algorithm can be applied in real systems for which the data retrieval power is sensitive.

In reference [83], the authors introduce a proactive caching strategy at the BS designed to optimize network throughput and improve user experience by incorporating user behavior-related information. To achieve this goal, the authors integrate a recommendation technique with BS caching, resulting in a temporal-spatial recommendation policy. This policy guides mobile users to request their preferred files at appropriate times and locations, effectively shaping local file popularity. To optimize both the recommendation and caching policies, a DQN with experience replay and fixed target network is employed. Notably, this approach does not rely on a priori knowledge of user preferences, the impact of recommendations on request probability, or mobility patterns. Through simulation results, the proposed approach demonstrates a reduction in cache misses compared to policies without any recommendation or temporal-spatial recommendation. This underscores its potential to enhance cache efficiency while respecting user preferences, thereby improving overall network throughput and user experience.

In [67], the authors introduce an actor-critic network aimed at minimizing end-to-end latency and reducing backhaul traffic. This study proposes a holistic approach that addresses the challenges arising from limited radio resources, caching storage, and computational capabilities by considering both content caching and computation offloading. The state vector encompasses parameters such as the number of requests, content sizes, popularity, Signal-to-Interference-plus-Noise Ratios (SINRs), among others. Meanwhile, the action vector represents decisions regarding Fog node assignment, subchannel allocation, caching, and computation offloading. To ensure the stability and effectiveness of the proposed algorithm, techniques such as fixed target networks and experience learning are employed to prevent divergence, while the Natural gradient method is utilized to avoid converging to local maxima. Through testing with varying numbers of UE and Fog nodes, it is revealed that the latency is significantly influenced by the ratio of Fog nodes to UE, emphasizing the importance of considering this ratio in the design of the system for minimizing latency and optimizing backhaul traffic.

In [64] , the authors introduce a cache replacement policy that takes into account user preferences and file popularity patterns. They investigate policies for both centralized and decentralized scenarios, aiming to optimize cache hit rates. A deep learning network with the Wolpertinger architecture, is employed due to its adaptability to online data, enabling the development of a long-term policy. Within this framework, a KNN component is introduced between the actor and the critic network , that aids in expanding the range of actions, preventing suboptimal decisions A single actor-critic architecture is utilized, where the action space involves either maintaining the cache status quo or replacing one selected content with the requested one. The state space comprises the feature space of the cached contents, while rewards are determined based on short-term and long-term cache hit rates. Training is conducted using the DDPG method. Tests reveal that the suggested framework can achieve competitive cache hit rates while significantly cutting runtime when compared to DQL based policies. Because of this, the suggested structure is effective at managing massive amounts of data.

In their subsequent study detailed in reference [68] the authors build upon their previous research in reference [64] , wherein they refine and adapt their approach for decentralized cooperative caching. This updated policy is designed to optimize both caches hit rate and transmission delay. Once again, they employ the Wolpertinger architecture, which has

been modified to specifically address this objective. Training in this approach is done using the Temporal Difference error, with a focus on minimizing the least squares temporal difference. They introduce a multi-actor single critic architecture, where each actor is assigned to a different base station, resulting in every actor having a different state and action space. The single critic is responsible for evaluating the corresponding state value for every possible action set combination. Through testing, the authors demonstrate the advantages of their approach in terms of adaptability and long-term stability compared to policy-based strategies. This highlights the effectiveness of their decentralized cooperative caching policy in achieving the desired optimization goals.

In their study [65], the authors introduce a cooperative algorithm designed to maximize hit rates utilizing a MAAC network. This algorithm integrates the actor network, which operates in edge servers to make caching policy decisions, while the critic network is situated in the central server. To address the variability in IoT data item sizes, the authors propose a dynamic adjustment of cooperation levels based on content size. To tackle the challenge of an exponentially growing action space with cache size, the authors introduce a caching policy selection scheme. This scheme defines caching policies by features, effectively reducing the action space to a manageable fixed-length vector, ensuring the validity of all actions. Additionally, the action space is adjusted to the size of the requested data item and current caching states. The reward system in this algorithm departs from a standard hit rate, instead redefining it as a weighted sum of local and neighboring server hit rates. The cooperation levels are dynamically adjusted through weights, emphasizing either local performance or assistance to neighbors based on the context. Moreover, the algorithm is designed to support both online work and offline training, facilitating efficient updates and collaboration among distributed edge servers. This cohesive approach ensures effective cooperation among actors, addressing challenges related to data item sizes and action space growth while optimizing hit rates in edge server caching policies.

In [35] , the proposed caching strategy aims to minimize the total discounted network cost, which includes transmission delay, by caching popular content. To achieve this goal, a two-step learning approach is implemented. Firstly, a neural network is trained using supervised learning, specifically a clustering LSTM, to predict the number of content requests based on historical data. This initial training involves predicting content requests and optimizing caching decisions. Then, a Supervised DDPG approach is proposed to

determine how much coded fraction of each file should be stored in each cache node based on the predicted content requests. To accelerate the training process, both the actor and critic networks are initially trained using supervised learning. During this phase, samples are generated by solving an approximate problem that minimizes the per-slot cost rather than the overall network cost. The strategy assumes the utilization of coded caching, wherein each file can be divided into several segments cached across various nodes in a distributed manner. Furthermore, the results demonstrate that the proposed SDDPG approach outperforms the per-slot optimization method, indicating its efficacy in improving caching efficiency and minimizing network costs.

4.4 Federated learning schemes

In federated learning schemes, cache hit rate and latency remain the primary performance metrics due to the utilization of collaborative caching strategies. These schemes recognize that evaluating their effectiveness solely based on cache hit rate is insufficient. Prediction accuracy is crucial, but the placement of content also holds significance in collaborative approaches. Therefore, employing cache hit rate alone does not provide a comprehensive evaluation of the solution's effectiveness. Instead, considering latency helps assess the relevance of content placement, contributing to a more thorough evaluation of these algorithms.

As mentioned earlier, federated learning is about training data in a decentralized way, but there isn't a predetermined way to do it. Different methods can be used, most commonly in the examined research, federated learning is combined with reinforcement learning, followed by supervised learning. This is in line with the observations that researchers tend to prefer reinforcement and supervised learning over unsupervised learning for edge caching with deep learning. Displaying their perceived effectiveness of unsupervised learning solving the problems associated with this domain. Federated learning scheme’s main objective focus is a joint endeavor to predict popular content and strategically place it.

Table 3.4: Summary of federated learning research for data caching

Study	Metric Target	Dl Architecture	Dl objective	Main Conclusions
[87]	Cache hit rate Latency	Hierarchical federated DRL	Proposes a hierarchical federated DRL model for content placement optimization	Clustering the edge devices by hierarchical architecture addresses the issues that rise from choosing either small or large groups.
[44]	Cache hit rate Latency	Federated double DQN	Decentralized cooperative framework for content popularity prediction	Lightweight and easy deployment
[88]	Cache hit rate Latency	Federated DQN	Proposes a D2D assisted collaborative edge caching framework	The attention mechanism solves the imbalance problem of local model quality
[70]	Classifica-tion accuracy	Federated aggregated GRU	Intrusion detection in wireless edge networks	Using attention mechanisms to assign weights to different devices based on importance increases the safety of the overall systems
[89]	Latency	Federated FNN	Distributed cluster-based user preference estimation to optimize caching placement	Mitigates the adverse effects of non-IID datasets in federated learning
[90]	Classifica-tion accuracy	Federated learning	Navigating constraints such as limited bandwidth and available uplink capacity	Optimizing user selection, transmit power and resource improves accuracy

In [87], the authors present the Hierarchical Federated Deep Reinforcement Learning (HFDRL) method, tailored for collaborative edge caching, utilizing Federated Deep Reinforcement Learning (FDRL) to optimize cache hit rates and diminish content access delays. The primary objective is to determine the optimal group size for cooperation by hierarchically clustering edge devices at multiple levels. This hierarchical architecture addresses challenges associated with selecting either small or large groups, allowing for the integration of benefits from both. These clusters not only collaborate in caching but also contribute to learning a comprehensive model for content replacement strategies. Moreover, the approach aims to enhance the performance of both local and global networks. Through testing, the authors demonstrate that this method leads to increased hit rates for both average and peak requests compared to single-level FDRL. This cohesive approach emphasizes the importance of hierarchical clustering in improving cache performance and reducing content access delays in collaborative edge caching scenarios

In [44] the FADE (Federated DRL-based Edge Caching Algorithm) framework introduces is an approach to optimizing caching strategies in wireless systems using federated deep reinforcement learning. Utilizing double DQN and experience replay, FADE refines the caching replacement process. FADE employs a federated learning architecture, wherein UEs collaboratively learn a shared predictive model, enabling decentralized model training while maintaining data localized on individual IoT devices. The workflow involves the dispersion of initial parameters from the central BS to UEs, iterative updates based on local computations, and subsequent aggregation of these updates at the central BS. The proposed convergence analysis ensures the effectiveness of the gradient descent algorithm, underscoring the robustness of the FADE algorithm. Additionally, the lightweight design of FADE is emphasized for efficient deployment in IoT devices, highlighting its suitability for real-world applications.

In [88], the authors introduce an attention-weighted federated approach utilizing reinforcement learning for D2D assisted collaborative caching, aimed at optimizing node selection and cache replacement policies. The proposed Adaptive Weighted Federated Deep Reinforcement Learning (AWFDRL) framework tackles the challenges in collaborative edge caching through a three-phase approach. The model release phase involves UEs associating with local BSs, reporting their states, and receiving the global model from the previous round. In the local DQN model training phase, each local agent

trains its model based on global parameters and local data. Subsequently, the aggregation phase utilizes an attention mechanism to calculate aggregation weights, considering training evaluation indicators like average reward, loss, hit rate, and more. This weighted federated aggregation accounts for heterogeneity in computing capacity, data quality, and model performance among UEs, thereby enhancing the robustness of the model. Furthermore, the convergence analysis ensures stability and effectiveness in local training and aggregation, providing a theoretical basis for the AWFDRL framework. Notably, the integration of an attention mechanism to regulate model weights during federated learning aggregation addresses the imbalance in local model quality, contributing to the overall cohesion and effectiveness of the proposed approach.

The authors of [70] present, FedAGRU (Federated Aggregated Gated Recurrent Unit), a FL algorithm tailored for intrusion detection in wireless edge networks. Essentially, this algorithm enhances the learning process of a central server from multiple edge devices without compromising sensitive data. FedAGRU employs an attention mechanism to assign varying weights to updates from different devices, depending on their significance to intrusion detection. Subsequently, these updates are combined, taking into account the importance of each device's contribution. Moreover, the algorithm addresses potential attacks, such as poisoning, which involves malicious attempts to compromise the integrity and reliability of ML models by injecting manipulated or fraudulent data during training. This is achieved by assessing the importance of each device's data and safeguarding the overall model from harmful influences.

In reference [89] the authors propose a framework for D2D-assisted Fog computing networks, integrating FL, content caching, and D2D communication to enhance communication efficiency and user experience. The framework introduces a clustered FL strategy to implement personalized caching based on user preferences. Additionally, the content caching model is designed to adapt to various scenarios, accommodating the density and mobility of users. Moreover, the paper presents a D2D cooperation model that facilitates content fetching from cooperative IDs via D2D communication links, effectively reducing traffic burden. The establishment of a D2D communication link between two devices necessitates short physical distance, membership in the same social community, and ensuring that the required content is cached in the recipient device. Furthermore, an

optimization problem is formulated to minimize content fetch delay while considering factors such as physical link quality, social connection strength, and transmission rate requirements. To address non-IID data and enhance learning efficiency, the paper introduces a Cluster-based User Preference Estimation (CUPE) algorithm, involving ID clustering and cluster-based federated learning. Through testing, the study demonstrates that updating the parameters of the local model via local training and cooperation with global parameters significantly improves network performance.

The authors of [90] present a scheme that addresses the challenge of training federated learning algorithms in realistic wireless networks, where bandwidth constraints limit uplink transmission. The article highlights the significant influence of wireless packet transmission errors on the FL model's parameter update process. The quality of training is directly impacted by wireless factors such as packet errors and resource availability. To overcome these challenges, the BS carefully selects users to execute the FL algorithm, aiming to create an accurate global model while minimizing the FL loss function. This integrated problem of joint learning, wireless resource allocation, and user selection is formulated as an optimization problem with the objective of minimizing the FL loss function. To find a solution, a closed-form expression for the expected convergence rate of the FL algorithm is derived to quantify the impact of wireless factors. Using this expected convergence rate, the optimal transmit power for each user is determined under a specified user selection and uplink Resource Block (RB) allocation scheme. Ultimately, the user selection and uplink RB allocation are optimized to minimize the FL loss function.

Chapter 5 Implementation Challenges and Research Directions

This chapter explores the challenges in the implementation of deep learning in edge computing and discusses the potential research directions. Addressing these challenges is crucial for enabling the mass adoption of deep learning algorithms for edge caching while additional research in promising directions may offer efficient ways to deal with those challenges and improve on the current proposed implementations.

5.1 Implementation challenges

5.1.1 The Cost of deep learning

The utilization of DL enhances performance efficiency in caching policies, yet it also introduces additional costs. These costs stem from the resources and time required for training and deploying DL models. Consequently, a trade-off arises between the savings achieved through DL-assisted caching policies and the resources needed to operate DL itself, resulting in potential profit, loss, or break-even outcomes. Hence, careful consideration is essential regarding when and where to implement DL. This is particularly crucial for various deep learning frameworks employing specialized processing units like GPUs. Addressing the cost of deep learning in edge caching scenarios necessitates the development of lightweight model architectures, efficient model compression techniques, and specialized hardware accelerators tailored for edge environments [18].

5.1.2 Unlabeled data

In edge caching scenarios, one of the significant challenges for supervised deep learning is the scarcity of labeled data. In edge caching environments, obtaining labeled data can be challenging due to several factors such as privacy concerns, limited communication bandwidth, and distributed nature of edge devices. Furthermore ,the data generated within the network may lack labels or contain mislabeled instances. Without sufficient labeled data, training deep learning models for caching decisions becomes difficult, as the models may struggle to generalize effectively to unseen data. Addressing the issue of unlabeled

data in edge caching scenarios requires innovative solutions such as semi-supervised learning techniques, transfer learning from related tasks or domains and active learning strategies to intelligently select informative data for labeling [13].

5.1.3 Dropped participants

D2D caching and federated learning strategies assume that participants' wireless connections remain constantly available. However, in reality, mobile devices involved, may intermittently lose connectivity, or drop out of the system due to network issues or energy limitations. The significant loss of participating devices during training can notably impair the FL system's performance, affecting metrics like accuracy and convergence rate. It’s imperative for new FL algorithms to be resilient to such dropouts and to anticipate scenarios where only a few participants remain connected for a training round. One potential remedy is for the FL model owner to offer dedicated or special connections, such as cellular networks, at no cost to participants, serving as an incentive to prevent dropouts [13].

Additionally, a large amount of dropped participants in D2D caching impacts the network in several ways like, the overall cache resources decreasing, redundancy diminishing, the cache hit rate may fluctuate, network fragmentation may occur and a rebalance of content may be needed. To mitigate the effects of dropped participants on D2D caching, network protocols and algorithms can be designed to dynamically adjust caching strategies, redistribute content among remaining participants, and optimize cache utilization based on network conditions and participant behavior [91].

5.1.4 Privacy concerns

The integration of deep learning models at the edge raises concerns about data privacy and security. Deep learning models trained on sensitive user data may inadvertently leak private information if not properly protected. Moreover, transmitting raw data from edge devices to centralized servers for model training raises privacy risks, as it exposes potentially sensitive information to interception or unauthorized access during transmission. Additionally, the deployment of complex deep learning models on resource-

constrained edge devices poses challenges in implementing robust privacy-preserving techniques, such as encryption and differential privacy, without compromising model performance and efficiency.

Privacy concerns in FL are addressed by enabling local model training, where only model parameters are exchanged with the FL server, thus ensuring the privacy of each participant. However, disclosing model updates during training could still leak sensitive information to adversaries or third parties. To address this, current strategies suggest security measures such as differential privacy and collaborative training methods. Nonetheless, these approaches lead to decreased model accuracy and demand significant computational resources on mobile devices [13].

5.1.5 Cost and monetization models

The economic aspects of caching in edge computing are often overlooked. Deploying deep learning models at the edge incurs costs related to hardware, software development, and maintenance, particularly in resource-constrained environments. Edge devices often have limited computational resources, necessitating the optimization of deep learning models for efficient inference and reduced energy consumption. Additionally, caching in user equipment and D2D communication raises concerns regarding the storage, power consumption, and data usage of mobile devices. Since not all UEs cache and share content equally, it creates an unequal communication environment. Hence, incentivizing models are crucial for UEs that engage in more caching and transmission activities compared to others. For instance, a basic incentive mechanism could involve offering higher download data rates and lower delays for UEs with a higher history of sharing/transmission. However, most D2D caching techniques lack such incentive mechanisms, necessitating the establishment of standards for practical UE cache deployments. Each processing entity should be rewarded based on their contributions to the learning process, as all entities benefit from optimal caching decisions [17].

5.2 Research Directions

5.2.1 Software defined networking

Software-Defined Networks (SDNs) are a modern approach to network architecture that separates the control plane from the data plane, allowing network administrators to centrally manage and orchestrate network resources through software-based controllers. This decoupling enables the centralization and programmable management of the control plane, while the data plane remains distributed across network devices. Utilizing the computational capabilities of nodes and the connections within a software-defined network (SDN) enables the creation of a decentralized and adaptable deep learning network [7], [80] .

5.2.2 Transfer Learning

Transfer learning is a machine learning technique where knowledge gained from solving one problem is applied to a different but related problem. Transfer learning allows pre-trained models, which have already been trained on a large dataset for a related task, to be reused or adapted for a new task with potentially less labeled data or in a different domain. These pre-trained models can serve as a starting point for training edge-specific caching models, enabling faster convergence and improved generalization performance with limited labeled data. Transfer learning offers notable resource efficiency, particularly when there is a strong relationship between the source and target problems [9], [38].

5.2.3 Deep learning for cache dimensioning

It's notable that very few studies utilize DL in making cache dimensioning decisions. This scarcity is primarily due to the limited availability of training datasets compared to content popularity prediction. Additionally, the allocation of cache sizes significantly impacts network performance and financial investments. Recently, network slicing has emerged as a crucial tool in enabling 5G networks to deliver multiple services with varied characteristics. These slices are formed on physical infrastructure, including network

storage, making it an intriguing area of study to determine memory space allocation to support content caching and other storage services [18].

5.2.4 Standard data set

A conventional dataset capturing user preferences and content popularity at the edge is necessary for comparative analysis in research and establishing baseline results. The MovieLens dataset, commonly used for evaluating recommendation systems, lacks edge network representation as it primarily features data relevant to cloud/CDN-based user access. Although various learning techniques have been employed with the MovieLens dataset to predict content popularity, it cannot be directly mapped to edge networks. Similarly, standard datasets for user mobility hinder baseline research efforts. Additionally, there is a need to discern and emphasize disparities between edge popularity and cloud popularity of content, as overlooking these differences can significantly impact cache population and cache hit rate [17].

Chapter 6 Conclusion

Utilizing deep learning for edge caching proves to be an efficient solution for managing the rapidly increasing network traffic. This work conducted a thorough investigation into various aspects of edge caching using deep learning techniques. It began with an overview of edge networks and caching, followed by an exploration of different caching strategies based on various caching positions and performance criteria. Subsequently, it presents an overview of ML algorithms, including supervised, unsupervised, reinforcement, and federated learning, along with showcasing representative deep learning models suitable for edge computing. Furthermore, this paper presents a summary of recent research papers, focusing on parameters such as, employed deep learning methods and the objectives behind their application. Finally, the implementation challenges and research directions of employing deep learning for caching are presented, including considerations such as the cost of deep learning, cost and monetization models , and integration with software-defined networks. This study aims to provide a comprehensive understanding of edge caching and its integration with deep learning methodologies.

Bibliography

[1] Rohit Shewale, “Internet User Statistics In 2024 — (Global Demographics),” https://www.demandsage.com/internet-user-statistics/.

[2] “Digital 2023 October Global Statshot Report — DataReportal – Global Digital Insights.” Accessed: Dec. 17, 2023. [Online]. Available: https://datareportal.com/reports/digital-2023-october-global-statshot

[3] “Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper - Cisco.” Accessed: Dec. 17, 2023. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[4] “Sandvine’s 2023 Global Internet Phenomena Report Shows 24% Jump in Video Traffic, with Netflix Volume Overtaking YouTube.” Accessed: Dec. 17, 2023. [Online]. Available: https://www.sandvine.com/press-releases/sandvines-2023-global-internet-phenomena-report-shows-24-jump-in-video-traffic-with-netflix-volume-overtaking-youtube

[5] “Global mobile video traffic 2022 | Statista.” Accessed: Dec. 17, 2023. [Online]. Available: https://www.statista.com/statistics/252853/global-mobile-video-traffic-forecast/

[6] “Monthly mobile data usage 2022 and 2028* | Statista.” Accessed: Dec. 17, 2023. [Online]. Available: https://www.statista.com/statistics/1100854/global-mobile-data-usage-2024/

[7] J. Yao, T. Han, and N. Ansari, “On Mobile Edge Caching,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2525–2553, 2019, doi: 10.1109/COMST.2019.2908280.

[8] M. Reiss-Mirzaei, M. Ghobaei-Arani, and L. Esmaeili, “A review on the edge caching mechanisms in the mobile edge computing: A social-aware perspective,” *Internet of Things*, vol. 22, p. 100690, Jul. 2023, doi: 10.1016/j.iot.2023.100690.

[9] N. Nomikos, S. Zoupanos, T. Charalambous, and I. Krikidis, “A Survey on Reinforcement Learning-Aided Caching in Heterogeneous Mobile Edge Networks,” *IEEE Access*, vol. 10, pp. 4380–4413, 2022, doi: 10.1109/ACCESS.2022.3140719.

[10] M. Sheraz *et al.*, “Artificial Intelligence for Wireless Caching: Schemes, Performance, and Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 631–661, 2021, doi: 10.1109/COMST.2020.3008362.

[11] M. McClellan, C. Cervelló-Pastor, and S. Sallent, “Deep Learning at the Mobile Edge: Opportunities for 5G Networks,” *Applied Sciences*, vol. 10, no. 14, p. 4735, Jul. 2020, doi: 10.3390/app10144735.

[12] F. Wang, M. Zhang, X. Wang, X. Ma, and J. Liu, “Deep Learning for Edge Computing Applications: A State-of-the-Art Survey,” *IEEE Access*, vol. 8, pp. 58322–58336, 2020, doi: 10.1109/ACCESS.2020.2982411.

[13] W. Y. B. Lim *et al.*, “Federated Learning in Mobile Edge Networks: A Comprehensive Survey,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020, doi: 10.1109/COMST.2020.2986024.

[14] S. ANOKYE, M. SEID, and S. Guolin, “A Survey on Machine Learning Based Proactive Caching,” *ZTE Communications*, vol. 17, no. 4, p. 46, Dec. 2019, doi: 10.12142/ZTECOM.201904007.

[15] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, “Convergence of Edge Computing and Deep Learning: A Comprehensive Survey,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020, doi: 10.1109/COMST.2020.2970550.

[16] M. Pooyandeh and I. Sohn, “Edge Network Optimization Based on AI Techniques: A Survey,” *Electronics (Basel)*, vol. 10, no. 22, p. 2830, Nov. 2021, doi: 10.3390/electronics10222830.

[17] J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi, “Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey,” *Journal of Network and Computer Applications*, vol. 181, p. 103005, May 2021, doi: 10.1016/j.jnca.2021.103005.

[18] Y. Wang and V. Friderikos, “A Survey of Deep Learning for Data Caching in Edge Network,” *Informatics*, vol. 7, no. 4, p. 43, Oct. 2020, doi: 10.3390/informatics7040043.

[19] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong, “The Characteristics of Cloud Computing,” in *2010 39th International Conference on Parallel Processing Workshops*, IEEE, Sep. 2010, pp. 275–279. doi: 10.1109/ICPPW.2010.45.

[20] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. YANG, and W. Wang, “A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications,” *IEEE Access*, vol. 5, pp. 6757–6779, 2017, doi: 10.1109/ACCESS.2017.2685434.

[21] K. Cao, Y. Liu, G. Meng, and Q. Sun, “An Overview on Edge Computing Research,” *IEEE Access*, vol. 8, pp. 85714–85728, 2020, doi: 10.1109/ACCESS.2020.2991734.

[22] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, “A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms,” *ACM Comput Surv*, vol. 52, no. 6, pp. 1–36, Nov. 2020, doi: 10.1145/3362031.

[23] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies, “The Case for VM-based Cloudlets in Mobile Computing,” *IEEE Pervasive Comput*, 2011, doi: 10.1109/MPRV.2009.64.

[24] V. Bahl, “emergence of micro datacenter (cloudlets/edges) for mobile computing,” 2015.

[25] M. Aazam and E.-N. Huh, “Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT,” in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, IEEE, Mar. 2015, pp. 687–694. doi: 10.1109/AINA.2015.254.

[26] C. C. Byers, “Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks,” *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, Aug. 2017, doi: 10.1109/MCOM.2017.1600885.

[27] M. Patel *et al.*, “Mobile-Edge Computing Contributing Organizations and Authors”. 2014. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf

[28] H. Wu, Y. Fan, Y. Wang, H. Ma, and L. Xing, “A Comprehensive Review on Edge Caching from the Perspective of Total Process: Placement, Policy and Delivery,” *Sensors*, vol. 21, no. 15, p. 5033, Jul. 2021, doi: 10.3390/s21155033.

[29] B. Agarwal, M. A. Togou, M. Ruffini, and G.-M. Muntean, “A Comprehensive Survey on Radio Resource Management in 5G HetNets: Current Solutions, Future Trends and Open Issues,” *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2495–2534, 2022, doi: 10.1109/COMST.2022.3207967.

[30] B. Ji *et al.*, “A Vision of IoV in 5G HetNets: Architecture, Key Technologies, Applications, Challenges, and Trends,” *IEEE Netw*, vol. 36, no. 2, pp. 153–161, Mar. 2022, doi: 10.1109/MNET.012.2000527.

[31] M. Peng, Y. Sun, X. Li, Z. Mao, and C. Wang, “Recent Advances in Cloud Radio Access Networks: System Architectures, Key Techniques, and Open Issues,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2282–2308, 2016, doi: 10.1109/COMST.2016.2548658.

[32] W. Wang, R. Lan, J. Gu, A. Huang, H. Shan, and Z. Zhang, “Edge Caching at Base Stations With Device-to-Device Offloading,” *IEEE Access*, vol. 5, pp. 6399–6410, 2017, doi: 10.1109/ACCESS.2017.2679198.

[33] D. Prerna, R. Tekchandani, and N. Kumar, “Device-to-device content caching techniques in 5G: A taxonomy, solutions, and challenges,” *Comput Commun*, vol. 153, pp. 48–84, Mar. 2020, doi: 10.1016/j.comcom.2020.01.057.

[34] T. X. Vu, S. Chatzinotas, and B. Ottersten, “Edge-Caching Wireless Networks: Performance Analysis and Optimization,” *IEEE Trans Wirel Commun*, vol. 17, no. 4, pp. 2827–2839, Apr. 2018, doi: 10.1109/TWC.2018.2803816.

[35] Z. Zhang and M. Tao, “Deep Learning for Wireless Coded Caching With Unknown and Time-Variant Content Popularity,” *IEEE Trans Wirel Commun*, vol. 20, no. 2, pp. 1152–1163, Feb. 2021, doi: 10.1109/TWC.2020.3030973.

[36] B. Tang and L. Kang, “EICache: A learning-based intelligent caching strategy in mobile edge computing,” *Peer Peer Netw Appl*, vol. 15, no. 2, pp. 934–949, Mar. 2022, doi: 10.1007/s12083-021-01266-4.

[37] M. K. Somesula, R. R. Rout, and D. V. L. N. Somayajulu, “Deadline-aware caching using echo state network integrated fuzzy logic for mobile edge networks,” *Wireless Networks*, vol. 27, no. 4, pp. 2409–2429, May 2021, doi: 10.1007/s11276-021-02578-2.

[38] F. Gabry, V. Bioglio, and I. Land, “On Energy-Efficient Edge Caching in Heterogeneous Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3288–3298, Dec. 2016, doi: 10.1109/JSAC.2016.2611845.

[39] T. T. Vu, D. T. Ngo, M. N. Dao, S. Durrani, and R. H. Middleton, “Spectral and Energy Efficiency Maximization for Content-Centric C-RANs With Edge Caching,” *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6628–6642, Dec. 2018, doi: 10.1109/TCOMM.2018.2866458.

[40] J. Shaikh, M. Fiedler, and D. Collange, “Quality of Experience from user and network perspectives,” *annals of telecommunications - annales des télécommunications*, vol. 65, no. 1–2, pp. 47–57, Feb. 2010, doi: 10.1007/s12243-009-0142-x.

[41] K. Qi, S. Han, and C. Yang, “Learning a Hybrid Proactive and Reactive Caching Policy in Wireless Edge Under Dynamic Popularity,” *IEEE Access*, vol. 7, pp. 120788–120801, 2019, doi: 10.1109/ACCESS.2019.2936866.

[42] Y. Jiang, H. Feng, F.-C. Zheng, D. Niyato, and X. You, “Deep Learning-Based Edge Caching in Fog Radio Access Networks,” *IEEE Trans Wirel Commun*, vol. 19, no. 12, pp. 8442–8454, Dec. 2020, doi: 10.1109/TWC.2020.3022907.

[43] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, “Cooperative Content Caching in 5G Networks with Mobile Edge Computing,” *IEEE Wirel Commun*, vol. 25, no. 3, pp. 80–87, Jun. 2018, doi: 10.1109/MWC.2018.1700303.

[44] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, “Federated Deep Reinforcement Learning for Internet of Things With Decentralized Cooperative

Edge Caching," *IEEE Internet Things J*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020, doi: 10.1109/JIOT.2020.2986803.

[45] Hao Che, Ye Tung, and Zhijun Wang, "Hierarchical Web caching systems: modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, Sep. 2002, doi: 10.1109/JSAC.2002.801752.

[46] E. Parrinello, A. Unsal, and P. Elia, "Fundamental Limits of Coded Caching With Multiple Antennas, Shared Caches and Uncoded Prefetching," *IEEE Trans Inf Theory*, vol. 66, no. 4, pp. 2252–2268, Apr. 2020, doi: 10.1109/TIT.2019.2955384.

[47] K. Zhang and C. Tian, "Fundamental Limits of Coded Caching: From Uncoded Prefetching to Coded Prefetching," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1153–1164, Jun. 2018, doi: 10.1109/JSAC.2018.2844958.

[48] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and D. I. Kim, "Distributed Deep Learning at the Edge: A Novel Proactive and Cooperative Caching Framework for Mobile Edge Networks," *IEEE Wireless Communications Letters*, vol. 8, no. 4, pp. 1220–1223, Aug. 2019, doi: 10.1109/LWC.2019.2912365.

[49] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning," *IEEE Netw*, vol. 33, no. 5, pp. 156–165, Sep. 2019, doi: 10.1109/MNET.2019.1800286.

[50] N. I. Osman, T. El-Gorashi, and J. M. H. Elmirghani, "The impact of content popularity distribution on energy efficient caching," in *2013 15th International Conference on Transparent Optical Networks (ICTON)*, IEEE, Jun. 2013, pp. 1–6. doi: 10.1109/ICTON.2013.6602957.

[51] M. Garetto, E. Leonardi, and S. Traverso, "Efficient analysis of caching strategies under dynamic content popularity," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, IEEE, Apr. 2015, pp. 2263–2271. doi: 10.1109/INFOCOM.2015.7218613.

[52] F. M. Harper and J. A. Konstan, “The MovieLens Datasets,” *ACM Trans Interact Intell Syst*, vol. 5, no. 4, pp. 1–19, Jan. 2016, doi: 10.1145/2827872.

[53] J. Ni, K. Zhang, and A. V. Vasilakos, “Security and Privacy for Mobile Edge Caching: Challenges and Solutions,” *IEEE Wirel Commun*, vol. 28, no. 3, pp. 77–83, Jun. 2021, doi: 10.1109/MWC.001.2000329.

[54] J. Zhang *et al.*, “Joint Resource Allocation for Latency-Sensitive Services Over Mobile Edge Computing Networks With Caching,” *IEEE Internet Things J*, vol. 6, no. 3, pp. 4283–4294, Jun. 2019, doi: 10.1109/JIOT.2018.2875917.

[55] V. Meena, K. Krithivasan, P. Rahul, and T. S. Praba, “Toward an Intelligent Cache Management: In an Edge Computing Era for Delay Sensitive IoT Applications,” *Wirel Pers Commun*, vol. 131, no. 2, pp. 1075–1088, Jul. 2023, doi: 10.1007/s11277-023-10469-2.

[56] D. Liu, B. Chen, C. Yang, and A. F. Molisch, “Caching at the wireless edge: design aspects, challenges, and future directions,” *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, Sep. 2016, doi: 10.1109/MCOM.2016.7565183.

[57] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, “Artificial Neural Networks-Based Machine Learning for Wireless Networks: A Tutorial,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3039–3071, 2019, doi: 10.1109/COMST.2019.2926625.

[58] L. Ale, N. Zhang, H. Wu, D. Chen, and T. Han, “Online Proactive Caching in Mobile Edge Computing Using Bidirectional Deep Recurrent Neural Network,” *IEEE Internet Things J*, vol. 6, no. 3, pp. 5520–5530, Jun. 2019, doi: 10.1109/JIOT.2019.2903245.

[59] W. H. Lopez Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, “Autoencoders,” in *Machine Learning*, Elsevier, 2020, pp. 193–208. doi: 10.1016/B978-0-12-815739-8.00011-0.

[60] S. Rathore, J. H. Ryu, P. K. Sharma, and J. H. Park, “DeepCachNet: A Proactive Caching Framework Based on Deep Learning in Cellular Networks,” *IEEE Netw*, vol. 33, no. 3, pp. 130–138, May 2019, doi: 10.1109/MNET.2019.1800058.

[61] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep Reinforcement Learning: A Brief Survey,” *IEEE Signal Process Mag*, vol. 34, no. 6, pp. 26–38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.

[62] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, “Collaborative Edge Computing and Caching With Deep Reinforcement Learning Decision Agents,” *IEEE Access*, vol. 8, pp. 120604–120612, 2020, doi: 10.1109/ACCESS.2020.3007002.

[63] W. Li, J. Wang, G. Zhang, L. Li, Z. Dang, and S. Li, “A Reinforcement Learning Based Smart Cache Strategy for Cache-Aided Ultra-Dense Network,” *IEEE Access*, vol. 7, pp. 39390–39401, 2019, doi: 10.1109/ACCESS.2019.2905589.

[64] C. Zhong, M. C. Gursoy, and S. Velipasalar, “A deep reinforcement learning-based framework for content caching,” in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, IEEE, Mar. 2018, pp. 1–6. doi: 10.1109/CISS.2018.8362276.

[65] Y. Zhang *et al.*, “Cooperative Edge Caching: A Multi-Agent Deep Learning Based Approach,” *IEEE Access*, vol. 8, pp. 133212–133224, 2020, doi: 10.1109/ACCESS.2020.3010329.

[66] M. Lei, Q. Li, R. Wu, A. Pandharipande, and X. Ge, “Deep Deterministic Policy Gradient-Based Edge Caching: An Inherent Performance Tradeoff,” in *2021 IEEE Global Communications Conference (GLOBECOM)*, IEEE, Dec. 2021, pp. 1–7. doi: 10.1109/GLOBECOM46510.2021.9685454.

[67] Y. Wei, F. R. Yu, M. Song, and Z. Han, “Joint Optimization of Caching, Computing, and Radio Resources for Fog-Enabled IoT Using Natural Actor–Critic Deep Reinforcement Learning,” *IEEE Internet Things J*, vol. 6, no. 2, pp. 2061–2073, Apr. 2019, doi: 10.1109/JIOT.2018.2878435.

[68] C. Zhong, M. C. Gursoy, and S. Velipasalar, “Deep Reinforcement Learning-Based Edge Caching in Wireless Networks,” *IEEE Trans Cogn Commun Netw*, vol. 6, no. 1, pp. 48–61, Mar. 2020, doi: 10.1109/TCCN.2020.2968326.

[69] C. Zhong, M. C. Gursoy, and S. Velipasalar, “A deep reinforcement learning-based framework for content caching,” in *2018 52nd Annual Conference on Information*

Sciences and Systems (CISS), IEEE, Mar. 2018, pp. 1–6. doi: 10.1109/CISS.2018.8362276.

[70] Z. Chen, N. Lv, P. Liu, Y. Fang, K. Chen, and W. Pan, "Intrusion Detection for Wireless Edge Networks Based on Federated Learning," *IEEE Access*, vol. 8, pp. 217463–217472, 2020, doi: 10.1109/ACCESS.2020.3041793.

[71] K. Thar, N. H. Tran, T. Z. Oo, and C. S. Hong, "DeepMEC: Mobile Edge Caching Using Deep Learning," *IEEE Access*, vol. 6, pp. 78260–78275, 2018, doi: 10.1109/ACCESS.2018.2884913.

[72] K. Thar, T. Z. Oo, Y. K. Tun, D. H. Kim, K. T. Kim, and C. S. Hong, "A Deep Learning Model Generation Framework for Virtualized Multi-Access Edge Cache Management," *IEEE Access*, vol. 7, pp. 62734–62749, 2019, doi: 10.1109/ACCESS.2019.2916080.

[73] L. Li *et al.*, "Deep Reinforcement Learning Approaches for Content Caching in Cache-Enabled D2D Networks," *IEEE Internet Things J*, vol. 7, no. 1, pp. 544–557, Jan. 2020, doi: 10.1109/JIOT.2019.2951509.

[74] S. M. Maher, G. A. Ebrahim, S. Hosny, and M. M. Salah, "A Cache-Enabled Device-to-Device Approach Based on Deep Learning," *IEEE Access*, vol. 11, pp. 76953–76963, 2023, doi: 10.1109/ACCESS.2023.3297280.

[75] Y. Zhang, Y. Li, R. Wang, J. Lu, X. Ma, and M. Qiu, "PSAC: Proactive Sequence-Aware Content Caching via Deep Learning at the Network Edge," *IEEE Trans Netw Sci Eng*, vol. 7, no. 4, pp. 2145–2154, Oct. 2020, doi: 10.1109/TNSE.2020.2990963.

[76] C. Zhang *et al.*, "Toward Edge-Assisted Video Content Intelligent Caching With Long Short-Term Memory Learning," *IEEE Access*, vol. 7, pp. 152832–152846, 2019, doi: 10.1109/ACCESS.2019.2947067.

[77] B. Liu *et al.*, "A deep learning-based edge caching optimization method for cost-driven planning process over IIoT," *J Parallel Distrib Comput*, vol. 168, pp. 80–89, Oct. 2022, doi: 10.1016/j.jpdc.2022.06.007.

[78] Q. Fan, X. Li, J. Li, Q. He, K. Wang, and J. Wen, "PA-Cache: Evolving Learning-Based Popularity- Aware Content Caching in Edge Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1746–1757, Jun. 2021, doi: 10.1109/TNSM.2021.3053645.

[79] Y.-T. Lin, C.-C. Yen, and J.-S. Wang, "Video Popularity Prediction: An Autoencoder Approach With Clustering," *IEEE Access*, vol. 8, pp. 129285–129299, 2020, doi: 10.1109/ACCESS.2020.3009253.

[80] W.-X. Liu, J. Zhang, Z.-W. Liang, L.-X. Peng, and J. Cai, "Content Popularity Prediction and Caching for ICN: A Deep Learning Approach With SDN," *IEEE Access*, vol. 6, pp. 5075–5089, 2018, doi: 10.1109/ACCESS.2017.2781716.

[81] F. Jiang, Z. Yuan, C. Sun, and J. Wang, "Deep Q-Learning-Based Content Caching With Update Strategy for Fog Radio Access Networks," *IEEE Access*, vol. 7, pp. 97505–97514, 2019, doi: 10.1109/ACCESS.2019.2927836.

[82] H. Zhou, T. Wu, H. Zhang, and J. Wu, "Incentive-Driven Deep Reinforcement Learning for Content Caching and D2D Offloading," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2445–2460, Aug. 2021, doi: 10.1109/JSAC.2021.3087232.

[83] K. Guo and C. Yang, "Temporal-Spatial Recommendation for Caching at Base Stations via Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 58519–58532, 2019, doi: 10.1109/ACCESS.2019.2914500.

[84] X. He, K. Wang, and W. Xu, "QoE-Driven Content-Centric Caching With Deep Reinforcement Learning in Edge-Enabled IoT," *IEEE Comput Intell Mag*, vol. 14, no. 4, pp. 12–20, Nov. 2019, doi: 10.1109/MCI.2019.2937608.

[85] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, "Reinforcement Learning-Based Optimal Computing and Caching in Mobile Edge Network," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2343–2355, Oct. 2020, doi: 10.1109/JSAC.2020.3000396.

[86] D. Li *et al.*, "Deep Reinforcement Learning for Cooperative Edge Caching in Future Mobile Networks," in *2019 IEEE Wireless Communications and Networking*

Conference (WCNC), IEEE, Apr. 2019, pp. 1–6. doi: 10.1109/WCNC.2019.8885516.

[87] F. Majidi, M. R. Khayyambashi, and B. Barekatain, "HFDRL: An Intelligent Dynamic Cooperate Cashing Method Based on Hierarchical Federated Deep Reinforcement Learning in Edge-Enabled IoT," *IEEE Internet Things J*, vol. 9, no. 2, pp. 1402–1413, Jan. 2022, doi: 10.1109/JIOT.2021.3086623.

[88] X. Wang, R. Li, C. Wang, X. Li, T. Taleb, and V. C. M. Leung, "Attention-Weighted Federated Deep Reinforcement Learning for Device-to-Device Assisted Heterogeneous Collaborative Edge Caching," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 154–169, Jan. 2021, doi: 10.1109/JSAC.2020.3036946.

[89] X. Huang, Z. Chen, Q. Chen, and J. Zhang, "Federated learning based QoS-aware caching decisions in fog-enabled internet of things networks," *Digital Communications and Networks*, vol. 9, no. 2, pp. 580–589, Apr. 2023, doi: 10.1016/j.dcan.2022.04.022.

[90] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A Joint Learning and Communications Framework for Federated Learning Over Wireless Networks," *IEEE Trans Wirel Commun*, vol. 20, no. 1, pp. 269–283, Jan. 2021, doi: 10.1109/TWC.2020.3024629.

[91] A. S. Ali, K. R. Mahmoud, and K. M. Naguib, "Optimal caching policy for wireless content delivery in D2D networks," *Journal of Network and Computer Applications*, vol. 150, p. 102467, Jan. 2020, doi: 10.1016/j.jnca.2019.102467.