Machine Learning for Next-generation Content Delivery Networks: Deployment, Content Placement, and Performance Management

Sepideh Malektaji

A thesis

In

The Concordia Institute

For

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Information and Systems Engineering) at

Concordia University

Montreal, Quebec, Canada

May 2022

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Sepideh Malektaji

Entitled: Machine Learning for Next-generation Content Delivery Networks: Deployment, Content Placement, and Performance Management

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and System Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

	Chair
Dr. John Xiuou Zhang	External Examiner
Dr. Hanan Lutfiyya	External to Program
Dr. Anjali Agrawal	Examiner
Dr. Chadi Assi	Examiner
Dr. Jamal Bentaher	Thesis Supervisor (s)
Dr. Roch Glitho	

Approved by

Dr. Mohammad Mannan — Graduate Program Director

May 31, 2022

Date of Defence

Dr. Mourad Debbabi — Dean, Gina Cody School of Engineering and Computer Science

ABSTRACT

Machine Learning for Next-generation Content Delivery Networks: Deployment, Content Placement, and Performance Management

Sepideh Malektaji, Ph.D.

Concordia University, 2022

With the explosive demands for data and the growth in mobile users, content delivery networks (CDNs) are facing ever-increasing challenges to meet end-users quality-of-experience requirements, ensure scalability and remain cost-effective. These challenges encourage CDN providers to seek a solution by considering the new technologies available in today's computer network domain. Network Function Virtualization (NFV) is a relatively new network service deployment technology used in computer networks. It can reduce capital and operational costs while yielding flexibility and scalability for network operators. Thanks to the NFV, the network functions that previously could be offered only by specific hardware appliances can now run as Virtualized Network Functions (VNF) on commodity servers or switches. Moreover, a network service can be flexibly deployed by a chain of VNFs, a structure known as the VNF Forwarding Graph or VNF-FG. Considering these advantages, the next-generation CDN will be deployed using NFV infrastructure. However, using NFV for service deployment is challenging as resource allocation in a shared infrastructure is not easy. Moreover, the integration of other paradigms (e.g., edge computing and vehicular network) into CDN will compound the complexity of content placement and performance management for the next-generation CDNs. In this regard, due to their impacts on final service and end-user perceived quality, the challenges in service deployment, content placement, and performance management should be addressed carefully. In this thesis, advanced machine learning methods are utilized to provide algorithmic solutions for the abovementioned challenges of the next generation CDNs.

Regarding the challenges in the deployment of the next-generation CDNs, we propose two deep reinforcement learning-based methods addressing the joint problems of VNF-FG's composition and embedding, as well as function scaling and topology adaptation. As for content placement challenges, a deep reinforcement learning-based approach for content migration in an edge-based CDN with vehicular nodes is proposed. The proposed approach takes advantage of the available caching resources in the proximity of the full local caches and efficiently migrates contents at the edge of the network. Moreover, for managing the performance quality of an operating CDN, an unsupervised machine learning anomaly detection method is provided. The proposed method uses clustering to enable easier performance analysis for next-generation CDNs. Each proposed method in this thesis is evaluated by comparison to the state-of-the-art approaches. Moreover, when applicable, the optimality gaps of the proposed methods are investigated as well.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Dr. Roch Glitho for his continuous support throughout my Ph.D., for his patience, motivation, and immersed kindness. Thanks for always believing in me and encouraging me even in my weakest moments.

I am also thankful to my thesis committee: Dr. Chadi Assi, Dr. Jamal Bentaher, Dr. Anjali Agrawal, and Dr. Hanan Lutfoyya, for their insightful and constructive comments and also for the questions which incented me to widen my research from various perspectives.

Furthermore, I would like to thank all my colleagues in TLSE Lab at Concordia University, especially my friends Marsa Rayani and Vahid Malaki. Also, Special thanks to Diala Naboulsi, Amin Ebrahimzadeh, and Dr. Halima Elbiaze for their help and advice.

I would like to thank my father, MohammadTaghi Malektaji, and my mother, Rakhshandeh Momeni, who helped me throughout my life and provided me with moral and emotional support. Words cannot express how grateful I am to my family for all of the sacrifices that they have made on my behalf.

Last but not least, I would like to express my very special thanks to my brother, Siavash Malektaji, the strongest and kindest person I know. Thank you brother; you have made such a huge impact on my life. I would not be the person I am today without you. You were and will always be my hero and role model in life.

I dedicate this Ph.D. thesis to my brother, his lovely wife, Sogol Asghari, and my beautiful yet-to-be-born niece, as they are the light of my life.

Contents

List of Figures

List of Tables

List of Acronyms

ANN	Artificial Neural Network
BDQN	Branching Dueling Q Network
CDDQN	Conventional Double Deep Q Network
CDN	Content Delivery Network
DBR	Download Bit Rate
DDJCE	Deep Dynamic Joint VNF-FG Composition and Embedding
DDQN	Double Deep Q Network
DNN	Deep Neural Network
DQL	Deep Q Learning
DQN-AF	Deep Q network with Action Filtering
DRL	Deep Reinforcement Learning
DRLCM	Deep Reinforcment Learning Content Migration
GBFSO	Greedy-BestFit Function Scaling-Only
HD	High Defenition
IaaS	Infrastructure as a Service
ILP	Integer Linear Programming
JFSTA	Joint VNF-FG Function Scaling and Topology Adaptation Algorithm
KPI	Key Performance Indicator
LRU	Least Recently Used
LSTM	Long Short Term Memory
MDP	Markov Decision Process
ML	Machine Learning
NFV	Network Function Virtualization
NRUA	Network Resource Utilization Analyzer
PBDQN	Plain Branching Dualing Q-network
QL	Quality Level
QoE	Quality of Experience

QoS	Quality of Service
RA	Resource Allocation
RFTAO	Random-FirstFit Topology Adaptation-Only
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SDA	Service Demand Analyzer
SRLCM	Simplified Reinforcment Learning Content Migration
SUMO	Simulation for Urban MObility
SVM	Support Vector Machine
TRLCM	Traditional Reinforcment Learning Content Migratio
VM	Virtual Machine
VNF-FG	Virtual Network Function Forwarding Graph
VNFR	Virtual Network Function Request
VoD	Video on Demand

Chapter 1

1. Introduction

1.1 Overview

Content Delivery Networks (CDNs) are a group of network elements organized for the efficient delivery of content to end-users over a large scale [1]. In this regard, CDNs rely on a set of strategically placed surrogate servers (a.k.a replica servers) to replicate content from an origin server (a.k.a content server) so that they can offer content delivery services with reduced latency. Today, CDNs play a pivotal role in the delivery of content to end-users across the internet. The demand for CDNs is predicted to increase exponentially, and its market is predicted to rise from $11.76 billion in 2019 to $49.61 billion in 2025 [2]. To that end, CDNs need to cope with the exponential data demand growth, straining their infrastructure. Moreover, they need to satisfy end-users quality-of-experience (QoE) requirements, ensure scalability and remain cost-effective. These challenges encourage CDN providers to seek solutions by considering the new technologies arising in the modern computer network domain.

Network Function Virtualization (NFV) is a network service deployment technology that has recently emerged. Thanks to the NFV, the network functions that previously could be offered only

by specific hardware appliances can now run as Virtualized Network Functions (VNF) on commodity servers or switches. In that regard, by virtualizing the network resources, NFV technology provides opportunities for network optimization, cost reduction, and scalability [3].

1.2 Challenges and Thesis Contributions

1.2.1 Challenges

Considering the advantages, the next-generation CDN will be deployed on shared infrastructure using NFV technology. However, the use of NFV for CDN deployment is not easy. To that end, not only new deployment challenges should be addressed but also with the integration of new technologies, one can expect new challenges in the operation and management of CDNs. In the following, some examples of these issues faced by modern CDNs are discussed:

- **VNF-FG resource allocation for CDN deployment:** Resource allocation in a shared and complex infrastructure is not easy. In the NFV ecosystem, a network service (e.g., a content delivery service) is a set of chained VNFs (called VNF forwarding graph or VNF-FG) through which the service traffic should traverse one by one. To this end, the following Resource Allocation (RA) problems [4] need to be addressed: 1) VNF-FG Composition: Determining the number and order of VNFs in VNF-FG, 2) VNF-FG embedding: Allocating physical resources to the VNFs and their connecting links, and 3) VNF-FG scheduling: efficient scheduling of the shared resources among embedded VNFs. Each of these RA challenges contributes to the costs and service quality of the final network service and thus should be addressed carefully.
- **VNF-FG adaptation for CDN deployment:** An already deployed and running VNF-FG that enables a content delivery service could become inefficient as the service demands increase and network conditions fluctuate [3]. To that end, not only the number of VNF instances needs to be adjusted (a process known as function scaling or horizontal scaling), but also the topology of the VNF-FG and its mapping to the physical network may need proper adjustments via so-called topology adaptation.
- **Content placement for CDN operation:** The next-generation CDN can be integrated with other paradigms (e.g., edge computing and vehicular network). The operational decisions in this heterogeneous and complex environment will be challenging. For example, for an edge-based CDN that consists of fixed and mobile caches, decisions on placement, migration, and removal of

contents are challenging. Since these decisions could have vital impacts on the content delivery quality, they should be carefully investigated [5].

- **Performance monitoring for CDN management:** Monitoring users' experience in the content delivery process is of paramount importance for CDNs. Throughout their operations, CDN providers target the satisfaction of users' expectations in terms of Quality of Experience (QoE). In this context, CDN providers need to acquire knowledge of users' QoE, identify QoE degradations, and investigate their potential root causes. However, due to the complexity of the CDN ecosystem, tracking the QoE evolution and detecting the degradation is quite challenging.

1.2.2 Thesis Contributions

Unfortunately, the challenges of the next-generation CDNs presented in Section 1.2.1 are not yet fully addressed. This Ph.D. thesis proposes the use of advanced Machine Learning (ML) algorithms to tackle these challenges in three categories of deployment, content placement, and performance management. To that end, it makes four main contributions. The two first contributions address the service deployment, whereas the third and fourth contributions target CDN content placement and performance management, respectively. These contributions are presented as follows

1.2.2.1 Dynamic Joint VNF Forwarding Graph Composition and Embedding: A Deep Reinforcement Learning Framework [6]

The first contribution is an ML-based deployment algorithm for joint VNF-FG composition and embedding. In the literature (e.g. [7-11]), the composition and embedding stages of VNF-FGs are usually targeted separately, which may result in undesired solutions. In this contribution, we propose a joint VNF-FG composition and embedding solution, which considers the variations of service demands while also accounting for dynamic network conditions. Specifically, our proposed solution relies on deep reinforcement learning empowered by two components for estimating dynamic parameters: network resource utilization and service demand analyzers. Moreover, to efficiently explore the problem's large discrete action space, we utilize a specialized branching Q-network and enhance it with an action filtering mechanism. We evaluate our proposed method against joint and disjoint composition and embedding heuristics as well as versus other deep

learning-based methods. Our results show that the proposed method can achieve up to a 95% improvement in embedding cost compared to our benchmarks.

1.2.2.2 Joint VNF-FG Function Scaling and Topology Adaptation for the Next-generation CDNs' Deployment [12]

The second contribution of this thesis is a joint VNF-FG function scaling and topology adaptation method. This method targets the necessary adaptations of already deployed and running VNF-FGs that have become inefficient by the increase of the service demands and fluctuations of network conditions. Given that function scaling and topology adaptation may have mutual correlations, a disjoint approach could lead to inefficient results. Thus, in our method, not only the number of VNF instances are adjusted, but also the topology of the VNF-FG and its mapping to the physical network are properly altered. In this contribution, we propose a deep reinforcement learning (DRL)-based joint framework, which takes advantage of a Deep Double Q network architecture enhanced with action filtering to jointly update the function and topology of the already deployed VNF-FGs. Our evaluation results show that the proposed method achieves up to a 93% cost improvement compared to our benchmarks.

1.2.2.3 Content placement for the Next-generation CDNs [13] [14]

The third contribution is a framework for content placement in edge-based CDNs with vehicular nodes. In this contribution, based on real-life situations, we consider a dynamic and heterogeneous environment consisting of mobile and fixed caches where contents have pre-assigned high and low priorities and developed a use case from a vehicular network to illustrate the motivation of our work. Our proposed method considers the available caching capacity in edge caches so that upon the arrival of high-priority contents, instead of just removing the low-priority contents from full caches, it migrates low-priority contents between edge caches to create enough space to accommodate high-priority contents. We implement our DRL migration agent with a deep double-Q learner method empowered by LSTM memory cells. The simulation results show up to 70% in cost improvements compared to the existing methods.

1.2.2.4 Performance Management for the Next-generation CDNs [15]

Users' viewing experience in the video delivery process is of paramount importance for Content Delivery Networks (CDNs). Throughout their operations, CDN providers target the satisfaction of

users' expectations in terms of Quality of Experience (QoE). In this context, CDN providers need to acquire knowledge of users' QoE and correlate observations through different video sessions to identify QoE degradations and investigate their potential root causes. In the absence of users' feedback on their QoE, CDN providers can monitor and analyze Key Performance Indicators (KPIs) throughout video sessions. This allows assessing the Quality of Service (QoS) offered to users, influencing their QoE. However, due to the large number of sessions handled by CDN operators, it is not possible to conduct such an analysis manually. In this work, we introduce a framework that allows to automatically group a large set of video sessions into a small number of representative clusters, with each cluster containing video sessions with similar patterns of KPIs. The framework builds upon a set of features representing the evolution of KPIs over a session. It relies on an unsupervised machine learning algorithm to form the clusters. We evaluate the framework over a real-world dataset with traffic logs relating to thousands of sessions. The obtained results underline the capabilities of the proposed framework.

1.3 Background Information

This subsection presents the background information that is relevant to our research domain and it covers the following topics: The general principles of CDN, service deployment, content placement, and performance management. Moreover, the general principles of machine learning methods will be discussed in this subsection.

1.3.1 General Principles of Content Delivery Networks

Content Delivery Networks (CDNs) consist of a collection of Web servers distributed over multiple locations with the main objective of delivering content to end-users with reduced latency. They were traditionally provisioned with static Web technologies but are now mostly integrated with modern technologies such as cloud computing. The main entities of a CDN are content servers (a.k.a origin servers), surrogate servers (sometimes called replica servers), and a controller [1]. Content servers hold the original copy of the content that end-users want to access. Surrogate servers are the servers in which the content of the origin server is replicated. They are deployed in strategic locations to enable the rapid delivery of content to end-users. They also enable load balancing and efficient resource usage. The controller uses certain criteria (e.g., content

availability, physical distance, and network conditions) to choose the most appropriate surrogate server for each end user's request and redirect the request to the selected server.

1.3.1.1 Cloud-based Content Delivery Networks

Cloud computing has several inherent advantages, such as scalability, on-demand resource allocation, flexible pricing model (pay-as-you-go), and easy applications and services provisioning [1]. CDNs can leverage these advantages. An example of a cloud-based CDN is MetaCDN [16]. In cloud-based CDN, replica servers are provisioned as cloud applications on top of Infrastructure as a Service (IaaS) [1]. However, cloud-based CDNs still face issues in meeting end-users expectations when it comes to latency. The distance between surrogate servers (possibly residing in the cloud) and end-users remains the main roadblock. Thereby the edge computing paradigm can be utilized by the next generation CDNs to address the issue of latency [17].

1.3.1.2 Cloud-based Content Delivery Networks with Edge Nodes

The next-generation CDN could be seen as an extended cloud-based CDN with edge caching resources. Indeed, moving the location of the caches closer to the edge of the network has the advantage of reducing the latency required for accessing and delivering users' requests [17]. In particular, caching at the edge node, such as base stations, roadside units, or even on vehicles' onboard units allows the delivery of content to mobile users with limited need for backhaul usage to connect to a remote surrogate server and thus to reduce the latency [17].

1.3.2 Deployment, Content Placement, and Performance Management of The Next-generation CDN

1.3.2.1 Deployment of the next-generation CDNs

The next-generation CDN is envisioned to be deployed on shared NFV-based infrastructure. Network Function Virtualization (NFV) [3] is a newly emerged network service deployment technology. The network functions that previously could be offered only by specific hardware appliances can now run as Virtualized Network Functions (VNF) on commodity servers or switches. Therefore, it reduces the capital and operational costs while yielding flexibility and scalability for the network operators. To this end, in the NFV paradigm, network services are deployed as an ordered set of VNFs called VNF-FG. In this regard, the so-called NFV resource allocation (NFV-RA) problem should be addressed [4]. The NFV-RA comprises three stages: (i)

composition of the VNF forwarding graph, (ii) embedding of the VNF-FG on the given substrate network, and (iii) scheduling of VNFs on the substrate nodes/links.

With the help of NFV, multiple virtual networks (e.g., content delivery networks) can be created and managed on top of shared physical infrastructure, and that is a significant opportunity to reduce capital and operational costs while yielding flexibility and scalability for network operators [3].

1.3.2.2 Content Placement for the next-generation CDNs

Content Placement (CP) algorithms [5] determine the selection of the contents to be stored in the surrogate servers in traditional CDNs. They directly affect meeting the end-user demands with the expected quality of service [5] and thus are quite important. They were conventionally categorized into either pull or push-based, considering how they are retrieved from origin servers to surrogate servers. With the integration of the new paradigms (e.g., Edge computing and NFV), the next-generation CDNs could consist of edge-based local caches in the vicinity of end-users, and they can be even dynamic. In this regard, advanced CP decisions are needed to determine the placement of content not only on surrogate servers but also on these edge caches as well.

1.3.2.3 Performance Management of the next-generation CDNs

In this contribution, we introduce a framework for the analysis of Key Performance Indicators (KPIs) in large-scale CDN systems so that in the absence of users' feedback on their QoE, CDN providers can monitor and analyze the performance evolutions throughout video sessions. Since it is not possible to conduct such an analysis manually, we introduce a framework that allows to automatically group a large set of video sessions into a small number of representative clusters, with each cluster containing video sessions with similar patterns of KPIs. The framework builds upon a set of features representing the evolution of KPIs over a session. It relies on an unsupervised machine learning algorithm to form the clusters. We evaluate the framework over a real-world dataset with traffic logs relating to thousands of sessions. The obtained results underline the capabilities of the proposed framework. Our framework employs an unsupervised machine learning algorithm to automatically form clusters of video sessions, presenting similar evolution of KPIs.

1.3.3 Machine Learning Algorithms

Machine Learning (ML) algorithms [18] are the process of automatic learning from a collected set of data or experiences. They build data-driven models by automatic analysis and thus provide efficiency and cost-effectiveness in computing processes [18]. Considering the availability of true labels for the data, ML methods can be categorized into supervised and unsupervised learnings. Moreover, Reinforcement Learning (RL) is another branch of the ML algorithm that learns with continuous interaction with the environment. The recent integration of Deep Neural Network (DNN) with RL has introduced a powerful ML tool called Deep Reinforcement Learning (DRL). A method that has been successfully and widely used in many different domains. Indeed, because of the interdisciplinary nature of ML algorithms in general, they play pivotal roles in various fields, including engineering, medical, and computing [18]. In the following subsections, we provide background on each of the ML classes (i.e., supervised, unsupervised, and reinforcement learning).

In supervised learning methods, the model is built over a collection of pairs of input and desired output [18]. The model will be trained so that a mapping function between input and output space be found. Consequently, the model will be tested over another input-output set (i.e., a testing set) that was not used for training. When the goal is to predict a continuous or quantitative output value, the corresponding problem to be solved is called regression, whereas the prediction of a categorical or qualitative output is known as a classification problem [18]. Support Vector Machines (SVMs), Artificial Neural Nets (ANN), logistic regression, naive Bayes, and random forests are some of the widely used supervised learning algorithms. Their application in communication systems includes channel decoder and email spam classification [19].

Q-Learning [20] is the most widely used reinforcement algorithm. Q-learning works by successively updating the evaluation of the long-term quality (the Q value) of actions at each state. It is a simple way for an agent to learn how to act optimally [20]. We note, however, that classic Q-learning is limited to tasks with a small number of states and actions [20]. Moreover, in the Q-learning algorithm, all the states should be met, and all the actions should be experienced. Those restrictions are impractical in most real-world problems, as they deal with environments that are extremely complex and dynamic, and their states are large and vary rapidly over time. The only way to learn anything in these types of dynamic situations (where we have dynamic state-space) is to generalize from previously experienced states to new states [20]. The required generalization

is often called function approximation [20]. To approximate the Q values for unmet states/actions, one can use a deep neural network (DNN)-based approach, which relies on nonlinear gradient-descent function approximation [20]. This approach eliminates the need for visiting all the state/action pairs to compute the Q values. First proposed in [21], this revival hybrid approach is now widely used in different domains under the so-called deep reinforcement learning (DRL) or deep Q-learning (DQL) method [19].

1.4 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 presents the motivating scenarios, the driven requirements, and a thorough review of the state-of-the-art. Chapter 3 presents a deep learning framework for joint VNF-FG composition and embedding for CDN deployment. In chapter 4, we discuss a joint function scaling and topology adaptation method. In chapter 5, a content placement method for CDN with mobile nodes is presented. Moreover, we present our unsupervised KPI clustering framework for CDN performance management in chapter 6. Finally, Chapter 7 concludes the thesis and presents the future work.

Chapter 2

2. Related Work

In this chapter, we first present a list of requirements for algorithms in CDNs, and then we survey the state-of-the-art accordingly.

2.1 Requirements

We divide the set of requirements into two classes of CDN general requirements and specific requirements that correspond to contributions made in this thesis.

2.1.1 General Requirements

QoS/QoE: The primary goal of CDN is to deliver quality content to the end-users. In this regard, The CDN algorithms not only should consider QoS metrics (such as latency jitter, etc.) but also should take into account the perceived quality of the end-users by considering indicative QoE metrics.

Cost: Another important objective in CDN algorithms is to ensure cost-efficiency**.** In this regard, the algorithm should be able to minimize the costs, which could be the budgets for the deployment or operation of the CDNs.

2.1.2 Specific Requirements

The requirements discussed in this section are specific to contributions made in this thesis and are divided into three classes: requirements for NFV-based deployment of CDN, CDN content placement, and performance management requirements.

2.1.2.1 Requirements for NFV-based deployment of CDN

Considering the advantages, the next-generation CDN will be deployed on shared infrastructure using NFV technology. In such an environment, the content delivery service is deployed by a set of chained VNFs (called VNF forwarding graph or VNF-FG) through which the service traffic should traverse one by one and finally reach the end-users. To this end, the following requirements should be satisfied:

Consideration of CDN's dynamic service demand: The incoming traffic demand for content delivery service is highly dynamic. For example, the demand for live streaming peaks during a football match, and these variations should be considered in the NFV-based deployment of CDNs.

Consideration of dynamic condition in CDN underlying physical network: Since the resources of the substrate network are shared among different service flows, the network resource congestion conditions are highly dynamic. For instance, the congested link between two physical nodes may dissolve shortly after a streaming application is terminated.

Joint CDN service's VNF-FG composition and embedding: Deployment of a CDN with disjoint VNF-FG composition and embedding may lead to violation of service requirements and consequently deteriorate the end-users perceived quality.

Joint CDN service's VNF-FG function scaling and topology adaptation: a well-performing CDN deployed by VNFs, could become inefficient as the service demands increase and network conditions fluctuate [3]. To that end, not only the number of VNF instances need to be adjusted (a process known as function scaling or horizontal scaling), but also the topology of the VNF-FG and its mapping to the physical network may need proper adjustments via so-called topology adaptation. Moreover, since function scaling and topology adaptation could have mutual

correlations, they need to be considered joint, or otherwise undesirable results leading to QoE degradation might happen.

2.1.2.2 Requirements for content placement in CDN with mobile edge nodes

The content placement in an edge-based CDN with vehicular nodes has a vital impact on the transmission delay and consequently on the end users' QoE. In this environment, it is crucial to account for the followings:

Consideration of mobility for both end-users and edge caches: An edge-based CDN [17] can consist of not only mobile end-users but also dynamic local caches that are deployed over vehicles.

Consideration of low- and high-priority content: In a CDN with mobile edge caches, the caching priority of contents could vary widely. As an example, consider a vehicular network consisting of autonomous and non-autonomous vehicles [23] [24]. In this network, the caching priority of certain contents, such as high definition maps (HD maps), would generally be higher than that of infotainment contents. Since vehicles' onboard sensors are limited to line-of-sight, autonomous vehicles rely heavily on these maps to plan precisely and maneuver correctly on the road. These machine-readable HD maps model the surface of the road to an accuracy of 10-20 cm and therefore have large volumes.

Consideration of the limited capacity of edge caches: Unfortunately, the local caches have limited capacities, and when they are fully occupied, it may sometimes be necessary to remove their lower-priority content to accommodate higher-priority content. At other times, it may be necessary to return previously removed content to local caches. Downloading this content from surrogate servers is costly from the perspective of network usage and potentially detrimental to the end-user QoE in terms of delay.

2.1.2.3 Requirements for Performance management in CDN

The CDN provider must track and analyze performance evolution. In the absence of users' feedback on their QoE, CDN providers can monitor and analyze Key Performance Indicators (KPIs) throughout video sessions. This would allow the CDN provider to learn about QoS and QoE offered to users, identify possible drops, and know whether they are caused by issues inside

or outside the CDN's domain. To that end, we derive the following requirements for a framework targeting QoS and QoE analysis for CDN providers.

Scalability: A CDN provider operates over a large scale, handling demand from a large set of users as well as a large set of content. A QoS and QoE analysis framework should therefore operate efficiently over a large scale.

Flexibility: Multiple KPIs that reflect users' QoE can be collected by the CDN provider. A QoS and QoE analysis framework needs to be flexible to account for multiple KPIs.

Automation: As analyzing and correlating KPIs cannot be handled manually over a large scale, a QoS and QoE analysis framework needs to offer automated procedures.

Fine granularity: Users' QoE can fluctuate over time throughout a single video session. A QoS and QoE analysis framework, therefore, needs to target the analysis of corresponding fluctuations.

2.2 Related work

In this section, we will discuss the works from the literature that are closely related to each of the thesis contributions. We first discuss and analyze the works related to service deployment and present them into two categories of the VNF-FG composition and embedding and VNF-FG function scaling and topology adaptation. After that, we review the works related to content placement and performance management in CDNs.

2.2.1 VNF-FG Composition and Embedding Related Work

In this section, we first review the few research works that target the joint problem of VNF-FG composition and embedding and then evaluate some of the recent disjoint methods. In our evaluation, we regard the general requirements and the requirements for NFV-based deployment of CDN discussed in Sections 2.1.1 and 2.1.2.1, respectively.

2.2.1.1 Joint VNF-FG Composition and Embedding

Despite its determinant value, the joint consideration of VNF-FG composition and embedding remains relatively under-examined; only relatively few works consider VNF-FG composition and

embedding together. Based on the structure of the heuristic, we classify these works into two categories of one- (e.g. [7], [25], [8], and [9]) and two-stage (e.g., [26], [10], [11], [27], and [28]) algorithms.

One-stage heuristics: In one-stage algorithms, for each requested VNF in a VNFR, the order and physical node that will host the VNF are determined simultaneously in a single step. Ref. [7] is one of the earliest one-stage works that considers VNF-FG composition and embedding simultaneously. The authors of [7] proposed their recursive CoordVNF heuristic for the coordination of VNF-FG composition and embedding with the objective of minimizing bandwidth usage. Comparable to CoordVNF is the so-called JoraNFV algorithm, proposed in [25], which coordinates the three phases of resource allocation, VNF-FG composition, embedding, and scheduling. The JoraNFV algorithm is designed to minimize a rather comprehensive cost model consisting of capital, operating, and link costs. To this end, the problem is formulated as a mixed-integer linear programming (MILP) problem, addressed by a single-stage heuristic. The correlation of VNF-FG composition and embedding and their joint impact on network operator revenue were extensively studied in [8], where the authors presented an ILP formulation of the chain composition and embedding problem with the objective of maximizing revenue of resource sharing. In [9], certain VNF embedding constraints, known as location constraints, are considered. These constraints enforce limitations on the number of nodes that can host specific VNFs and thus complicate the joint problem of VNF-FG composition and embedding. After presenting an ILP formulation of the joint problem, these authors proposed an exact method followed by a greedy-based heuristic algorithm for large-scale problems.

Two-stage heuristics: Two-stage algorithms begin with the composition (or selection) of a single VNF-FG (or a subset of all possible VNF-FGs). In this composition stage, not only VNFRs but also some embedding-related restrictions will be observed. The composed VNF-FG (or the selected subset of them) will then go through the embedding stage. In this stage, with an embedding objective (e.g., minimizing the costs, maximizing revenue, etc.), either the algorithm produces (if possible) an embedding solution for the composed VNF-FG or the most appropriate solution among the previously selected VNF-FGs satisfying the embedding objective will be chosen. If embedding of the composed VNF-FG is not possible or the objective is not well satisfied by either of the VNF-FGs in the selected set, the algorithm returns to the composition stage and looks for

alternative VNF-FG(s). For example, in [26], the authors proposed a two-stage heuristic algorithm to jointly compose and embed the service requests. In the first stage, an order of VNFs (i.e., a VNF-FG) is determined such that the embedding cost is minimized. To this end, the proposed heuristic considers the following two aspects: (i) the location of the candidate node to host a given VNF and (ii) the ratio of the outgoing data rate over the incoming data rate for a given VNF. The allocation of resources for the composed VNF-FG is then carried out in the next stage (the embedding stage). In this stage, if any assigned node becomes over-utilized, a node splitting mechanism is triggered, which alters the structure of the composed VNF-FG. As a result, the acceptance ratio of the service requests will be improved.

Relatively only a few two-stage heuristic works consider QoS parameters such as availability, latency, and reliability in their joint VNF-FG composition and embedding solutions. As for the availability of a given service function, the failure of both embedding nodes and VNFs should be considered. To this end, the authors of [10] introduced the concept of traffic-weighted availability, which is defined as the fraction of traffic that can be supported considering the availability state of the underlying infrastructure. With the help of this newly introduced metric, the authors of [10] solve the VNF-FG composition and embedding problem while guaranteeing availability. However, the method presented in [10] is only applicable for applications that can tolerate a certain range of bandwidth reduction. Alternatively, a probability-based availability and backup model for VNF-FG is presented in [11], where the authors propose a two-stage heuristic for VNF-FG composition and embedding while ensuring availability.

Latency and reliability, two important QoS metrics, were considered in [27] and [28], respectively. In [27], the authors target latency minimization for applications that require different service functions on forwarding and backward traffic. To this end, they define a problem called Hybrid VNF-FG Composition and Embedding and, considering the substrate network provisioning state, propose two approximation algorithms that minimize the latency of the constructed service function. The work in [28] studies the trade-off between reliability and per-server load in a bipartite forwarding graph using a probabilistic model. For the joint optimization of a VNF-FG and the embedding, they provide a mixed ILP formulation to maximize end-to-end reliability and propose a two-stage Block coordinate descent (BCD)-based heuristic to solve it. We note that none of the existing joint composition and embedding solutions supports service demand

dynamics (The first requirement in Section 2.1.2.1) or network condition variations (The second requirement in Section 2.1.2.1). Therefore, these two requirements remain unmet by all these related works.

Table 2.1 VNF-FG composition and embedding related work evaluation

Requirements / Related Works	Requirements				
	General Requirements		VNF-based Deployment Requirements		
	QoS/ QoE	Cost	VNF-FG Composition and embedding	Service Demand Dynamics	network Condition Dynamics
M. Beck *et al.*, [7]	✓	✓	✓(Joint approach)	x	x
Araujo *et al.*, [8]	✓	✓	✓(Joint approach)	x	x
Spinnewyn *et al.*, [9]	✓	✓	✓(Joint approach)	x	x
Gour *et al.*, [10]	✓	✓	✓(Joint approach)	x	✓
Wang *et al.*, [11]	✓	✓	✓(Joint approach)	x	✓
Chen *et al.*, [29]	✓	✓	Composition Only	x	x
Bian *et al.*, [30]	✓	x	Composition Only	x	x
Ning *et al.*, [31]	✓	✓	Composition Only	x	✓
Pham *et al.*, [34]	✓	✓	Embedding Only	x	x
Wu *et al.*, [35]	✓	✓	Embedding Only	✓	x
Pei *et al.*, [36]	✓	✓	Embedding Only	x	✓

2.2.1.2 Disjoint VNF-FG Composition and Embedding

Here we first present the disjoint approaches targeting VNF-FG composition and then proceed with recently published embedding-only studies. In this regard, we review both Machine Learning (ML)-based and non-ML-based methods.

VNF-FG Composition: Following an intent-based service design approach, the authors in [29] propose a semantic-based VNF-FG composer called CompRess, in which users' intents are translated into multiple possible VNF-FGs. However, their VNF-FG composition design does not provide a particular solution as to the best ordering pattern for requested VNFs. In [30], researchers formulate the VNF-FG composition problem as a non-cooperative game to reduce request latency while considering (fixed) network congestion. The work in [31] considers the heterogeneity of today's Internet traffic and proposes a dynamic DRL-based VNF-FG composition that regards traditional IP-based traffic (known as background traffic) patterns and updates the VNF-FG chaining accordingly, aiming at minimizing the service's flow path delay. In [32], the authors first formulate the VNF-FG composition problem as a binary integer program and then use a deep belief network (a supervised learning model) to obtain the best chaining strategy. Their strategy aims at minimizing the end-to-end delay while respecting VNF-FG's request constraints. Despite the consideration of the substrate network dynamics, both works discussed above (i.e. [31] and [32]) unrealistically assume service requests to be static.

VNF-FG Embedding: As exhaustively surveyed in [4] and more recently [33], there are multiple approaches (e.g., Markov approximation, Game theory, DRL, etc.) for VNF embedding. For example, [34] uses a sample-based Markov approximation approach and further enhances it with a matching game to tackle the complexity of the VNF-FG embedding problem. Game theory is also widely used to address the challenges in VNF embedding. In [35], for instance, researchers propose a user-network cooperation-based method to minimize the VNF-FG embedding cost while considering the service response time in a dynamic workload (but fixed network congestion) condition. In fact, the automated learning ability of DRL approaches makes them suitable for the dynamic VNF-FG embedding problem, and there are indeed numerous DRL-based embedding works. Some examples are [36], [37], [38], [39], [40], and [41]. In [36], the authors propose a DRL-based method called “DDQN-VNFPA.” Their proposed learning method includes offline

training while aiming to minimize the embedding cost. Although substrate network dynamics are considered in "DDQN-VNFPA," the service network requests are assumed to be fixed.

Embedding cost minimization was also targeted in [39], in which the authors propose "DeepOpt," an RL embedding method that is empowered with Graph Neural Network (GNN) instead of the more commonly used feedforward neural networks to improve the applicability of their solution for different network structures. However, their approach considers static network conditions and service demands.

The work in [37] proposes an actor-critic-based algorithm called "UNREAL_MD," in which the feedback of the environment is quantified by a queuing model that evaluates the delay of the embedded VNF-FG. "NFVdeep," an adaptive online method that considers not only the variation of VNFR requirements but also the changes in substrate network conditions, is proposed in [38]. This work aims to minimize the operating cost while maximizing the total throughput. It uses a serialization-and-backtracking method to handle the large action space of the embedding problem. More recently, in [40], researchers propose "DDQP," a dynamic DRL-based method to robustly deploy both active and standby VNF-FG instances considering both the variation of service demand and network conditions with the objective of minimizing resource waste. The dynamics of underlying substrate networks are also considered in [41], in which complex VNFs are first decomposed into smaller VNF components, and then a DRL-based delay minimizing scheme decides on their placements. To improve the results, the authors apply both the experience reply and target network mechanisms in their scheme. To summarize, even though the related works described above all exploit the advanced features of a DRL method, to the best of our knowledge, no work has applied a DRL approach to solving VNF-FG composition and embedding problems jointly while considering the dynamics of substrate network conditions and service requests.

2.2.2 VNF-FG Adaptation Related Work

We classify the related work of VNF-FG deployment into two groups of function scaling and topology adaptation. We note that while there exist a number of works on function scaling only and topology adaptation only, the joint approach has not been studied before. In our evaluation, we consider the general requirements and the requirements for NFV-based deployment of CDN discussed in Sections 2.1.1 and 2.1.2.1, respectively

Table 2.2 VNF-FG adaptation related work evaluation

Requirements / Related Works	Requirements				
	General Requirements		VNF-FG Adaptation Requirements		
	QoS/ QoE	Cost	VNF-FG Function Scaling and Topology Adaptation	Service Demand Dynamics	network Condition Dynamics
Fei *et al.*, [42]	✓	x	Function Scaling Only	✓	x
Panday *et al.*, [44]	✓	x	Function Scaling Only	✓	x
Subramanya *et al.*, [45]	✓	x	Function Scaling Only	✓	x
Luo *et al.*, [46]	✓	✓	Function Scaling Only	✓	x
Lang *et al.*, [47]	✓	✓	Function Scaling Only	✓	x
Houidi *et al.*, [48]	✓	✓	Topology Adaptation Only	✓	x
Liu *et al.*, [49]	✓	✓	Topology Adaptation Only	✓	✓

2.2.2.1 VNF-FG Function Scaling

Adjusting the number of instantiated VNFs in response to a rise in network traffic has been investigated in some recent works (e.g., [42], [43], [44], [45], [46], and [47]. Most of these works rely on predicting the rise of VNF demand, which is considered as a trigger for scaling. In [42], a

simple pre-determined threshold-based load monitoring mechanism was used for initiating the scaling procedure. However, the complications of calculating these thresholds and their dependent parameters are not discussed. In [44], a Gated Recurrent Unit (GRU) was proposed as a resource demand predictor. However, once the increase in service demand is predicted and the action of scaling is decided, the whole service function chain is scaled, though the imposed cost of such a decision was not discussed. In [45], the authors targeted the VNF auto-scaling in multi-domain networks by leveraging centralized, and federated learning prediction approaches. The future numbers of VNF instances are calculated as a function of the predicted traffic demands. However, only a general form of resource allocation cost is considered, while other costs such as instantiation of new VNFs and state copying are ignored. Both Refs. [44] and [45] use oversimplified cost models. Similar to [45], a VNF deployment and migration method for heterogeneous edge and cloud environments was proposed in [43]. However, the migration in their work is considered as a mechanism for accommodating the rejected VNF-FG requests rather than a scaling mechanism for the original VNF-FG. In [46], a dependent rounding online algorithm was proposed to ensure that enough VNF instances are deployed in case of network traffic fluctuations. In [47], a supervised learning agent (which is trained by generated labeled data) determines the required number of VNF instances according to the changes in service demand. While Refs. [46] and [47] consider a rather comprehensive cost model, the topologies of the VNF-FGs are assumed to be fixed. In other words, in these works, the connectivity of the VNFs in a service chain does not change in the scaling process.

2.2.2.2 VNF-FG Topology Adaptation

Here, we review the existing works that target VNF-FG topology adaptation (or expansion) in response to a rise in service demand. We review the papers that make modifications in the structure of the VNF-FG but exclude those works that preserve the initial VNF-FG and only apply the modifications to the embedding graph, e.g., [38], [40]. In [48], the authors studied the VNF-FG extension problem and proposed two heuristics, namely, Steiner Tree-based and eigendecomposition-based algorithms, to minimize the rejection of extension requests when the demand increases. In their work, the initially embedded graph remains unaltered while additional nodes and links are simply attached to the edge of the graph as a response to the changes in the demand of the users. However, there are situations where the whole VNF-FG needs to be modified

or that adding new VNFs just to the edge of the graph could not be feasible. For instance, a particular VNF may need to be reached in between two other VNFs and not at the edge of the graph. The authors of [49] considered the dynamicity of mobile users as well as the changes in their service demand, aiming to adjust the deployed VNF-FG accordingly. To that end, they first formulated the problem as an Integer Linear Programming (ILP) and solved it using the Column Generation (CG) approach. However, in their work to decrease the complexity, the order of VNFs is reserved. Also, [49] only considers VNF-FGs with simple typologies, thus having limited applicability in real-world scenarios. None of these works provide a general framework where both function scaling and topology adaptation can both be leveraged.

2.2.3 CDN Content Placement Related Work

Using the general requirements and the specific requirements for content placement for CDN with mobile edge nodes discussed in Sections 2.1.1 and 2.1.2.2, respectively, we discuss the related work for content placement. We first review the existing research works that target the edge content placement and delivery problem in CDNs while considering a content priority scheme. Next, we review the recent DRL-based approaches in the edge caching domain.

2.2.3.1 Content Placement Approaches with content priority schemes

There are very few works that consider specific content priority schemes in CDNs with edge nodes. Most of them focus on priority content dissemination rather than caching technology. The work in [50], for instance, proposes a priority-based content propagation scheme that accelerates safety content delivery for a set of moving vehicles and provides the forwarding of non-safety contents based on popularity. Similarly, in [51], an information-centric dissemination protocol for safety information in vehicular ad-hoc networks was proposed. The authors of [52] proposed an architecture that uses a data cognitive engine to determine user priority (based on the users' health situation) and allocates edge resources (including edge caching resources) accordingly through a resource-cognitive engine. However, none of the above-mentioned works consider the limited caching capability of edge caches in their solutions, and so none of them propose a strategy for content eviction.

Table 2.3 Content placement related work evaluation

Requirements / Related Works	Requirements				
	General Requirements		Content Placement Requirements		
	QoS/ QoE	Cost	High- Low priority contents	Mobility	Limited Size of Edge Caches
Khan et al., [50]	✓	x	✓	✓	x
Meuser et al., [51]	✓	x	✓	✓	✓
Chen et al., [52]	✓	x	✓	x	x
Zhu et al., [53]	x	✓	x	✓	x
Hu et al., [55]	✓	x	x	✓	x
He et al., [56]	✓	✓	x	x	✓
Yu *et al.,* [57]	✓	✓	x	✓	✓
Qiao *et al.,* [58]	✓	✓	x	✓	x
Gomaa *et al.,* [59]	x	✓	x	x	✓

2.2.3.2 DRL-based approaches for edge content caching

The use of deep reinforcement learning (DRL) has become quite popular in the networking domain. In a recent work, the authors of [19] conducted a comprehensive survey on DRL applications for solving a variety of networking problems (e.g., dynamic network access, wireless

caching, and data rate control). Specifically, as stated in [19], the adoption of DRL for edge caching has received more attention than other networking issues. Zhu et al. [53] advocated the use of DRL by examining key challenges in mobile edge caching and then mapping them with unique DRL aspects. The existing edge caching DRL-based approaches can be classified into two categories: (i) works that use DRL for learning specific caching parameters (e.g., content popularity [54] or cache expiration time), and (ii) DRL approaches that target multiple aspects in their caching policy design [55],[56] (e.g., networking and computation). Falling into the first category, the authors in [54] propose a DRL-based cache replacement scheme for a single BS, where the content popularity is learned by considering the cache hit rate as the system reward. Similarly, in their recent work [57], Yu et al. propose a federated learning approach to predict content popularity for connected vehicles and provide a mobility-aware cache replacement policy. Many recent works in the vehicular network domain belong to the second category. For instance, Hu et al. [55] proposed integrated networking, caching, and computing optimization framework for connected vehicles that sets both operational excellence and cost efficiency as objectives. They adopted deep reinforcement learning to overcome the high level of complexity caused by the joint optimization problem. However, they do not suggest a strategy for the case of full edge caches. Similarly, in the recent work of Qiao et al. [58], DRL is utilized to solve the joint optimization of content placement and delivery problems in the vehicular networks, formulated as a double time-scale Markov decision process. Gomaa et al. [59] proposed a dynamic orchestration framework for communication, caching, and computing resources in a software-defined and virtualized vehicular network. They applied DRL to obtain a close-to-optimal policy for integrated resource allocation. However, no concrete solution for the case of full edge caches is proposed. Considering that resources in edge caches are indeed limited, it is quite probable that these caches become fully occupied. Therefore, having no strategy for these cases is a notable shortcoming of these DRL-based edge caching methods.

2.2.4 CDN Performance Management Related Work

In the following, we present CDN performance management works and evaluate them according to the general and specific set of requirements listed in Sections 2.1.1 and 2.1.2.3.

Several studies have aimed at investigating the impact of specific events in the CDN system on the QoS and QoE over individual video sessions. Fan et al. [60] studied the impact of changes in

the CDN redirection mechanism on the latency perceived by users. The study was conducted over a large scale. However, only one KPI was considered, i.e., latency. Casas et al. [61] also analyzed the impact of events in CDNs on the QoE of users. To this end, the authors propose automated approaches that operate over a large scale and consider multiple KPIs. However, their work does not consider the fluctuations of KPIs throughout sessions. Shafiq et al. [62] studied the impact of network dynamics on user abandonment behavior. Their analysis was conducted over a large scale, over multiple KPIs, and relies on an automated approach. Nevertheless, KPIs fluctuations over individual sessions were not considered.

Other studies have aimed at identifying the mapping between QoS and QoE metrics in CDNs using traffic datasets. Li et al. [63] studied the correlations between video download throughput and user engagement. A large-scale analysis of a single KPI, i.e., download throughput, is conducted. However, it does not account for its fluctuations. Lian et al. [64] studied correlations between performance and QoE metrics. The correlations were considered for multiple KPIs through automated approaches. However, the evolution of KPIs is not considered, and the analysis is led over a small scale. Orsolic et al. [65] predict the QoE level of a session according to a set of KPI features. The evolution of multiple KPIs is considered over individual sessions with an automated approach for prediction. However, the study is conducted over a small scale.

Multiple studies focus on the analysis of anomalies based on CDN traffic datasets. Giordano et al. [66] propose a method to identify changes in the CDN cache selection policy. Their method operates over a large scale, considers multiple KPIs, and is automated. Nevertheless, it does not account for the evolution of KPIs throughout sessions. Wu et al. [67] focus on the detection of video freeze events in video sessions. They propose an automated method that operates over a large scale, considering the evolution of only one KPI over each session, i.e., the inter-segment duration. Dimopoulos et al. [68] propose a framework to diagnose the root cause of mobile video QoE issues. The framework adopts an automated approach that covers multiple KPIs. However, it operates over a small scale and does not account for the evolution of KPIs across video sessions. In turn, Zhu et al. [69] also target the diagnosis of QoE issues. They propose an automated approach that allows identifying the root cause of large latency increases over a large scale. Nevertheless, they only account for latency and do not consider its evolution through video sessions.

Table 2.4 Performance management related work evaluation

Requirements / Related Works	Requirements					
	General Requirements		Performance Management Requirements			
	QoS/ QoE	Cost	Scalability	Flexibility	Automation	Fine Granularity
Fan *et al.*, [60]	✓	✓	✓	x	x	✓
Casas *et al.*, [61]	✓	x	✓	✓	✓	x
Shafiq *et al.*, [62]	✓	✓	✓	✓	✓	x
Li *et al.*, [63]	✓	x	✓	x	x	x
Lian *et al.*, [64]	✓	x	x	✓	✓	✓
Orsolic *et al.*, [65]	✓	✓	x	✓	✓	✓

In summary, none of the previous studies meets all requirements listed in Sections 2.1.1 and 2.1.2.3. Some studies were conducted over a small scale, and thus did not meet our scalability requirement. Many works do not consider multiple KPIs, leaving the flexibility requirement unsatisfied. All but two papers introduce automated procedures for their studies, meeting the automation requirement. As for fine granularity, only two studies fulfilled that requirement. While none of the previous works meets all our requirements, our KPIs analysis framework does accomplish that. Our framework forms clusters of video sessions, presenting a similar evolution of KPIs, using unsupervised machine learning tools. To the best of our knowledge, we are the first to investigate the evolution of KPIs throughout video sessions using unsupervised machine learning tools. Our framework provides a clear understanding of the evolution of KPIs throughout video sessions. It operates by considering the fine-grained evolution of multiple KPIs, throughout

each session, it adopts an automated approach and operates over a large scale, thereby meeting all requirements listed in Sections 2.1.1 and 2.1.2.3

2.3 Conclusion

In this chapter, we first presented sets of general and contribution-specific requirements. After that, we surveyed the related work. Table 2.1and Table 2.2 provide a summary of the reviewed papers, respectively. For each paper, we show the requirements which are met and the ones which are not met. As it can be seen, none of the reviewed works satisfy all our requirements.

Chapter 3

3. Joint VNF-FG Composition and Embedding for CDN Deployment[1]

3.1 Introduction

Deploying a CDN with the NFV paradigm is not easy. For example, considering the complex and dynamic nature of a CDN, resource allocation (RA) remains a challenging topic for the deployment of content delivery services. On the other hand, the two seemingly separated stages of a VNF forwarding graph (i.e., composition and embedding) can have inter-related impacts on the eventual service performance and the embedding cost and thus should be jointly considered. This chapter focuses on the joint VNF-FG composition and embedding problem and proposes a joint framework where the variations of service demands as well as dynamic network conditions are

[1] This chapter is based on a submitted paper:

- Sepideh Malektaji, Amin Ebrahimzadeh, Marsa Rayani, Vahid Maleki Raee, Halima Elbiaze, and Roch Glitho, *"Dynamic Joint VNF Forwarding Graph Composition and Embedding: A Deep Reinforcement Learning Framework" revised version* submitted to IEEE Transactions On Network and Service Management.

simultaneously considered. To manage the complexity of the problem, we formulate it as a Markov Decision Process and design our Reinforcement Learning (RL)-based framework that relies on a Q-learning approach [20], which makes simultaneous decisions regarding both the ordering and embedding of requested VNFs. Next, to cope with the huge discrete multi-dimensional action space, we utilize a variant of the Deep Q Network (DQN) approach, the Branching Dueling Q network (BDQN) [70], and further enhance it with an action filtering mechanism [71]. This step reduces the action space and helps explore the problem search space more efficiently. Given the set of VNF service requests (VNFRs), along with their QoS requirements, our proposed framework calculates the mapped forwarding graph of each VNFR, which not only determines a proper ordering of the requested VNFs (i.e., VNF-FG composition), it also specifies the hosted physical nodes and links for the requested VNFs and their connections (i.e., VNF-FG embedding). More specifically, the obtained mapped forwarding graph minimizes the accumulated embedding cost over the usage service period while meeting the VNFR-specific service throughput requirements. In doing so, we empower our framework with the so-called resource utilization analyzer and service demand analyzer, which estimate the time-varying service demand and network resource utilization, respectively.

The remainder of this chapter is organized as follows: we first provide the system model and problem formulation. We then present our proposed DRL-based joint framework in detail, followed by the performance evaluation of the framework. Finally, in the last subsection, the conclusion will be provided for this chapter.

3.2 System Model and Problem Formulation

3.2.1 System Model

In our system model, to differentiate the duration of the data collecting from the actual service usage periods, we view time as two consecutive intervals, namely, pre-service usage time and service usage time. Pre-service usage time is of duration T_0 and refers to $t < T_0$, whereas service usage time corresponds to $t \geq T_0$.

1) ***Substrate network:*** Substrate network is modeled as a directed graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ and $\mathcal{E}$ are the sets of $|\mathcal{N}|$ physical nodes and $|\mathcal{E}|$ directed links, respectively. Each $e_{p,q} \in \mathcal{E}$ represents the direct physical link connecting physical nodes n_p and n_q, $\forall n_p, n_q \in$

$\mathcal{N}$. Also, we let the set $\mathcal{P}_{p,q}^{max}$ contain all the paths such as $\varphi_{p,q}$ connecting the node n_p to n_q with a maximum path length of φ^{max} hops. Accordingly, $|\varphi^{max}|$ is the number of all existing paths in G with a maximum of φ^{max} hops. Each node $n_q \in \mathcal{N}$ has a processing capacity $W_{\text{CPU}}(n_p)$. Similarly, each link $e_{p,q} \in \mathcal{E}$ has a bandwidth capacity $W_{\text{BW}}(_{p,q})$.

2) ***VNF service request (VNFR):*** Let $\mathcal{R}$ be the set of $|\mathcal{R}|$ received VNFRs, each with a unique ID number. Services of the VNFRs should become available during the *service usage time* (i.e., t < T0). Further, we assume that VNFR $r \in \mathcal{R}$ is attributed to the following parameters [26]:

1) Ingress node $n_{r,s} \in \mathcal{N}$ and egress node $n_{r,d} \in \mathcal{N}$, which are the physical nodes from which the first VNF of the requested VNF-FG originates and at which the last VNF terminates, respectively.

2) The initial entering data rate at the beginning of the *service usage time* denoted by:

$$d_{r,s} = \{d_{r,s}(t) | t = T_0\}$$

Where $d_{r,i}(t)$ denotes the entering data rate to VNF $f_{r,i} \in F_r$ for t ≥ T0.

3) Set $F_r = \{f_{r,1}, f_{r,2}, \dots, f_{r,|F_r|}\}$ of $|F_r|$ required VNFs along with their dependency graph D_r, which is an acyclic-directed graph with $|F_r|$ vertices and $|L_r|$ directed links representing the dependency relations between VNFs. We denote $\zeta(D_r)$ as the Degree of Freedom (DoF) [103] of the dependency graph D_r, which is given by:

$$\zeta(D_r) = |\mathcal{F}_r| \times (|\mathcal{F}_r| - 1) - |\mathcal{L}_r| \qquad (3\text{-}1)$$

4) Ratio $R_{f_{r,i}}$ of outgoing data rate to incoming data rate of VNF $f_{r,i} \in F_r$.

5) Processing resource demand $P_{f_{r,i}}$ per bandwidth for VNF $f_{r,i} \in F_r$.

We assume that all the service request parameters explained above remain unchanged during the service usage period (i.e., t ≥ T_0) except for the entering data rate $d_{r,s}(t)$, which may vary over time. Figure 3.1 illustrates an example of a VNFR and its specifications.

3) ***Mapped forwarding graph****:* Let the mapped forwarding graph M_r of VNFR $r \in \mathcal{R}$ be an ordered list of $|M_r|$ distinct elements. Each element of M_r is a 3-tuple containing three components $f_{r,i}, n_j$, and $\varphi_{k,j}$, where $f_{r,i} \in F_r$, $n_j \in \mathcal{N}$, and $\varphi_{k,j} \in \mathcal{P}_{k,\ j}^{max}$. To that end, we denote the h^{th} element of M_r as follows:

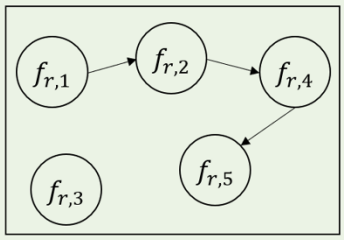
$$\sigma_{r,h}^{i,j,k} = < f_{r,i},\ n_j,\ \varphi_{k,j} > \qquad (3\text{-}2)$$

VNFR:

$\mathrm{ID}_r = 001,\ n_{r,s} = n_1,\ n_{r,d} = n_8,\ d_{r,s} = 2$ Gb/s, $q_r^{min} = 0.5$ Gbps

VNFs (i.e. $\mathcal{F}_r$)	CPU Demand (i.e. $P_{f_{r,i}}$)	Data Rate (i.e. $R_{f_{r,i}}$)
$f_{r,1}$	50	150%
$f_{r,2}$	30	60%
$f_{r,3}$	50	120%
$f_{r,4}$	20	40%
$f_{fr,5}$	10	80%

Dependency Graph D_r: $|\mathcal{F}_r| = 5, |\mathcal{L}_r| = 3,\ \zeta(D_r) = 17$



Substrate Network:

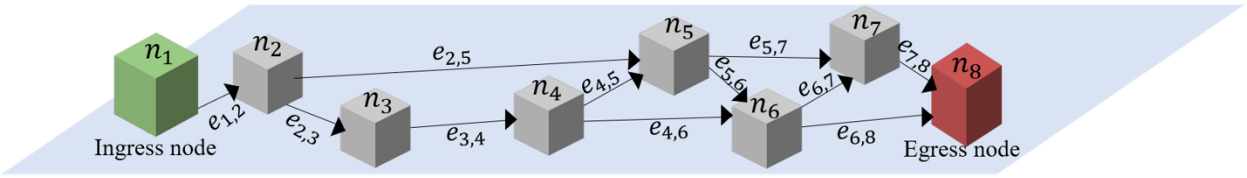


Figure 3.1 Example of a VNF service request, request specifications, and a dependency graph.

Additionally, let us define the function $\Pi_b(\sigma_{r,h}^{i,j,k})$ that returns component b (b = 1, 2, 3) of $\sigma_{r,h}^{i,j,k}$.

$$\Pi_b(\sigma_{r,h}^{i,j,k}) = \begin{cases} f_{r,i} & if\ b = 1 \\ n_j & if\ b = 2 \\ \varphi_{k,j} & if\ b = 3 \end{cases} \qquad (3\text{-}3)$$

As such, element h of M_r, which is $\sigma_{r,h}^{i,j,k} =< f_{r,i},\ n_j,\ \varphi_{k,j} >$, indicates that the h-th VNF in the composed VNF-FG for request $r \in \mathcal{R}$ is $\Pi_1(\sigma_{r,h}^{i,j,k}) = f_{r,i} \in \mathcal{F}_r$ and this VNF should be embedded in the physical node $\Pi_2(\sigma_{r,h}^{i,j,k}) = n_j \in \mathcal{N}$ and connected to the previously embedded VNF, i.e., $\Pi_1(\sigma_{r,h-1}^{i,j,k})$, through the embedding path $\Pi_3(\sigma_{r,h}^{i,j,k}) = \varphi_{k,j} \in \mathcal{P}_{k,j}^{max}$. Here, $n_k = \Pi_2(\sigma_{r,h-1}^{i,j,k})$, while assuming:

$$\Pi_2(\sigma_{r,0}^{i,j,k}) = n_s \ \text{ and } \ \Pi_2\left(\sigma_{r,|M_r|}^{i,j,k}\right) = n_d \qquad (3\text{-}4)$$

Since each physical node $n_j \in \mathcal{N}$ can host more than one VNF, we define $K_{M_r,j}$ as the set of VNFs of M_r that are hosted on the physical node n_j. We also define $\mathcal{E}_r \subseteq \mathcal{E}$ and $\mathcal{N}_r \subseteq \mathcal{N}$ as the sets of physical links and nodes involved in M_r, respectively. Note that $\mathcal{N}_r$ can also contain the

forwarding nodes which do not host any VNFs. Therefore, we define $\mathcal{N}'_r \subseteq \mathcal{N}_r$, as the set of physical nodes that, according to M_r host at least a VNF $f_{r,i} \in F_r$.

4) ***Resource utilization and service demand:*** Resources of physical nodes and links are shared among different service flows. Given that the resource consumption patterns of these flows are dynamic, available resources on substrate nodes and links will change frequently. Let $u_j(t)$ and $u_{v,z}(t)$ represent the CPU and bandwidth utilization of node $n_j \in \mathcal{N}$ and link $e_{v,z} \in \mathcal{E}$ for t ≥ 0, respectively.

5) ***QoS model:*** Service throughput as an important QoS parameter is critical to be considered in NFV resource allocation problems. In this regard, we consider service throughput as our QoS parameter specified by the service level agreement (SLA). We denote the minimum acceptable throughput as q_r^{min}, and define it as the minimum acceptable amount of outgoing data rate of service request $r \in \mathcal{R}$ from the egress node $n_{r,d} \in \mathcal{N}$. Let $q_{M_r}(t)$ be the end-to-end throughput of the mapped forwarding graph M_r at time t, t ≥ T0. Since the graph topology of the M_r can be decomposed into sequential and/or parallel structural patterns, $q_{M_r}(t)$ will be equal to the minimum throughput provided by the physical nodes and links in M_r .

6) ***Cost model:*** We model the embedding cost $C_{M_r}(t)$, ∀t ≥ T0, of mapped forwarding graph $M_r = (\sigma_{r,1}, \dots, \sigma_{r,|M|})$ as follows:

$$C_{M_r}(t) = \left(\sum_{h=1}^{h=|M_r|} C_{\sigma_{r,h}}(t) \right) + |\mathcal{N}_r| \times \nu \qquad (3\text{-}5)$$

where ν is the activation cost per physical node, $|\mathcal{N}_r|$ is the number of activated nodes in $\mathcal{N}_r$, and $C_{\sigma_{r,h}}(t)$is the embedding cost of element $\sigma_{r,h}^{i,j,k} =< f_{r,i},\ n_j,\ \varphi_{k,j} > \in M_r, \forall t \geq T_0,$ given by

$$C_{\sigma_{r,h}}(t) = d_{r,i}(t) \times P_{f_{r,i}} \times \gamma_j + \xi_{i,j} + \sum_{v,z|e_{v,z} \in \varphi_{k,j}} \left(d_{r,i}(t) \times R_{f_{r,i}} \times \Gamma_{v,z} \right), \qquad (3\text{-}6)$$

Where γ_j and $\Gamma_{v,z}$ are the costs of utilizing a unit of CPU and a unit of bandwidth resource from node n_j and link $e_{v,z}$, respectively. Also, $\xi_{i,j}$ denotes the fixed cost of instantiating VNF $f_{r,i}$ at node n_j. We note that the term $d_{r,i}(t) \times P_{f_{r,i}}$ in Eq. (3-6) computes the CPU demand of $f_{r,i}$ from its hosting node n_j at time t, $\forall t \geq T_0$. Also, $d_{r,i}(t)$ and $P_{f_{r,i}}$ denote the entering data rate

Table 3.1 Input Parameters and Variables

System model and formulations parameters	
$\mathcal{N}$	Set of $\lvert\mathcal{N}\rvert$ number of physical nodes
$\mathcal{E}$	Set of $\lvert\mathcal{E}\rvert$ number of physical links
$\mathcal{P}_{p,q}^{max}$	Set of all the paths from node n_p to node n_q with a maximum path length of ϕ^{max} hops
$W_{\text{CPU}}(n_p)$	Processing capacity of the physical node $n_p \in \mathcal{N}$
$W_{\text{BW}}(e_{p,q})$	Bandwidth capacity of the physical link $e_{p,q} \in \mathcal{E}$
q_r^{min}	The minimum acceptable throughput for the VNFR $r \in \mathcal{R}$
$q_{M_r}(t)$	The end-to-end throughput of the mapped forwarding graph M_r at time t
$\mathcal{F}_r$	Set of $\lvert\mathcal{F}_r\rvert$ number of required VNFs for VNFR $r \in \mathcal{R}$
D_r	The dependency graph representing the relations between VNFs of the $\mathcal{F}_\nabla$
$\mathcal{L}_r$	Set of $\lvert\mathcal{L}_r\rvert$ number of directed links in dependency graph D_r
$\zeta(D_r)$	Degree of Freedom (DoF) for dependency graph D_r
$d_i(t)$	The entering data rate to VNF $f_i \in \mathcal{F}(t)$ at time t.
$u_j(t)$	The CPU utilization of node $n_j \in \mathcal{N}$ at time t.
$u_{e_{v,z}}(t)$	The bandwidth utilization of the link $e_{v,z} \in \mathcal{E}$ at time t.
$R_{f_{r,i}}$	The ratio of the outgoing data rate to the incoming data rate of VNF $f_{r,i} \in \mathcal{F}_r$.
$P_{f_{r,i}}$	The Processing resource demand per bandwidth for VNF $f_{r,i} \in \mathcal{F}_r$.
M_r	A mapped forwarding wih $\lvert M_r\rvert$ distinct elements
$\mathcal{N}_r$	The set of physical nodes involved in M_r
$\mathcal{E}_r$	The set of physical links involved in M_r
$\mathcal{N}'_r$	The set of physical nodes that, according to M_r host at least one VNF $f_{r,i} \in F_r$
$\mathcal{E}'_r$	The set of physical links involved in M_r
$\sigma_{r,h}^{i,j,k}$	The h^{th} element of M_r, indicating the placement of $f_{r,i}$ on physical node n_j while using the physical link $\varphi_{k,j}$ for connection.
$\mathcal{K}_{M_r,j}$	The set of VNFs of $\mathcal{F}_r$ that as dictated by M_r are hosted on the physical node n_j.
$\Gamma_{v,z}$	The cost of utilizing a unit of bandwidth in the physical link $e_{v,z}$.
γ_j	The cost of utilizing a unit of CPU in the physical node n_j.
$\xi_{i,j}$	The fixed cost of instantiating VNF $f_{r,i}$ at node n_j
ν	The activation cost per active physical node

and processing resource demand per bandwidth for VNF $f_{r,i}$, respectively. In Eq. (3-6), the term $d_{r,i}(t) \times R_{f_{r,i}}$ accounts for the bandwidth resource required for connecting $f_{r,i}$ to the next VNF in M_r.

3.2.2 Problem Formulation

In the following, we formally formulate the problem under study, while Table 3.1 delineates the important parameters and variables. Following the system model presented above, M_r would be a candidate joint composition and embedding solution for VNFR $r \in \mathcal{R}$. To better explain the structure of a candidate solution, let us consider an examplar solution (i.e. M_r) for the VNFR depicted in Fig. 3.1. Let M_r be as follows:

$$\mathrm{M}_r = [< f_{r,5}, n_2, [e_{1,2}] >, < f_{r,4}, n_4, [e_{2,3}, e_{3,4}] >, < f_{r,4}, n_4, [e_{2,3}, e_{3,4}] >, < f_{r,2}, n_5, [e_{4,5}] >, < f_{r,1}, n_7, [e_{5,7}] >, < f_{r,3}, n_8, [e_{7,8}] >] \quad (3\text{-}7)$$

As such, M_r not only suggests a VNF-FG for VNFR r (i.e., $[f_{r,5}, f_{r,4}, f_{r,2} f_{r,1}, f_{r,3}]$), but it also determines its embedding graph. The latter, for instance, indicates that the first VNF in the suggested VNF-FG, $f_{r,5}$, should be placed on node n_2 and connected to the previous hosting node (i.e., the ingress node n_1) through path $e_{1,2}$ (which here happens to be a single link). The second VNF in the VNF-FG, $f_{r,4}$, should be placed on node n_4 and connected to the previous hosting node n_2 through path $[e_{2,3}, e_{3,4}]$. Similarly, other elements determine the mapping of the composed VNF-FG (i.e. $[f_{r,5}, f_{r,4}, f_{r,2} f_{r,1}, f_{r,3}]$).

To ensure feasibility, a candidate composition and embedding solution M_r should be evaluated for various constraints, as the resources of the substrate network are shared among different flows. It is therefore critical to not only verify whether the selected links/nodes meet the given utilization constraints during the service usage time but also make sure that QoS constraints are not violated. To this end, a candidate solution M_r is *feasible* only if it satisfies the following constraints:

Constraint 1: The throughput of the given solution M_r should meet the required throughput q_r^{min} during the service usage time:

$$q_r^{min} \leq q(M_r, t), \forall t \geq T_0 \quad (3\text{-}8)$$

Constraint 2: The processing demands of VNFs should be smaller than the available processing resources of their hosting nodes in $\mathcal{N}'_r$:

$$\frac{\left(\sum_{\forall(r,i)|f_{r,i}\in\kappa_{M_r,j}} d_{r,i}(t)\times P_{f_i}\right)}{W_{\mathrm{CPU}}(n_j)} \leq \overbrace{1-u_j(t)}^{Avail.\ processing\ ratio} \quad (3\text{-}9)$$
$$\forall t > T_0, \forall n_j \in \mathcal{N}'_r,$$

where the term $d_{r,i}(t) \times P_{f_{r,i}}$ is the CPU demand of $f_{r,i} \in K_{M_r,j}$. The right-hand side of Eq. (3-9) computes the ratio of the available processing resource at the physical node $n_j \in \mathcal{N}'_r$ at time $t \geq T_0$, while the left-hand side is the ratio of the cumulative demanded processing resource of all VNFs hosted by the node n_j to the processing capacity of the node n_j.

Constraint 3: Similar to Constraint 2, this constraint ensures that the bandwidth demands of VNFs in M_r is smaller than the available bandwidth resources of the corresponding embedding links in $\mathcal{E}_r$:

$$\frac{\sum_{\forall(r,i)|f_{r,i}\in\kappa_{M_r,j}} d_{r,i}(t)\times \mathcal{R}_{f_i,i}}{W_{\mathrm{BW}}(e_{v,z})} \leq \overbrace{1-u_{v,z}(t)}^{Avail.\ bandwidth\ ratio} \quad (3\text{-}10)$$
$$\forall t \geq T_0 \forall n_j \in \mathcal{N}'_r, \forall (v,z)|e_{v,z} \in \varphi_{j,k}, n_k, n_j \in \mathcal{N}'_r$$

where the right-hand side of the equation computes the ratio of the available bandwidth resource at the physical link $e_{v,z} \in \varphi_{j,k}$, where $\varphi_{j,k}$ is the path that, according to M_r, connects node n_j to n_k, while the left-hand side is the ratio of cumulative bandwidth demand from the link $e_{v,z}$ to its bandwidth capacity.

With all these considerations in mind, we define our objective function as follows:

$$\begin{matrix} min_{M_r|\forall r\in\mathcal{R}} & \int_{T_0\leq t} C_{M_r}(t)\cdot dt \\ s.t. & Constraints\ 1,2,and\ 3 \end{matrix} \quad (3\text{-}11)$$

which aims to find the least costly mapped forwarding graph for the given VNFR $r \in \mathcal{R}$ while satisfying the QoS and capacity constraints given by Eqs. (3-8), (3-9), and (3-10).

3.3 Deep Reinforcement Learning for Joint VNF-FG Composition and Embedding

The VNF-FG composition and embedding sub-problems have both proven to be NP-hard [72] [73]. Moreover, variations of network resources as well as date rate demands of the incoming

requests further compound this complexity. In the following, we present the main MDP components designed for our joint composition and embedding problem.

3.3.1 System States, Actions, and Reward

Let S represent the set of states. The state is denoted by Eq. (3-12):

$$s_t =< ID_r, M_r, A_r >^t \quad (3\text{-}12)$$

The state contains three main components: (i) ID ID_r of the selected VNFR $r \in \mathcal{R}$, for which we seek to find a joint composition and embedding solution, (ii) constructed mapped forwarding graph M_r for the request specified by ID_r, and (iii) an auxiliary graph A_r indicating the VNFs in F_r, which are not yet chained into the VNF-FG and also not yet mapped to the physical nodes. Moreover, A_r also reflects the dependency relation between the remaining VNFs indicated by D_r. Note that the system state s_t does not include the dynamics of network resource utilization and/or service demands as these dynamic parameters are considered in the system reward computations, to be discussed later on. We denote $\mathcal{A}(s_t)$ as the total set of actions in the state $s_t = < ID_r, M_r, A_r >^t \in S$. The action is denoted by Eq. (3-13):

$$a_t =< f_{select}, n_{select}, \varphi_{select} >^t \in \mathcal{A}(s_t) \quad (3\text{-}13)$$

The action is a 3-tuple containing the following components: (i) selected VNF f_{select}^t from F_r, as the next candidate VNF in the constructed mapped forwarding graph specified by M_r, (ii) selected physical node n_{select}^t for hosting f_{select}^t, and (iii) selected path φ_{select}^t to connect n_{select}^t (i.e., host node of f_{select}^t), to n_{select}^{t-1} (i.e., host node of f_{select}^{t-1}). After selecting action a_t, the system state will transition to s_{t+1}. As such, $a_t =< f_{select}, n_{select}, \varphi_{select} >^t$ will be added to M_r.

Also A_r will be updated by omitting f_{select}^t and all its incoming links. This is done to relax the dependency relation of other VNFs and the selected VNF f_{select}^t. In other words, since the VNF f_{select}^t already exists in M_r and hence the dependency relation is already satisfied its dependent VNFs in F_r could be selected for the next VNFs in future elements of M_r until it is completed and consequently, the VNF-FG is fully composed.

Next, we design our reward function, which needs to take into account the embedding cost and constraints 1, 2, and 3, given by Eqs. (3-8)-(3-10). To that end, we define the reward of selecting the action $a_t \in \mathcal{A}(s_t)$ in the state $s_t \in S$ as follows:

$$R(s_t, a_t) = \begin{cases} -\left(C_{M_r}(t) + \Omega \times |A_r|\right) & Eqs.(3-8)-(3-10) are\ true, \\ -\infty & otherwise, \end{cases} \quad (3\text{-}14)$$

where we assign a negative infinity to the reward if M_r is not feasible; otherwise, the reward considers the cost of M_r along with the penalizing term $\Omega \times |A_r|$, where Ω is the penalty coefficient and $|A_r|$ is the number of remaining VNFs of F_r that have not been mapped yet.

3.3.2 RL and DRL

MDP framework provides the necessary mathematical formalism for applying the Reinforcement Learning (RL) approach, which is a widely used strategy to solve complex problems. In RL, an agent automatically learns the dynamic parameters and updates its decisions through its interactions with the environment. One of the most widely used RL strategies is the so-called Q-learning [20], which successively updates the evaluation of the long-term quality (also known as Q value) of actions at each state. It is known that Q-learning is a simple yet effective way for an agent to learn how to act optimally [20]. However, in most real-world problems with large state/action spaces, Q-learning becomes inefficient as exploring all the states and taking all the possible actions could be impossible [21]. The designed action in our problem includes the selection of a path in a substrate network.

Clearly, the number of paths even in a small network could grow large, thus leading to a large action space. A viable way of learning efficiently in such environments with huge state/action space is to use function approximation for estimating the Q value [42]. The revival hybrid approach of combing Deep Neural Network (DNN) with RL algorithm has proven to be effective, and it is now widely being used in different domains under the so-called deep reinforcement learning (DRL), also known as deep Q-learning (DQL). In this work, to approximate the Q values for unmet states/actions, we use a specialized variant of the deep dueling Q network [20] and enhance it with a *branching* [70] technique to ease the complexity of computing Q values along with *action filtering* [71] technique to reduce the size of the action space. In the following, we present our proposed framework and the customized structure of the utilized Q-network.

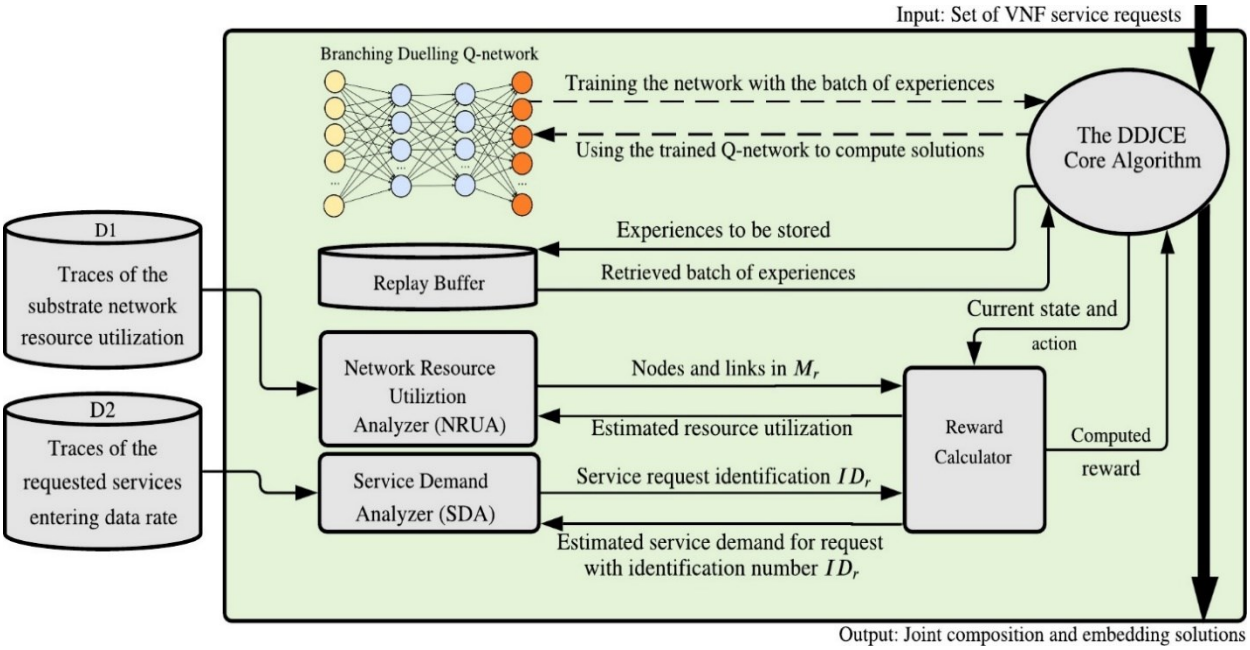


Figure 3.2 High-level view of the proposed framework, its components and their interactions.

3.3.3 Deep Dynamic Joint VNF-FG Composition and Embedding (DDJCE) Framework

Fig. 3.2 illustrates our proposed Deep Dynamic Joint VNF-FG Composition and Embedding (DDJCE) framework. As shown in Fig. 3.2, the "DDJCE Core Algorithm" is the main component of our framework. It receives the set of VNF service requests as the input and computes the joint composition and embedding solutions in the form of mapped forwarding graphs. To this end, through its searching process, the "DDJCE Core Algorithm" constructs our system states and then makes a set of decisions (e.g., the next VNF to be added to the VNF-FG and its embedding node and path) after formulating them as the actions of the system. After constructing the state-action pairs, the "DDJCE Core Algorithm" sends them to the our "Reward Calculator" component, which in turn interacts with two analyzers, namely, Network Resource Utilization Analyzer (NRUA) and Service Demand Analyzer (SDA), to collect the required estimated parameters (e.g., resource utilization and service demand data rate) for calculation of the reward. Once the reward is computed, it will be sent to DDJCE Core Algorithm, where the experience (containing the current state, action, reward, and also the next state) is collected and sent to the replay buffer to be stored. Once the replay buffer is sufficiently filled, the "DDJCE Core Algorithm" extracts random batches of samples, which are then used to train a specialized Branching Duelling Q-network (BDQN). Once the specialized Q-network is trained, the DDJCE core algorithm uses it to obtain the mapped

forwarding graphs for the received requests. In the following, we describe the various components in technically greater detail and then present the structure of our utilized Q-network.

***DDJCE Core Algorithm**:* Our proposed "*DDJCE Core Algorithm*," which is illustrated in Algorithm 1, performs the following tasks: (i) generating the experiences to be stored in the replay buffer, (ii) randomly extracting the experience batches, (iii) training the BDQN network with these batches, and (iv) calculating the mapped forwarding graph solutions using the trained Q-network To be more specific, the algorithm starts with random initialization of the BDQN network (see line 1 of Algorithm 1). Also, we set a memory for storing the experiences (i.e., line 2 of Algorithm 1). All received VNFRs are collected in $\mathcal{R}$ (i.e., line 3 of Algorithm 1). Request index r and state index i are then initialized (i.e., line 4 of Algorithm 1). Next, we select a random request r from $\mathcal{R}$ and initialize the loop index *iter* (i.e., line 7 of Algorithm 1). The loop index '*iter*' is incremented at each step until a predefined maximum number *max_iteration* of iterations (i.e., line 8 of Algorithm 1). We initialize the mapped forwarding graph M_r of request r to an empty list, whereas the auxiliary graph A_r is initialized with a copy of the dependency graph D_r of request r (i.e., line 9 of Algorithm 1). Also, in order to set the starting point to the ingress node of request r, we initialize the variable n^0_{select} to $n_{r,s}$ (i.e., line 10 of Algorithm 1). The state s_{i-1} is also built by making a 3-tuple $< ID_r, A_r, M_r >$. In line 12 of Algorithm 1, we identify VNFs belonging to A_r that has no outgoing arrows (i.e., does not depend on any other VNFs) and collect them in set B_i. Moreover, the physical nodes that are ready to host at least an element of B_i are collected in the variable O_i (i.e., line 13 of Algorithm 1). Similarly, all the possible paths with a maximum length of Φ^{max} that connect n^0_{select} to the request egress node $n_{r,d}$ are collected in the variable $\emptyset_i$ (i.e., line 14 of Algorithm 1). We construct the $\mathcal{A}(s_{i-1})$ of possible actions in the current state as follows (i.e., line 15 of Algorithm 1):

$$\mathcal{A}(s_{i-1}) = B_i \times O_i \times \emptyset.$$

Following an ε-greedy strategy, the algorithm switches between the exploration and exploitation phases (i.e., lines 16-17 of Algorithm 1). If we are in the exploration phase, a random action a_i is selected as follows (i.e., line 18 of Algorithm 1):

$$a_i =< f^i_{select}, n^i_{select}, \phi^i_{select} > \in \mathcal{A}(s_{i-1}).$$

On the other hand, if the algorithm is in the exploitation phase, we select the action that maximizes the Q-value rather than selecting a random one (i.e., line 20 of Algorithm 1). Next, we update A_r by removing f^i_{select} and all its incoming links (i.e., line 22 of Algorithm 1). Moreover, we update M_r by appending a_i to it (i.e., line 23 of Algorithm 1). Next, we calculate the reward using Eq. (3-14) which requires not only the information of the updated M_r and A_r, but also estimations of other parameters (e.g., resource utilization and demand data rate). Thus, the calculation of reward is done by a separate component called "Reward Calculator," which will be explained later on. In line 25 of Algorithm 1, we set the state s_i to $< ID_r, A_r, a_i >$ and then increment the request counter *iter* and state counter i by one. We then create a 4-tuple experience as follows:

Algorithm 1 The DDJCE Core Algorithm

1: Initialize the BDQN network
2: Initialize a replay buffer
3: Initialize $\mathcal{R}$ as the set of all VNFRS.
4: $r = \{\}, i = 1$
5: **while** $\mathcal{R}$ not empty **do**
6: Update $\mathcal{R}$ by excluding r
7: Select a random request $r \in \mathcal{R}$, initialize $iter = 1$
8: **while** $iter <$ max_iteration **do**
9: Set $M_r = [\,], A_r = D_r$
10: Set $n^0_{select} = n_{r,s}, s_{i-1} =< ID_r, A_r, M_r >$
11: **while** A_r not empty **do**
12: Set $B_i = \{f_{r,j} | f_{r,j} \in F_r$ and $f_{r,j}$ has no outgoing arrows in $A_r\}$
13: Set $O_i = \{n_p | n_p \in \mathcal{N}$ and n_p can host at least a member of $B_i\}$
14: Set $\emptyset_i = \{\phi^{max}_{p,q} | \phi^{max}_{p,q} \in \phi_{p,q},\ n_p = n^{|M^r|}_{select},\ n_q = n_{r,d}$, and $\phi^{max}_{p,q}$ consists of maximum ϕ^{max} hops $\}$.
15: Construct the action space $\mathcal{A}(s_{i-1}) = \{B_i \times O_i \times \emptyset_i\}$
16: Generate a random number $\lambda \in [0, 1]$
17: **if** $(\lambda < \varepsilon)$ **then** ▷ Exploration phase
18: Select a random action $a_i =< f^i_{select}, n^i_{select}, \phi^i_{select} > \in \mathcal{A}(s_{i-1})$
19: **else** ▷ Exploitation phase
20: Choose a_t which maximizes $Q^{Select}(s_i, a_i; \theta_i)$
21: **end if**
22: Update A_r by deleting f^i_{select} and all its incoming arrows.
23: Update M_r by appending a_i.
24: Send M_r and A_r to the reward computation component and receive $\mathbb{R}(s_{i-1}, a_i)$.
25: Set $s_i =< ID_r, A_r, M_r >, iter = iter + 1, i = i + 1$
26: Store $< s_{i-1}, a_i, \mathbb{R}(s_i, a_i), s_i >$ as experience in replay buffer.
27: **if** enough experiences stored in replay buffer **then**
28: Extract a batch of experiences and use it to train Q network
29: **end if**
30: **end while**
31: **end while**
32: **end while**

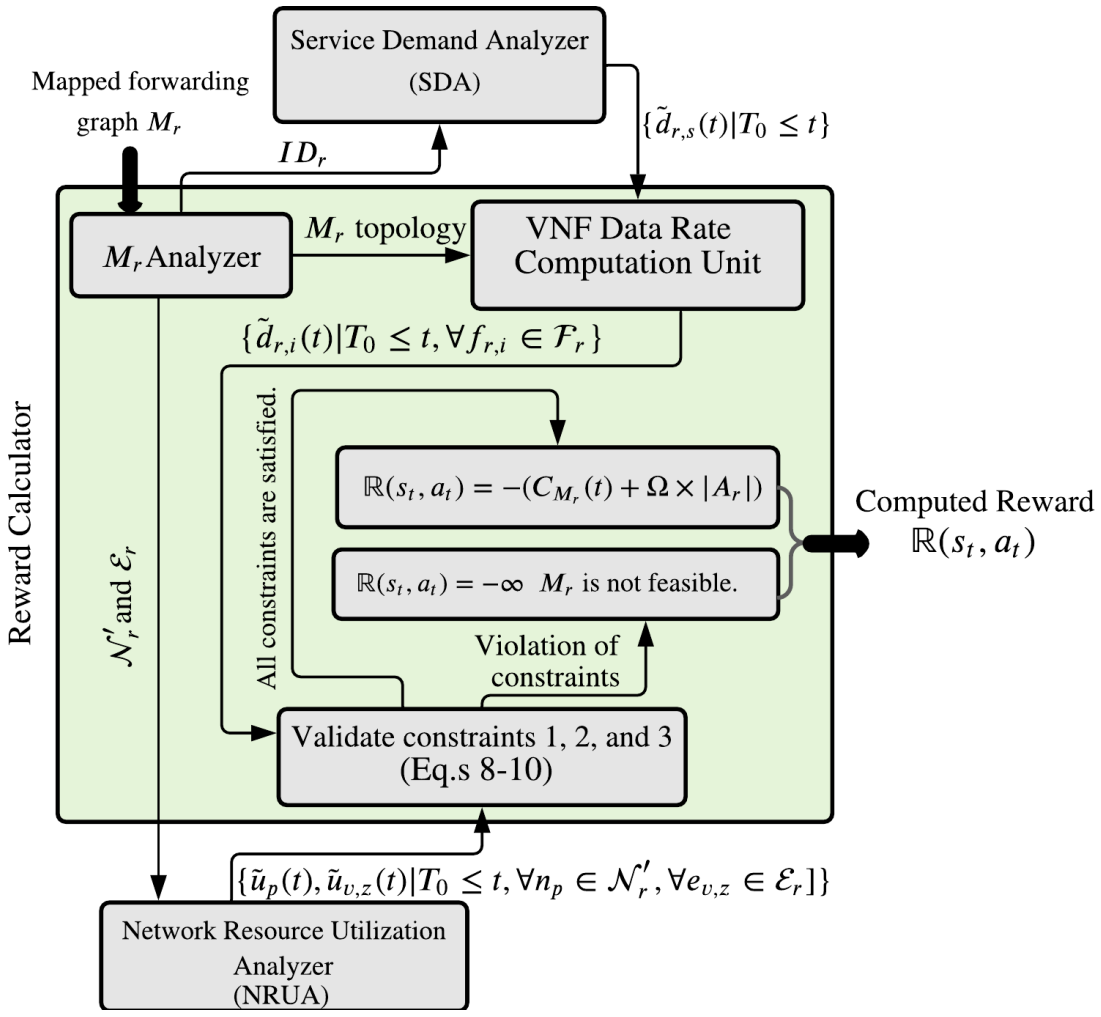


Figure 3.3 Overview of our Reward Calculator component and its interactions with SDA and NRUA.

$$< s_{i-1}, a_i, R(s_i, a_i), s_i > \qquad (3\text{-}15)$$

which is sent to the replay buffer to be stored (see line 26 of Algorithm 1). Next, the algorithm checks whether the replay buffer contains enough experiences (i.e., line 27 of Algorithm 1). If it is filled with enough experiences, a random batch of size β will be extracted and will be sent to the specialized Q-network for training (i.e., line 28 of Algorithm 1). This process is repeated until A_r becomes empty, which indicates that all the VNFs and their dependency links are considered in M_r and a candidate solution for request r is obtained. The algorithm continues to generate further solutions for the same request (i.e., lines 8-31 of Algorithm 1) until the number of iterations reaches a predefined threshold, when we move on to the next request (i.e., lines 5-32 of Algorithm 1). This process is repeated until $\mathcal{R}$ becomes empty, meaning that all the given requests have been considered for training our specialized Q-network, thus making it ready to generate desired joint composition and embedding solutions.

Reward Calculator: Fig. 3.3 illustrates an overview of our Reward Calculator component. In our design, reward represents the short-term consequence of a selected action in a given state. To that end, our reward calculation relies on evaluating the mapped forwarding graph M_r and the auxiliary graph A_r , which are updated after selecting a_t in s_t (see line 24 of Algorithm 1). After receiving the updated mapped forwarding graph from DDJCE Core Algorithm, the "M_r Analyzer" inside Reward Calculator identifies the used physical nodes and links (i.e., $\mathcal{N}'_r$ and ε_r) as well as the topology of M_r. The identified physical resources (i.e., $\mathcal{N}'_r$ and ε_r) will be sent to the Network Resource Utilization Analyzer (NRUA), which estimates their utilization values for the usage period $t \geq T_0$. Moreover, the corresponding request identifier ID_r is sent to Service Demand Analyzer (SDA), which

estimates the corresponding initial entering data rate $\check{d}_{r,s}(t)$ for the usage period $t \geq T_0$. Having the topology of M_r, the estimated initial entering data rate $\check{d}_{r,s}(t)$ of service request r as well as the ratio $R_{f_{r,i}}$ of outgoing data rate to incoming data rate of VNFs in F_r (available from VNFR specification), our VNF Data Rate Computation Unit computes the amount of data rate that will enter each VNF at time $t \geq T_0$. Specifically, the output of the VNF Data Rate Computation Unit is given by:

$$\{\tilde{d}_{r,i}(t) \mid T_0 \leq t, \forall f_{r,i} \in F_r\} \tag{3-16}$$

where $\check{d}_{r,i}(t)$ is the estimated data rate entering the VNF $f_{r,i} \in F_r$ in M_r. The estimated data rate entering each VNF $f_{r,i} \in F_r$ in the VNF-FG can be obtained by multiplying the estimated data rate $\check{d}_{r,i-1}(t)$ of the previous VNF by the ratio $R_{f_{r,i}}$ of outgoing to the incoming data rate of the selected VNF. Receiving the estimated values, the next step for computing the reward value is to validate the constraints 1, 2, and 3 given by Eq. (3-8)-(3-10) (see Fig. 3.3). Once the constraints are validated, the reward is obtained using Eq. (3-14) and sent to DDJCE Core Algorithm for further processing (see Algorithm 1). In the following, we describe NRUA and SDA in detail.

As shown in Figs. 3.2 and 3.3, the Reward Calculator interacts with two components, NRUA and SDA. NRUA is responsible for estimating the resource availability of the nodes and links identified by the Reward Calculator for the service usage time period $t < T_0$. Letting $\mathcal{N}'_r$ and $\mathcal{E}_r$ to denote the sets of physical nodes and links identified by the Reward Calculator, the output of NRUA is as follows:

$$\{\tilde{u}_p(t), \tilde{u}_{v,z}(t) | T_0 \leq t, \forall n_p \in \mathcal{N}'_r, \forall e_{v,z} \in \mathcal{E}_r\} \quad (3\text{-}17)$$

where $\tilde{u}_p(t)$ and $\tilde{u}_{v,z}(t)$ are the estimated resource utilization of node $n_p \in \mathcal{N}'_r$ and physical link $e_{v,z} \in \mathcal{E}_r$, for the service usage time period $t \geq T_0$, respectively. The estimation in NRUA is carried out by tracking and analyzing the traces available in dataset D1, which contains the past observations of resource utilization of physical nodes and links for the pre-service usage time period $t < T_0$. Formally stated, D1 contains the followings:

$$D1 \ni \{u_p(t), u_{v,z}(t) | t < T_0, \forall n_p \in \mathcal{N}', \forall e_{v,z} \in \mathcal{E}\} \quad (3\text{-}18)$$

Similarly, SDA estimates the initial data rate of a service request specified by the Reward Calculator. Specifically, the output of SDA is as follows:

$$\{\tilde{d}_{r,s}(t) \mid T_0 \leq t\} \quad (3\text{-}19)$$

where $\tilde{d}_{r,s}(t)$ is the estimated initial entering data rate of request r, which is carried out by tracking and analyzing the traces available in dataset D2. Similar to D1, D2 contains the past observations of the requested services' initial entering data rate for the pre-service usage time period of $t < T_0$. Formally stated, D2 contains the followings:

$$D2 \ni \{d_{r,s}(t) | \ t < T_0, \forall r \in \mathcal{R}\} \quad (3\text{-}20)$$

3.3.4 Branching Dueling Q network with Action Filtering

Recall that a single action a_t in our design contains three dimensions $f^t_{select}, n^t_{select}, and\ \varphi^t_{select}$. This makes our problem belong to the domain of *multi-discrete* action space problems [70]. It is important to note that a DQN may have a large number of neurons in its output layer. This, as a

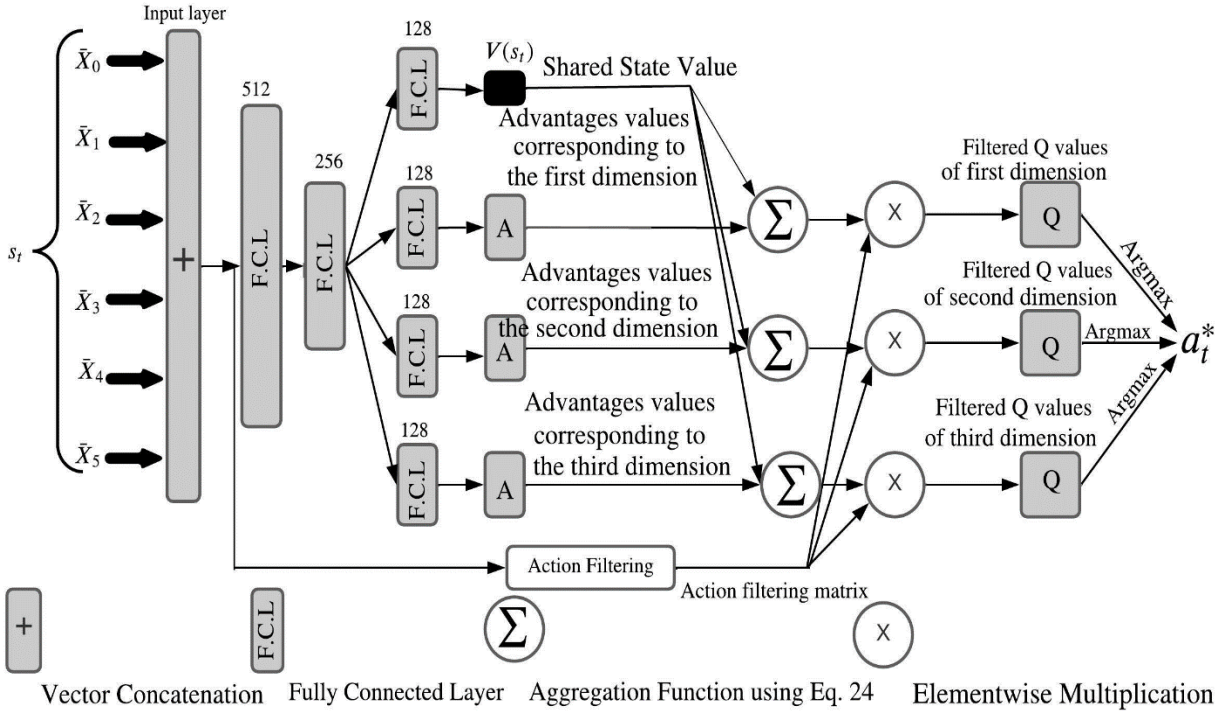


Figure 3.4 Architecture of the utilized BDQN enhanced with action filtering mechanism. The number of neurons of each layer is shown on top of the layer.

result, requires a huge amount of memory, thus leading to inefficient training [70]. Moreover, such a simple structure is prone to overestimation and an unstable training process [70]. In the following, after discussing these issues in more detail, we present our solution to overcome them. Typically, the deep neural network in the DQL algorithm is responsible for estimating the Q value Q(s, a) of a given state and action using a set of parameters θ. Using the well-known Bellman equation [20], the parameters of the Q-network are updated using the gradient descant update rule formulated as follows:

$$\theta_{t+1} = \theta_t + \alpha\left(Y_t^Q - Q(s_t, a_t; \theta_t)\right)\nabla_{\theta_t} Q(s_t, a_t; \theta_t) \qquad (3\text{-}21)$$

Where

$$Y_t^Q = R_{t+1} + \gamma \max 1_a\, Q(s_{t+1}, a; \theta_t) \qquad (3\text{-}22)$$

which represents the target value in a conventional DQN algorithm. In Eq. 22, R_{t+1} is the immediate reward for taking action a_t in state s_t and α is the gradient step size. According to Eq.s 21 and 22, the parameters of the Q-network are updated using a target value, which is computed using the same set of parameters θ_t, causing an oscillated training, a problem known as moving

Q-targets, which can lead to overestimation [20]. To tackle this, the widely used strategy proposed by [21] is to use two Q-networks simultaneously: (1) an online network: a network with parameters θ and (2) a target network with parameter θ' where the two parameters are switched symmetrically after passing the predefined steps in training. Accordingly, Eq. (3-22) is updated as follows:

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a; \theta_t); \theta_t') \qquad (3\text{-}23)$$

It is proven that the method of using two Q-networks, also known as *Double DQN (DDQN)*, leads to better performance compared to conventional DQN methods [20]. However, it is still inapplicable to our framework, as our problem has a large discrete multi-dimensional action space, which makes it difficult to explore with conventional DQN or even DDQN networks [21]. To overcome this issue, we adopt a Branching Dueling Q Network (BDQN) [70], which can be viewed as a variant of the Dueling Q-network [81]. The key idea behind this is that for large action space problems, the Q value can be estimated without redundant computations for low-/similar-valued actions in a given state. For example, in our problem, all the actions that suggest using the already placed VNFs as the next element in VNF-FG can be accounted as such low- and similar-valued actions. To this end, the DQN structure is modified into an architecture with separate streams, each stream corresponding to an action dimension as well as a specialized stream for sharing an estimated (scalar) representation of state value denoted by $V(s_t)$. For instance, consider the multi-dimensional action $a_t =< a_1, \dots, a_d, \dots, a_N >$ with N dimensions, where each action dimension a_d can have n discrete values $a_{d,1}, \dots, a_{d,n}$. According to [70], instead of computing Q values for each action, we can use Q values of each dimension (i.e., $Q_d(s_t, a_d)$), which can be obtained as follows:

$$Q_d(s_t, a_d) = V(s_t) + \left(A_d(s_t, a_d) - \frac{1}{n} \sum_{i=1,\dots,n} A_d(s_t, a_{d,i}) \right) \qquad (3\text{-}24)$$

Where $A_d(s_t, a_d)$ is the advantage function representing the state-dependent value of the dimension a_d in s_t, which is a concept borrowed from the Dueling DQN architecture [70]. The value of $V(s_t)$ and $A_d(s_t, a_d)$ are tuned simultaneously in the training process of BDQN.

The architecture of our proposed Q-network is shown in Fig. 3.4, where we further enhance the BDQN architecture with an action space reduction technique called *action filtering* [71]. First, the action filtering component decodes the current state s_t to reconstruct its building components $ID_r, A_r, and\ M_r$. Next, each component is investigated, and the set of invalid actions is identified.

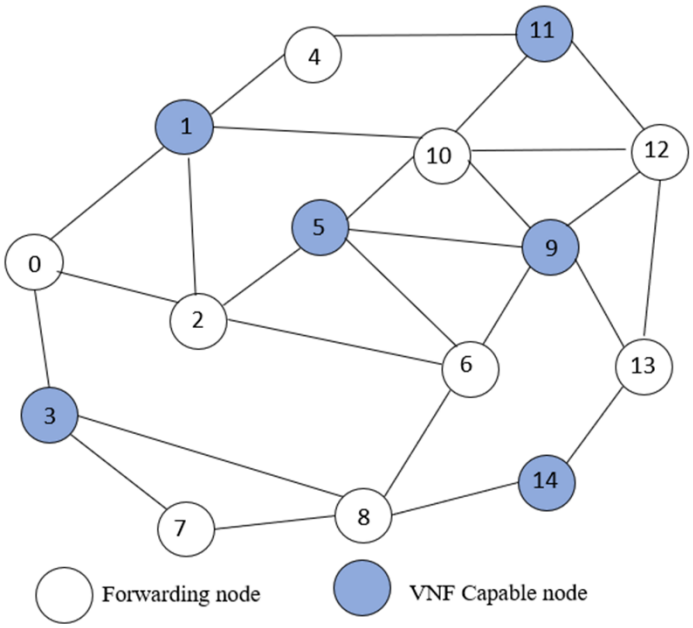


Figure 3.5 Substrate network topology comprised of forwarding and VNF-capable nodes and bidirectional links [74] [78]

Then, a corresponding binary action filtering matrix $\mathbb{B}_{s_t}$of size d × n (d is the number of dimensions and n being the number of possible actions in each dimension) is generated. We set $b_{i,j}$ of $\mathbb{B}_{s_t}$ to zero, if the corresponding action is invalid; otherwise, it is set to one. This matrix will be injected into the network to explicitly modify the Q values corresponding to these actions. For instance, consider $\sigma_{r,|M_r|} =< f_{r,3}, n_8, [e_{7,8}] >$ as the last element of the M_r (given by Eq. (3-7)) and $\varphi_i^t = [e_{5,3}, e_{3,2}]$ as a candidate path. Clearly, any action that contains φ_i^t in its third dimension should be marked invalid because φ_i^tdoes not originate from the node n_8 and cannot embed the connecting link between $f_{r,3}$ and the next selected VNF of M_r. Accordingly, the element $b_{3,i}$of matrix $\mathbb{B}_{s_t}$will be set to be zero, indicating that the corresponding action is invalid.

3.4 Performance Evaluation

In this section, we evaluate our proposed algorithm. Our DDJCE core algorithm and its interactions with the BDQN network are implemented in the OpenAI Gym toolkit [75] via a customized environment. The simulation scenarios are con- ducted on Google Cloud Platform using a VM instance with 10 vCPUs and 37 GB memory. As for the BDQN, we use Tensorflow 2.5.0 [76] to instantiate the network. Moreover, to ensure convergence and model efficiency, we

utilize Keras Tuner Framework [77] to automate the search for tuning the network hyper-parameters.

3.4.1 Simulation Settings

We consider the substrate network topology shown in Fig. 3.5, which has widely been used for performance evaluation in closely related works, e.g., [74], [78]. Accordingly, in our different scenarios, we add/drop links/nodes to scale up/down the network when needed. Moreover, the CPU and bandwidth capacities of nodes and links are set randomly in the range [1-5] GHz, and [100 Mbps, 1 Gbps]. As for the resource requests and nodes utilization, to ensure a realistic scenario, we use the recently published Google cluster trace data set [79][80] (published in 2020), available with the platform Google BigQuery, which is an extension of the previously published dataset from 2011. Detailed statistic analysis for this data set is accessible from [80]. Moreover, the embedding costs related parameters ζ, γ, Γ, and ξ are set to 10, 5, 5, and 1000 (all in unit of currency), respectively. We consider the same parameter settings for all the following simulation scenarios unless otherwise stated.

3.4.2 Optimality Gap

Since finding the optimal solution for the joint composition and embedding problem is time-consuming, we scale down the network shown in Fig. 3.5 into a small network with only nine nodes and 26 links and perform an exhaustive search to find the optimal joint composition and embedding for just a single service request of 4 VNFs with a dependency graph that has 46 degrees of freedom. We used an exhaustive search approach to find the optimum solution to the cost minimization problem formulated by Eq. (3-5) while satisfying Constraints 1, 2, and 3. To evaluate the optimality gap of our proposed framework along with the impact of training on the results, we conduct the comparison by collecting the solutions found by our framework as well as their corresponding embedding costs every 500 episodes using Eq. (3-5).

Fig. 3.6 shows the optimality gap measured by the absolute relative difference (in percentage) between the embedding cost achieved by our DDJCE framework and that of the optimal solution. We observe from Fig. 3.6 that the optimality gap is significant during the initial 500 episodes. Nevertheless, as the model keep training, the optimality gap decreases significantly. More

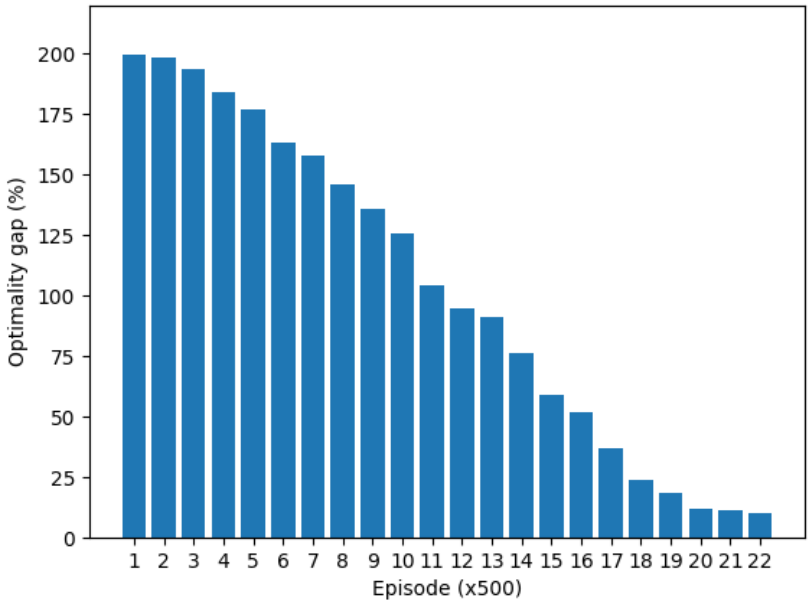


Figure 3.6 Optimality gap (in percentage) of our proposed DDJCE framework vs. the number of episodes.

specifically, starting from an optimality gap of 200%, our proposed DDJCE framework reduces the embedding cost down to 10.63% after it is sufficiently trained. An important aspect to note here is that it takes more than 5 hours for the exhaustive search to find the optimal solution in such a small scenario, which deals with only a single service request with only four requested VNFs, while our framework requires 2 hours of training to reduce the optimality gap by 90%. Clearly, considering multiple and more complex requests would compound the complexity, thus prolonging the search process of the exhaustive search approach even more.

3.4.3 Convergence and Performance Comparison with other Deep Learning Methods

In the following, we investigate the convergence behavior of our proposed DDJCE framework against two other Deep Q- learning-based methods with different Q-network structures:

(i) DDQN [21] consists of two conventional Q-networks and (2) PBDQN [70], a Plain Branching Dualing Q-network without any action filtering. While the DDQN is the most widely used improvement version of DQN, the PBDQN is designed to improve the performance of DDQN even further in environments with large discrete action spaces. To tune the hyper-parameters automatically (including the learning rate), we have used Keras Tunner for all the three Q-

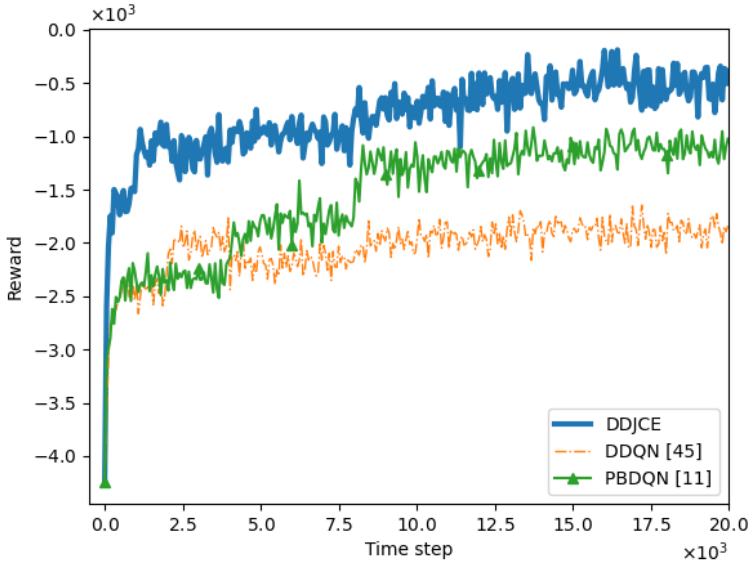


Figure 3.7 Reward vs. time step for different methods.

networks: DDJCE, DDQN, and PBDQN. Moreover, they all receive the same batch size (30 samples in each batch) and use the same technique to draw samples from their individual experience replay buffers. For this scenario, we use the standard substrate network shown in Fig. 3.5 as the DDQN could not converge in the scaled version of this network.

Fig. 3.7 illustrates the reward vs. time step for different methods. We observe from this figure that the DDQN and PBDQN methods perform closely during the initial episodes, but they eventually converge to different values. Also, despite the momentary superiority of DDQN (around time steps 2500 to 5000), the PBDQN shortly surpasses the DDQN method, converging to a higher reward. The extra training required for constructing a shared representation of states could be the reason for such behavior during the initial time steps. This emphasizes the importance of training for PBDQN, which continues to evolve into exploring even more rewarding solutions, as opposed to DDQN, which seems to be trapped in local optima shortly after 5000-time steps.

According to Fig. 3.7, the behavior of our proposed DDJCE is significantly different from the DDQN and PBDQN methods, as it achieves a much higher reward within the initial 100 steps and continues to evolve into even higher rewards. We believe that our deployed action filtering technique is the key reason behind such superior performance of our proposed DDJCE framework. This is because by decreasing the number of actions in each state, the search space is reduced as well, enabling a more efficient search process toward better solutions.

3.4.4 Performance Comparison with Joint and Disjoint Composition and Embedding Heuristics

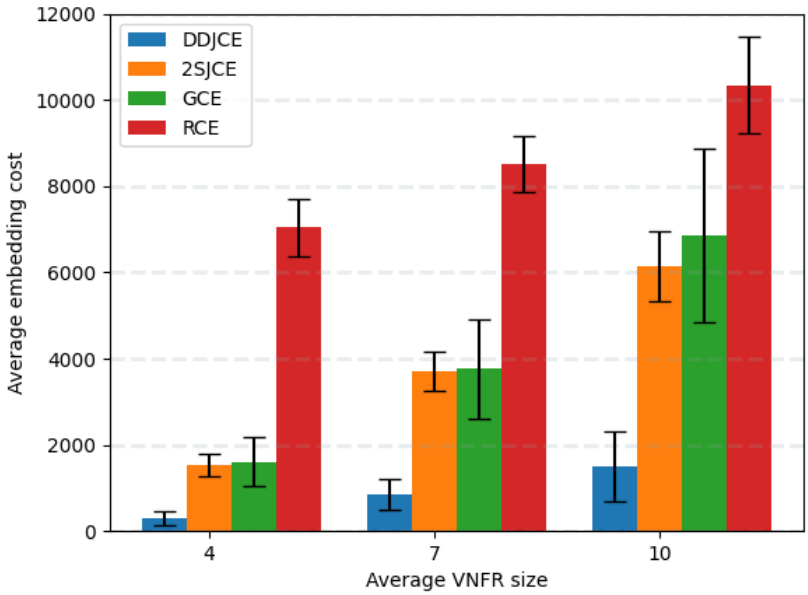


Figure 3.8 Average embedding cost vs. average VNFR size for four different methods.

Next, we evaluate the performance of our proposed DDJCE framework against three benchmarks, namely, RCE [82], GCE [83], and 2SJCE [84]. While the RCE and GCE methods are widely used disjoint approaches based on random and greedy heuristics, respectively, the 2SJCE method relies on a two-stage joint embedding and composition approach. However, it does not consider the variations of resource utilization and/or service demands and solely focuses on stationary values of those parameters at the arrival time of VNFRs. While RCE chooses random solutions from different possible VNF-FG embeddings, GCE takes the VNF requests along with their VNF-FGs as the input and then searches for less costly embedding solutions. The 2SJCE method follows a two-stage search procedure to obtain both the composition and embedding solutions for each VNFR. For this scenario, we use the standard substrate network shown in Fig. 3.5 and run our simulations for different values of VNFR size (i.e., the average number of VNFs in each VNFR) from 4 to 10. Fig. 3.8 illustrates the average embedding cost vs. VNFR size for different methods plotted with their corresponding 95% confidence intervals. As shown in Fig. 3.8, the average embedding cost of the RCE method, which relies on a random composition and embedding, is always significantly larger than other methods. This is because the only criteria

considered in RCE are the feasibility of the resultant VNF- FG composition and embedding solution. On the other hand, the average embedding cost of the GCE method is smaller compared to that of the RCE method. However, we observe from Fig. 3.8 that the GCE method is associated with wide error bars, which indicate that the embedding cost varies strongly across different VNFRs. Specifically, while GCE may be able to find low-cost embedding solutions for a few VNFRs, it fails to achieve such high-quality solutions for the others. Interestingly, we observe that the minimum cost achieved by the GCE method is even smaller than that of DDJCE, though the embedding costs of other requests are so large that on average GCE performs inferior in comparison with our proposed method. This happens because our proposed DDJCE method aims to obtain low-cost solutions while considering all the requests simultaneously, whereas the GCE (and also RCE) investigate the requests sequentially, without considering the consequences of the embedding outcome of a solution on those of others. On the other hand, as it can be observed in Fig. 3.8, the 2SJCE method, which is a joint approach, performs similar to GCE, as it fails to consider the fluctuations of dynamic parameters (i.e., resource utilization and service demand) and their impact on embedding cost. Moreover, given that the 2SJCE method cannot estimate the network condition, it fails to take advantage of the soon-to-be freed overloaded resources for VNF-FG embedding.

3.4.5 Impact of VNF Dependency

In this scenario, we evaluate the impact of VNF dependency (measured in terms of average degree-of-freedom (DoF)) on the performance of our proposed DDJCE method in comparison with RCE, GCE, and 2SJCE benchmarks. Recall that a small value of DoF indicates a strong dependency between VNFs of a request, whereas a high value shows a loose dependency between them (see also Eq. (3-1)). Average embedding cost vs. DoF for the different methods under consideration is shown in Fig. 3.9. There we observe that the average embedding cost of our proposed DDJCE method decreases as the average DoF increases. The 2SJCE method also shows the same kind of behavior, as it is able to decrease the average cost for an increasing DoF. This is because by increasing DoF, the number of possible solutions for VNF-FG composition increases, which in turn increases the chance of finding a VNF-FG that may lead to a less costly embedding solution.

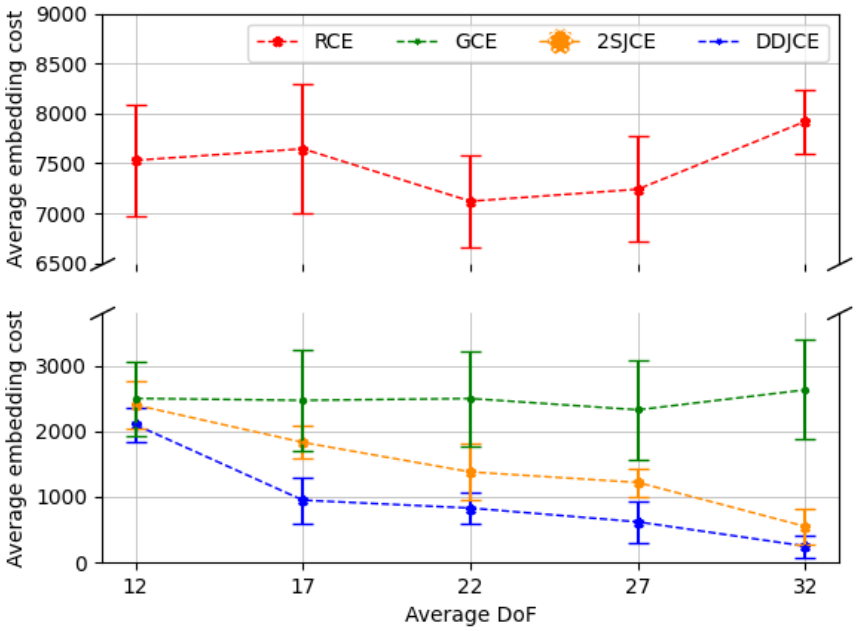


Figure 3.9 Average embedding cost vs. average DoF for four different methods.

Unlike DDJCE and 2SJCE method, we observe from Fig. 3.9 that increasing the DoF does not have any meaningful impact on the performance of RCE and GCE methods, as they do not have any particular strategy to explore through different VNF-FGs to decrease the embedding cost.

3.4.6 Scalability

In the following, we explore the scalability of our proposed DDJCE method. To do so, we increase the size of the standard substrate network shown in Fig 3.5 to contain as many as 20 nodes and 88 links. Also, we consider a batch of 10 service requests, where the average number of VNFs for each request is increased to 10. Under these conditions, we evaluate the cost improvement (in percentage) of our proposed DDJCE framework with respect to the RCE, GCE, and 2SJCE methods. Fig. 3.10 shows the cost improvement vs. the number of nodes. As shown in Fig. 3.10, even though there is a slight drop in the improvement of our proposed DDJCE method with respect to RCE method when the number of nodes increases from 7 to 10 (which may be due to the chaotic behavior of the RCE method), the cost improvement gradually increases up to 95%. In addition, we observe significant improvement (of 75%) with respect to the 2SJCE method, indicating the beneficial impact of estimating the dynamic parameters, which is deployed in our proposed DDJCE method. Such improvements demonstrate that the DDJCE framework is not only

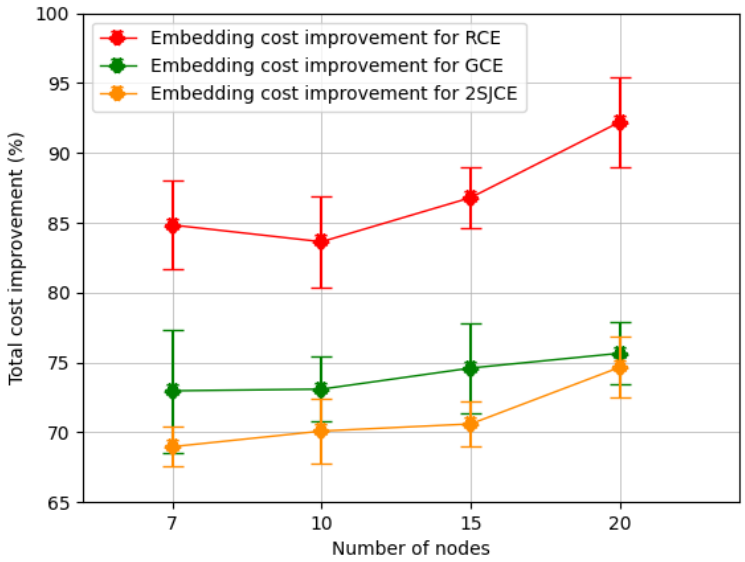


Figure 3.10 Total embedding cost improvement (in percentage) of our proposed DDJCE method vs. the number of nodes, with respect to other methods. The shaded region shows the values obtained in different runs of our proposed DDJCE method.

applicable in scaled scenarios but also becomes more effective in larger networks where more embedding opportunities for a given VNF-FG are held. We believe that the reason for this superiority is rooted in the sophisticated structure of the neural network that is used as a function approximator in DDJCE framework, which automatically explores the large search space of the given (scaled) scenario. We note that since the cost improvement achieved by our proposed DDJCE method slightly varies in different runs of the algorithm, we reflect these variations via the shaded regions shown in Fig. 3.10.

3.5 Conclusion

In this chapter, we have proposed a deep reinforcement learning-based joint VNF-FG composition and embedding framework for the deployment of the next generation of CDNs. We evaluate the performance of our proposed framework with different structures for Q-networks and also conduct evaluations against the widely used joint and disjoint composition and embedding methods. Compared to the benchmarks, the proposed framework improves the embedding cost up to a 95%. Moreover, using the optimal solution found by the exhaustive search as the basis, we investigate the optimality gap of our framework.

Chapter 4

4. Joint VNF-FG Function Scaling and Topology Adaptation for CDN Deployment[1]

4.1 Introduction

After the initial deployment, to remain efficient, a CDN needs to be adapted, taking into account the changing factors in its ecosystem. Moreover, considering the frequent changes in user demand, application requirements, and traffic conditions, the performance degradation of an already deployed content delivery service could be quite probable. In this regard, the initial VNF-FG of the content delivery service and the service embedding map need to be modified and scaled, if needed.

In this chapter, we propose a joint function scaling and topology adaptation method, which supports not only the horizontal scaling but also VNF reordering and connectivity changes in a given VNF-FG. Given that the VNF-FG composition and embedding problems have been proved

[1] This chapter is based on a submitted paper:

- Sepideh Malektaji, Amin Ebrahimzadeh, Halima Elbiaze, and Roch Glitho, *"Joint VNF-FG Function Scaling and Topology Adaptation using Deep Reinforcement Learning"* submitted to IEEE Transactions on Emerging Topics In Computing .

to be NP-hard [72][73], the complexity of the problem at hand (which can be considered as VNF-FG re-composition and re-embedding) is also NP-hard. To tackle this complexity, we formulate the problem as a Markov Decision Process (MDP) and solve it using our proposed Reinforcement Learning (RL)-based framework, which relies on a Q-Learning approach [20]. Our proposed framework aims to jointly execute function scaling and topology adaptation on the given VNF-FG. To cope with the huge discrete multi-dimensional action space, we utilize a Deep Double Q Network (DQN) and further enhance it with an action filtering mechanism [71]. This step is instrumental in reducing the action space, thus helping explore the problem search space more efficiently. Given the QoS threshold, our proposed framework identifies the necessary modifications of the original VNF-FG and determines a proper embedding that minimizes the re-embedding cost. We evaluated the performance of our proposed framework against different network architectures and conducted performance evaluations comparing with both joint and disjoint benchmarks. The results show that our proposed method achieves up to a 93% cost improvement compared to the benchmarks.

The rest of this chapter is organized as follows. First, it presents the system model, followed by the formulation of the targetted problem. Then, it discusses the proposed DRL-based joint function scaling and topology adaptation. After that, it presents the simulation parameters and settings, followed by the validation results. We will conclude this chapter at the end.

4.2 System Model

1) Substrate network: $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ and $\mathcal{E}$ are the sets of $|\mathcal{N}|$ physical nodes and $|\mathcal{E}|$ directed links, respectively. Each $e_{p,q} \in \mathcal{E}$ represents the direct physical link connecting physical nodes n_p and n_q, $\forall n_p, n_q \in \mathcal{N}$. Also, we let the set $\mathcal{P}_{p,q}^{max}$ contain all the paths such as $\varphi_{p,q}$ connecting the node n_p to n_q with a maximum path length of φ^{max} hops. Each node $n_q \in \mathcal{N}$ has a processing capacity $W_{\text{CPU}}(n_p)$. Similarly, each link $e_{p,q} \in \mathcal{E}$ has a bandwidth capacity $W_{\text{BW}}(_{p,q})$.

2) VNF service request (VNFR): We assume that a VNF request is attributed to the following parameters [26]:

1) Ingress node $n_s \in \mathcal{N}$ and egress node $n_d \in \mathcal{N}$, which are the physical nodes from which the first VNF of the requested VNF-FG originates and at which the last VNF terminates, respectively.

2) Set $F(t) = \{f_1, f_2, \dots, f_{|F(t)|}\}$ of $|F(t)|$ number of required VNFs at time t, along with their dependency graph D, which is an acyclic directed graph representing the dependency relations between VNFs. Note that in our formulation, all instances, regardless of their VNF types, are indexed individually so that two different VNF instances could have similar types (e.g., load balancer, firewall, etc.)

3) The entering data rate to the ingress node (i.e., n_s) denoted by $d_s(t)$. Correspondingly, $d_i(t)$ denotes the entering data rate to VNF $f_i \in F(t)$ at time t.

4) Ratio R_{f_i} of outgoing data rate to incoming data rate of VNF $f_i \in F(t)$.

5) Processing resource demand P_{f_i} per bandwidth for VNF $f_i \in F(t)$.

We assume that all the service request parameters explained above remain unchanged except for the entering data rate $d_s(t)$, and the set of required VNFs $F(t)$, which may vary over time due to an increase in service demand and function scaling, respectively.

3) ***Mapped forwarding graph****:* Let the mapped forwarding graph $M(t)$be an ordered list of $|M(t)|$ distinct elements at time t. Each element $< \varphi_{k,j},\ f_i,\ n_j,\ \varphi_{j,p} >^{t}$ is a 4-tuple containing four components $\varphi_{k,j},\ f_i, n_j,$ and $\varphi_{j,p}$, where $\varphi_{k,j} \in \mathcal{P}_{k,j}^{max}, f_i \in F(t), n_j \in \mathcal{N},$ and $\varphi_{j,p} \in \mathcal{P}_{j,p}^{max}$. As such, h^{th} element of $M(t)$, which is $\sigma_{h,k,j,i,p}^{t} =< \varphi_{k,j},\ f_i,\ n_j,\ \varphi_{j,p} >^{t}$, indicates that at time t, the VNF $f_i \in F(t)$ should be embedded in the physical node $n_j \in \mathcal{N}$ and connects to the previously embedded VNF, through the embedding path $\varphi_{k,j}$. Moreover, the embedded VNF should be connected to the next embedding node through the path $\varphi_{j,p}$. To sum up, each element of the $M(t)$ specifies not only the hosting node for each VNF, but also its connections (i.e., entering and outgoing paths) to the previous and next embedded VNFs. This structure, as a result, allows formulating VNF-FGs with complex topologies. Moreover, we define $\mathcal{N}_M(t)$ and $\mathcal{E}_M(t)$ as the sets of nodes and links involved in the structure of $M(t)$, respectively.

4) Resource utilization: Resources of physical nodes and links are shared among different service flows. Given that the resource consumption patterns of these flows are dynamic, the available

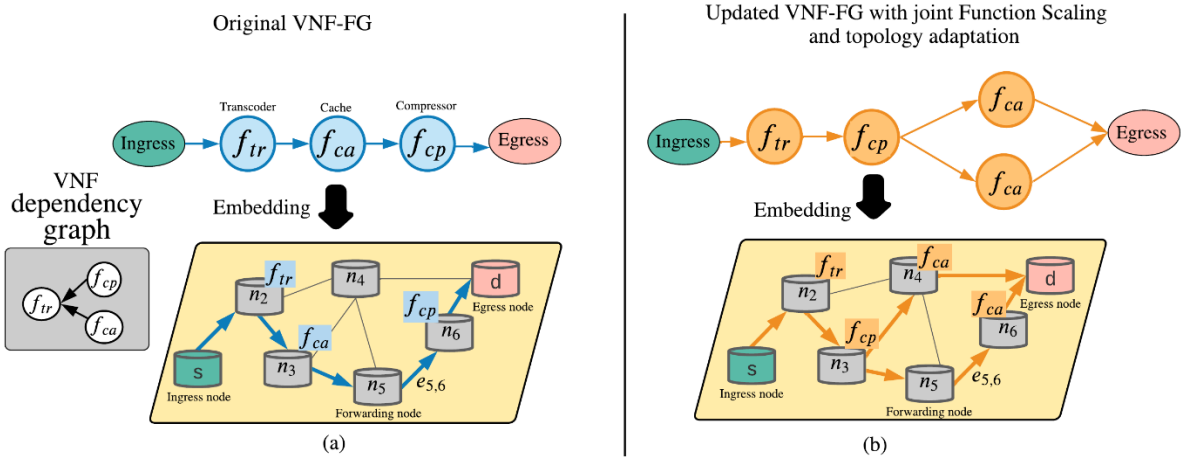


Figure 4.1 (a) Original VNF-FG and its embedding to the substrate network, (b) joint function scaling and topology adaptation techniques.

resources on the substrate nodes and links will change frequently. Let $u_j(t)$and $u_{v,z}(t)$ represent the CPU and bandwidth utilization of node $n_j \in \mathcal{N}$ and link $e_{v,z} \in \mathcal{E}$ in time t, respectively.

5) QoS model: Considering service throughput as our QoS parameter, let q^{min} denote the minimum acceptable throughput, which is defined as the minimum acceptable amount of outgoing data rate of the service request from the egress node $n_d \in \mathcal{N}$. Let $q_M(t)$ estimate the end-to-end throughput of the mapped forwarding graph $M(t)$ at time t. Since the graph topology of the $M(t)$ can be decomposed into sequential and/or parallel structural patterns, $q_M(t)$ will be equal to the minimum throughput provided by the physical nodes and links in $M(t)$.

4.3 Problem Formulation

Let us consider the time $t = T_0$ as the critical time instant for the mapped forwarding graph $M(t)$, where $q\big(M(t = T_0)\big) < q^{min}$. We assume that this condition triggers the adjustment of $M(t)$. Let M_1 and M_2 denote the original and modified mapped forwarding graphs at time $t = T_0$, respectively, which are given by:

$$M_1 = \mathrm{M}(T_0^-) \text{ and } M_1 = \mathrm{M}(T_0^+) \qquad (4\text{-}1)$$

Table 4.1 Input parameters and Variables

System model and formulations parameters	
$\mathcal{N}$	Set of $\lvert\mathcal{N}\rvert$ number of physical nodes
$\mathcal{E}$	Set of $\lvert\mathcal{E}\rvert$ number of physical links
$\mathcal{P}_{p,q}^{max}$	Set of all the paths from node n_p to node n_q with a maximum path length of ϕ^{max} hops
$\varphi_{k,j}$	the path that, according to M_2, connects node n_K to n_J (i.e., an outgoing path from n_K).
$W_{\text{CPU}}(n_p)$	Processing capacity of the physical node $n_p \in \mathcal{N}$
$W_{\text{BW}}(e_{p,q})$	Bandwidth capacity of the physical link $e_{p,q} \in \mathcal{E}$
q^{min}	The minimum acceptable throughput
$q_M(t)$	The end-to-end throughput of the mapped forwarding graph $M(t)$ at time t
$\mathcal{F}(t)$	Set of $\lvert\mathcal{F}(t)\rvert$ number of required VNFs for VNFR $r \in \mathcal{R}$ at time t
D	The dependency graph representing the relations between VNFs.
$d_i(t)$	The entering data rate to VNF $f_i \in \mathcal{F}(t)$ at time t.
$u_j(t)$	The CPU utilization of node $n_j \in \mathcal{N}$ at time t.
$u_{e_{v,z}}(t)$	The bandwidth utilization of the link $e_{v,z} \in \mathcal{E}$ at time t.
R_{f_i}	The ratio of the outgoing data rate to the incoming data rate of VNF $f_i \in \mathcal{F}(t)$.
P_{f_i}	The Processing resource demand per bandwidth for VNF $f_i \in \mathcal{F}()$.
$M(t)$	A mapped forwarding wih $\lvert M(t)\rvert$ distinct elements at time t
M_1	The original mapped forwarding graph, $M_1 = \{M(t) \lvert t \leq T_0\}$
M_2	The modified mapped forwarding graph $M_2 = \{M(t) \lvert t > T_0\}$
$\mathcal{N}_{M_2}$	The set of physical nodes involved in M_2
$\mathcal{E}_{M_2}$	The set of physical links involved in M_2
$\sigma_{h,k,j,i,p}^{t}$	The h^{th} element of $M(t)$, indicating the placement of f_i on physical node n_j while using the physical links $\varphi_{k,j}$ and $\varphi_{j,p}$ as incoming and outgoing paths.
$\mathcal{K}_{M_2,j}$	The set of VNFs of that as dictated by M_2 are hosted on the physical node n_j.
$C_{\delta_{1,2}}$	The differentiated resource usage between M_1 and M_2
C_{new}	The aggregation cost of instantiating new VNFs
C_{mig}	The aggregation cost of state copying for migrating VNFs

For illustration, let us consider the examples depicted in Figs. 4.1.a and 4.1.b, where a content delivery service is realized via the deployment of three VNFs, namely, transcoder VNF f_{tr}, compressor VNF f_{cp}, and cache VNF f_{ca}. As illustrated in Fig. 4.1.a, the original VNF-FG of the content delivery service with the three VNFs f_{tr}, f_{cp}, and f_{ca} is given. In this service, the transcoding function should be executed before the compressing and caching functions, and this order restriction is represented by the arrows from VNFs f_{cp} and f_{ca} to VNF f_{tr} in the given dependency graph. Let us assume that node n_5 is a forwarding node which is incapable of hosting any VNF. Figure 4.1.a depicts the embedding map of the service, which specifies the allocation of

physical resources to the VNF-FG. Consider a scenario in which the service demand increases and function f_{ca} hosted on node n_3 is particularly overloaded. Moreover, let us assume that the link $e_{5,6}$ connecting nodes n_5 to n_6 is also congested. In a joint function scaling and topology adaptation approach, as depicted in Fig. 4.1.b, a new instance of the overloaded VNF f_{ca} is instantiated, followed by modifying the order of functions, thus altering the topology of the VNF-FG (see Fig. 4.1.b). This in turn, leads to a modified service embedding map, where not only a smaller amount of load is carried by the congested link $e_{5,6}$, but also the two instances of VNF f_{ca} can be executed in parallel. Using the system model presented in Section 4.2, M_1 and M_2 can be presented as follows:

$$\mathrm{M}_1 = [< [e_{s,2}], f_{tr}, n_2, [e_{2,3}] >, < [e_{2,3}], f_{ca}, n_3, [e_{3,5}, e_{5,6}] >, < [e_{3,5}, e_{5,6}], f_{cp}, n_6, [e_{6,d}] >] \quad (4\text{-}2)$$

$$\mathrm{M}_2 = [< [e_{s,2}], f_{tr}, n_2, [e_{2,3}] >, < [e_{2,3}], f_{ca}, n_3, [e_{3,5}, e_{5,6}] >, < [e_{2,3}], f_{cp}, n_3, [e_{3,4}] > , < [e_{3,4}], f_{ca}, n_4, [e_{4,d}] >, < [e_{3,5}, e_{5,6}], f_{ca}, n_6, [e_{6,d}] >] \quad (4\text{-}3)$$

where $e_{s,2}, e_{6,d}$ and $e_{4,d}$ are the links connecting the ingress node n_s to n_2 and nodes n_6 and n_4 to egress node n_d, respectively. In Transforming M_1 to M_2 *b*esides new paths and changes in VNFs' order (i.e., topology adaptation), a new instance of f_{ca} is also created so that the two instances of this VNF are hosted on different nodes n_5 to n_4 (i.e., a realization of function scaling). Cost C_{Total} of transforming the original mapped forwarding graph M_1 to the modified mapped forwarding graph M_2 can be broken down into three partial costs as follows:

1- $C_{\delta_{1,2}}$ which is the differentiated resource usage between M_1 and M_2 and is computed by subtracting the accumulated physical resources (computing and bandwidth) consumption costs of elements in M_1 from that of M_2. Moreover, to encourage consolidation and limit the number of active computing resources (i.e., active physical nodes), the difference in the number of active nodes between M_1 and M_2 is also accounted for in this partial cost.

2- C_{new} which is the aggregation cost of instantiating new VNFs [85], which are those that are newly added to M_2 comparing to M_1, and

3- C_{mig} which is the aggregation cost of state copying for migrating VNFs, which are those that, in transitioning of M_1 to M_2, move from one physical node to the other node.

We then define our cost minimization problem as follows:

$$\min_{M_2|M_1}\left(C_{total} = C_{\delta_{1,2}} + C_{new} + C_{mig}\right) \tag{4-4}$$

Subject to Constraints 1-3, which are described in the following:

Constraint 1: This constraint ensures that the throughput of $M_2\left(i.e., q(M_2)\right)$satisfies the minimum acceptable throughput threshold q^{min}:

$$q_{M_2}(\mathrm{t}) \geq q^{min} \tag{4-5}$$

Constraint 2: The processing demands of VNFs in M_2 should be smaller than the available processing resources of their hosting nodes:

$$\frac{\left(\sum_{\forall f_i | f_i \in \kappa_{M_2,j}} d_i(t) \times P_{f_i}\right)}{W_{\mathrm{CPU}}(n_j)} \leq \overbrace{1 - u_j(t)}^{Avail.\ processing\ ratio} \tag{4-6}$$
$$\forall t > T_0, \forall n_j \in \mathcal{N}_{M_2}$$

where $\mathcal{N}_2$ denotes the set of all physical nodes involved in M_2 and $\kappa_{M_2,j}$ denotes all the VNFs embedded on node $n_j \in \mathcal{N}_{M_2}$. The term $d_i(t) \times P_{f_i}$is the CPU demand of $f_i \in \kappa_{M_2,j}$. The right-hand side of Eq. (4-6) represents the ratio of the available processing resources at the physical node $n_j \in \mathcal{N}_{M_2}$ at time $t > T_0$, while the left-hand side is the ratio of the cumulative requested processing resources of all VNFs hosted by the node n_j to the processing capacity of the node n_j.

Constraint 3: Similar to Constraint 2, this constraint ensures that the bandwidth demands of VNFs in M_2 are smaller than the available bandwidth resources of the corresponding embedding links in $\mathcal{E}_{M_2}$ (i.e., the set of all the physical links involved in M_2):

$$\frac{\left(\sum_{\forall f_i | f_i \in \kappa_{M_2,j}} d_i(t) \times \mathcal{R}_{f_i}\right) \times \frac{1}{|\varphi_{j,out}|}}{W_{\mathrm{BW}}(e_{v,z})} \leq \overbrace{1 - u_{v,z}(t)}^{Avail.\ bandwidth\ ratio} \tag{4-7}$$
$$\forall t > T_0, \forall n_j \in \mathcal{N}_{M_2}, \forall (v,z) | e_{v,z} \in \varphi_{j,k}, n_k, n_j \in \mathcal{N}_{M_2}$$

where the right-hand side is the ratio of the available bandwidth resource at the physical link $e_{v,z} \in \varphi_{j,k}$, where $\varphi_{j,k}$ is the path that, according to M_2, connects node n_j to n_k (i.e., an outgoing path from n_j). Also, $|\varphi_{j,out}|$ is the number of all outgoing paths from n_j, including $\varphi_{j,k}$, which, as M_2 dictates, connect n_j to the next VNF host nodes. The left-hand side of Eq. (4-7) is the ratio of cumulative bandwidth demand from the link $e_{v,z}$ to its bandwidth capacity.

4.4 Deep Reinforcement Learning For Efficient VNF-FG Function Scaling and Topology Adaptation

In the following, we present our proposed DRL-based solution. The search space of the problem under study is not only large, but it can also grow exponentially as the problem size increases. To tackle the problem in a computationally efficient manner, we first design an MDP [20] framework tailored to our problem and then leverage it to propose our DRL solution, which relies on Deep Q Learning (DQL).

4.4.1 System States, Actions, and Rewards

Let S represent the set of states. The state denoted by $s_i =< M_2, A >^i \in S$ contains two main components: (i) The modified mapped forwarding graph M_2 and (ii) an auxiliary graph A specifying the VNFs in $F(t)$, including the new instances added as the result of function scaling, which are not yet chained into M_2. Furthermore, we denote $\mathcal{A}$ as the total set of actions in the state $s_i \in \mathcal{S}$. The action $a_j =< \varphi^j_{1,select}, f^j_{select}, n^j_{select}, \varphi^j_{2,select} > \in \mathcal{A}(s_i)$ is a 4-tuple containing the following components: (i) the selected entering path $\varphi^j_{1,select}$ to connect n^j_{select} (i.e., the host node of f^j_{select}) to n^{j-1}_{select} (i.e., the host node of f^{j-1}_{select}), (ii) VNF f^j_{select} selected as the next candidate VNF in the modified mapped forwarding graph M_2, (iii) physical node n^j_{select} selected to host f^j_{select}, and (iv) outgoing path $\varphi^j_{2,select}$ selected to connect n^j_{select} (i.e., the host node of f^j_{select}) to n^{j+1}_{select} (i.e., the host node of f^{j+1}_{select}). After selecting action a_j the system state will transition to s_{i+1}. As such, the element $\sigma_{i+1} = a_j$, will be added to M_2. Consequently, the auxiliary graph A will be updated by omitting f^j_{select} indicating that this VNF instance is now considered in the M_2.

Next, we design our reward function, which takes into account the cost and Constraints 1, 2, and 3 given by Eqs(4-4)(4-7). We define the reward $R(s_i, a_j)$ of selecting action $a_j \in \mathcal{A}(s_i)$ in state $s_i \in \mathcal{S}$ as follows:

$$R(s_i, a_j) = \begin{cases} -(C_{total} + \Omega \times |A|) & Eqs.\ (4-5)-(4-7)\ are\ true, \\ -\infty & otherwise, \end{cases} \tag{4-8}$$

where

$$C_{total} = C_{\delta_{1,2}} + C_{new} + C_{mig} \quad (4\text{-}9)$$

In Eq. (4-8), we assign a negative infinity value to the reward if, according to constraints 1, 2, and 3, M_2 is not feasible; otherwise, the reward considers the total cost of transforming M_1 to M_2 (i.e., C_{total}) along with the penalizing term $\Omega \times |A|$, where Ω is the penalty coefficient and $|A|$ is the number of VNF instances in $\mathcal{F}(t)$ that remain to be mapped.

4.4.2 Deep Q Learning for VNF-FG Function Scaling and Topology Adaptation

The MDP components presented in Section 4.4.1 provide the required mathematical formalism for applying Q-Learning, which is a widely used branch of reinforcement learning algorithms [20]. By successively updating the evaluation of the long-term quality (also known as the Q value) of actions at each state, Q-Learning provides a simple yet effective way for an agent to learn how to act optimally [20]. However, in most real-world problems with large state/action spaces, Q-Learning becomes inefficient, as exploring all the states and taking all the possible actions is often impossible [21]. In our designed MDP components, as presented in Section 4.4.1, action $a_j \in \mathcal{A}(s_i)$ includes the selection of entering and outgoing paths (i.e., $\varphi^j_{1,select}$ and $\varphi^j_{2,select}$) for each VNF-host node pair. Clearly, even in a small network, the number of paths could become extremely large, thus leading to a very large action space. In such environments with huge state/action spaces, a possible way for learning efficiently is to approximate the Q values of state and actions [21]. To that end, using a Deep Neural Network (DNN) that estimates these Q values (DQN) has proven to be effective [21], and it is now widely used in different domains under the so-called deep reinforcement learning (DRL) label, also known as deep Q-Learning (DQL). However, the use of nonlinear function approximations (such as DNN) can lead to unstable or even diverge results [21]. This is due to the fact that the true values of the Q function used for training, calculated by summation of the current reward and maximum Q value of the next state, is a function of the Q value itself and thus vary by changes in the estimation of Q values.

Using two deep Q networks (also known as Double Deep Q-Learning or DDQN) can eliminate the correlations between the true Q values and their estimations, thus stabilizing the results.

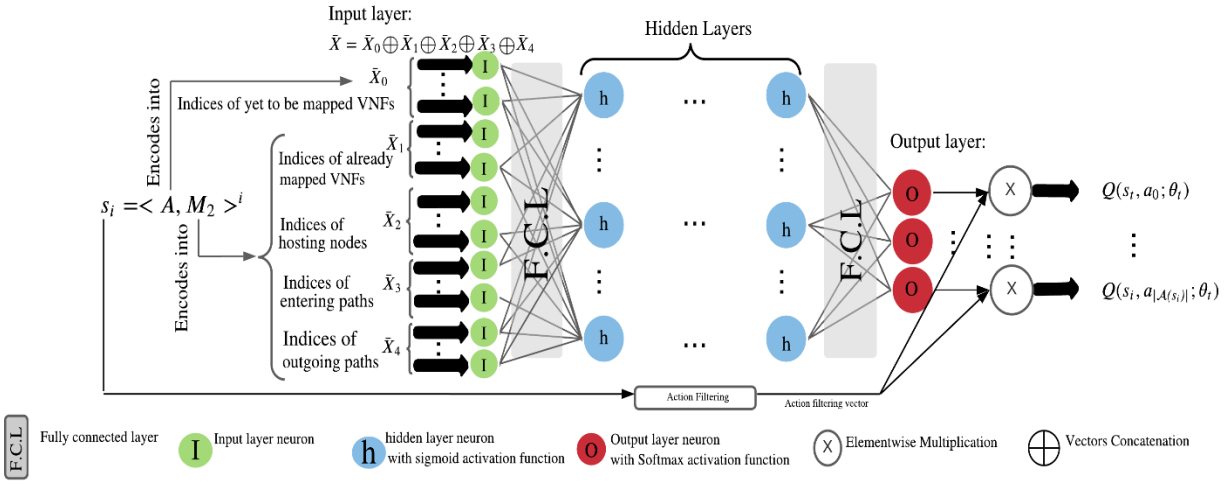


Figure 4.2 Schematic view of *DQN-Selection* network enhanced with action filtering technique.

In this work, we utilized two DQNs, namely, $DQN - Selection$ with parameter θ_t and $DQN - Evaluation$ with parameter θ'_t. Both networks are fed with the same batch of experiences (where an experience comprises a state, action, reward, and next state). Also, after a predefined number of iterations, θ'_t is updated by the value of θ_t. In this setting, in exploiting phase, the actions are selected by $DQN - Selection$, whereas the output of $DQN - Evaluation$ is treated as the true values for tuning of $DQN - Selection$ parameter (i.e., θ_t) in backpropagation calculations.

Yet another effective technique for problems with huge action space is the so-called Action Filtering [71], which can further reduce the action space. In this technique, in each state, the Q values for irrelevant actions are explicitly set to zero. For instance, as discussed in Section 4.4.1, in our problem, action $a_j =< \phi^j_{1,select}, f^j_{select}, n^j_{select}, \phi^j_{2,select} >$ comprises the entering and outgoing paths (i.e., $\phi^j_{1,select}$ and $\phi^j_{2,select}$). However, only a subset of paths, which can connect the host nodes of two successive embedded VNFs (successive in accordance to M_2), should be considered as candidates for $\phi^j_{1,select}$ and $\phi^j_{2,select}$. Also, the nodes that can not be reached within a predetermined maximum number of hops from ingress or egress nodes should be excluded for hosting of VNFs. In our framework, a separate action filtering entity in *DQN-Selection* receives

Algorithm 1 Joint VNF-FG Function Scaling and Topology Adaptation (JFSTA) Algorithm

Initialize the ***DQN-Selection*** network with $\theta_t = \theta_0$
Initialize the ***DQN-Evaluation*** network with $\theta'_t = \theta'_0$
Initialize a replay buffer
Initialize iter = 1, i = 0
while $iter <$ max_iteration **do**
    Set $M_2 = [\,], A = \{F(t) | t \leq T_0\}, iter = iter + 1$
    Set $F_{overloaded} = \{\mathcal{K}_{M_1,j} | q(n_j) = q^{min}(\mathcal{N}_{M_1})\}$
    while $q(M_2) < q^{min}$ **do**
        Select a random VNF f_{rand} from $F_{overloaded}$.
        Update A by adding a new instance of f_{rand}.
        Set $s_i =< M_2, A >$
        while A not empty **do**
            Set $B_i = \{f_j | f_j \in A, f_j$ does not depend on other VNFs in $A\}$
            Set $O_i = \{n_p | n_p \in \mathcal{N}$ and n_p can host at least a member of $B_i\}$
            Set $\emptyset_i = \{$ all paths with maximum ϕ^{max} hops$\}$.
            Construct the action space $\mathcal{A}(s_i) = \{\emptyset_i \times B_i \times O_i \times \emptyset_i\}$
            Generate a random number $\lambda \in [0, 1]$
            if $(\lambda < \varepsilon)$ **then** ▷ Exploration phase
                Select a random action
                $a_j =< \varphi^j_{1,select}, f^j_{select}, n^j_{select}, \varphi^j_{2,select} > \in \mathcal{A}(s_i)$
            else ▷ Exploitation phase
                Choose a_j which maximizes $Q^{\text{DQN-Selection}}(s_i, a_j; \theta_i)$
            end if
            Update M_2 by appending a_j.
            Update A by deleting f^j_{select}.
            Compute the reward $\mathbb{R}(s_i, a_j)$ with Eq.8.
            Set $s_{i+1} =< M_2, A >$
            Store $< s_i, a_j, \mathbb{R}(s_i, a_j), s_{i+1} >$ as experience in replay buffer.
            if enough experiences stored in replay buffer **then**
                Extract a batch of experiences
                Feed the batch to **DQN-selection and DQN-evaluation**.
                Collect the outputs of **DQN-evaluation** as true values.
                Use the true values to train **DQN-selection**.
            end if
            Set $i = i + 1$
            Every $\bar{\tau}$ iterations set $\theta'_t = \theta_t$
        end while
    end while
end while

each state and identifies such low-value actions in that state, and correspondingly generates an action filtering vector with size $|\mathcal{A}|$. This vector contains binary coefficients for each possible action so that by multiplying to the output of the last layer of $DQN - Selection$ explicitly set the Q values of irrelevant actions to zero. Figure 4.2 depicts a schematic view of $DQN - selection$ with its state-input encoding and action filtering entity.

4.4.3 Joint Function Scaling and Topology Adaptation (JFSTA) Algorithm

Algorithm 1 presents the pseudo-code of our proposed framework. The algorithm starts with random initialization of the two Q networks, *DQN-Selection* and *DQN-Evaluation*. Then a memory for storing the experiences is initialized (line 3 of Algorithm 1) while setting the main loop and state iterating indices $iter$ and i as 1 and 0, respectively (line 4 of Algorithm 1). The loop index $iter$ is compared with a predefined maximum number *max_iteration* (line 5 of Algorithm 1). If loop index $iter$ is smaller than *max_iteration*, the main loop is executed (lines 5-38). The modified mapped forwarding graph M_2, which is the desired output, is initialized by an empty list while the auxiliary graph is initialized as $A = \{F(t) | t \leq T_0\}$. Therefore, at this point (line 6), A contains all the VNFs in the original mapped forwarding graph (i.e., the VNFs in M_1). Next, the overloaded VNFs in the original VNF-FG (i.e., set $F_{overloaded}$) are identified. As such, $F_{overloaded}$ contains all the VNFs of M_1 that reside on the node with the lowest throughput (line 7 of Algorithm 1). Next, the minimum throughput condition (Constraint 1) is checked. If the minimum throughput $q(M_2)$ of M_2 is smaller than q^{min} we try to modify M_2. To that end, a random VNF f_{rand} is selected from the set $F_{overloaded}$. The selected VNF is actually a candidate for function scaling. Next, the auxiliary graph A is updated by adding a new instance of f_{rand}. Note that to cover the cases where more than one instance of f_{rand} is needed, f_{rand} is not dropped from $F_{overloaded}$ so that re-selecting f_{rand} in future iterations remains possible. Next, the state $s_i =< M_2, A >$ is built (line 11 of Algorithm 1). At this point, our original problem is already downgraded to a smaller problem of finding a properly mapped forwarding graph for VNFs inside A. M_2 is then constructed by considering all the VNFs specified by A (see the most inner WHILE loop presented by lines 12-36). In line 13, we identify the VNFs belonging to A that, according to the dependency graph (i.e., graph D), do not depend on any other VNFs in this set and collect them in the set B_i. Moreover, the physical nodes that are ready to host at least one element of B_i are collected in the set O_i (line 14 of Algorithm 1). Similarly, all the possible paths with a maximum length of ϕ^{max} are collected in the set O_i (line 15 of Algorithm 1). In line 16, $\mathcal{A}(s_i)$, which is the set of all possible actions in the current state (i.e., s_i), is constructed as $\mathcal{A}(s_i) = O_i \times B_i \times O_i \times O_i$.

Following an ε-greedy strategy, the algorithm switches between the exploration and exploitation phases. In the exploration phase, a random action $a_j =< \phi^j_{1,select}, f^j_{select}, n^j_{select}, \phi^j_{2,select} > \in \mathcal{A}(s_i)$ is selected. In the exploitation phase, the algorithm chooses the action that, according to *DQN-Selection*, maximizes the Q-value (line 21 of Algorithm 1). Next, M_2 is updated by appending a_j (lines 23 of Algorithm 1). A is also updated by removing the VNF instance f^j_{select} indicating that it is already considered in M_2 (line 24 of Algorithm 1). Next, we calculate the reward using Eq. (4-8). Then the next state $s_{i+1} = < M_2, A >$ will be constructed. Then, the experience $< s_i, a_j, R(s_i, a_j), s_{i+1} >$, will be sent to the replay buffer to be stored (see line 27 of Algorithm 1). Next, the algorithm checks whether the replay buffer contains enough experience (line 28 of Algorithm 1). If so, a random batch of size β will be extracted and sent to *DQN-Selection* and *DQN-Evaluation* for training (line 30 of Algorithm 1). A forward pass to *DQN-Evaluation* provides the true values for *DQN-Selection*, which will be used for parameter tuning of this Q network (lines 31 and 32 of Algorithm 1). If the replay buffer does not contain enough experiences, the batch extraction and training will be postponed to future iterations. Increasing the state index i by one (line 34 of Algorithm 1), this process is repeated until A becomes empty, which indicates that all the VNFs in M_1 along with the new instances are now considered in M_2. If the throughput of the constructed M_2 is larger than q^{min}, M_2 is a candidate solution. The algorithm continues to generate further solutions until the number of iterations reaches a predefined threshold, indicating that the training of Q-networks is completed.

4.5 Performance Evaluation

In the following, we evaluate the performance of our proposed algorithm, which has been implemented using the OpenAI Gym toolkit [72] via a customized environment. The simulations were conducted on a Google Cloud Platform using a VM instance with 10 vCPUs and 37 GB of memory. For implementing the Q networks, we used Tensorflow 2.5.0 [76]. To ensure convergence and model efficiency, we utilized the Keras Tuner Framework [77] to automate the tuning of the hyper-parameters.

In our evaluations, we have considered the substrate network topology that is shown in Fig. 3.5 [74][78]. This substrate network topology consists of 15 nodes (including forwarding and VNF-capable node) and 52 bidirected links. Accordingly, in our scenarios, we add/drop links/nodes to

scale up/down this substrate network as needed. The CPU and bandwidth capacities of nodes and links were set randomly in the range [1-5] GHz and [100 Mbps, 1 Gbps], respectively. As for the resource requests and node utilization, to ensure a realistic scenario, we used values proportional to the recently published Google cluster traces [79] available with the platform Google BigQuery. The number of VNF instances in the original VNF-FG is selected in the range [3, 10]. The embedding map of the original VNF-FG is also set randomly. In these settings, for each episode of our simulation scenario, a random substrate node of the original embedding map is selected as the overloaded node (i.e., the service throughput bottleneck) and, consequently, the VNFs hosted on this node are considered candidates for the function scaling. Moreover, to avoid exploding the action size, we limit the maximum number of instances and number of branches (i.e., number of entering/outgoing paths to/from each VNF hosting node) to 2 and 3, respectively. The cost of new instantiation and state copying of a single VNF is set to 100 and 10 (in unit of currency), respectively. The costs of consuming a unit of CPU and bandwidth are 10 and 5 (both in unit of currency), respectively.

4.5.1 Optimality Gap

To evaluate the performance of our proposed JSTAF algorithm against the optimal solution, we consider a small-size problem instance. To do so, we select a subset of the substrate network shown in Fig 3.5, which consists of 9 nodes and 26 bidirected links. We use the exhaustive search approach to find the optimal solution of joint function scaling and topology adaptation of three randomly composed and embedded VNF-FGs, each comprising 5 VNFs. Figure 4.3. depicts the gap (measured in terms of the absolute relative difference in percentage) between the solutions found by our algorithm in every 700 episodes for different learning rates of 1.0, 0.1, 0.01, 0.001, and the average cost of the optimally modified VNF-FGs. As shown in the figure, the value of the learning rate has a significant impact on the optimality gap and its convergence. More specifically, higher values of learning rate (i.e., between 0.1 and 1.0) lead to poor and diverged results. This happens mainly because with the large values of the learning rate, rather than reaching the exact optimal values, the parameters of the neural network oscillate around these optimal settings on each update. On the other hand, too small values of learning rate (i.e., below 0.001) could lead to a prolonged convergence. We observe in Fig. 4.3 that the optimality gap of the learning rate of 0.001 is trapped in local optima, thus hitting a plateau at 300%.

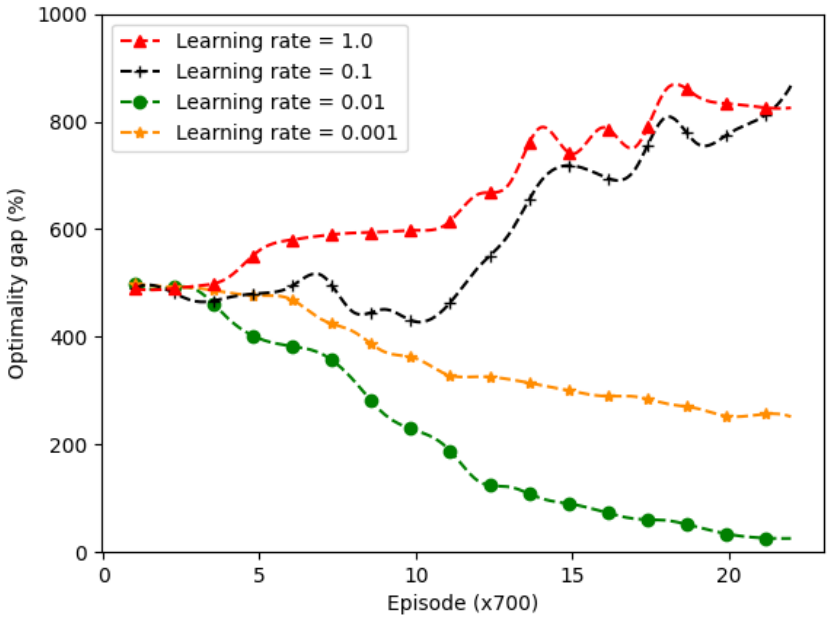


Figure 4.3 Optimality gap (in percentage) of our proposed JFSTAF framework vs. the number of episodes.

This demonstrates that the gradient descent algorithm used in our proposed JFSTAF algorithm is very sensitive to the learning rate parameter, which can be tuned carefully. As suggested by our obtained results shown in Fig. 4.3 we set the learning rate to 0.01 from now on. We also observe from Fig. 4.3 that the gap is significant during the initial 700 episodes. This observation is not surprising since in the initial episodes, the JFSTAF algorithm merely explores random solutions. As the number of episodes grows, the optimality gap gradually decreases. More starting, starting with an optimality gap of 500%, our JFSTAF algorithm reduces the gap down to 33% after being sufficiently training. An important consideration here is that the exhaustive search method takes more than 7 hours to find the optimal solutions for the considered small-size VNF-FGs, whereas it takes our proposed JFSTAF algorithm about 2 hours to be sufficiently trained until it reduces the optimality gap by 80%. This highlights the scalability of our proposed algorithm, especially for large-size problem instants and more complex VNF-FGs.

4.5.2 Convergence and Performance Comparison with Other Deep Learning Network Architectures

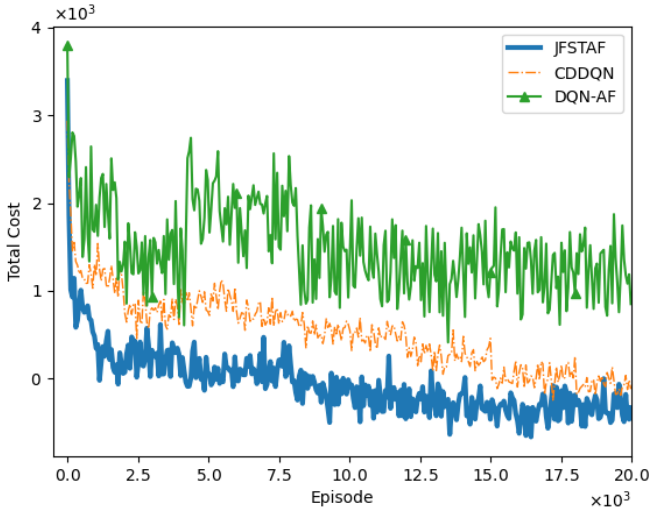


Figure 4.4 Total cost vs. episode for different deep Q-Learning architectures.

To evaluate the convergence performance of our proposed VNF-FG Joint Function Scaling and Topology Adaptation Framework (JFSTAF) and its architecture design choices, we compare it with two widely used and classical Q-network architectures: (i) DQN-AF: a single Q-network [86] with action filtering entity and (ii) CDDQN: a Conventional Double Deep Q network without action filtering [21]. We have fed all three Q-networks JFSTAF, DQN-AF, and CDDQN with the same batch size of 30 samples and used the same technique to draw samples from their individual experience replay buffers. Figure 4.4 depicts the convergence performance of different methods. We observe from the figure that the DQN-AF method performs poorly compared to other methods JFSTAF and CDDQN. More specifically, even after 20,000 episodes, the DQN-AF method converges to a very large cost value of around 1500 unit of currency. This is a direct consequence of the limitation of a single Q-network because the Q network is trained by its own Q value estimations. On the other hand, the CDDQN and JFSTAF methods converge to almost the same cost value. However, it is evident from Fig. 4.4 that our proposed JFSTAF method converges at a higher rate (around episode 8000) compared to the CDDQN method. This demonstrates the empowering impact of the action filtering technique, which reduces the action size efficiently by identifying more valuable actions, thus allowing for faster convergence.

4.5.3 Performance Comparison with Disjoint Method

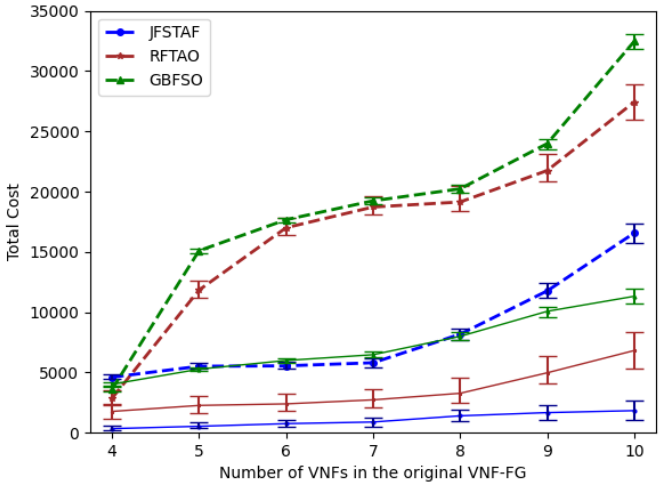


Figure 4.5 Total cost vs. number of VNFs in the original VNF-FG (performance comparison with disjoint methods).

Next, we examine the performance gains of our JFSTAF in comparison with heuristic-based function scaling-only and topology adaptation-only methods. Since greedy and random heuristics are widely used in the literature [82], [83], [3] for comparison, we design a Greedy-BestFit Function Scaling-Only (GBFSO) [83] and a Random-FirstFit Topology Adaptation-Only (RFTAO) [82] algorithms. In our design, the GBFSO algorithm increases the number of overloaded VNF instances (i.e., the VNFs that are hosted on the overloaded substrate node in the original mapped forwarding graph) to the maximum possible number of instances per VNF (i.e., 3 in our scenario) and embed them in a way that it imposes the least possible embedding cost using the best-fit approach. The RFTAO algorithm, on the other hand, randomly reorders the existing VNF instances while respecting the dependency graph and embeds them to the nearest available nodes from their original host using the first fit approach. Moreover, in our simulation, we realized two scenarios, A and B. In scenario A, the minimum acceptable throughput is increased by 20%, whereas in scenario B, an 80% increase has been made to the minimum acceptable throughput.

Figure 4.5 illustrates the total cost vs. the number of VNFs in the original VNF-FGs. In this figure, two sets of curves are depicted. The solid curves represent scenario A (where the minimum acceptable throughput is increased by 20%), whereas the dashed curves represent Scenario B (where an 80% increase has been made to the minimum acceptable throughput). Clearly, satisfying the higher minimum throughput requires more resources, thus imposing higher costs for all algorithms under consideration. However, we observe from Fig. 4.5 that our proposed JFSTAF

algorithm outperforms the two disjoint methods in both Scenarios A and B for a given number of VNFs increasing from 4 to 10. This highlights the beneficial impact of the joint consideration of function scaling and topology adaptation in cases with divergent requests' parameters. Specifically, according to Fig. 4.5, our proposed JFSTAF achieves a performance gain of up to 73% and 93% compared to the RFTAO and GBFSO methods, respectively. Moreover, according to Fig. 4.5, GBFSO imposes higher costs compared to the RFTAO algorithm. This happens because even if the most efficient embedding solution has been selected for the modified VNF-FG, the GBFSO algorithm always maximizes the number of VNF instances, thus leading to the consumption of more resources. The results also show that the random ordering of VNFs (without any function scaling) in the RFTAO algorithm leads to a high cost. This is mainly due to the fact that reordering the VNFs in the VNF-FG results in the utilization of new paths, thus increasing the resource consumption cost as well as the state copying costs. Finally, we observe that adding 6 VNFs to the original VNF-FG makes the costs of the RFTAO and GBFSO algorithms (averaged over Scenarios A and B) about five times larger. By contrast, our proposed JFSTAF algorithm experiences a maximum of only 3.2 times increase in the total cost.

4.5.4 Performance Comparison with a Joint Method

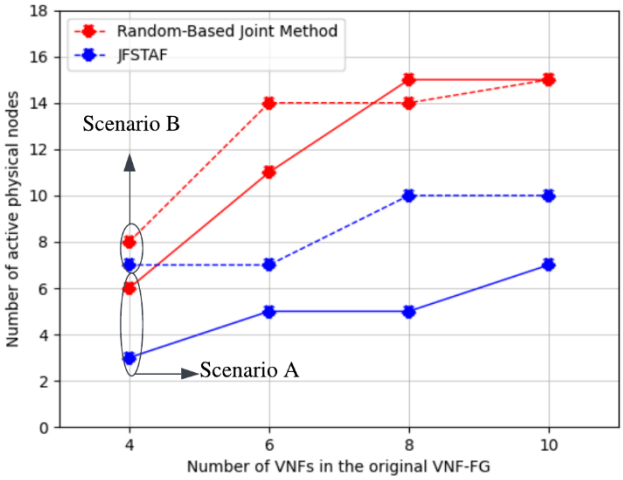


Figure 4.6 Number of active physical nodes vs. number of VNFs in the original VNF-FG (performance comparison with a joint method).

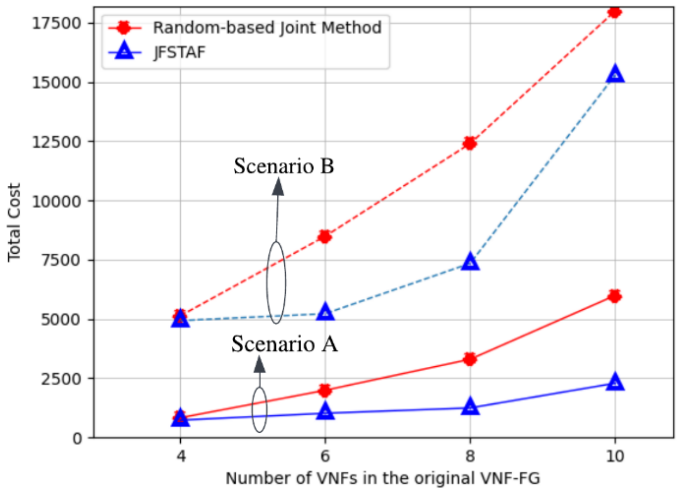


Figure 4.7 Total cost vs. number of VNFs in the original VNF-FG (performance comparison with a joint method).

Given that there is no existing work on joint function scaling and topology adaptation, we implement a random-based joint method as our benchmark, where the order, connectivity, and the number of VNF instances are randomly modified, and the first feasible mapped forwarding graph (according to Eqs. (4-5)-(4-7)) is selected as the solution. We refer to this method as the random-based joint approach and compare its performance with our proposed JFSTAF algorithm in terms of the number of active physical nodes for Scenarios A and B, explained in Section 4.4.3. Figure 4.6 illustrates the number of active physical nodes vs. the number of VNFs in the original VNF-FG. Clearly, having a higher acceptable minimum throughput enforces both methods to place VNF instances on different nodes to encourage parallel executions, which leads to a higher throughput. Nevertheless, as shown in Fig. 4.6, our proposed JFSTAF algorithm requires a smaller number of active nodes in both Scenarios A and B, mainly because it directly takes into account this parameter in its cost model (i.e., see Eq. (4-4)). The random-based joint benchmark, on the other hand, scatters the VNF instances across existing physical nodes arbitrarily since it ignores the impact of the number of active physical nodes on the total cost in its search process.

Finally, the total cost vs. the number of VNFs in the original VNF-FGs for the two joint methods, random based and JFSTAF, in Scenarios A and B is depicted in Fig 4.7. As shown in the figure, our proposed JFSTAF algorithm outperforms the random-based joint benchmark in both Scenarios A and B for a given number of VNFs in the original VNF-FG. We note that for the small number

of VNFs in the original VNF-FG, the total costs of both methods are almost the same in both Scenarios A and B. This is because there are not many choices for reordering of VNFs, as the number of VNFs is too small. Also, the choices for function scaling are rather limited. On the other hand, as the number of VNFs increases, the difference between our proposed JFSTAF method and the random-based joint method becomes significant, achieving a maximum cost improvement of 60%.

4.6 Conclusions

In this chapter, we studied the VNF-FG scaling problem, which arises from changes in user demand, application requirements, and traffic conditions. To that end, we proposed our joint function scaling and topology adaptation method, which supports not only the horizontal scaling but also VNF reordering and connectivity changes in a given VNF-FG. Converting the problem to an MDP framework, we have designed state, action, and reward components accordingly. To tackle the issue with a large state/action space, we approximate Q values by using two Deep Q networks while applying an action filtering technique to further reduce the size of the action space. We evaluated the performance of our proposed framework against different network architectures and conducted performance evaluations comparing with both joint and disjoint benchmarks. The results show that our proposed method achieves up to a 93% cost improvement compared to the benchmarks.

Chapter 5

5. Content Placement for CDN[1]

5.1 Introduction

Once a CDN is deployed, different parameters should be optimized to ensure efficient operation. To that end, consider an edge-based CDN with caches close to end-users. Even though utilizing edge caches and bringing the contents closer to end-users potentially provide increased QoS, it also imposes costs for CDN providers. For example, the contents might be needed to migrate between edge nodes after their initial placement. Making the content migration decision is challenging since it corresponds to different costs such as content uploading\downloading costs and bandwidth occupation costs. Moreover, the edge cache's characteristics, such as their specific cost model, limited capacity, and the possibility to be mobile, made the initial content placement and migration decisions complex. Moreover, in a CDN that delivers contents of low- or high priority, the migration decisions would be even more challenging because the cost model and the

[1] This chapter is based on the following published papers:

- Sepideh Malektaji, Somayeh Kianpisheh, and Roch Glitho,"*Purging-Aware Content Placement in Fog-Based Content Delivery Networks.*" In 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), pp. 1-3. IEEE, 2018

- Sepideh Malektaji, Amin Ebrahimzadeh, Halima Elbiaze, Roch Glitho, and Somayeh Kianpishe. "*Deep Reinforcement Learning-based Content Migration for Edge Content Delivery Networks with Vehicular Nodes*." IEEE Transactions on Network and Service Management 2021.

acceptable QoS level for high-priority content might not be the same as those for low-priority contents.

This chapter proposes a content migration method for edge content delivery networks with vehicular nodes. In such CDNs, local caches can offload their contents to neighboring edge caches whenever feasible instead of removing them when fully occupied. This process ensures that more contents remain in the vicinity of end-users. We propose a deep reinforcement learning approach to selecting which contents to migrate and to which neighboring cache to migrate while minimizing the corresponding cost. Our simulation scenarios realized up to a 70% reduction of content access delay cost compared to conventional strategies with and without content migration.

The remainder of this chapter is organized as follows: we first provide the system model and present our optimization formulation. We then present our proposed DRL-based content migration method in detail, followed by the performance evaluation of the framework. Finally, the conclusion will be provided for this chapter in the last subsection.

5.2 System Model

Figure 5.1 illustrates a high-level view of our system model with an example of the problem under study. In Fig. 5.1, certain mobile caches, such as autonomous vehicles in an area (referred to as targeted caches), send their requests for high-priority contents to the CDN controller. These high-priority contents could be HD maps and are denoted by C^{High}. Some targeted mobile caches, such as vehicle 'A' in Fig. 5.1, may already be fully occupied with low-priority contents denoted by C^{Low}. This condition triggers our proposed algorithm to find a desirable solution. The content migration algorithm would free up enough space in target caches to accommodate high-priority contents. However, instead of dropping the low-priority contents, the algorithm provides a new placement for them so that the low-priority contents migrate to nearby available caching resources and thus remain in the vicinity of the target caches. Such a content migration consumes resources both from the host and the destination node. It also consumes scarce bandwidth resources and thus is not cost-free. Our algorithm considers these costs and accordingly proposes a low-cost content migration solution. It also determines a low-cost delivery strategy for high-priority content from edge caches to target caches.

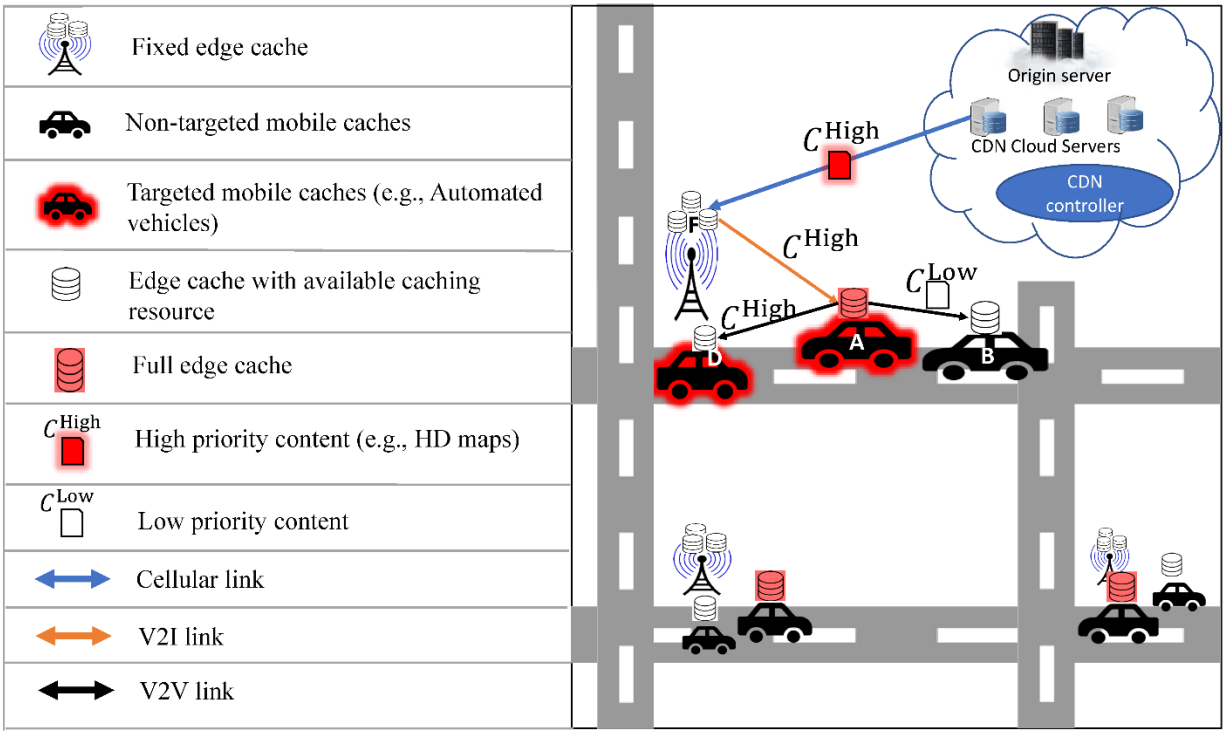


Figure 5.1 System view and an example of an edge-based CDN with vehicular nodes

For example, in Fig. 5.1, the content migration solution could be as follows: Assume a low-priority content C^{Low} fully occupies vehicle 'A''s mobile cache. However, vehicle 'A' needs to cache a high-priority content C^{High}. Considering the different migration solution costs, the algorithm could come up with the following strategy: The low-priority content C^{Low} cached in vehicle 'A' should be migrated to neighbor cache 'B' via a vehicle-to-vehicle (V2V) link [87]. This would free up enough space in the cache of vehicle 'A' to store C^{High}. As for the high-priority content delivery strategy, fixed cache 'F' can deliver C^{High} to mobile cache 'A' via a vehicle-to-infrastructure (V2I) link [87]. The received high-priority content C^{High} can then be delivered from vehicle 'A' to the other targeted caches such as vehicle 'D' via a V2V link. This example is valid based on the assumption that vehicle 'A' and both vehicles 'B' and 'D' stay within the coverage ranges of fixed cache `F' and vehicle `A', respectively, long enough for the contents to be transferred successfully. We note that any other migration and delivery strategies could impose a higher cost on the system. In the following, we present the modeling of our considered content migration problem, followed by an explanation of the cost calculation for content migration.

Edge Caches: we consider a CDN system that consists of mobile and fixed edge caches as well as CDN cloud servers. Let $M = \{e_j\}_{j=1}^{M}$ and $F = \{e_f\}_{f=1}^{F}$ denote the sets of mobile and fixed caches with a total number of M and F caches, respectively. The total set $E = M \cup F$ consists of all the fixed and mobile caches, consisting of a total of $N = M + F$ edge caches. Each cache $e_i \in E$ has limited caching and processing capacities denoted by $L_{stor}(e_i)$ and $L_{proc}(e_i)$ respectively.

Coverage regions of caches: Each cache (fixed or mobile) $e_i \in E$ has a circular coverage region with a diameter ℓ_i. The coverage area of fixed caches is usually larger than mobile caches, and it can cover a segment of a bidirected road or an intersection of the road. We denote the set of fixed caches that have a road intersection in their coverage as $F^+$and those that only cover straight segments of roads as F^-. Note that the total set of fixed edge caches is $F = F^+ \cup F^-$.

Locations and mobility of caches: Let $l_i(t)$ be the location of the cache $e_i \in M$ at time t, and l_f be the location of the fixed cache $e_f \in F$. The velocity of mobile cache $e_i \in M$ at time t, is denoted by v_i(t). For the movement of mobile caches, we adopt a probabilistic model, where mobile caches follow a probabilistic approach in the selection of their direction in a grid-like environment. At each intersection, the mobile cache chooses to keep moving in the same direction or to change direction. The probability of going straight is denoted by μ_S while taking a left or a right occurs with the probability of μ_L and μ_R, respectively.

Contents: Contents in the considered CDN system have either low or high priority. Let $C^{High}(t)$ and $C^{Low}(t)$ denote the high- and low-priority content sets with sizes of size($C^{Low}(t)$) and size($C^{High}(t)$.), respectively. $C^{High}(t)$ and $C^{Low}(t)$ contain $|C^{Low}(t)|$ and $|C^{High}(t)|$ number of individual contents denoted as $c_h^{high} \in C^{High}(t)$ and $c_l^{low} \in C^{Low}(t)$, respectively. Following the above-mentioned notations, $C(t) = C^{Low}(t) \cup C^{High}(t)$ denotes the total contents at the edge caches at time *t*, where $|C(t)|$ represents the total number of individual contents $c_k \in |C(t)|$, and size($C(t)$) is the total size of the set $C(t)$ in bytes.

Requests: Let $R_{k,i}(t)$ denotes the set of requests for content $c_k \in C(t)$ received by cache $e_i \in E$ at time t. $|R_{k,i}(t)|$ also denotes the exact total number of such requests at time t. These requests might come from the users in coverage of e_i or requests of other caches redirected to it. We assume each request needs W processing units for fulfillment. Therefore $Req^{Max}(e_i) =$

$\frac{L_{proc}(e_i)}{W}$ specifies the maximum number of requests that e_i can serve simultaneously given that it has the requested content.

Target caches: In this paper, we label a mobile cache that must locate high-priority content as 'Target Cache'. One example of such mobile caches is autonomous vehicles that need to repeatedly cache an updated version of the HD maps (i.e., high-priority contents) [87]. Let $Q(t) = \{e_q\}_{q=1}^{Q}$ denotes the set of target caches for contents $C^{High}(t)$. Note that the set of target caches can change in time. If at least one target cache in the set $Q(t)$ does not have enough free storage to accommodate $C^{High}(t)$, then the content migration strategy should be applied. As a result of applying content migration, a new placement solution for existing low priority contents will be obtained where target caches free space to store the high priority contents. Moreover, our algorithm also determines the best delivery strategy for high-priority content. Our proposed algorithm also determines the best delivery strategy for the high-priority content. Note that, as mentioned earlier, we assume a hierarchical structure in CDN caches so that the contents first arrive at fixed caches (i.e., RSUs) and from there are distributed to mobile caches. Thus, the delivery strategy for high-priority content considers the transmission of $C^{High}(t)$ from fixed caches to target caches.

Delay: The average communication delay $D_{i,j}(t, B)$ for transmitting a data of length B (in bytes) from the edge cache e_j to e_i, where e_j and $e_i \in E$ at time *t* is estimated as follows

$$D_{i,j}(t,B) = \begin{cases} 0 & if\ i = j \\ \frac{B}{£_{i,j}(t)} + \tau_{i,j} & i \neq j, \end{cases} \quad (5\text{-}1)$$

where $£_{i,j}(t)$ and $\tau_{i,j}$ are the data rate and propagation delay between edge cache e_i to e_j at time *t*, respectively. Further, we model the average communication delay between the edge caches and remote cloud server as a fixed value d_∞(which is dominated by the propagation delay, assuming that the remote cloud server is located hundreds of miles away).

Power consumption and bandwidth occupation: Migrating contents from one edge cache to another consumes electrical power, as the source and destination edge caches need to upload and download the content, respectively. We define g_i and $p_i$as the power consumption cost on edge cache e_i for uploading and downloading one byte, respectively. In addition, network bandwidth will be occupied while migrating the contents. we denote $\emptyset$ as the bandwidth occupation cost for transmitting one byte for one unit of distance in the edge network.

Table 5.1 Input Parameters and variables

Network Parameters	
$\mathcal{M}$	Set of M number of mobile edge caches
$\mathcal{F}$	Set of F number of fixed edge caches
$\mathcal{E}$	Set of $N = M + F$ number of edge caches, $\mathcal{E} = \mathcal{M} \cup \mathcal{F}$
$\mathcal{F}^{+}$	Set of fixed caches covering a road intersection
$\mathcal{F}^{-}$	Set of fixed caches covering a straight road segment
g_i	Power consumption cost of e_i for uploading one byte
p_i	Power consumption cost of e_i for downloading one byte
ϕ	Bandwidth cost of one byte transmitted for one hop
d_{∞}	Delay of transmitting a byte from cloud servers to edge caches
$C(t)$	Set of all contents at time t including low- and high-priority contents, $C(t) = C^{\text{High}}(t) \cup C^{\text{Low}}(t)$
$\mathcal{Q}(t)$	Set of target caches for $C^{High}(t)$
$\mathcal{D}(t)$	Set of dominating caches at time t
$v_i(t)$	The instantaneous speed of mobile cache e_i at time t
ℓ_i	Diameter of circular coverage of edge cache e_i
e_i	Edge cache i ($e_i \in \mathcal{E}$)
$R_{\max}(e_i)$	Maximum number of requests e_i can serve simultaneously
$\delta_{i,j}(t)$	Sojourn time of edge cache e_j in coverage of edge cache e_i
$r_{i,j}$	Length of road path in coverage of e_i traversed by e_j when $e_i, e_j \in \mathcal{M}$
$r^{\mathcal{S}}_{f,u}$	The length of straight road path within the coverage of fixed edge cache e_f traversed by $e_u \in \mathcal{M}$
$r^{\mathcal{L}}_{f,u}$	The length of left road path within the coverage of fixed edge cache e_f traversed by $e_u \in \mathcal{M}$
$r^{\mathcal{R}}_{f,u}$	The length of right road path within the coverage of fixed edge cache e_f traversed by $e_u \in \mathcal{M}$
$\mu_{\mathcal{S}}$	The probability that mobile caches follow a straight road.
$\mu_{\mathcal{R}}$	The probability that mobile caches take a right turn.
$\mu_{\mathcal{L}}$	The probability that mobile caches take a left turn.
$L_{stor}(e_i)$	Caching capacity of $e_i \in \mathcal{E}$
$L_{proc}(e_i)$	Processing capacity of $e_i \in \mathcal{E}$
$l_i(t)$	Location of edge cache $e_i \in \mathcal{E}$
γ	Delay (1ms) cost to access one byte of low-priority content
ψ	Delay (1ms) cost to access one byte of high-priority content
$R_{k,i}(t)$	Set of requests for content $c_k \in C(t)$ received by e_i at time t
$D_{i,j}(t, B)$	Delay of transmitting B bytes from e_i to e_j at time t
$\mathcal{L}_{i,j}$	average data rate between edge cache e_i and e_j.
$\tau_{i,j}$	Propagation delay between edge cache e_i and e_j.
λ_{c_k}	Average request rate for content c_k
$\flat_{i,j,l}(t)$	Number of downloaded bytes of c_l^{low} from e_i to e_j. $e_i, e_j \in \mathcal{E}$
$\vartheta_{u,f,h}(t)$	Number of downloaded bytes of c_h^{high} from $e_f \in \mathcal{F}$ to $e_u \in \mathcal{D}(t)$
$\Im_{u,f,h}$	Delay of transmitting c_h^{high} from $e_f \in \mathcal{F}$ to $e_u \in \mathcal{D}(t)$

5.3 Optimization Formulation for Content Migration

In this section, we introduce the set of input parameters and decision variables considered in our formulation and then explain our objective function and constraints. Table 5.1 delineates some of the important inputs and variables used in our formulation.

1) $y_{i,l}(t)$: A binary decision variable which is 1 when low priority content c_l^{low} is in the edge cache e_i at time t (otherwise, it is 0).

2) $x_{f,h}(t)$: A binary decision variable which is 1 when high priority content c_h^{high} is in the fixed edge cache e_f at time t (otherwise, it is 0).

3) $z_{i,j,k}(t)$: An integer decision variable between 0 and Req^{Max} (e_i). This variable identifies the number of requests for content $c_k \in C(t)$ redirected from e_j to e_i at time *t*. Note that for the special cases where $i = j$ the parameter $z_{i,j,l}(t)$ represents the number of requests for content c_k which are received and also directly processed by e_j itself.

5.3.1. Content Migration Cost

Content migration cost at time t consists of three partial cost components, namely $\mathbf{C_1(t)}$, $\mathbf{C_2(t)}$ and $\mathbf{C_3(t)}$. The first partial cost, $C_1(t)$, is the cumulative cost of power consumption associated with uploading contents from edge caches, given by

$$C_1(t) = \sum_{i=1}^{N} \sum_{l=1}^{|C^{low}(t)|} \Lambda_{l,i}.\left(y_{i,l}(t-1) - y_{i,l}(t)\right)^{+} \quad (5-2)$$

Where

$$(A-B)^{+} = \begin{cases} 1, & if\ A > B \\ 0, & otherwise. \end{cases} \quad (5-3)$$

For each content c_l^{low}, $\left(y_{i,l}(t-1) - y_{i,l}(t)\right)$ is equal to 1 when contentc_l^{low} is uploaded from e_i. In this case, the non-negligible cost of $\Lambda_{l,i} = size(c_l^{low}).g_i$ will be imposed on the system due to content uploading.

Similarly, the cumulative cost of power consumption $\mathbf{C_2}(\boldsymbol{t})$ associated with downloading contents from edge caches at time *t* is given by

$$\mathrm{C}_2(t) = \sum_{i=1}^{N} \sum_{l=1}^{|C^{low}(t)|} \mathrm{V}_{l,i}.\left(y_{i,l}(t-1) - y_{i,l}(t)\right)^{+} \qquad (5-4)$$

In this case, the non-negligible cost of $\mathrm{V}_{l,i} = size(c_l^{low}).p_i$ will be imposed on the system due to content downloading.

The cost $\mathbf{C_3}(\boldsymbol{t})$ of bandwidth occupation involved in migrating contents between edge caches is given by

$$\mathrm{C}_3(\boldsymbol{t}) = \sum_{i,j=1,i\neq j}^{N} \sum_{l=1}^{|C^{low}(t)|} \boldsymbol{\Delta_l}.\left|\boldsymbol{l_i}(\boldsymbol{t}) - \boldsymbol{l_j}(\boldsymbol{t})\right|.\left(\boldsymbol{y_{i,l}}(\boldsymbol{t-1}) - \boldsymbol{y_{i,l}}(\boldsymbol{t})\right)^{+}.\left(\boldsymbol{y_{j,l}}(\boldsymbol{t}) - \boldsymbol{y_{j,l}}(\boldsymbol{t-1})\right)^{+} \qquad (5\text{-}5)$$

Where the term $\left(y_{i,l}(t-1) - y_{i,l}(t)\right)^{+}.\left(y_{j,l}(t) - y_{j,l}(t-1)\right)^{+}$ becomes non-zero only when content c_l^{Low} is uploaded from e_i and downloaded into e_j. Moreover, letting Φ be the bandwidth occupation cost for transmitting one byte over a unit of distance, $\Delta_l = Size\ (c_l^{Low}).\Phi$ will be equal to the bandwidth occupation cost for transferring c_l^{Low} over a unit of distance. As (5-5) suggests, the associated bandwidth occupation cost can be calculated by multiplying Δ_l by the distance $\left|l_i(t) - l_j(t)\right|$ between source and destination. The content migration cost, C_M is then obtained by summing the three partial costs $\mathbf{C_1}(\boldsymbol{t})$, $\mathbf{C_2}(\boldsymbol{t})$, and $\mathbf{C_3}(\boldsymbol{t})$ accumulated over the observation time period $[t_0, t_k]$ as follows:

$$\mathrm{C}_M = \int_{t_0}^{t_k} [\mathrm{C}_1(\mathrm{t}) + \mathrm{C}_2(\mathrm{t}) + \mathrm{C}_3(t)].dt \qquad (6)$$

5.3.2. Delay cost of low-priority contents

To calculate the delay cost of low-priority contents, we assume that the content popularity follows a Zipf distribution [56] [59], with α being the Zipf slope ($0 < \alpha < 1$). Assuming c_l^{low} is the l'th most popular content, the probability of content c_l^{low} being requested is $\frac{1}{\rho.l^{\alpha}}$, where $\rho = \sum_{l=1}^{size(C^{Low}(t))} 1/l$. With the assumption that the low-priority content requests follow a Poisson process with parameter β [56][59], the average request rate λ_l of content c_l^{low} can be calculated by $\lambda_l = \frac{\beta}{\rho.l^{\alpha}}$. The delay cost $\mathbf{C_A}$ of accessing low-priority contents is given by:

$$\mathrm{C}_A = \int_{t_0}^{t_k} \sum_{i,j=1,i\neq j}^{N} \sum_{l=1}^{|C^{Low}(t)|} \boldsymbol{\lambda_l}.\boldsymbol{\gamma}\left[\boldsymbol{D_{i,j}}(\boldsymbol{t}, \mathrm{P}_{i,j,l}).\boldsymbol{z_{i,j,l}}(\boldsymbol{t}).\boldsymbol{y_{i,l}}(\boldsymbol{t}) + \boldsymbol{d_{\infty}}.\left(|\boldsymbol{R_{l,i}}(\boldsymbol{t})| - \boldsymbol{z_{i,j,l}}(\boldsymbol{t}).\boldsymbol{y_{i,l}}(\boldsymbol{t})\right)\right].\mathbf{dt}, \qquad (5\text{-}7)$$

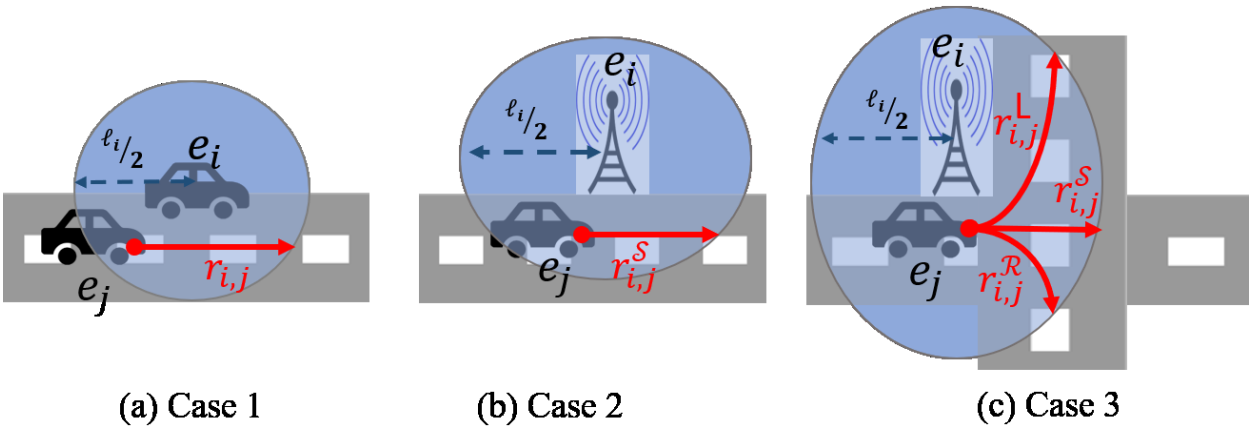


Figure 5.2 Illustration of the three different cases for calculating the sojourn time $\boldsymbol{\delta_{i,j}(t)}$

Where the term $z_{i,j,l}(t).y_{i,l}(t)$ counts the number of requests for content c_l^{Low} that are sent to the edge cache e_iAnd $\boldsymbol{\gamma}$ is the cost of a unit delay for accessing one byte of low-priority content. However, some of the requests may not be fulfilled successfully, which occurs when the receiving edge node leaves the radio coverage of the transmitter node before the whole content has been transmitted.

To compute the successfully transmitted bytes of c_l^{Low}We first obtain the sojourn time $\delta_{i,j}(t)$ in the coverage area of e_i. Depending on the type of the caches (i.e., fixed or mobile) and their area, we may deal with one of the following three cases to calculate the sojourn time $\delta_{i,j}(t)$ as shown in Fig. 5.2.

Case 1: The two caches e_j and e_i are both mobile, i.e., e_j,$e_i \in M$ (see Fig 5.2a). In this case, the sojourn time $\delta_{i,j}(t)$ is given by Eq. (5-8a)

$$\delta_{i,j}(t) = [\tfrac{\ell_i}{2} - |l_i(t) - l_j(t)|]^+ . \frac{r_{i,j}}{|v_i(t) - v_j(t)|}, \quad \text{if } e_j, e_i \in M \qquad (5\text{-}8a)$$

In Eq. (5-8a) $r_{i,j}$ is the length of road path within the coverage of e_i traversed by e_j (see Fig. 5.2a.) In this equation, the term $[\tfrac{\ell_i}{2} - |l_i(t) - l_j(t)|]^+$is equal to 1 only if e_j resides within the coverage of e_i at time t; otherwise, it is zero. Also, $|v_i(t) - v_j(t)|$ is the relative speed of e_j with respect to e_i.

Case 2: In this case, e_j is mobile, whereas e_i is a fixed cache covering a straight road segment; i.e., $e_j \in M$ and $e_i \in F^-$ (see Fig. 5.2b). We can then calculate $\delta_{i,j}(t)$ as follows:

$$\delta_{i,j}(t) = [\frac{\ell_i}{2} - |l_i - l_j(t)|]^+ . \frac{r^{s}_{i,j}}{v_j(t)}, \quad \text{if } e_j \in M \text{ and } e_i \in F^- \tag{5-8b}$$

Where $r^{s}_{i,j}$ is the length of straight road path within the coverage of fixed edge cache e_i traversed by e_j (see Fig. 5.2b.).

Case 3: In the third case, e_j is mobile and e_i is a fixed cache covering a road intersection; i.e., $e_j \in M$ and $e_i \in F^+$ (see Fig. 5.2c). We can then calculate $\delta_{i,j}(t)$ as follows:

$$\delta_{i,j}(t) = [\frac{\ell_i}{2} - |l_i - l_j(t)|]^+ . \frac{\eta_s r^{s}_{i,j} + \eta_L r^{L}_{i,j} + \eta_R r^{R}_{i,j}}{v_j(t)}, \text{ if } e_j \in M \text{ and } e_i \in F^+ \tag{5-8c}$$

where $r^{s}_{i,j}$ is the length of straight road path within the coverage of fixed edge cache e_i traversed by e_j, while η_s is the probability that mobile caches follow a straight road. $r^{L}_{i,j}$ is the length of left road path within the coverage of fixed edge cache e_i traversed by e_j, while η_L is the probability that a mobile cache takes a left turn. Further, $r^{R}_{i,j}$ is the length of the right road path within the coverage of fixed edge cache e_i traversed by e_j, while η_R is the probability that a mobile cache takes a right turn. We note that upon facing an intersection along its path, a mobile cache follows the straight road or takes a left or right turn with probabilities η_s, η_L , and η_R , respectively.

By setting $D_{i,j}(t, B) = \delta_{i,j}(t)$ in Eq. (5-1), the number of bytes that can be transferred from e_i to e_j can be computed (i.e., the term $[\delta_{i,j}(t) - \tau_{i,j}].\pounds_{i,j}(t)$). Specifically, the number of successfully transmitted bytes of c_l^{Low} from edge cache e_i to e_j can be obtained as follows:

$$Þ_{\boldsymbol{i,j,\iota}}(t) \;=\; min\{\boldsymbol{y_{i,l}(t)}.([\delta_{i,j}(t) - \tau_{i,j}].\pounds_{i,j}(t), size(c_l^{Low})\} \tag{5-9}$$

It should be noted that in Eq. (5-9), any requests for the remaining bytes of c_l^{Low} that can not be served from any edge caches are assumed to be redirected to cloud servers for fulfillment.

5.3.3. Delay cost of high-priority contents

Target cache $e_q \in Q(t)$, can download the high priority contents $C^{High}(t)$ either directly from fixed caches or from other target caches that have already received the high priority contents.

However, if e_q can not receive $C^{High}(t)$ from any other edge caches due to, for instance, their isolated location or high speed, it should download all or the remaining parts of those contents from CDN cloud servers.

We define the so-called dominating cache Ɗ(t) as a subset of target caches that can transmit the contents to the rest of the target caches with at most σ hops. Note that Ɗ(t) $\subseteq Q(t)$ and that it is identified by means of graph theory. Let the contact graph $G(t) = (e_q|\forall q \in Q(t), E(t))$ be the representation of the target caches' topology at time *t*, where E(t) is the set of edges showing the connectivity among the target caches. In this regard, edge $\xi_{q,p}(t) \in E(t)$ exists if and only if the target caches e_q to e_p are in the transmission range of each other at time *t*. The set of dominating nodes in the contact graph can be identified in polynomial time by the algorithm proposed in [58].

Depending on its path, dominating cache $e_u \in D(t)$ receives a high-priority content $c_h^{high} \in C^{High}(t)$ in a continuous manner while switching from one fixed cache range to another. The number of bytes in content c_h^{high} downloaded from e_f varies according to the amount of time $e_u \in D(t)$ spends in the coverage area of the fixed cache e_f. Recall that $\delta_{i,j}(t)$ (estimated by Eq. (5-8)) is the duration time that $e_u \in D(t)$ remains within the coverage range of fixed cache e_f. Using $\delta_{u,f}(t)$, the number of successfully transmitted bytes of content c_h^{high} from fixed cache e_f to target cache $e_u \in D(t)$ can be calculated as follows:

$$\mathrm{V}_{u,f,h}(\mathrm{t}) = \min\left\{x_{f,h}(t)\,.\,([\delta_{i,j}(t) - \tau_{i,j}].\pounds_{i,j}(t)), size(\,c_h^{high})\right\} \tag{5-10}$$

Considering (5-1), the delay in downloading high-priority content c_h^{high} from the fixed edge cache e_f to $e_u \in D(t)$ can be obtained as:

$$\Im_{\,u,f,h} = D_{u,f}(t, \mathrm{V}_{u,f,h}(\mathrm{t})) \tag{5-11}$$

We note that when $e_u \in D(t)$ is out of the range of any fixed cache, the request should be redirected to the cloud, and the remaining portions of the high-priority content downloaded from CDN cloud servers. The high-priority content delivery cost for target caches can then be obtained as follows:

$$C_D = \int_{t_0}^{t_k} \left(\sum_{u|e_u \in D(t)} \sum_{h=1}^{|C^{High}(t)|} \sum_{f=1}^{\mathrm{F}} . \left[\Im_{\,u,f,h} + \left[|R_{h,f}(t)| - z_{f,u,h}(t).x_{f,h}(t)\right].d_\infty\right]\psi \right).dt \tag{5-12}$$

Where ψ is the delay cost per second of downloading one byte of high-priority content.

5.3.4. Objective Function and Constraints

The objective is to minimize the total cost as an aggregation of content migration cost and content delay cost in a given CDN system. Let w_M, w_A and w_D denote the weights of the costs $C_M, C_A, and\ C_D$, respectively. The objective function Φ over the observation time period $[t_0, t_k]$ is then given by

$$min\ \Phi = w_M.\mathrm{C}_M + w_A.\mathrm{C}_A + w_D.\mathrm{C}_D \qquad (5-13)$$

subject to the following constraints:

$$\sum_{l=\{1,..|C^{low}(t)|\}} \sum_{j=\{1,..|M\}} z_{i,j,l}(t).\, y_{i,l}(t) \le R_{max}(e_i),\ \ \forall\, t_0 \le t \le t_T,\ \forall 1 \le i \le M \qquad (5-14a)$$

$$\sum_{u|e_u \in D(t)} \sum_{h=\{1,..,|C^{High}(t)|\}} \left(\left[{}^{\ell_f}/_{2} - \left(l_u(t) - l_f\right) \right]^{+} .\, x_{f,h}(t) \right) + \qquad (5\text{-}14b)$$

$$\sum_{l=\{1,..,|C^{Low}(t)|\}} \sum_{j=\{1,..|M\}} z_{f,j,l}(t).\, y_{f,l}(t) \le R_{max}(e_f) \qquad \forall\, t_0 \le t \le t_k,\ \forall 1 \le f \le F$$

$$\sum_{l=\{1,..|C^{low}(t)|\}} y_{v,l}(t).\, Size(c_l^{Low}) \le \quad L_{stor}(e_v) \qquad \forall\, t_0 \le t \le t_k,\quad \forall e_v \in \{M - Q(\mathrm{t})\} \qquad (5\text{-}14c)$$

$$\sum_{l=\{1,..|C^{low}(t)|\}} y_{q,l}(t).\, Size(c_l^{Low}) + size(C^{High}(t))) \le L_{stor}\left(e_q^{MEC}\right) \qquad (5\text{-}14d)$$

$$\forall\, t_0 \le t \le t_k,\quad \forall q | e_q \in Q(t)$$

$$\sum_{l=\{1,..|C^{low}(t)|\}} y_{f,l}(t).\, Size(c_l^{Low}) + \sum_{h=\{1,..|C^{High}(t)|\}} x_{f,h}(t).\ \ \le L_{stor}\left(e_f^{FEC}\right) \qquad (5\text{-}14e)$$

$$\forall\, t_0 \le t \le t_k, \forall f | e_f \in F$$

Constraint (5-14a) ensures that the maximum number of content requests that can be served simultaneously from the mobile edge cache $e_i \in M$ is not exceeded. Similarly, constraint (5-14b) indicates a set of constraints on the number of requests for contents that can be handled

simultaneously by the fixed cache $e_f \in F$. Note that $\left[{\ell_f}/{2} - \left(l_u(t) - l_f\right)\right]^+$ computes the number of dominating caches covered by e_f at time *t*. Therefore, the number of requests for downloading high-priority contents is calculated by the first term in constraint (5-14b), while the second term computes the number of requests for low-priority contents. Constraint (5-14c) represents the capacity constraints for non-target caches $e_v \in \{M - Q(t)\}$. Similarly, the capacity constraints of target mobile caches are specified by constraint (5-14d). Note that each target mobile cache, in addition to the contents already cached in it, should also have space for high-priority contents; this is ensured by constraint (5-14d). Finally, constraint (5-14e) represents the capacity constraints on fixed caches.

5.4 RL-based Content Migration

Here we define the main components of the MDP in our content migration problem.

5.4.1 System States

The state of the system at time *t* should represent (i) the placement of both low- and high-priority contents on the edge caches at that time and (ii) the content delivery state of the system. The delivery state of the system is defined as the participation level of each edge cache in the delivery of requested contents. This participation level is quantified by the number of redirection requests that each edge cache performs at time *t*. To formally present the system states set, we define $\mathbb{Y}^{c_l}(t)$, $\mathbb{X}^{c_h}(t)$ and $\mathbb{Z}^{c_l}(t)$ as the realization sets of random variables $y_{i,l}(t), x_{f,h}(t), z_{i,j,l}(t)$ at time *t*, respectively. We note that $\mathbb{Y}^{c_l}(t)$, $\mathbb{X}^{c_h}(t)$ and $\mathbb{Z}^{c_l}(t)$ are given by:

$$\mathbb{Y}^{c_l}(t) = [Y_{1,l}(t), Y_{2,l}(t) \dots, Y_{N,l}(t)] \tag{5-15}$$

$$\mathbb{X}^{c_h}(t) = [X_{1,h}(t), X_{2,h}(t) \dots, X_{F,h}(t)] \tag{5-16}$$

and

$$\mathbb{Z}^{c_l}(t) = \begin{matrix} [Z_{1,1,l}(t) & Z_{1,2,l}(t) & \dots, & Z_{1,N,l}(t) \\ \vdots & \ddots & & \vdots \\ Z_{N,1,l}(t) & Z_{N,2,l}(t), & \dots, & Z_{N,N,l}(t)] \end{matrix} \tag{5-17}$$

where $Y_{i,l}(t), X_{f,h}(t), and\ Z_{i,j,l}(t)$ are the exact values of random variables $y_{i,l}(t), x_{f,h}(t), and\ z_{i,j,l}(t)$ at time *t*, respectively. We then encapsulate $\mathbb{Y}^{c_l}(t)$, $\mathbb{X}^{c_h}(t)$ and $\mathbb{Z}^{c_l}(t)$ in the vector $\mathcal{X}^{c_h,c_l}$ given by

$$\mathcal{X}^{c_h,c_l} = [\mathbb{Y}^{c_l}(t),\ \mathbb{X}^{c_h}(t) \text{ and } \mathbb{Z}^{c_l}(t)\ ] \tag{5-18}$$

Finally, the state s_t of the system at time *t* can be calculated as follows:

$$S_t = [\cup \chi^{c_h,c_l}(t) \colon c_h \in C^{High}(t), c_l \in C^{Low}(t)] \tag{5-19}$$

5.4.2 System Actions

The agent can take action by migrating, caching, or dropping the contents from edge caches or by redirecting requests between them. To better explain these possible actions, we divide them into three types. The first type of action is to migrate, cache, or drop the low-priority content $c_l \in C^{Low}(t)$ in edge caches (both fixed and mobile caches). We refer to this action type as "Act.Type1". This type of action will cause changes in the values of $\mathbb{Y}^{c_l}(t+1)$ with respect to $\mathbb{Y}^{c_l}(t)$. To represent this type, a binary vector $a_{c_l}^{\mathbb{Y}}$(t) of size N is used, where N is the total number of edge caches. A value of 1 for the i'th element of the vector $a_{c_l}^{\mathbb{Y}}$(t) indicates a zero to one or vice versa change in the i'th element of $\mathbb{Y}^{c_l}(t)$ (i.e. $Y_{i,l}(t)$'s value), whereas a value of 0 would indicate no change in the value of $Y_{i,l}(t)$. The second type of action, "Act.Type2", is the caching (or dropping) of the high-priority content $c_h \in C^{High}(t)$ on (from) fixed caches for their later delivery to targeted caches. The effect on the values of $\mathbb{X}^{c_h}(t)$ will be similar to that of "Act.Type1", and we represent it by a binary vector $a_{c_h}^{\mathbb{X}}$(t) of size F, where F is the number of fixed edge caches. The third type of action, "Act.Type3", considers redirecting the low-priority contents' requests between edge caches. This type of action affects the values of $\mathbb{Z}^{c_l}(t)$ and we denote it by a binary matrix $a_{c_l}^{\mathbb{Z}}$(t) of size N×N. Thus, three types of action can be recognized, "Act.Type1", "Act.Type2" and "Act.Type3", implemented by $a_{c_l}^{\mathbb{Y}}$(t), $a_{c_h}^{\mathbb{X}}$(t), and $a_{c_l}^{\mathbb{Z}}$(t) respectively, each indicating the possible changes in the corresponding state vectors' values. The overall action a_t at time *t* is then summarized by

$$a_t = \{< a_{c_l}^{\mathbb{Y}}(\mathrm{t}), a_{c_h}^{\mathbb{X}}(\mathrm{t}), a_{c_l}^{\mathbb{Z}}(\mathrm{t}) > \mid c_l \in C^{Low}(t)\,, c_h \in C^{High}(t)\} \qquad (5\text{-}20)$$

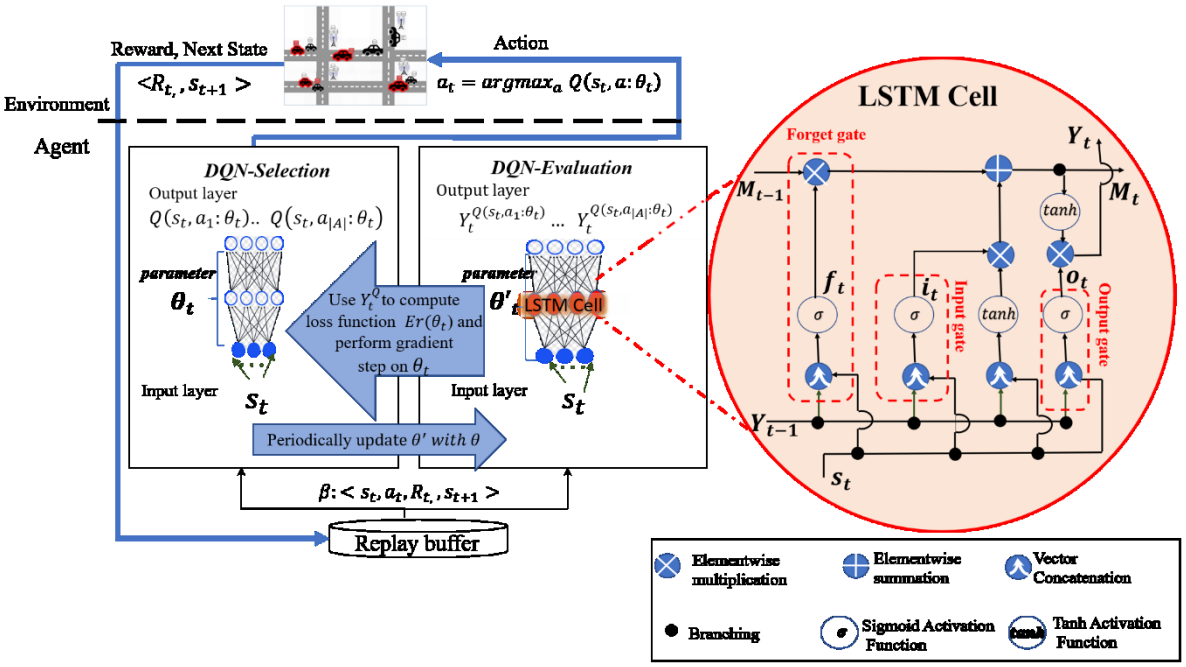


Figure 5.3 A schematic view of the agent and its interactions with the environment, including the structure of our deployed LSTM cell

5.4.3 Reward Function

Upon performing an action, the agent needs an immediate feedback to assess the short-term quality of the performed action. This feedback is quantified by the value of a reward function. Our reward function $\mathbb{R}(s_t, a_t)$is given by

$$\mathbb{R}(s_t, a_t) = -(\mathrm{C}_M(t) + \mathrm{C}_A(t) + \mathrm{C}_D(t)) \qquad (5\text{-}21)$$

which is defined based on the aggregation of the migration cost $\mathrm{C}_M(t)$, the low-priority contents' access delay $\mathrm{C}_A(t)$, and the high-priority contents download cost $\mathrm{C}_D(t)$ the current time slot. Note that the reward function computed by (5-21) only quantifies the short-term impact of the performed action a_t as an immediate feedback.

5.4.4 Design of the deep RL agent

Unlike non-learning approaches, the RL agent automatically learns the ever-changing environment and updates its decisions through its interactions. Figure 5.3 illustrates a schematic view of our agent and these interactions. We will refer to Fig. 5.3 and Algorithm 1 as we explain the theoretical steps of our work. Specifically, our approach is based on Q-learning, one of the most widely used RL strategies [20]. Q-learning works by successively updating the evaluation of the long-term quality (the Q value) of actions at each state. It is a simple way for an agent to learn how to act optimally [20]. We note, however, that classic Q-learning is limited to tasks with a small number of states and actions [20][21]. Moreover, in the Q-learning algorithm, all the states should be met, and all the actions should be experienced. Those restrictions are impractical in our problem, as it deals with an environment that is extremely complex and dynamic, and its states are large and vary rapidly over time. The only way to learn anything in these types of dynamic situations (where we have dynamic state-space) is to generalize from previously experienced states to new states [20]. The required generalization is often called function approximation. In this study, to approximate the Q values for unmet states/actions, we use a deep neural network (DNN)-based approach, which relies on nonlinear gradient-descent function approximation. This approach eliminates the need for visiting all the state/action pairs to compute the Q values. First proposed in [86], this revival hybrid approach is now widely used in different domains under the so-called deep reinforcement learning (DRL) or deep Q-learning (DQL) method.

In this work, since our problem concerns a sequential decision-making process, we exploit an advanced version of DQL, a double deep Q-network (DDQN) with LSTM memory cells. In the rest of this section, we first explain the motivation for choosing this specific Q network architecture, and then we discuss the limitations of conventional recurrent neural networks and explain how LSTM memory cells can overcome those limitations, ending with our DDQN algorithm for content migration.

5.4.5 DDQN

Q-Learning is a model-free reinforcement algorithm to estimate Q values for state-action pairs. The Q value of a state-action pair can be interpreted as an expected discounted reward accumulated over a long time period. As an example, in a given state $s \in S$ (S being the set of all the states) with two possible actions $a_1, a_2 \in A$ (A being the set of all the possible actions), if $Q(s, a_1) > Q(s, a_2)$, then choosing a_1 over a_2 will result in a higher accumulated reward over the long term. The detailed mathematical explanation can be found within the well-known Bellman equation [20].

The Q-Learning algorithm starts by initializing the Q values for all state-action pairs by setting them to zero. Next, it recursively computes and updates the Q value of a given pair as follows:

$$Q^{new}(s,a) = Q^{old}(s,a) + \alpha.[R + \gamma.max_a Q(s',a) - Q^{old}(s,a)] \qquad (5\text{-}22)$$

Where R is the reward of performing action $a \in A$ in state $s, s' \in S$ is the next state, $\alpha \in [0,1]$ denotes the learning rate, and $\gamma \in [0,1]$ is the discounting rate. The Q update continues until all the states are met, and all the actions have been experienced. At this point, the final Q value, Q∗(s, a), determines the best action $a^* \in$ A at a given state as follows:

$$a^* = argmax_a Q^*(s,a)$$

It is important to note that in our considered problem, there is no final state, especially given that the states vary over time (mobile caches move and new high-priority contents arrive). Therefore, an approximation method is required for the Q values of unmet states [21]. In a deep Q network (DQN), a multi-layered neural network is utilized to estimate the Q values. At state s_tThe learning agent takes action a_t based on policy ε, which is initially purely random and gradually improves as the agent becomes more experienced. Let us denote the reward and the resulting state as R_t and s_{t+1}, respectively. The tuple $e_t =< s_t, a_t, R_{t+1}, s_{t+1} >$ represents the experience of the agent at time *t* stored in a buffer called the experience replay buffer. Periodically, the samples of the agent's experience will be drawn randomly to form the learning batches. These learning batches are then used to feed the DQN and update the estimated Q values.

For a given state-action pair $<s_t, a_t>$, $Q(s_t, a_t; \theta_t)$is the DQN current estimation of the Q value. Here, θ_t is the parameter of the Q network at time *t*. The gradient descent update rule for the parameter θ_t will be applied as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^Q - Q(s_t, a_t; \theta_t)).\nabla_{\theta_t} Q(s_t, a_t; \boldsymbol{\theta}_t) \qquad (5\text{-}23)$$

where α is the gradient step size and Y_t^Q denotes the target Q value with the current parameter θ_t, which is calculated by

$$Y_t^Q = \boldsymbol{R}_t + \gamma \max_a Q(s_{t+1}, a; \boldsymbol{\theta}_t) \qquad (5\text{-}24)$$

With the update rule (5-23), the parameter θ_t of the DQN will be tuned so that $Q(s_t, a_t; \theta_t)$ moves towards Y_t^Q with step size α. Note, however, that in doing so, Y_t^Q itself is computed by the maximum value of $Q(s_{t+1}, a; \theta_t)$. This loop, in turn, will cause an over-optimistic and unstable approximation of the Q values, which can degrade the accuracy of the results [21]. This can be avoided using a technique first proposed by Van Hasselt [21], where two DQNs are trained in parallel. The first DQN, $Q^{Select}(s, a; \theta_t)$, referred to as *DQN-Selection*, with parameter θ_t is used for the selection of actions, whereas the second DQN, $Q^{Eval}(s, a; \theta_t)$, referred to as *DQN-evaluation*, with parameter θ'_t is trained for the evaluation of the actions. With these settings, the target Q value for the DDQN will be computed as follows:

$$Y_t^Q = \boldsymbol{R}_t + \gamma \,.\, Q^{Eval}(s_{t+1}, argmax_a Q^{Select}(s_{t+1}, a; \theta_t); \boldsymbol{\theta'}_t) \qquad (5\text{-}25)$$

Accordingly, the error function Er(θ_t) of DDQN at time *t* is given by:

$$Er(\boldsymbol{\theta}_t) = \frac{1}{2}[Y_t^Q - Q^{Select}(s_{t+1}, a; \theta_t)]^2 \qquad (5\text{-}26)$$

After each forward pass, Er(θ_t) will be recalculated. Following the back propagation procedure and the derivation chain rule, the contribution of each DDQN parameter to the error will be obtained. The gradient descent update rule that is given by Eq. (5-23) uses this calculated value to update the parameters. Periodically, the values of θ_t will be copied to θ'_t. After sufficient training, the parameters will be tuned such that the error value becomes quite small. For illustration, we depict the interactions of two networks Q^{Select} and Q^{Eval} in Fig. 5.3. The decoupling of the selection and evaluation Q-networks in the learning process has proven to be successful in reducing over-optimism and therefore producing more stable and reliable learning results [21].

5.4.6 DDQN with LSTM cells

As the DDQN continues to learn, the impact of some important experiences in the distant past could be replaced by more recent experiences. This problem, which is also referred to as the vanishing gradient [21], [86], [88], is a well-known obstacle in the learning path of gradient-based

approaches such as RNN [90]. The vanishing gradient makes the learning process time-consuming and may lead to inaccurate results [90]. Consider the gradient update rule in a DNN given by Eq. (5-23). After passing many gradient update steps and when *t* becomes large enough, the error and the gradient term $(Y_t^Q - Q(s_t, a_t; \theta_t)).\nabla_{\theta_t} Q(s_t, a_t; \boldsymbol{\theta}_t)$ becomes so small that the values of $\boldsymbol{\theta}_t$ do not change significantly. Insufficient decaying error backflows to the initial layers of the neural network, thus hampering the learning process [88]. To avoid this issue, the authors of [90] have suggested using long short-term memory (LSTM) cells, which are deployed in the hidden layers of the given DNN to ensure the flow of decaying error in the backpropagation process in later learning steps, thereby allowing the learning process to continue. It is worth noting that the LSTM architecture is now widely used in many DNN applications [86], [88], [90] and has been proven to outperform the simple feedforward DNNs [90].

Figure 5.3 depicts the structure of our deployed LSTM cell. The LSTM cell is comprised of three inputs, M_{t-1}, Y_{t-1}, and s_t, which are the previous memory state of the cell, the previous output of the cell (i.e., the previous predicted value), and the current input of the network, respectively. The two inputs M_{t-1} and Y_{t-1} of the cell are initialized to be all zeros at time t=0. The LSTM cell outputs two vector values, $Y_t, and\ M_t$ which are the current output (i.e., predicted value), and the current memory state of the cell, respectively. As shown in Fig. 5.3, an LSTM cell consists of three gates: (i) forget, (ii) input, and (iii) output, each containing a sigmoid activation function denoted by $\sigma(x) = (1 + e^{-x})^{-1}$ The output of the sigmoid functions of the forget, input, and output gates are f_t, i_t, and o_t, respectively. Each of these activation functions has its own weights and bias as follows: W_f and b_f for the forget gate, W_i and b_i for the input gate, and W_o and b_o for the output gate. All these parameters are randomly initialized at the beginning. With these settings, the forward pass formulas of an LSTM cell are as follows:

$$f_t = \sigma\big(W_f[Y_{t-1}, s_t] + b_f\big),$$

$$i_t = \sigma(W_i[Y_{t-1}, s_t] + b_i),$$

$$o_t = \sigma(W_o[Y_{t-1}, s_t] + b_o),$$

$$M_t = M_{t-1} \otimes f_t \oplus (i_t \otimes \tanh([Y_{t-1}, s_t])),$$

$$Y_t = \tanh(M_t) \otimes o_t,$$

where $[Y_{t-1}, s_t]$ is the concatenation of vectors Y_{t-1} and s_t, while the element-wise multiplication and summation are denoted as $\otimes$ and $\oplus$, respectively, and $tanh$ is the hyperbolic tangent function. During training, the cell parameters W_i, b_i, W_o, b_o, W_f, and b_f are tuned using

Algorithm 1 DDQN-based Content Migration

Initialize the ***DQN-selection*** network with $\theta_t = \theta_0$
Initialize the ***DQN-evaluation*** network with $\theta'_t = \theta'_0$
Initialize two vectors $a^{\mathbb{Y}}_{c_l}$, $a^{\mathbb{X}}_{c_h}$ of size N and a matrix $a^{\mathbb{Z}}_{c_l}$ of size $N \times N$.
Initialize replay buffer $\mathcal{D}$
for episode $k = 1$ to K **do**
    Observe the current content placement and delivery state of the environment and construct s_1 as Eq. (19).
    for time slot $t = 1$ to T **do**
        Generate a random number $\lambda \in [0, 1]$
        if $(\lambda < \varepsilon)$ **then** ▷ Exploration phase
            reset $a^{\mathbb{Y}}_{c_l}(t)$, $a^{\mathbb{X}}_{c_h}(t)$, and $a^{\mathbb{Z}}_{c_l}(t)$ with zero values.
            $Action\_Types = [\texttt{Act.Type1}, \texttt{Act.Type2}, \texttt{Act.Type3}]$
            Choose a random integer $i \in \{0, 1, 2\}$
            if $(Action\_Types[i] == \texttt{Act.Type1})$ **then**
                Choose a random integer number $j \in \{0, \ldots, N-1\}$
                Assign a value of 1 to the j^{th} element of $a^{\mathbb{Y}}_{c_l}(t)$.
            else if $(Action\_Types[i] == \texttt{Act.Type2})$ **then**
                Choose a random integer number $l \in \{0, \ldots, N-1\}$
                Assign a value of 1 to the l^{th} element of $a^{\mathbb{X}}_{c_h}(t)$.
            else if $(Action\_Types[i] == \texttt{Act.Type3})$ **then**
                Choose two random integer numbers $m, n \in \{0, \ldots, N-1\}$
                Assign a value of 1 to the $(m, n)^{th}$ element of $a^{\mathbb{Z}}_{c_l}(t)$.
            end if
            construct a_t as Eq. (20).
        else ▷ Exploitation phase
            Choose a_t which maximizes $Q^{Select}(s_t, a_t; \theta_t)$
        end if
        Execute a_t
        Observe the next state s_{t+1} and reward R_t
        Store $< s_t, a_t, R_t, s_{t+1} >$ in replay buffer $\mathcal{D}$
        Get a batch of experiences β randomly sampled from $\mathcal{D}$
        Set target Q value Y^Q_t :
            $Y^Q_t = R_t + \gamma \cdot Q^{Eval}(s_{t+1}, \text{argmax}_a Q^{Select}(s_{t+1}, a; \theta_t); \theta'_t)$
        Set the average error function $Er(\theta_t)$:
            $Er(\theta_t) = \frac{1}{\beta} \sum_{i=1}^{\beta} (Y^Q_i - Q^{Select}(s_i, a_i; \theta_t))^2$
        Update θ_t with the gradient descent update rule:
            $\theta_{t+1} = \theta_t - \alpha \frac{\partial Er(\theta_t)}{\partial \theta_t}$
        Every $\bar{\tau}$ iterations set $\theta'_t = \theta_t$
    end for
end for

the back propagation, and stochastic gradient descent update rules explained in Section 5.4.4. Note that in our proposed algorithm, the LSTM cell is embedded in the hidden layers of the *DQN-Evaluation*, as shown in Fig. 5.3.

Algorithm 1 illustrates the main steps of our double deep Q-Learning algorithm used for solving our content migration problem. The algorithm starts with observing the initial state, s_1. A series of iterations are then followed while the algorithm switches between exploration and exploitation phases. Parameters ϵ (exploration rate) and λ (a random value in the range $[0,1]$) are used to control these phases. A random action and its type are selected in the exploration phase (see lines 10-23), while in the exploitation phase, *DQN-selection* will determine the action (see line 25). The action, reward, and next state are then collected and stored in buffer $\mathcal{D}$ (line 29). A batch of experiences is then randomly retrieved from $\mathcal{D}$ (line 30). The target value of *DQN-selection* (i.e., Y_t^Q) can be computed by the use of *DQN-evaluation* (line 31). This target value will be used for computing the error function $Er(\theta_t)$Which is the average error of all samples of β (line 32). The parameters of *DQN-selection* will be updated by performing a gradient descent step on $Er(\theta_t)$ with respect to θ_t (line 33). Finally, every $\bar{\tau}$ steps, the parameters of the *DQN-selection* are copied to *DQN-evaluation* (line 34).

5.5 Performance Evaluation

To ensure that our simulated evaluations are conducted based on realistic scenarios, we used the SUMO (Simulation for Urban MObility) simulator [91]. As for the DDQN, we used TensorFlow 1.6.0 [76], Google's open-source machine learning library. In particular, we utilize "*tf.contrib.rnn.LSTMCell*" and "*keras.models*" classes to instantiate the two four-layer DNNs, *DQN-selection* and *DQN-Evaluation*, with LSTM cells in hidden layers of the latter DNN. To assure convergence, we rely on the Keras class "*ReduceLRONplatue*" to automatically update the learning rate. All the simulation tests were conducted on a machine with 2.67 GHz Intel Xeon CPU E5640 and 32 GB of memory.

5.5.1 Simulation Settings

In our simulations, we considered an n × n bidirected road grid environment [92] where each grid cell covers an area of 0.25 km^2. The number of grid cells, mobile caches, and fixed caches are specified in each evaluation scenario. In this grid structure, mobile caches move with an average

velocity of 30 km/hour, and with the parameters η_S = 0.5, η_L= 0.25 and $\eta_{\mathcal{R}}$ = 0.25, set according to the Manhattan model, the most popular model for mobility in urban areas [92]. Each fixed cache has a capacity of 1.5 GB of memory and four processing units. Each mobile cache was provided with a 700 MB memory and one processing unit.

We assume that the fixed edge caches can handle up to 20 requests at a time and that each mobile cache can handle a maximum of 5 requests simultaneously. While the fixed edge caches have a 60-meter diameter circular coverage, the mobile caches can cover a circular area of only 10 meters in diameter. In addition, 40 low-priority contents of various sizes from 50 to 120 MB are randomly placed so that 8 of the mobile caches are fully occupied. In the beginning, we assume that 5 of these full mobile caches are targeted caches.

In the simulation scenario, high-priority contents arrive according to a Poisson process with an average arrival rate of 5 contents per time unit. The sizes of these contents are similar to the sizes of the low-priority contents' ranges: from 50 to 120 MB. We set the delay for transmitting 1 MB of content (high or low priority) from cloud servers to an edge cache node (fixed or mobile) and the delay from each cache node to the users of its coverage as 0.5 and 0.2 milliseconds [93], respectively.

The average cost of the power consumption required by edge caches to upload and download 1 MB is set to 2 units of currency, while the average cost of transferring 1 MB between the edge caches for one hop is set at three units. The costs of each second of delay in accessing 1 MB of low- and high-priority content are set to 5 and 10 units of currency, respectively (timely access to high-priority content is critical, and so its delay costs twice as much).

To set up the learning process, the actions selected in the initial 700 time slots are totally random and are used initially to fill the experience replay buffer in order to start the learning process. In each time slot (for t>700), 30 samples of experience are extracted from the replay buffer to form the learning batch.

5.5.2 Comparison with the optimal solution

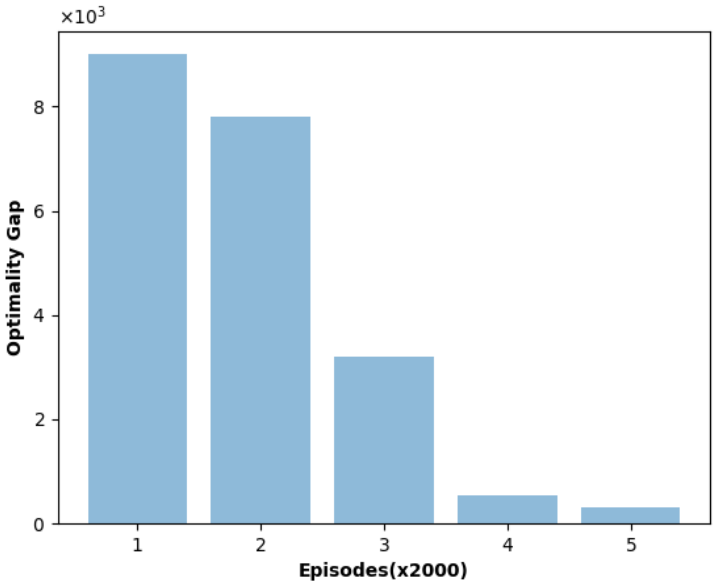


Figure 5.4 Proposed algorithm's optimality gap

To evaluate the performance of our proposed DRLCM algorithm against that of the optimal solution, we consider a small-scale scenario with only two mobile and two fixed edge caches. The road structure in this scenario is a two × two bidirected road grid, with the other parameters the same as the scenario previously explained. We let our learning method collect experiences about the environment while learning for a maximum of 10,000 episodes. We pause the simulation scenario every 2,000 episodes and perform an exhaustive search to find the optimal solution considering the positioning of mobile caches and content arrivals at that time. Figure 5.4 shows the absolute difference between the optimal solution and the DRLCM at every 2,000th episode. While the method is still in its exploration phase, the gap is considerably higher for the initial episodes. However, it decreases significantly as the episodes pass. At the end of 10,000 episodes, the DRLCM managed to decrease this gap by more than 97%. An important consideration here is that it takes more than 1 hour for each exhaustive search to find the optimal solution in this small-scale scenario, and that is only one snapshot of the whole system. Clearly, it is not possible to conduct exhaustive searches each time mobile edge caches change their position.

5.5.3 Performance comparison with existing deep learning methods

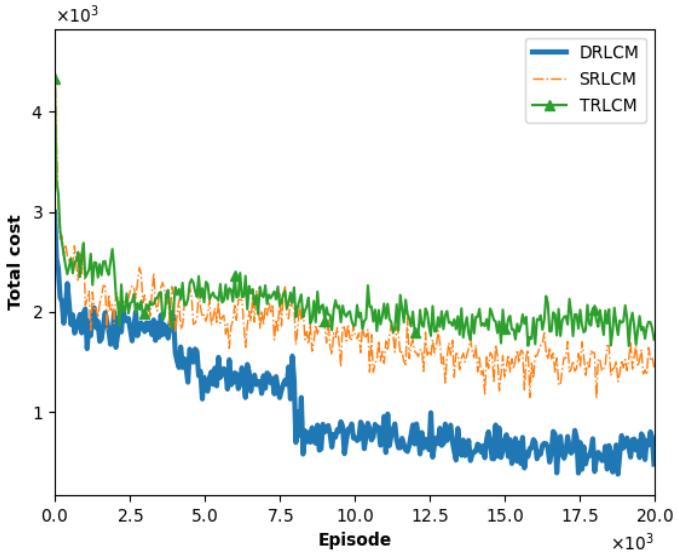


Figure 5.5 Total cost vs. episode evolution

We investigate the convergence performance of our proposed DRLCM with two other deep learning approaches, namely, SRLCM, a simplified Q-Learning version with Double RNNs and no LSTM cells [55], and TRLCM, a learning method with a single RNN and no LSTM cells [86], i.e., a traditional deep Q-Learning method. While the learning structure of the SRLCM method has been widely used in many recent studies [55],[56], and [94], TRLCM represents a classical version of the deep Q-Learning approach [86]. The evaluation scenario consists of 12 fixed and 20 mobile edge caches in a 5×5 road grid environment. Figure 5.5 depicts the total cost (in unit of currency) vs. episodes for different methods. According to Fig. 5.5, all three deep learning-based methods perform closely for the first episodes. This is mainly due to the fact that at the beginning, there is no knowledge about the environment, and so all the methods choose somewhat random actions. However, due to their different learning structures, they converge to different values. The policy learning of TRLCM seems to stop soon after completing 7,500 episodes, whereas the total costs achieved by the SRLCM and DRLCM methods keep decreasing. Finally, around the 15,000'th episode, the SRLCM method reaches a cost value of 1,500 and levels out. In contrast, our proposed DRLCM method continuously decreases the total cost as the number of episodes increases. Clearly, our proposed DRLCM method outperforms the other two deep learning methods. This high performance is attributed to the use of LSTM memory cells, which allow the DRLCM agent to remember the most valuable experiences that it had in its past observations.

5.5.4 Performance Comparison with Non-learning Methods

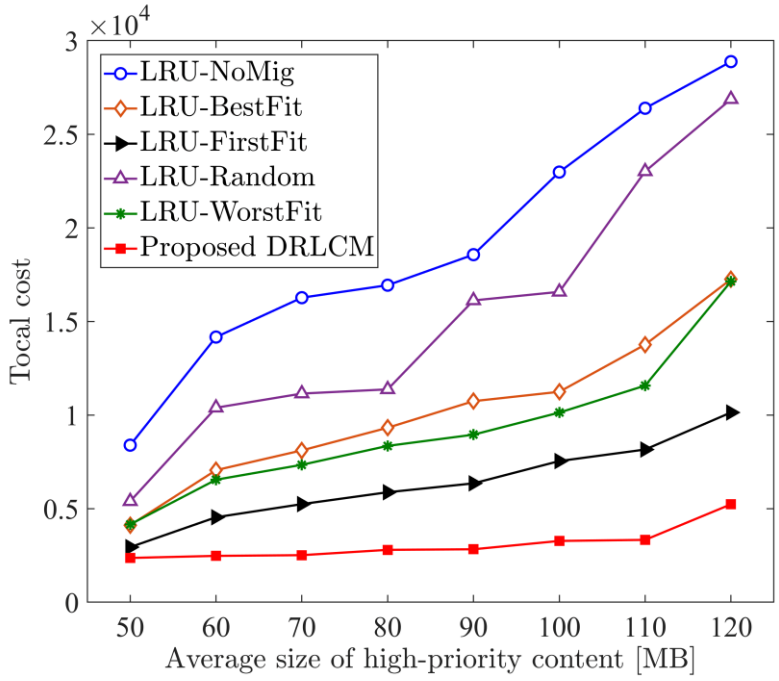


Figure 5.6. Total cost vs. average size of high-priority content

In the next set of evaluation scenarios, we compare the performance of our proposed method with methods based on the least recently used (LRU) eviction strategy [8], which is the most common non-learning cache replacement method. The five LRU-based approaches are explained below:

LRU-NoMig: LRU contents are deleted from the targeted full caches to free up space for the newly arrived high-priority contents.

LRU-FirstFit: LRU contents are migrated from the targeted full caches to the closest edge caches that have enough capacity to store them.

LRU-BestFit: LRU contents are migrated from the targeted full caches to the edge caches with the minimum caching capacities that can accommodate the migrated contents.

LRU-WorstFit: LRU contents are migrated from the targeted full caches to the edge caches with maximum caching capacities that can accommodate the migrated contents.

LRU-Random: LRU contents are migrated from the targeted full caches to the random edge caches with enough space to accommodate the migrated contents.

We compare the performance of our proposed method with methods based on the least recently used (LRU) eviction strategy [8], which is the most common non-learning cache replacement method. The five LRU-based approaches are LRU-NoMig**,** LRU-FirstFit**,** LRU-BestFit**,** LRU-WorstFit, and LRU-Random. The total cost vs. the average size of arrived high-priority contents (in MB) is shown in Fig. 5.6, which helps to compare the performance of the five non-learning methods with that of our proposed DRLCM method. As shown in Fig. 5.6, when the size of high-priority contents increases from 50 to 120 MB, the cost increases in all methods, which is expected, as all methods try to free up more space to accommodate such high-priority contents. Therefore, a larger amount of content will be migrated/deleted, and higher costs will be imposed. Further, we observe from Fig. 5.6 that the costly process of re-downloading the deleted contents imposes the largest cost to the LRU-NoMig method. The cost of the LRU-Nomig is even slightly larger than that of the LRU-Random approach, which randomly migrates LRU contents to the available edge caches instead of deleting them.

5.5.5 Scalability and Cost Improvement Percentages

we increased the total number of edge caches from 32 to 75 and the high-priority content sizes from 50 to 120 MB. Figure 5.7 depicts the improvement of cost (in percentage) vs. the number of edge cashes. It can be inferred from Fig. 5.7 that the improvement made by our proposed DRLCM method not only remains for a scaled version of the scenario but increases up to 70% in comparison with the LRU-NoMig approach, which does not support any content migration. This observation reveals the value of an appropriate decision to keep the content at the ed ge instead of performing

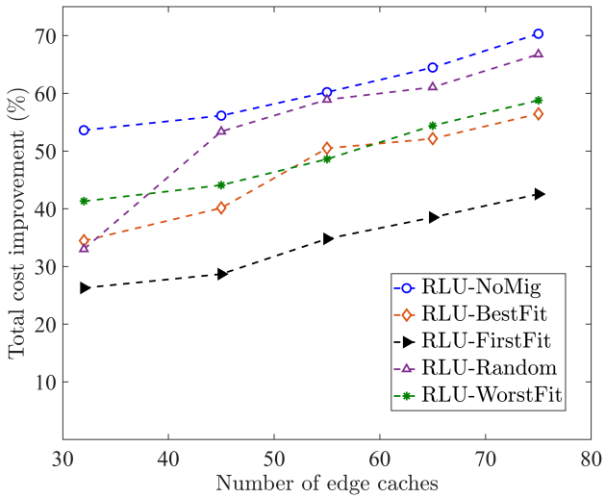


Figure 5.7. Total cost improvement

content deletions. Note that even though increasing the number of edge caches makes the scenario more complex, it ensures that more caching and processing resources become available at the edge.

5.5 Conclusions

In this chapter, we have proposed a deep reinforcement learning (DRL) content migration technique for a hierarchical edge-based CDN. Based on real-life situations, we considered a dynamic and heterogeneous environment consisting of mobile and fixed caches where contents have pre-assigned high and low priorities and developed a use case from a vehicular network to illustrate the motivation of our work. Our proposed method considers the available caching capacity in edge caches so that upon the arrival of high-priority contents, instead of just removing the low-priority contents from full caches, it migrates low-priority contents between edge caches to create enough space to accommodate high-priority contents. We implemented our DRL migration agent with a deep double-Q learner method empowered by LSTM memory cells. The simulation results show up to 70% in cost improvements compared to the existing methods.

Chapter 6

6. Performance Management for CDNs[1]

6.1 Introduction

Performance management is an essential task for CDN providers. In that regard, they need to acquire knowledge of users' QoE and correlate observations through different video sessions to identify QoE degradations and investigate their potential root causes. In the absence of users' feedback on their QoE, CDN providers can monitor and analyze Key Performance Indicators (KPIs) throughout video sessions. This allows for assessing the Quality of Service (QoS) offered to users, influencing their QoE. However, due to the large number of sessions handled by CDN operators, it is not easy to conduct such an analysis. Tens of thousands of video requests are received by a country-wide CDN provider on a daily basis, according to our investigations. Analyzing and correlating the KPIs among corresponding sessions is simply not possible manually. Automated approaches are thus needed to allow for this analysis over a massive set of sessions. In this chapter, we focus on analyzing the evolution of KPIs across video sessions for

[1] This chapter is based on a published paper:

- Sepideh Malektaji, Diala Naboulsi, Roch Glitho, Alexander Polyantsev, Ali El Essaili, Cyril Iskander, and Richard Brunner. "*Video sessions KPIs clustering framework in CDNs*." In 2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC), pp. 1-6. IEEE, 2019.

QoS and QoE analysis, using unsupervised machine learning tools. Moreover, we propose a framework that allows the automatic formation of clusters of video sessions, presenting similar evolution of KPIs. We capture the dynamics of KPIs over each session through a set of representative features. Using the k-means clustering algorithm, we build upon collected features to form clusters of video sessions, with each containing similar sessions in terms of KPIs evolution. The framework is evaluated over a real-world traffic dataset covering thousands of sessions collected over the infrastructure of a country-wide CDN provider. We show that our framework allows distinguishing meaningful clusters. The obtained results underline the capabilities of the proposed framework.

The rest of the chapter is organized as follows. First, we introduce the proposed framework, including the KPI representations and the clustering algorithm used in this framework. We then cover the evaluation of the framework, and finally, we conclude the work in the last subsection.

6.2 KPIs Clustering Framework

In this section, we present our video sessions clustering framework. Our framework allows video sessions to be grouped into a set of clusters, each associated with a specific pattern in the evolution of KPIs. The evolution of a KPI over time, throughout a video session, can be captured through an irregular time series representation. Various clustering approaches can be employed accordingly: raw data-based clustering, feature-based clustering, and model-based clustering [14]. Video sessions typically last for a long duration. Therefore, the time series representing the evolution of their KPIs are also long. Moreover, they have unequal lengths, representing the user's watching time of a particular video content. In contrast to other approaches, a feature-based clustering approach allows us to cope with these two aspects. We, therefore, adopt this type of approach in our framework. In the following, we first describe the features that we employ to represent the evolution of KPIs throughout a session. The similarity measure on which the clustering step relies to group sessions is then presented, followed by a description of the clustering algorithm. Finally, we describe how clusters are selected.

6.2.1 KPIs Representation

Our framework operates over a set of sessions S. We employ $s \in S$ to refer to an individual session in S. A session s spans over a time interval T_s, a set of time instants t. We consider a set I of KPIs. The evolution of KPI $i \in I$ throughout session $s \in S$ is captured via an irregular time series $i_s = \{ i_s^t , \ \forall t \in T_s\}$ where i_s^t is the value of KPI $i \in I$ at time instant $t \in T_s$ over session $s \in S$.

We rely on the derived time series representation of KPIs i_s to extract a set of features V that capture the evolution of KPIs throughout a session. The set of features V has been chosen as the set of features considered in [96]. There, the authors showed that the set of features V allows to successfully represent the evolution of a time series. Each feature $v \in V$ captures a specific facet of the evolution of the KPI in question throughout a session. For each KPI $i \in I$, we extract the following six features V for a session $s \in S$, as detailed below.

Average. The average value of KPI $i \in I$ of all samples during session $s \in S$ is obtained with Equation (1).

$$\mu_{s,i} = \frac{1}{|T_s|} \sum_{t \in T_s} i_s^t \tag{6-1}$$

Standard deviation. The standard deviation of KPI $i \in I$, over all its samples in session $s \in S$, is derived with Equation (6-2).

$$\sigma_{s,i} = \sqrt{\frac{1}{|T_s|} \sum_{t \in T_s} \left(i_s^t - \mu_{s,i}\right)^2} \tag{6-2}$$

Skewness. Skewness is a measure that characterizes the shape of a mathematical distribution. It captures the level of symmetry in the distribution with respect to its central point. For a KPI $i \in I$

over a session $s \in S$, it is derived with Equation (6-3).

$$W_{s,i} = \frac{1}{|T_s|\sigma_{s,i}^3} \sum_{t \in T_s} (i_s^t - \mu_{s,i})^3 \tag{6-3}$$

Kurtosis. Kurtosis is another measure that allows characterizing the shape of a mathematical distribution. It captures whether the distribution is heavy-tailed or light-tailed. A distribution with a light tail tends to have low kurtosis. For a KPI $i \in I$ over session $s \in S$, the Kurtosis can be obtained using Equation (6-4).

$$K_{s,i} = \frac{1}{|T_s|\sigma_{s,i}^4} \sum_{t \in T_s} \left(i_s^t - \mu_{s,i}\right)^4 \tag{6-4}$$

Energy. Energy measures the strength of a time series. It is obtained based on the non-uniform Discrete Fourier transform (DFT) of a time series. Assuming $f_{s,i}^j \in F$ is the jth discrete Fourier component for KPI $i \in I$ in session $s \in S$, and that F is the complete set, the energy is derived with Equation (6-5).

$$E_{s,i} = \frac{1}{|F|} \sum_{f_{s,i}^j \in F} |f_{s,i}^j| \tag{6-5}$$

MLE. The Maximum Lyapunov Exponent measures the randomness for a time series by quantifying the average logarithmic rate of separation of two nearby subsets of the time series. For a KPI $i \in I$ over a session $s \in S$, it is obtained based on Equation (6-6).

$$M_{s,i} = \lim_{t \to \infty} \frac{1}{T_s} ln \frac{|i_s^{t+\delta} - i_s^t|}{|i_s^{t_0+\delta} - i_s^{t_0}|} \tag{6-6}$$

6.2.2 Similarity Measure

After deriving the six features for each KPI $i \in I$, over session $s \in S$, we construct a vector V_s encompassing these features for a session $s \in S$ as follows:

$$V_s = \left[\mu_{s,i}, \sigma_{s,i}, W_{s,i}, K_{s,i}, E_{s,i}, M_{s,i} \;\middle|\; \forall i \in I\right]$$

Each of the features is rescaled to the interval [0,1] by considering the minimum and maximum value of the feature of interest across all observations. Rescaling is done so that each feature contributes approximately proportionately to the similarity measure. We use $v_{s,i}$ to refer to the rescaled feature $v \in V$ of KPI $i \in I$ over session $s \in S$. This allows us to define a rescaled vector V_s' of features for a session $s \in S$, as follows:

$$V_s' = [\, v_{s,i} \mid \forall i \in I, v \in V]$$

We capture the degree of similarity between a pair of sessions s and r by calculating the Euclidean distance between the corresponding rescaled vectors using Equation (6-7).

$$d(s,r) = \left[\sum_{v \in V}\sum_{i \in I}\left(v_{s,i} - v_{r,i}\right)^2\right]^{1/2} \tag{6-7}$$

6.2.3 Clustering Algorithm

To obtain the set of video sessions clusters, we employ the widely-known k-means clustering algorithm [97]. It has been selected due to its superior performance compared to other approaches, e.g., hierarchical clustering techniques, as shown in the literature [97] as well as our experiments. k-means clustering algorithm is an unsupervised machine learning algorithm that allows grouping a set of observations into a given number k of clusters. It relies on a vectorial representation of observations, in our case, the derived vectors V_s', for each session $s \in S$ and for a similarity measure among them, in our case computed based on the Euclidean distance, as described previously. Given a random initial selection of k centroids for clusters, the algorithm operates by alternating between the two following steps until no more changes are possible.

Assignment step: For each session, the algorithm computes the average distance between the session and all k centroids. Then, the session is assigned to the cluster with the smallest distance.

Update step: Once all sessions are assigned to a cluster in the assignment step, the centroid for each cluster is updated. The new centroid is obtained by computing the mean for vectors that correspond to the sessions in the cluster.

6.2.4 Selection of Clusters

The k-means clustering algorithm allows clustering video sessions into a given number k of clusters. Multiple strategies exist for choosing the best k value. In our work, we use clustering indices to compare multiple clustering solutions and choose the best one. For that, we run k-means algorithm for different values of k. A clustering index can then compare the different solutions and select the best value of k for the final clusters. We combine the results of the following three indices.

The Calinski-Harabasz (CH) index [98] quantifies the dispersion level among clusters against within clusters dispersion. The best value of k is considered as the one leading to the largest value of CH.

The Silhouette (SI) index [99], for a single data point, in our case a session, allows measuring how similar it is to the cluster where it belongs compared to other clusters. The average value of the Silhouette index over all the sessions indicates the consistency level in the clustering. The larger the value, the better the clustering.

The Davies Bouldin (DB) index [100] evaluates consistency in a clustering solution. It measures the within-cluster distances against between-cluster distances. A lower value of the DB index translates into a better clustering.

Combining the indices: To combine the outcome of the CH, SI, and DB indices to find the proper number of clusters, we follow a ranking method. For each index, we assign a rank to each value of k. The value of k that has the highest aggregated rank, according to the different indices, is considered to represent the best clustering solution.

6.3 Performance Evaluation

We summarize here our assessment of our framework, starting with a presentation of the dataset we utilized. The clusters of sessions that the framework generated are described next.

6.3.1 Dataset

The dataset we used to evaluate our framework was collected over the infrastructure of a real-world CDN provider. This CDN provider offers both VoD and live video content. The dataset encompasses sessions covering the transfer of these two types of content at a country-scale level. The initial dataset was collected for several days in 2016, with tens of thousands of content requests received on a daily basis. For each content request, the transfer of each of its chunks was tracked through data logs, with diverse information relating to the client, the content, and the streaming node. For our evaluations, we operate over a subset of 6000 sessions occurring on a typical working day. For our evaluations, we considered two major KPIs, the Download Bit Rate (DBR) and the Quality Level (QL). The DBR is the rate at which bits are transferred from the surrogate server to the user. The QL represents instead the bit rate at which the video is encoded. Depending on the streaming technique used, the QL can change over time according to a user's requests (e.g., as in the case of adaptive bit rate streaming). Aside from being available in the dataset, these two KPIs have been selected for our evaluation because of their correlation with users' QoE, as shown in previous studies ([101] and [62]). However, our framework is generic enough to account for other KPIs that can be collected over the CDN system.

6.3.2 Sessions Clusters

We now discuss the outcome of our framework after its application to the obtained dataset. We start from the number of clusters selected. Recall that to obtain the best number of clusters, we rely on different clustering indices with multiple runs of the k-means clustering algorithm. We considered cases ranging from two to nine clusters. Among these options, the different clustering indices ranked the case of nine clusters as the best choice. Therefore, in the rest of the section, we analyze the corresponding nine clusters. We first examine the sessions that are grouped into the same clusters. We portray in Fig. 6.1 the evolution of the DBR and the QL over two sessions selected from cluster 4 and cluster 2, respectively. For cluster 4, we can see in Fig. 6.1.a and

Fig.6.1.b that the two sessions present a similar evolution in terms of DBR and QL, with strong upwards variations in DBR and a constant QL.

In turn, the sessions selected from cluster 2, shown in Fig. 6.1.c and Fig. 6.1.d, present similar DBR and QL variations; each is characterized by a single prominent peak for DBR and a constant QL with a few drops at the beginning. These observations illustrate our framework's ability to identify similarities in the KPI patterns between sessions.

Moreover, by comparing the sessions in Fig. 6.1, we can observe that the sessions in cluster 4 are significantly different from those in cluster 2, which also shows the capability of our framework to separate sessions with distinct KPIs patterns into different clusters. Similar observations also hold for the other clusters. We observed that sessions grouped inside the same cluster present very similar patterns in terms of DBR and QL evolution, while the sessions in different categories are clearly distinguishable. Furthermore, the identified patterns are informative for CDNs on users' QoS and QoE, as follows.

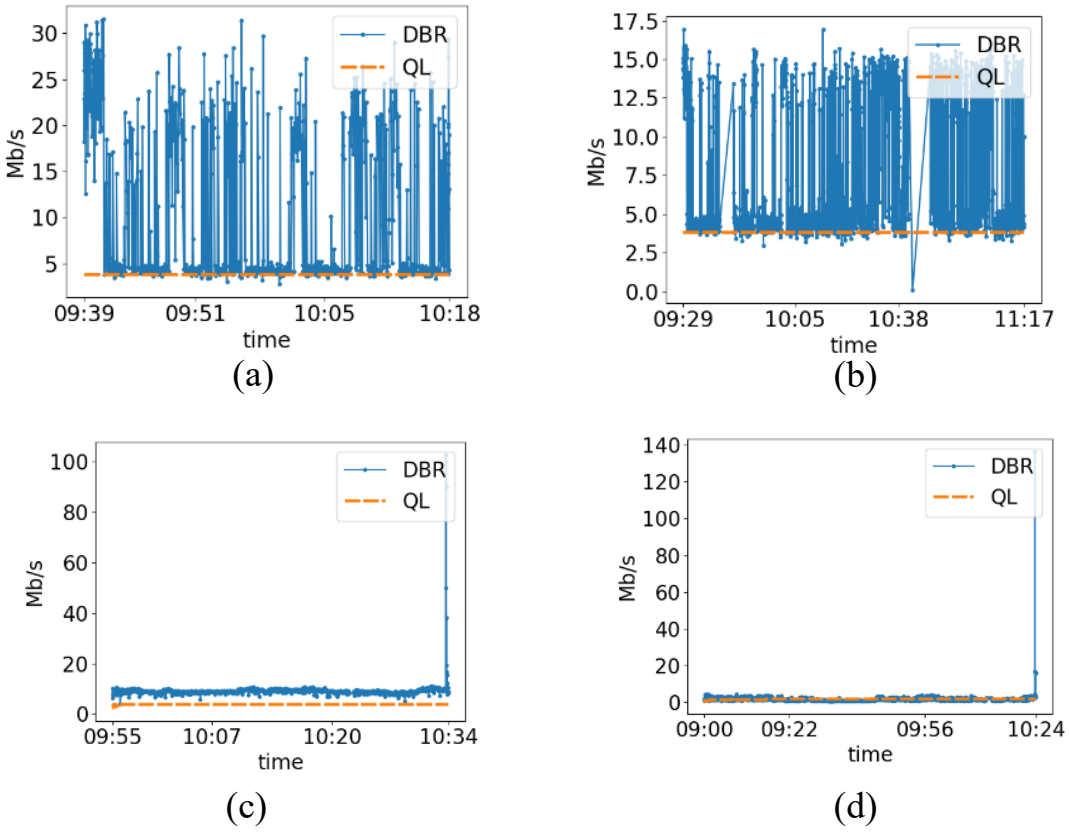
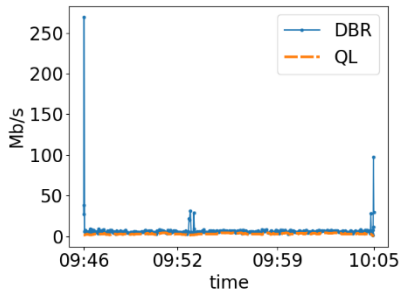
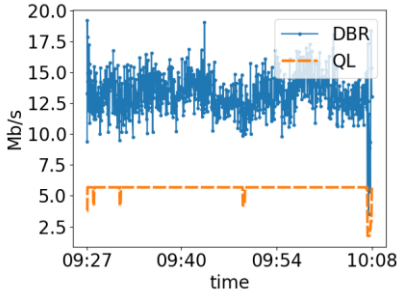


Fig. 6.1 Sessions (a,b) in cluster 4; Sessions (c,d) in cluster 2.

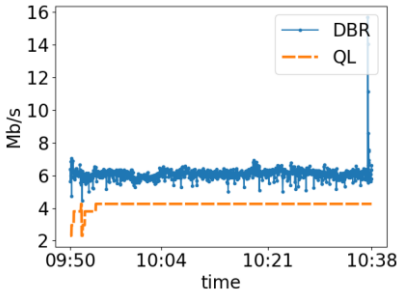
In Fig. 6.2, we portray the evolution of the DBR and QL for a sample session from each cluster. The sample session is the closest session to the centroid of the cluster and is, therefore, representative of the patterns in the corresponding cluster. As can be seen, each cluster presents distinct patterns for KPIs evolution. Cluster 3 has strong variations in both DBR and QL. This



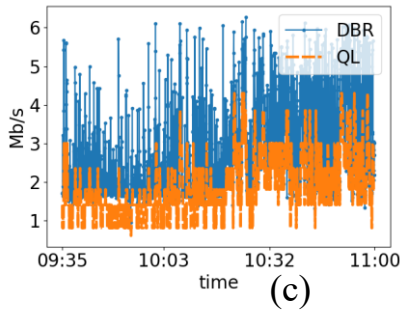
(a) Cluster 0- A few peaks of DBR at the starting and ending points of the session, variations in QL.



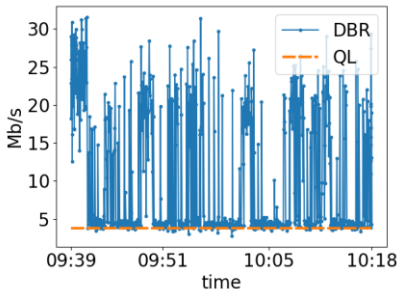
(b) Cluster 1- Variations and a few drops in DBR, a few drops in QL.



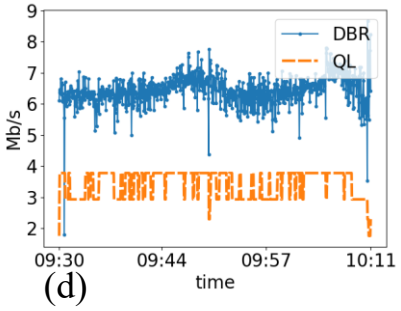
(c) Cluster 2- A single peak of DBR at the end of the session, constant QL with a few drops at the beginning.



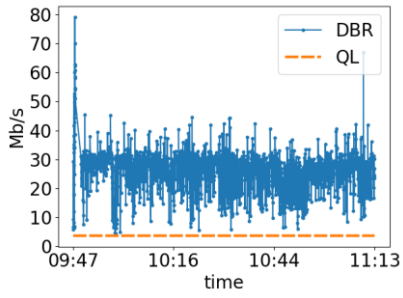
(d) Cluster 3- Strong variations in both DBR and QL.



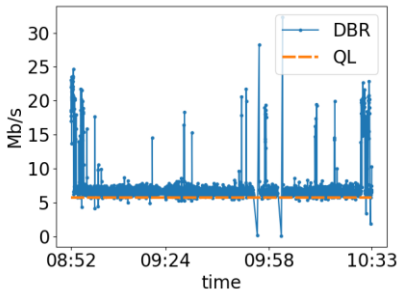
(e) Cluster 4- Upward variations in DBR, constant QL.



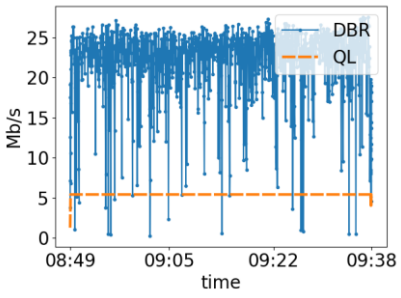
(f) Cluster 5- Variations in both DBR and QL, mostly towards smaller values.



(h) Cluster 6- Variations in DBR with peaks at starting and ending of the session, constant QL.



(i) Cluster 7- Upward variations in DBR, constant QL.



(j) Cluster 8- Significant drops in DBR and a few drops in QL.

Fig. 6.2 Summary of obtained clusters.

reflects unstable network conditions throughout the video streaming, leading to frequent switching in QL of the viewed content. These frequent switches can significantly degrade the users' QoE [102]. It is thus critical for the CDN provider to identify this pattern. Similar behavior is perceived in Cluster 5, with variations in both DBR and QL. However, in this case, variations are less prominent. Important variations in DBR and QL are also observed in Cluster 0. Nevertheless, a single high peak in DBR enables the distinction of corresponding sessions in a separate cluster.

Clusters 1, 2, and 8 present a few QL drops with distinct DBR patterns. Cluster 2 shows a constant QL with a few drops at the beginning and a single peak of DBR at the end of the session. The low QL values at the beginning of a streaming session are important to analyze as they have a significant influence on the user's decision to carry on a streaming session [102]. Similarly, Cluster 1 is characterized by a few drops in QL together with variations and drops in DBR. There, the drops in QL and DBR at the end of the session are especially critical, as they could have resulted in the user abandoning the session. Similarly, cluster 8 presents a few drops in QL and some notable drops in DBR. Particularly important are the DBR drops that go below the stable QL and can reach zero. Such drops can lead to undesirable video stalling behavior that affects users' QoE [102].

In contrast to the other patterns, clusters 6, 4, and 7 present a constant QL, including sessions with constant bitrate streaming. Cluster 6 is characterized by variations in DBR with peaks at its session's starting and ending points. However, as the DBR values remain greater than the constant QL, based on these two KPIs, the video streaming would go smoothly. Both Cluster 4 and Cluster 7 exhibit a constant QL with upward variations in DBR that are more prominent in Cluster 4. However, both these clusters also present DBR drops below QL levels that can lead to interruptions in video streaming. Overall, we notice the framework is capable of identifying meaningful clusters of sessions with distinct KPIs patterns that are informative for CDNs on users' QoS and QoE.

6.4 Conclusion

In this work, we introduce a framework for the analysis of KPIs in large-scale CDN systems. The framework employs an unsupervised machine learning algorithm to automatically form clusters of video sessions, presenting similar evolution of KPIs. We evaluate the framework over a real-world dataset. The results underline its ability to create meaningful clusters.

Chapter 7

7. Conclusion and Future Work

7.1 Conclusion

Due to the inherently complex and dynamic nature of CDNs, several issues need to be addressed to realize the next generation of CDNs. These issues include CDN deployment, content placement, and performance management. This thesis proposes a number of ML-based solutions to address these issues.

7.1.1 CDN Deployment

To provide a cost-efficient CDN deployment solution, in chapter 3, we proposed a deep reinforcement learning-based joint VNF-FG composition and embedding framework that considers the variations of service demands as well as the substrate network congestion. The proposed method improved the embedding cost by up to 95%. Moreover, we demonstrated that

our proposed method could reach as close as 10.63% to the optimal solution within an acceptable time duration.

Moreover, to enable adaptability in CDN deployment, in chapter 4, we addressed another resource allocation issue. In there, we proposed a joint function scaling and topology adaptation method, which supports not only the horizontal scaling but also VNF reordering and connectivity changes in a given VNF-FG. We evaluated the performance of our proposed framework against different neural network architectures and conducted performance evaluations comparing with both joint and disjoint benchmarks. The results show that our proposed method achieves up to a 93% cost improvement compared to the benchmarks.

7.1.1 CDN Content Placement

To meet the QoS/QoE requirements in the operation of an edge-based CDN, we proposed a deep reinforcement learning (DRL) content placement and migration technique that considers the available caching capacity in the end-users neighboring edge caches. Our proposed method eliminates the need for time-consuming retransmission of the selected content from remote servers by enabling a content migration strategy. The simulation results show up to 70% in cost improvements compared to the existing methods.

7.1.1 CDN Performance Management

To provide an automated performance management solution for CDN, we introduce a framework for analyzing KPIs in large-scale CDN systems. The framework employs an unsupervised machine learning algorithm to automatically form clusters of video sessions, presenting similar evolution of KPIs. We evaluate the framework over a real-world dataset. The results underline its ability to create meaningful clusters that can provide valuable insights for further root cause analysis for CDNs.

7.2 Future Work

This thesis presented significant contributions toward realizing the next generation of CDNs by addressing challenges in the deployment, content placement, and performance management of CDNs. Yet, there exist several research directions for the future.

7.2.1 CDN Deployment Future Work

In chapter 3, to provide a dynamic resource allocation solution, we employed two analyzers that use historical data and estimate dynamic parameters such as service demand and physical network congestion. However, an interesting future research direction could be integrating advanced ML algorithms to build predictive models for these parameters. Such models can help CDN providers to design proactive solutions enabling faster and more efficient deployment and adaptation strategies.

7.2.2 Content Placement Future Work

As a future research direction, our work in chapter 5 can be extended by considering an additional caching layer consisting of drone caches. Even though the use of caches installed on drones provides flexibility for content delivery, their high mobility compounds the complexity of the problem and hence requires further investigations.

7.2.3 Performance Management Future Work

The performance management framework proposed in chapter 6 can be extended to provide a root cause analysis solution. To that end, the patterns discovered in each cluster could be further analyzed using cutting-edge pattern recognition ML tools. Once the clusters with performance degrading patterns are identified, interesting root cause information can be fetched from the sessions belonging to those clusters. For instance, if most video sessions in the problematic cluster are served by the same server, the server might be overloaded and could be the root cause of performance degradation.

Bibliography:

[1] Zolfaghari B, Srivastava G, Roy S, Nemati HR, Afghah F, Koshiba T, Razi A, Bibak K, Mitra P, Rai BK. Content delivery networks: State of the art, trends, and future roadmap. ACM Computing Surveys (CSUR). 2020 Apr 16;53(2):1-34.

[2] "Content delivery network (CDN) market—Growth, trends, and forecast (2020–2025)," Mordor Intell., Hyderabad,India,Rep.,2019.[Online].Available:https://www.mordorintelligence.com/industryreports/content-delivery-market

[3] He M, Alba AM, Basta A, Blenk A, Kellerer W. Flexibility in softwarized networks: Classifications and research challenges. IEEE Communications Surveys & Tutorials. 2019 Jan 14;21(3):2600-36.

[4] Herrera JG, Botero JF. Resource allocation in NFV: A comprehensive survey. IEEE Transactions on Network and Service Management. 2016 Aug 5;13(3):518-32.

[5] Salahuddin MA, Sahoo J, Glitho R, Elbiaze H, Ajib W. A survey on content placement algorithms for cloud-based content delivery networks. IEEE Access. 2017 Sep 19;6:91-114.

[6] Sepideh Malektaji, Amin Ebrahimzadeh, Halima Elbiaze, Roch Glitho, "Dynamic Joint VNF Forwarding Graph composition and placement: A Deep Reinforcement Learning Framework" submitted work IEEE Transactions on Network and Service Management.

[7] M. T. Beck and J. F. Botero, "Coordinated allocation of service function chains," in Proc. IEEE Global Communications Conference (GLOBE-COM), 2015, pp. 1–6.

[8] S. M. Ara´ujo, F. S. de Souza, and G. R. Mateus, "A composition selection mechanism for chaining and placement of virtual network functions," in Proc. IEEE International Conference on Network and Service Management (CNSM), 2019, pp. 1–5.

[9] B. Spinnewyn, P. H. Isolani, C. Donato, J. F. Botero, and S. Latr, "Coordinated service composition and embedding of 5G location-constrained network functions," IEEE Transactions on Network and Service Management, vol. 15, no. 4, pp. 1488–1502, 2018.

[10] R. Gour, G. Ishigaki, J. Kong, and J. P. Jue, "Availability-guaranteed slice composition for service function chains in 5G transport networks," IEEE/OSA Journal of Optical Communications and Networking, vol. 13, no. 3, pp. 14–24, 2021.

[11] M. Wang, B. Cheng, S. Zhao, B. Li, W. Feng, and J. Chen, "Availability-aware service chain composition and mapping in NFV-enabled net-works," in Proc. IEEE International Conference on Web Services (ICWS), 2019, pp. 107–115.

[12] Sepideh Malektaji, Amin Ebrahimzadeh, Halima Elbiaze, Roch Glitho, “Joint VNF-FG Function Scaling and Topology Adaptation using Deep Reinforcement Learning” submitted work IEEE Transactions on Emerging Topics in Computing.

[13] Sepideh Malektaji, Amin Ebrahimzadeh, Halima Elbiaze, Roch Glitho, and Somayeh Kianpishe. “Deep Reinforcement Learning-based Content Migration for Edge Content Delivery Networks with Vehicular Nodes.” IEEE Transactions on Network and Service Management 2021.

[14] Sepideh Malektaji, Somayeh Kianpisheh, and Roch Glitho,“Purging-Aware Content Placement in Fog-Based Content Delivery Networks.” In 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), pp. 1-3. IEEE, 2018.

[15] Sepideh Malektaji, Diala Naboulsi, Roch Glitho, Alexander Polyantsev, Ali El Essaili, Cyril Iskander, and Richard Brunner. “Video sessions KPIs clustering framework in CDNs.” In 2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC), pp. 1-6. IEEE, 2019.

[16] Broberg J, Buyya R, Tari Z. MetaCDN: Harnessing ‘Storage Clouds’ for high-performance content delivery. Journal of Network and Computer Applications. 2009 Sep 1;32(5):1012-22.

[17] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow and P. A. Polakos, "A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges," in *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416-464, Firstquarter 2018, doi: 10.1109/COMST.2017.2771153.

[18] Sun, Yaohua, et al. "Application of machine learning in wireless networks: Key techniques and open issues." *IEEE Communications Surveys & Tutorials* 21.4 (2019): 3072-3108.

[19] Luong, Nguyen Cong, et al. "Applications of deep reinforcement learning in communications and networking: A survey." *IEEE Communications Surveys & Tutorials* 21.4 (2019): 3133-3174.

[20] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[21] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. No. 1. 2016

[22] Benkacem I, Taleb T, Bagaa M, Flinck H. Optimal VNFs placement in CDN slicing over multi-cloud environment. IEEE Journal on Selected Areas in Communications. 2018 Mar 12;36(3):616-27.

[23] Wang W, Lan R, Gu J, Huang A, Shan H, Zhang Z. Edge caching at base stations with device-to-device offloading. IEEE Access. 2017 Mar 7;5:6399-410.

[24] Su Z, Hui Y, Xu Q, Yang T, Liu J, Jia Y. An edge caching scheme to distribute content in vehicular networks. IEEE Transactions on Vehicular Technology. 2018 Apr 9;67(6):5346-56.

[25] Wang L, Lu Z, Wen X, Knopp R, Gupta R. Joint optimization of service function chaining and resource allocation in network function virtualization. IEEE Access. 2016 Nov 17;4:8084-94.

[26] Li J, Shi W, Ye Q, Zhuang W, Shen X, Li X. Online joint VNF chain composition and embedding for 5G networks. In2018 IEEE Global Communications Conference (GLOBECOM) 2018 Dec 9 (pp. 1-6). IEEE.

[27] Zheng D, Peng C, Liao X, Tian L, Luo G, Cao X. Towards latency optimization in hybrid service function chain composition and embedding. InIEEE INFOCOM 2020-IEEE Conference on Computer Communications 2020 Jul 6 (pp. 1539-1548). IEEE.

[28] Kang J, Simeone O, Kang J. On the trade-off between computational load and reliability for network function virtualization. IEEE Communications Letters. 2017 Apr 25;21(8):1767-70.

[29] Chen X, Yu H, Xu S, Du X. CompRess: Composing overlay service resources for end-to-end network slices using semantic user intents. Transactions on Emerging Telecommunications Technologies. 2020 Jan;31(1):e3728.

[30] Bian S, Huang X, Shao Z, Gao X, Yang Y. Service chain composition with resource failures in NFV systems: A game-theoretic perspective. IEEE Transactions on Network and Service Management. 2020 Dec 16;18(1):224-39.

[31] Ning Z, Wang N, Tafazolli R. Deep Reinforcement Learning for NFV-based Service Function Chaining in Multi-Service Networks. In2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR) 2020 May 11 (pp. 1-6). IEEE.

[32] Pei J, Hong P, Li D. Virtual network function selection and chaining based on deep learning in SDN and NFV-enabled networks. In2018 IEEE International Conference on Communications Workshops (ICC Workshops) 2018 May 20 (pp. 1-6). IEEE.

[33] Yang S, Li F, Trajanovski S, Yahyapour R, Fu X. Recent advances of resource allocation in network function virtualization. IEEE Transactions on Parallel and Distributed Systems. 2020 Aug 17;32(2):295-314.

[34] Pham C, Tran NH, Ren S, Saad W, Hong CS. Traffic-aware and energy-efficient vNF placement for service chaining: Joint sampling and matching approach. IEEE Transactions on Services Computing. 2017 Feb 20;13(1):172-85.

[35] Wu B, Zeng J, Ge L, Shao S, Tang Y, Su X. Resource allocation optimization in the NFV-enabled MEC network based on game theory. InICC 2019-2019 IEEE International Conference on Communications (ICC) 2019 May 20 (pp. 1-7). IEEE.

[36] Pei J, Hong P, Pan M, Liu J, Zhou J. Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks. IEEE Journal on Selected Areas in Communications. 2019 Dec 13;38(2):263-78.

[37] N. Yuan, W. He, J. Shen, X. Qiu, S. Guo and W. Li, "Delay-Aware NFV Resource Allocation with Deep Reinforcement Learning," NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1-7, doi: 10.1109/NOMS47738.2020.9110377.

[38] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, “NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning,” in Proc. ACM International Symposium on Quality of Service, 2019, pp. 1–10.

[39] P. Sun, J. Lan, J. Li, Z. Guo and Y. Hu, "Combining Deep Reinforcement Learning With Graph Neural Networks for Optimal VNF Placement," in IEEE Communications Letters, vol. 25, no. 1, pp. 176-180, Jan. 2021, doi: 10.1109/LCOMM.2020.3025298.

[40] L. Wang, W. Mao, J. Zhao and Y. Xu, "DDQP: A Double Deep Q-Learning Approach to Online Fault-Tolerant SFC Placement," in IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 118-132, March 2021, doi: 10.1109/TNSM.2021.3049298.

[41] X. Fu, F. R. Yu, J. Wang, Q. Qi and J. Liao, "Dynamic Service Function Chain Embedding for NFV-Enabled IoT: A Deep Reinforcement Learning Approach," in IEEE Transactions on Wireless Communications, vol. 19, no. 1, pp. 507-519, Jan. 2020, doi: 10.1109/TWC.2019.2946797.

[42] Fei X, Liu F, Jin H, Li B. FlexNFV: Flexible network service chaining with dynamic scaling. IEEE Network. 2020 Feb 11;34(4):203-9.

[43] Zhang Q, Liu F, Zeng C. Online Adaptive interference-aware VNF deployment and migration for 5G network slice. IEEE/ACM Transactions on Networking. 2021 May 25;29(5):2115-28.

[44] Pandey S, Hong JW, Yoo JH. GRU and EdgeQ-Learning based Traffic Prediction and Scaling of SFC. In2021 IEEE 7th International Conference on Network Softwarization (NetSoft) 2021 Jun 28 (pp. 124-132). IEEE.

[45] Subramanya T, Riggio R. Centralized and federated learning for predictive VNF autoscaling in multi-domain 5G networks and beyond. IEEE Transactions on Network and Service Management. 2021 Jan 11;18(1):63-78.

[46] Luo Z, Wu C. An online algorithm for VNF service chain scaling in datacenters. IEEE/ACM Transactions on Networking. 2020 Mar 24;28(3):1061-73.

[47] Lange S, Kim HG, Jeong SY, Choi H, Yoo JH, Hong JW. Predicting vnf deployment decisions under dynamically changing network conditions. In2019 15th International Conference on Network and Service Management (CNSM) 2019 Oct 21 (pp. 1-9). IEEE.

[48] Houidi O, Soualah O, Louati W, Zeghlache D. Dynamic VNF forwarding graph extension algorithms. IEEE Transactions on Network and Service Management. 2020 Apr 28;17(3):1389-402.

[49] Liu J, Lu W, Zhou F, Lu P, Zhu Z. On dynamic service function chain deployment and readjustment. IEEE Transactions on Network and Service Management. 2017 Jun 5;14(3):543-53.

[50] Khan I, Zhang T, Xu X, Shan S, Khan A, Ahmad S. Priority-based content dissemination in content centric vehicular networks. In2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC) 2018 May 25 (pp. 2005-2009). IEEE.

[51] Meuser T, Richerzhagen B, Stavrakakis I, Nguyen TA, Steinmetz R. Relevance-aware information dissemination in vehicular networks. In2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM) 2018 Jun 12 (pp. 588-599). IEEE.

[52] Chen M, Qian Y, Hao Y, Li Y, Song J. Data-driven computing and caching in 5G networks: Architecture and delay analysis. IEEE Wireless Communications. 2018 Feb 28;25(1):70-5.

[53] Zhu H, Cao Y, Wang W, Jiang T, Jin S. Deep reinforcement learning for mobile edge caching: Review, new features, and open issues. IEEE Network. 2018 Nov 29;32(6):50-7.

[54] Zhong C, Gursoy MC, Velipasalar S. A deep reinforcement learning-based framework for content caching. In2018 52nd Annual Conference on Information Sciences and Systems (CISS) 2018 Mar 21 (pp. 1-6). IEEE.

[55] Hu RQ. Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning. IEEE Transactions on Vehicular Technology. 2018 Aug 27;67(11):10190-203.

[56] He Y, Zhao N, Yin H. Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. IEEE Transactions on Vehicular Technology. 2017 Oct 6;67(1):44-55.

[57] Yu Z, Hu J, Min G, Zhao Z, Miao W, Hossain MS. Mobility-aware proactive edge caching for connected vehicles using federated learning. IEEE Transactions on Intelligent Transportation Systems. 2020 Aug 31;22(8):5341-51.

[58] Qiao G, Leng S, Maharjan S, Zhang Y, Ansari N. Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks. IEEE Internet of Things Journal. 2019 Oct 22;7(1):247-57.

[59] Gomaa H, Messier GG, Williamson C, Davies R. Estimating instantaneous cache hit ratio using markov chain analysis. IEEE/ACM transactions on Networking. 2012 Dec 10;21(5):1472-83.

[60] Fan X, Katz-Bassett E, Heidemann J. Assessing affinity between users and CDN sites. InInternational Workshop on Traffic Monitoring and Analysis 2015 Apr 21 (pp. 95-110). Springer, Cham.

[61] Casas P, D'Alconzo A, Fiadino P, Bär A, Finamore A, Zseby T. When YouTube does not work—Analysis of QoE-relevant degradation in Google CDN traffic. IEEE Transactions on Network and Service Management. 2014 Dec 4;11(4):441-57.

[62] Shafiq MZ, Erman J, Ji L, Liu AX, Pang J, Wang J. Understanding the impact of network dynamics on mobile video user engagement. ACM SIGMETRICS Performance Evaluation Review. 2014 Jun 16;42(1):367-79.

[63] Li Z, Wu Q, Salamatian K, Xie G. Video delivery performance of a large-scale VoD system and the implications on content delivery. IEEE Transactions on Multimedia. 2015 Mar 27;17(6):880-92.

[64] Li W, Spachos P, Chignell M, Leon-Garcia A, Zucherman L, Jiang J. Understanding the relationships between performance metrics and QoE for over-the-top video. In2016 IEEE International Conference on Communications (ICC) 2016 May 22 (pp. 1-6). IEEE.

[65] Orsolic I, Pevec D, Suznjevic M, Skorin-Kapov L. A machine learning approach to classifying YouTube QoE based on encrypted network traffic. Multimedia tools and applications. 2017 Nov;76(21):22267-301.

[66] Giordano D, Traverso S, Grimaudo L, Mellia M, Baralis E, Tongaonkar A, Saha S. YouLighter: A cognitive approach to unveil YouTube CDN and changes. IEEE Transactions on Cognitive Communications and Networking. 2015 Jun;1(2):161-74.

[67] Wu T, Huysegems R, Bostoen T. Scalable network-based video-freeze detection for HTTP adaptive streaming. In2015 IEEE 23rd International Symposium on Quality of Service (IWQoS) 2015 Jun 15 (pp. 95-104). IEEE.

[68] Dimopoulos G, Leontiadis I, Barlet-Ros P, Papagiannaki K, Steenkiste P. Identifying the root cause of video streaming issues on mobile devices. In Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies 2015 Dec 1 (pp. 1-13).

[69] Zhu Y, Helsley B, Rexford J, Siganporia A, Srinivasan S. LatLong: Diagnosing wide-area latency changes for CDNs. IEEE Transactions on Network and Service Management. 2012 Jul 6;9(3):333-45.

[70] Tavakoli A, Pardo F, Kormushev P. Action branching architectures for deep reinforcement learning. InProceedings of the AAAI Conference on Artificial Intelligence 2018 Apr 29 (Vol. 32, No. 1).

[71] Kanervisto A, Scheller C, Hautamäki V. Action space shaping in deep reinforcement learning. In2020 IEEE Conference on Games (CoG) 2020 Aug 24 (pp. 479-486). IEEE.

[72] Li D, Hong P, Xue K, Pei J. Virtual network function placement and resource optimization in NFV and edge computing enabled networks. Computer Networks. 2019 Apr 7;152:12-24.

[73] Chemodanov D, Calyam P, Esposito F. A near optimal reliable composition approach for geo-distributed latency-sensitive service chains. InIEEE INFOCOM 2019-IEEE Conference on Computer Communications 2019 Apr 29 (pp. 1792-1800). IEEE.

[74] G. Wang, G. Feng, T. Q. Quek, S. Qin, R. Wen, and W. Tan, “Reconfiguration in network slicing—optimizing the profit and performance,” IEEE Transactions on Network and Service Management, vol. 16, no. 2, pp. 591–605, 2019.

[75] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI gym,” arXiv preprint arXiv:1606.01540, 2016.

[76] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., “Tensorflow: A system for large-scale machine learning,” in 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 2016, pp. 265–283.

[77] T. O’Malley, “Hyperparameter tuning with Keras Tuner,” 2020.

[78] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y.-C. Liang, “Network slice reconfiguration by exploiting deep reinforcement learning with large action space,” IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2197–2211, 2020

[79] J. Wilkes, “Yet more Google compute cluster trace data,” Google research blog, 2020.

[80] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, “Borg: the next generation,” in Proc. ACM European Conference on Computer Systems, 2020, pp. 1–14

[81] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in Proc. PMLR International Conference on Machine Learning, 2016, pp. 1995–2003.

[82] X. Lin, D. Guo, Y. Shen, G. Tang, and B. Ren, “DAG-SFC: Minimize the embedding cost of SFC with parallel VNFs,” in ACM Proc. International Conference on Parallel Processing, 2018, pp. 1–10

[83] H. Cao, J. Du, H. Zhao, D. X. Luo, N. Kumar, L. Yang, and F. R.Yu, “Resource-ability assisted service function chain embedding and scheduling for 6G networks with virtualization,” IEEE Transactions on Vehicular Technology, vol. 70, no. 4, pp. 3846–3859, 2021.

[84] Z. Wang, J. Zhang, T. Huang, and Y. Liu, “Service function chain composition, placement, and assignment in data centers,” IEEE Transactions on Network and Service Management, vol. 16, no. 4, pp. 1638–1650, 2019.

[85] Jahromi NT, Kianpisheh S, Glitho RH. Online VNF placement and chaining for value-added services in content delivery networks. In2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN) 2018 Jun 25 (pp. 19-24). IEEE.

[86] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y. Mastering the game of go without human knowledge. nature. 2017 Oct;550(7676):354-9.

[87] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen,“Toward efficient content delivery for automated driving services: An edge computing solution,” IEEE Network, vol. 32, no. 1, pp. 80–86, Jan. 2018.

[88] J. Liu, A. Shahroudy, D. Xu, A. C. Kot, and G. Wang, “Skeleton-based action recognition using spatio-temporal LSTM network with trust gates,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 12, pp. 3007–3021, Nov. 2017.

[89] Y. Bin, Y. Yang, F. Shen, N. Xie, H. T. Shen, and X. Li, “Describing video with attention-based bidirectional LSTM,” IEEE Transactions on Cybernetics, vol. 49, no. 7, pp. 2631–2641, May 2018.

[90] K. Greff, R. K. Srivastava, J. Koutn´ık, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 10, pp. 2222–2232, July 2016.

[91] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, “Recent development and applications of SUMO-simulation of urban mobility,” International Journal on Advances in Systems and Measurements, vol. 5, no. 3&4, Dec. 2012.

[92] A. Hanggoro and R. F. Sari, “Performance evaluation of the Manhattan mobility model in vehicular Ad-hoc networks for high mobility vehicle,” in Proc. IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), 2013, pp. 31–36.

[93] G. Li, J. Wang, J. Wu, and J. Song, “Data processing delay optimization in mobile edge computing,” Wireless Communications and Mobile Computing, vol. 2018, Feb. 2018.

[94] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, “Migration modeling and learning algorithms for containers in fog computing,” IEEE Transactions on Services Computing, vol. 12, no. 5, pp. 712–725, Feb.2018.

[95] Liao TW. Clustering of time series data—a survey. Pattern recognition. 2005 Nov 1;38(11):1857-74.

[96] Teemu et al. "Feature-based clustering for electricity use time series data." Adaptive and Natural Computing Algorithms, 401-412, 2009.

[97] Xu R, Wunsch D. Survey of clustering algorithms. IEEE Transactions on neural networks. 2005 May 9;16(3):645-78.

[98] Caliński T, Harabasz J. A dendrite method for cluster analysis. Communications in Statistics-theory and Methods. 1974 Jan 1;3(1):1-27.

[99] Wang K, Wang B, Peng L. CVAP: validation for cluster analyses. Data Science Journal. 2009 Apr 24:0904220071-.

[100] Davies DL, Bouldin DW. A cluster separation measure. IEEE transactions on pattern analysis and machine intelligence. 1979 Apr(2):224-7.

[101] Menkovski V, Exarchakos G, Liotta A, Sánchez AC. Quality of experience models for multimedia streaming. International Journal of Mobile Computing and Multimedia Communications (IJMCMC). 2010 Oct 1;2(4):1-20.

[102] Juluri P, Tamarapalli V, Medhi D. Measurement of quality of experience of video-on-demand services: A survey. IEEE Communications Surveys & Tutorials. 2015 Feb 6;18(1):401-18.

[103] Klein, D. J., and M. Randić. "Innate degree of freedom of a graph." *Journal of Computational Chemistry* 8.4 (1987): 516-521.