



PEOPLES DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
University Mohammed Khider-BISKRA
Faculty of The Exact Sciences, Natural and Life Sciences
Department of Computer Science

Ordre Number :RTIC06/M2/2021

THESIS

Presented to obtain the diploma of academic master in

Computer Science

Path: Network and Information and Communication Technology

**Deep Reinforcement Learning Based
Approach for Service Function Chain
Deployment in 5G Networks**

By:

ELBEY NOUR ELIMANE

Members of the jury:

Boukhrouf Djemaa	M.C.B	President
Ayad Soheyb	M.C.A	Supervisor
Ben Dahmene Asma	M.A.A	Member

Academic year 2021-2022

Acknowledgement

First and foremost, praised be Allah the Almighty, through whose mercy all good things are achieved.

Firstly, I would like to express my special appreciation and gratitude to my supervisor, Dr. Soheyb Ayad, for giving me the opportunity to pursue my aspirations. He believed in me and gave me this wonderful opportunity to do research. I will always be indebted to him for this. I also appreciate his continuous support and valuable guidance throughout my master's period, and I consider myself fortunate to have him as a supervisor.

I would also like to thank Mr. Benhaya Bilal, who has been involved throughout the thesis, giving me valuable suggestions and helping to achieve a correct development of the thesis. My long-lasting thanks go to Dr. Fouad Bekkari, who always pushed me forward with his advice and support.

I also take this opportunity to show my gratitude to all my professors in the computer science department. I would further like to express my heartfelt gratitude to the jury members for agreeing to judge this work.

I sincerely thank all my family and my friends for their continuous encouragement, support, and love throughout the time I have dedicated to this project.

Thank you all for the unconditional emotional support during the journey towards my goal.

Abstract

5G networks are intended to simultaneously support a wide range of applications with a high variety of requirements which brings a diversity of use cases for mobile networks and a growing number of demands. The development of 5G relies on new techniques such as Software Defined Networks (SDN), Network Function Virtualisation (NFV) and Service Function Chain (SFC) technologies. SDN allows the separation of control and data planes. NFV decouples network functions from the hardware that performs them through virtualisation. SFC is a popular service paradigm that has been proposed to derive maximum benefits from both NFV and SDN in 5G networks.

The conventional approaches to service and infrastructure management, which require complete and perfect knowledge of the system, are inefficient. In this context, Deep Reinforcement Learning (DRL), which marked a success in solving complicated control and decision-making problems by allowing network entities to learn, build knowledge, and make optimal decisions independently, inspired us to suggest this study, where we focus on optimising the resource utilisation of the fifth generation networks. We propose a Deep Q-learning (DQN) based DRL approach for optimal SFC deployment in 5G networks. The objective of the proposed approach is to deploy SFCs automatically and support the heterogeneity of SFC network requirements. Experimental results show that the approach proposed can achieve superior performance in solving SFC deployment problem, where the average return are 15.28% compared with the Q-learning based Reinforcement Learning (RL) approach 7.4%.

Key words: 5G networks, Software Defined Network (SDN), Network Function Virtualisation (NFV), Service Function Chain (SFC), Deep Reinforcement Learning (DRL).

الملخص

تهدف شبكات الجيل الخامس (5G) إلى دعم مجموعة واسعة من التطبيقات في وقت واحد مع مجموعة متنوعة من المتطلبات التي تجلب مجموعة متنوعة من حالات الاستخدام لشبكات المحمول وعدد متزايد من الطلبات. يعتمد تطوير شبكات الجيل الخامس على تقنيات جديدة مثل الشبكات المعرفة بالبرمجيات (SDN) ، وتقنيات المحاكاة الافتراضية لوظيفة الشبكة (NFV) وسلسلة وظائف الخدمة (SFC). تسمح الشبكات المعرفة بالبرمجيات بفصل الجزء الخاص بالتحكم عن الجزء الخاص بالبيانات. تفصل تقنيات المحاكاة الافتراضية لوظيفة الشبكة ووظائف الشبكة عن الأجهزة من خلال المحاكاة الافتراضية. سلسلة وظائف الخدمة هي نموذج خدمة شائع تم إقتراحه لتحقيق أقصى قدر من الفوائد من كل من SDN و NFV في شبكات الجيل الخامس . الأساليب التقليدية لإدارة الخدمات والبنية التحتية ، والتي تتطلب معرفة كاملة ومثالية للنظام ، أصبحت غير فعالة. في هذا السياق ، نظرا للنجاح الباهر الذي حققه التعلم العميق المعزز (DRL) في حل مشاكل التحكم المعقدة واتخاذ القرار من خلال السماح لكيانات الشبكة بالتعلم وبناء المعرفة واتخاذ القرارات المثلثي بشكل مستقل ، إقترحنا هذه الدراسة التي تركز على تعظيم الاستفادة من موارد شبكات الجيل الخامس ، حيث إستعملنا نموذجاً قائماً على التعلم العميق المعزز (DQN) لنشر سلاسل وظائف الخدمة بالطريقة الأمثل. الهدف من النموذج المقترح هو نشر سلاسل وظائف الخدمة تلقائياً ودعم عدم تجانس متطلباتها. تظهر النتائج التجريبية أن النموذج المقترح يمكن أن يحقق أداءً فائقاً في حل مشكلة نشر سلاسل وظائف الخدمة ، حيث يبلغ متوسط العائد 15.28٪ مقارنةً بنهج التعلم المعزز القائم على Q-learning 7.4 ٪.

الكلمات المفتاحية: شبكات (5G) ، الشبكة المعرفة بالبرمجيات (SDN) ، المحاكاة الافتراضية لوظيفة الشبكة (NFV) ، سلسلة وظائف الخدمة (SFC) ، التعلم المعزز العميق (DRL) .

Contents

Acknowledgement	I
Abstract	II
List of Abbreviation	VII
List of Figures	IX
List of Tables	XI
List of Algorithms	XII
List of listings	XIII
General introduction	1
1 5G Network on General	4
1.1 Introduction	4
1.2 Mobile Network Evolution	5
1.3 5G Network	5
1.3.1 5G Architecture	6
1.3.2 5G Key Performance Indicators (KPI)	7
1.3.3 5G Enabling Technologies	9
1.4 Network Function Virtualization (NFV)	12
1.4.1 NFV architecture	12
1.5 Software Defined Network (SDN)	14
1.5.1 SDN architecture	14
1.6 Service Function Chain (SFC)	16
1.6.1 Static service function chaining	17
1.6.2 Dynamic service function chaining	17
1.6.3 Service function chaining architecture	18

1.6.4	SDN/NFV architecture for SFC deployment	20
1.7	Conclusion	21
2	Machine Learning and Networking	22
2.1	Introduction	22
2.2	Machine Learning	22
2.2.1	Machine Learning Types	23
2.2.2	Difference between Machine Learning types	26
2.3	Deep Reinforcement Learning	27
2.3.1	Policy and Value function	27
2.3.2	Exploration vs Exploitation	28
2.3.3	Deep Q-learning	29
2.4	Machine Learning in Networking	31
2.4.1	Special Considerations to deploy ML in Networking	34
2.4.2	Machine Learning for 5G Network	34
2.5	Related Works	36
2.5.1	DRL for NFV-based Service Function Chaining in Multi-Service Networks	36
2.5.2	Online Service Function Chain Deployment with DRL in 5G networks	36
2.5.3	Adaptive Online SFC Deployment with Deep Reinforcement Learning	37
2.5.4	Q-Learning based SFC deployment on Edge Computing Environment	37
2.5.5	RL based QoS/QoE-aware Service Function Chaining in Software-Driven 5G Slices	37
2.5.6	Discussion	38
2.6	Conclusion	39
3	Design	40
3.1	Introduction	40
3.2	System Modeling	40
3.2.1	Network Definition	40
3.2.2	Service Function Chain Request	40
3.2.3	Problem Formulation	41
3.2.4	Markov Decision Process (MDP) for SFC deployment	41
3.3	General Architecture	42
3.4	Detailed Architecture	43
3.4.1	Source and Destination nodes	44
3.4.2	Network Function Virtualization	44

3.4.3	Software Define Network	45
3.4.4	Network topology	45
3.4.5	DRL model architecture	46
3.5	Reinforcement Learning model architecture	55
3.5.1	RL based approach for SFC deployment algorithm	55
3.5.2	RL based approach for SFC deployment process	57
3.6	Conclusion	58
4	Experimental study and results	59
4.1	Introduction	59
4.2	Development tools	59
4.2.1	IDE (integrated development environment)	60
4.2.2	Programming language	60
4.2.3	libraries	60
4.3	Implementation	61
4.3.1	Environment	61
4.3.2	Deep Q-learning Agent	63
4.3.3	Q-learning Agent	67
4.4	Results	69
4.4.1	DQN models evaluation results	70
4.4.2	DQN vs Q-learning comparison results	73
4.5	Final discussion	75
4.6	Conclusion	76
	General conclusion	77
	Bibliography	79

List of Abbreviation

ITU-R	International Telecommunication Union- Radiocommunication
IMT-2020	International Mobile Telecommunications-2020, original name of 5G
3GPP	3rd Generation Partnership Project
KPIs	Key Performance Indicators
5G NR	5G New Radio
NFV	Network Function Virtualization
SDN	Software Defined Network
SFC	Service Function Chain
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
CUs	Centralised Units
DUs	Distributed Units
RAN	Radio Access Network
RRUs	Remote Radio Units
PHY	Physical layer
MAC	Medium Access Control
VNFs	Virtual Network Functions
AUSF	Authentication Server Function
UDM	Unified Data Management
AMF	Access and Mobility Management Function
SMF	Session Management Function
IP	Internet Protocol
PCF	Policy Control Function
NF	Network Function
NSSF	Network Slice Selection Function
UE	User Equipment
AF	Application Function

UPF User plane Function
DN Data Network
eMBB Enhanced Mobile Broadband
IMT-advanced International Mobile Telecommunications- advanced, 4G
CDF Cumulative Distribution Function
mMTC Massive Machine Type Communications
URLLC Ultra-Reliability Low Latency Communications
MIMO Massive Multiple-Input Multiple-Output
Mm-wave Millimeter -wave signals
NS Network slicing
WLAN Wireless Local Area Network
CAPEX Capital Expenses
OPEX Operating Expenses
NFVI NFV infrastructure
MANO NFV Management and Orchestration
ONF Open Network Foundation
NSH network service header
IETF Internet Engineering Task Force
SF Service Function
SFF Service Function Forwarder
SFP Service Function Path
LSA Latent Semantic Analysis
PCA Principal Component Analysis
MDP Markov Decision Processes
DNN Deep Neural Networks
DQN Deep Q-learning Network
QoS-E Quality of Service-Experience
HMM Hidden Markov Model
OSPF Open Shortest Path First
HTTP Hypertext Transfer Protocol
CNN Convolutional neural network
DOV Direction Of Arrival
TDD Test-driven development
DDQN Dueling Double Deep Q-Network
MILP mixed-linear programming

List of Figures

1.1	Overview of the evolving mobile technologies [5].	5
1.2	5G network architecture [2].	6
1.3	5G Key Performance Indicators (KPI) [7].	8
1.4	MIMO [1].	10
1.5	Beam Forming [18].	10
1.6	Small Cells [8].	11
1.7	Millimeter-wave signals (Mm-wave) [21].	11
1.8	Network Slicing architecture [2].	12
1.9	ETSI NFV reference architecture [12].	13
1.10	ONF SDN reference architecture [22].	15
1.11	SFC request example [11].	17
1.12	SFC Static [20].	17
1.13	SFC dynamic [20].	18
1.14	IETF SFC reference architecture [16].	19
1.15	SDN/NFV based dynamic Service Function Chaining [11].	21
2.1	Supervised Learning [25].	23
2.2	Unsupervised Learning [25].	24
2.3	Interaction between an agent and its environment [25].	25
2.4	Difference between SL, UL and RL [43].	26
2.5	Interaction between an DRL agent and its environment [24].	27
2.6	Deep Q Network operation [2].	31
2.7	The typical workflow of machine learning for networking (MLN) [29].	32
3.1	General architecture for the proposed model.	43
3.2	Detailed architecture.	44
3.3	SNDlib dfn-bwin topology [56].	46
3.4	DQN vs Q-learning.	46

3.5	DQN workflow.	49
3.6	Q-network and target-network.	50
3.7	SFC deployment approach in time steps.	50
3.8	DRL based SFC deployment process.	55
3.9	RL based SFC deployment process.	58
4.1	Used tools.	59
4.2	SNDlib and environment link data.	62
4.3	SFC requests sample.	62
4.4	DQN training phase reward.	66
4.5	SFC request used in evaluation.	69
4.6	DQN agents reward.	70
4.7	DQN agents resource consumption.	71
4.8	DQN agents bandwidth consumption.	72
4.9	DQN vs Q-learning reward.	73
4.10	DQN vs Q-learning resource consumption.	74
4.11	DQN vs Q-learning bandwidth consumption.	75

List of Tables

2.1	Relationships between Network application and MLN workflow [29].	33
2.2	Brief summary of corresponding ML-driven approaches to cope with the demand of the 5G standards[34].	35
2.3	comparison between related works.	39
4.1	Environment configuration	61
4.2	Hyperparameter configuration	63
4.3	Hyperparameter configuration	67
4.4	Comparison between all models	76

List of Algorithms

1	DQN algorithm	30
2	Experience replay pseudo code	47
3	ϵ -greedy pseudo code	48
4	DQN SFC deployment	52
5	Reset function	53
6	SFC request function	53
7	Start function	54
8	Step function	54
9	Q-learning SFC deployment	56
10	agent_learn function	57

List of listings

4.1	Used libraries	63
4.2	DQN agent initialization	63
4.3	Build model	64
4.4	Memorize function	64
4.5	ϵ -greedy function	65
4.6	DQN agent learning function	65
4.7	Used libraries	67
4.8	Q-learning agent initialization	67
4.9	<i>check_state_exist</i> function	68
4.10	ϵ -greedy function	68
4.11	Q-learning agent learning function	69

General introduction

Information and Communication Technologies (ICT) have been identified as a key component in social and economic growth because they have produced new benefits and efficiencies that have never been experienced before. Since 2009, when 4G network services were introduced, people have been able to use internet services on their devices. Despite the advancements in 4G network technologies, it is difficult to provide mobile services that require high speed, rapid response, high dependability, and energy efficiency. For that reason, these features have become critical requirements for future 5G services.

Research on 5G services was performed by the International Telecommunication Union-Radio-communication (ITU-R) [1], the 3rd Generation Partnership Project (3GPP) and the Next Generation Mobile Networks (NGMN) Alliance. They proposed the usage scenarios as a group of three categories: enhanced Mobile Broadband (eMBB), massive Machine-Type Communications (mMTC), and Ultra-Reliable and Low-Latency Communications (URLLC). Peak data rate, area traffic capacity, network energy efficiency, connection density, latency, mobility, spectrum efficiency, and user-experienced data rate are chosen as key performance indicators (KPIs) [4]. The details are discussed in section (1.3.2). In 5G networks, service providers aim to deploy their services flexibly and quickly to take in the specific requirements of diverse services, and they face many problems, especially in network performance and availability.

In traditional network architecture, each network function, such as load balancer (LB), firewall (FW), and network address translation (NAT), requires expensive hardware for its deployment. Because of continued growth in the number of users and application services, using a hardware-based network to store and transfer a huge amount of data according to specified requirements is a risky task. For that, network providers may benefit from three of 5G new emerging technologies:

1. **Network Function Virtualization (NFV)** has emerged as an innovative network architecture paradigm that uses virtualization technology to abstract the network node functions from hardware.

2. **Software Defined Networking (SDN)** is a complementary technology that allows programmatic control of network functions and enables flexible and efficient network management.
3. **Service Function Chain (SFC)** is group of chained Virtual Network Functions (VNF) in the network.

To overcome the limits of traditional network architecture and the problems that network providers have faced in service deployment, they must make sure that the service requirement meet the computational and network capacity constraints. The SFCs need to be deployed in a specific way where the allocation of resources and placement of virtual network functions will be performed dynamically. Hence, there is a need for a dynamic and automatic service deployment model to save effort and time.

In view of the dynamic deployment of the service function chain, the existing deployment methods have some shortcomings. Most SFC deployment mechanisms have only solved one problem, which is determining deployment location or achieving optimal targets, but in our work, we try to solve both of them. In order to achieve node and link load balancing, we propose an intelligent SFC deployment model based on a Deep Reinforcement Learning approach. With Deep Q-learning, which has a powerful learning capacity to be widely adopted to solve complex problems, our proposed DQN can efficiently provide a high-reward SFC deployment solution to each arriving request, considering its resource demand and the current resource utilization. The main contributions of this study is:

- We use an MDP model to formulate the SFC deployment problem to capture real-time network variations, where network variations are dynamically and continuously expressed as MDP state transitions.
- We propose our model DQN based DRL approach to automatically deploy SFC requests with different requirements.
- We implement a Q-learning based RL approach for SFC deployment to make an effective comparison with the DQN approach.

The rest of this dissertation is structured as follows:

- **chapter 1** Provides a literature review in the context of this study, 5G networks, and the emerging technologies that help overcome 4G limitations.
- **chapter 2** We introduce machine learning mechanisms and their applications in 5G networks. Then we focus on Deep Reinforcement Learning, especially the DQN algorithm. Finally, we discuss the related work.
- **chapter 3** Represent the design of our approach. Specifically, the general and detailed designs, as well as the algorithm compared.

- **chapter 4** Illustrate the implementation of our system, presenting the used tools and technologies for the development of our system and provides evaluation results. This section also evaluates DQN model by drawing comparisons with Q-learning approaches.

Chapter 1

5G Network on General

1.1 Introduction

5G is the fifth-generation mobile network. It is a new global standard after 1G, 2G, 3G and 4G networks, in the World Mobile Conference in Europe (2015) ITU-Radiocommunication (ITU-R) set the vision of International Mobile Telecommunications-2020 (IMT-2020), various 5G services are presented in a vision document [1]. In Release 15 [3], 3GPP has specified the frequency band allocated for the 5G network, it also highlighted the three leading Key Performance Indicators (KPIs) that define various use cases of 5G New Radio [4]. 5G network has many applications scenario in: enhancing agricultural productivity, advanced healthcare and improved manufacturing operations. 5G enable a new kind of network that is designed to connect virtually everyone and everything together including machines, object and devices.

In this chapter we introduce different 5G enabling technologies that are supported in this thesis. We start with giving basic knowledge about 5G and previous generation networks. Then, we present Network Function Virtualization (NFV) which is a promising paradigm to solve the problems of the traditional network approach. After that, we introduce Software Defined Network, SDN is another networking architecture used to separate the control plane from the data plane. Finally, we focus on Service Function Chain, SFC is a popular service paradigm that has been proposed to derive maximum benefits from both NFV and SDN. SFC using SDN and NFV, facilitates implementation of 5G network services.

1.2 Mobile Network Evolution

The evolution of mobile networks shown in **figure 1.1** which present the evolving generations of mobile technologies in terms of data rate, mobility, coverage and spectral efficiency. As the mobile technologies are growing, the data rate, mobility, coverage and spectral efficiency increases. It also, shows that the 1G and 2G technologies use circuit switching while 2.5G and 3G uses both circuit and packet switching and the generations from 3.5G to 5G are using only packet switching. Moreover that, it differentiates between unlicensed spectrum and licensed spectrum. All the evolving generations use the licensed spectrum while the WiFi, Bluetooth and WiMAX are using the unlicensed spectrum.

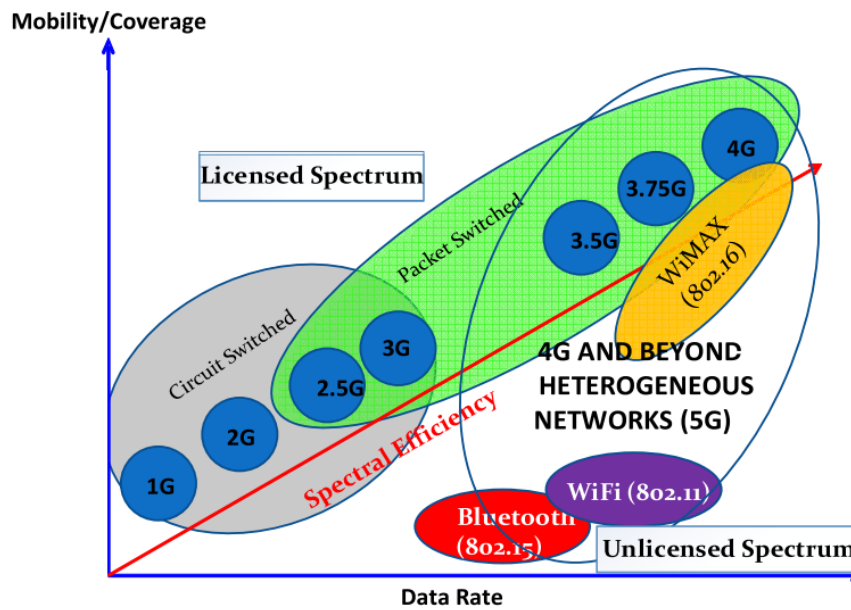


Figure 1.1: Overview of the evolving mobile technologies [5].

1.3 5G Network

While 4G was focused to provide mobile broadband communications, 5G is being designed to become a key asset in the introduction of the digital technologies in multiple economic and societal sectors. 5G infrastructures are expected to play a key role on the evolution of sectors such as the industry 4.0, automotive and mobility, transportation, healthcare system, energy industry. The massive demand for higher data rates and bandwidth with the increased number of users led to the deployment of the 5G network. 5G networks are expected to support higher connectivity, improved capacity, high-speed data rates, and low latency.

1.3.1 5G Architecture

The research has proposed several architectures to overcome 4G limitations in order to fully benefit 5G advantages. **Figure 2.2** shows the basic architecture of 5G network as presented in [2], the Radio Access Network (RAN) is divided and virtualized into server-based Distributed Units (DUs) and then centralized into server-based Centralised Units (CUs), reducing the proprietary hardware to Remote Radio Units (RRUs).

- **The RRU** handles parts of the physical layer (PHY), analog to digital conversion, filtering, power amplification as well as the digital beam forming functionality.
- **The Distributed Unit (DU)** is close to the RRU and runs the Radio Link Control, Medium Access Control (MAC), and parts of the PHY layer. It provides digital processing, including signal modulation, encoding and scheduling. It is a logical node that includes a subset of the gNodeB (5G base station) functions.
- **The Centralized Unit (CU)** is a logical node that provides support for higher layers of the protocol stack. It includes the gNodeB functions like Transfer of user data, mobility control, radio access network sharing, positioning, session management, with the exception of the functions that are allocated exclusively to the DU. The CU controls the operation of several DUs.

This centralized deployment makes load-balancing between different RRUs possible. That is why, in most cases, the DU will be collocated with RRUs. 5G Core components running as virtualized network functions (VNFs) in the cloud. 5G infrastructure will enable the inter-connectivity among the different emerging technologies like Massive MIMO network, Cognitive Radio network, and small-cell networks.

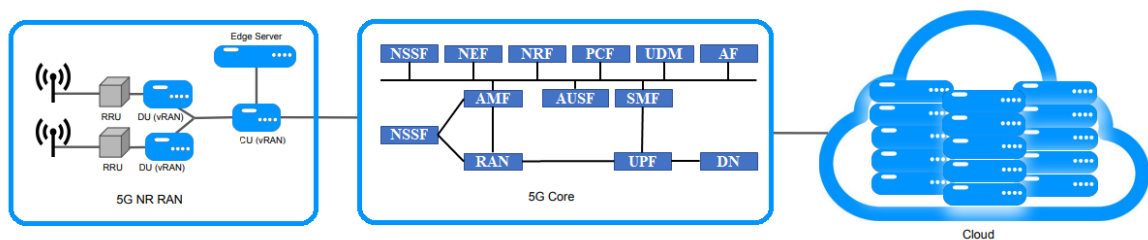


Figure 1.2: 5G network architecture [2].

The authors in [2] define 5G core Network function in **figure 1.2** as:

- **AUSF** (Authentication Server Function) provides a unified framework for authentication issues (for 3GPP access as well as non-3GPP access).

- **UDM** (Unified Data Management) contains data that was related to HSS (Home Subscriber Server).
- **AMF** (Core Access and Mobility Management Function) has different functionalities such as access authentication and authorization, registration management and mobility management.
- **SMF** (Session Management Function) is responsible for session management and some other functionalities, such as allocation of IP addresses, and controlling the policy enforcement and QoS (establishment of sessions).
- **PCF** (Policy Control Function) is related to policy framework and provides policy rules to NFs in the control plane.
- **NSSF** (Network Slice Selection Function) determines the serving AMF for the UE and selects network slice instances for it (in addition to network slicing concept, network slice instances provide specific services to different enterprises).
- **AF** (Application Function) provides services to 3rd parties.
- **UPF** (User plane Function) is responsible for everything related to user data.
- **DN** (Data Network) is internet access or services from operators and 3rd parties.
- **NEF** 5G Network Exposure Function (NEF) facilitates secure, robust, developer-friendly access to the exposed network services and capabilities of 5G network.
- **NRF** The Network Repository Function is one of the key functions of the 3GPP Service-Based Architecture for 5G Core networks, acting as a central Services Discovery broker for all Network Functions (NFs) in the 5G Core.

1.3.2 5G Key Performance Indicators (KPI)

Recommendation ITU-R M.2083 [1] defines eight key “Capabilities for IMT-2020”. The intent of these requirements is to ensure that IMT-2020 technologies are able to fulfil the objectives of IMT-2020 and to set a specific level of performance that each proposed set of radio interface technologies needs to achieve. 5G New radio specifies three primary use cases that are specified as the Next Generation KPI. The three KPIs discussed that future networks will provide a vast set of services and features that are summarized in **figure 1.3**.

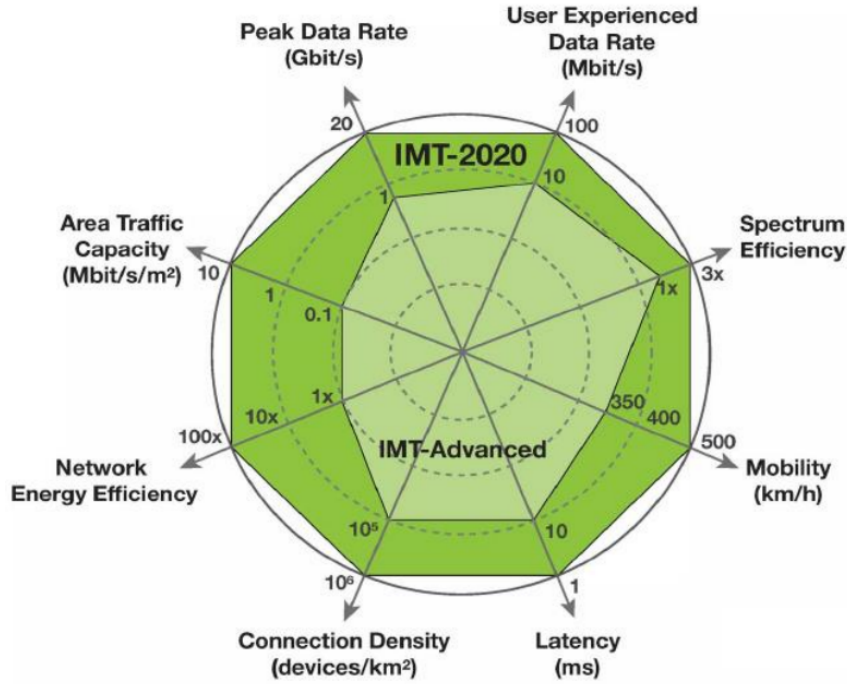


Figure 1.3: 5G Key Performance Indicators (KPI) [7].

According to [1] the three KPIs are:

1.3.2.1 enhanced Mobile Broadband(eMBB)

Enhancing the current MBB service will enable new applications with higher data rate demands over a uniform coverage area. The essential requirements to enable eMBB are presented below:

- a. **Peak data rate** Peak data rate is planned to increase and support high-demand data-driven use cases. IMT-2020 systems will be required to deliver 20-times higher data rate than the previous technology specification, from 1 *Gb/s* in IMT-advanced (4G) to 20 *Gb/s* in IMT-2020 (5G).
- b. **User experienced data rate** User experienced data rate is defined as the 5% point of the Cumulative Distribution Function (CDF) of the user throughput over active time, measured in a dense urban environment. IMT-2020 intends to brace 10-times higher user experienced data rate compared to IMT-advanced, from 10 *Mbit/s* to 100 *Mbit/s*.
- c. **Spectrum efficiency** The minimum requirements for peak spectral efficiencies in IMT-2020 are 30*bit/s/Hz* for downlink, and 15 *bit/s/Hz* for uplink. The peak spectral efficiency denotes the maximum data rate under ideal conditions normalized by channel bandwidth. The available spectrum will extend from 3 *GHz* in 4G to 30 *GHz* in 5G.
- d. **Area traffic capacity** Area traffic capacity refers to the total traffic throughput served per

geographic area. The target value for area traffic capacity increased from 0.1 Mbits/s/m^2 on 4G to 10 Mbit/s/m^2 in 5G.

1.3.2.2 massive Machine Type Communications(mMTC)

Another key characteristic of 5G communication services is the scalable connectivity demand for the expanding number of wireless network-enabled devices, focusing on the efficient transmission of small payloads over an extended coverage area. The two central requirements to enable mMTC are:

- a. **Network energy efficiency** Network energy efficiency is important for eMBB and it is expected to increase from $1x$ on IMT-advanced to $100x$ for IMT-2020.
- b. **Connection Density** A big challenge for 5G systems is to connect a massive number of devices to the internet, from 100 thousand connections per km^2 in 4G to 1 million connections per km^2 in IMT-2020.

1.3.2.3 Ultra-Reliability Low Latency Communications (URLLC)

Forthcoming network services, remote surgery, mission-critical applications, vehicle-to-vehicle (V2V) communications, high speed train connectivity and smart industry applications, will prioritize extreme reliability, low-latency and mobility, over data rates. The crucial requirements to enable URLL communications are:

- a. **Mobility** Mobility is described as the maximum mobile station speed at which a defined QoS can be achieved (in km/h).
- b. **Latency** Latency is probably one of the most influential performance measures of 5G. A reliable 5G system requires extremely low latency, even a few milliseconds (ms) can make an enormous difference. The requirements for IMT-2020 give no room for unbounded delay, from an admissible 10 ms in 4G, to less than 1 ms in the specification for 5G.

1.3.3 5G Enabling Technologies

Research has proposed technologies that reinforce the enhancement of 4G technologies in order to achieve 5G requirements in a more efficient and intelligent way. Some of this technologies are:

1.3.3.1 Massive Multiple-Input Multiple-Output (MIMO)

MIMO is an antenna technology for wireless communications, in which multiple antennas are used at both the transmitter and the receiver as **figure1.4** shows.

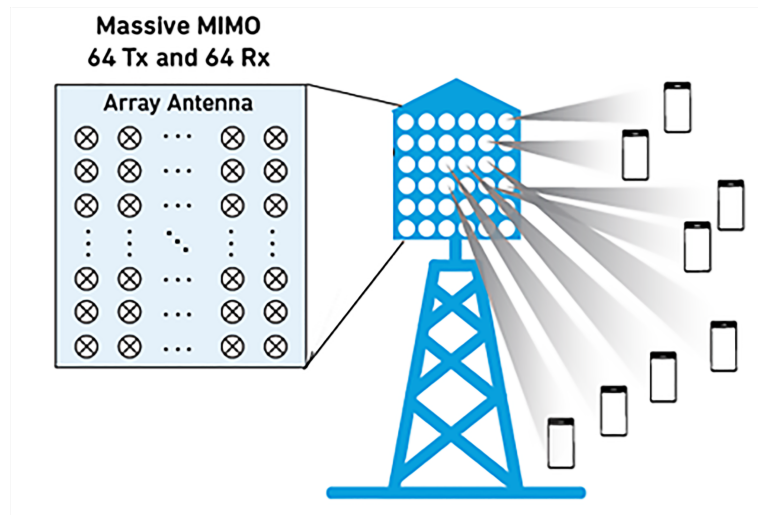


Figure 1.4: MIMO [1].

1.3.3.2 Beam Forming

Digital Beam forming can be utilized for 5G portable correspondences and can point a flag from a sender to a receiver when they in observable pathway. **Figure 1.5** show that instead of broadcasting toward each path it would enable the base station to center stream of information to particular client.

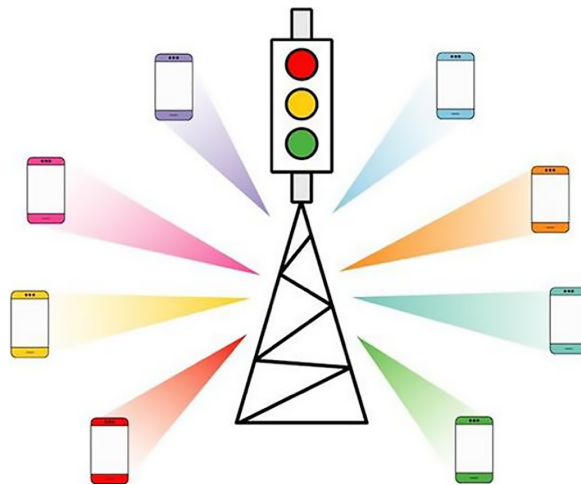


Figure 1.5: Beam Forming [18].

1.3.3.3 Small Cell Networks

Based on **figure 1.6** a small cell is essentially a scaled down base station (Macro Cell) that splits up a cell site into smaller pieces. The principal objective of small cells is to expand the full-scale cells edge information limit, speed and general system proficiency.

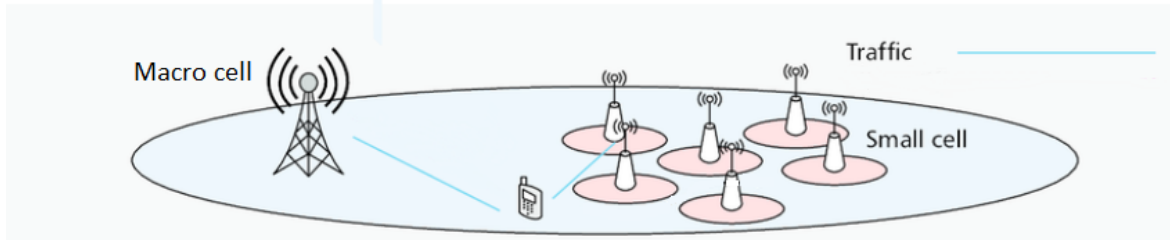


Figure 1.6: Small Cells [8].

1.3.3.4 Millimeter-wave signals (Mm-wave)

Taking into account the rapid increase in data traffic, a new technology mm-Wave is introduced in 5G to improve cell capacity and enhance the capabilities of new technologies adopted in 5G. As shown in **figure 1.7** the availability of bands in the range of 20-100 GHz makes mmWave a lucrative prospect in the design of 5G networks.

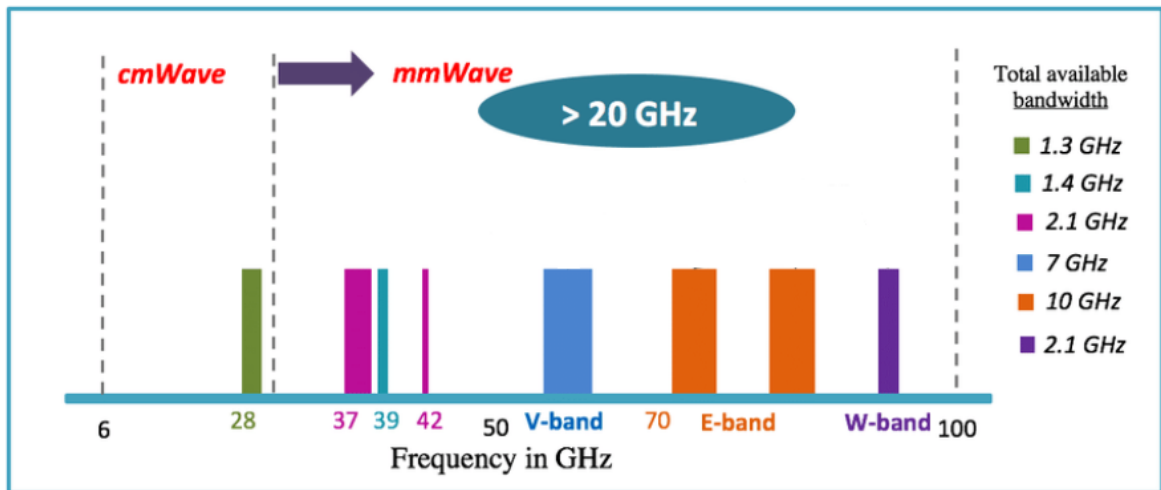


Figure 1.7: Millimeter-wave signals (Mm-wave) [21].

1.3.3.5 Network slicing (NS)

Through the joint use of SDN and NFV, a multitude of independent virtual networks can be abstracted from a single network. Each of these networks is specialised to meet the specific requirements of an end-to-end service. This idea of splitting the network into multiple isolated networks is a new paradigm called network slicing [9]. Its purpose is to logically adapt the management of network infrastructure and resources to meet the promise of 5G.

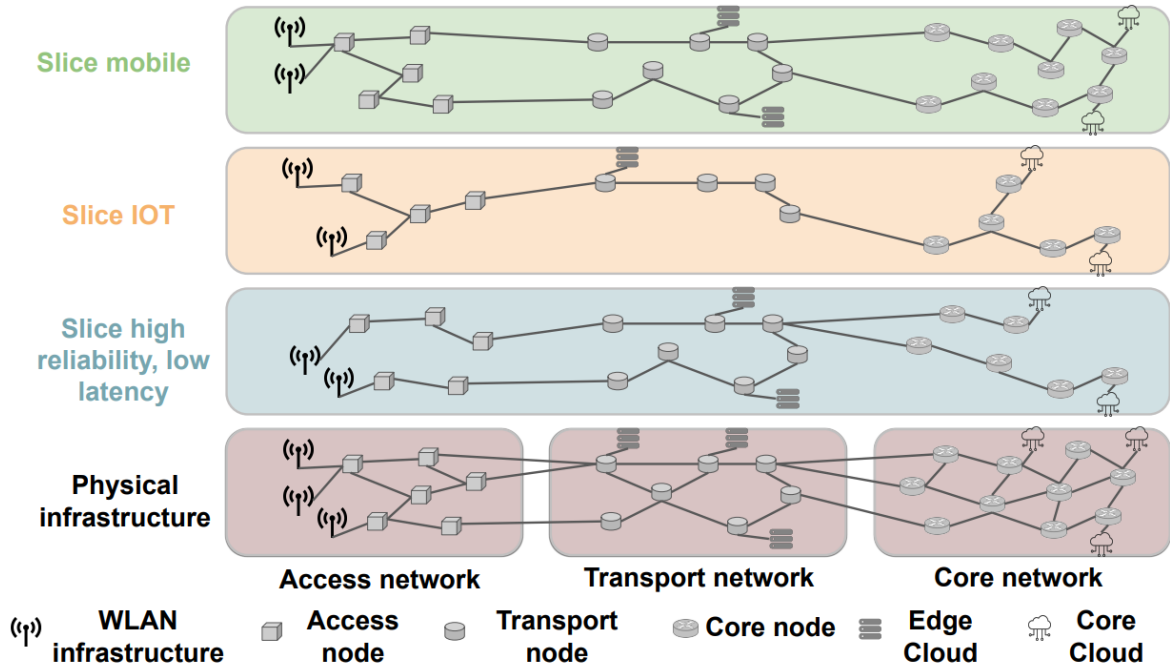


Figure 1.8: Network Slicing architecture [2].

An example of the network slicing architecture can be seen on **figure 1.8**. The network infrastructure is on the last layer and covers the access network, the transport network and the core network. In this example there are three slices, each slice has several entry points on the access network by the help of Wireless Local Area Network (WLAN) infrastructure. Then every slices passe through network functions on edge clouds in the transport network. Finally they connect to datacenters in the core network.

1.4 Network Function Virtualization (NFV)

NFV emerging as a breakthrough in 5G systems, decouples physical hardware and underlying network functions and let the network functions run centrally on generic cloud servers, thus providing advantages in scalability and flexibility. NFV greatly reduces Capital Expenses (CAPEX) required to buy hardware devices and saves Operating Expenses (OPEX) by aggregating resources for virtual network functions that run on a centralized server pool [11].

1.4.1 NFV architecture

According to ETSI, the overall architecture of NFV consists of four key elements: NFV Infrastructure (NFVI), Virtual Network Functions (VNFs), hypervisors, and NFV Management and Orchestration (MANO), as shown in **figure 1.9**. Specifically, the main component of NFV is VNFs which is software implementations of network functions, running on a generic cloud infrastructure.

VNFs are deployed upon the NFVI that includes virtual computation, virtual storage, and virtual network resources. These virtual resources, created by hypervisors, bring virtualization over physical hardware resources within the network. In particular, hardware resources might include networking, computing and storage infrastructures. Moreover, the NFV MANO framework controls the provisioning of VNFs, the configuration of VNFs and the infrastructure they run on. MANO can also chain several VNFs to activate an end-to-end service [13].

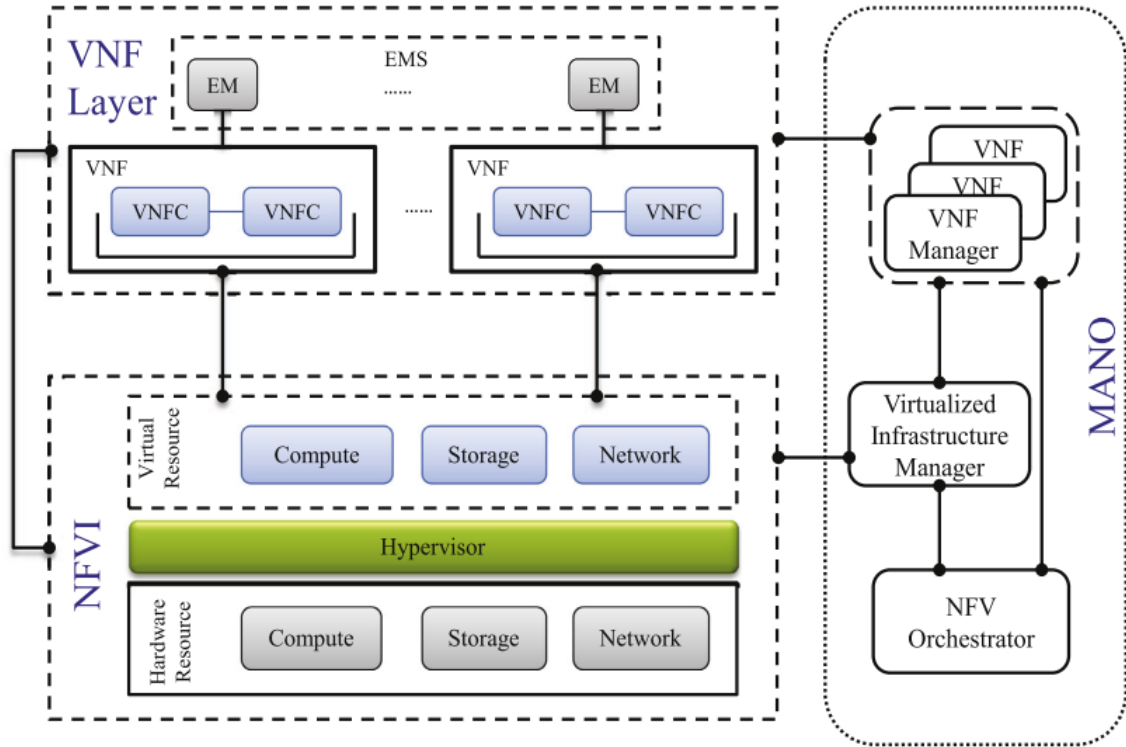


Figure 1.9: ETSI NFV reference architecture [12].

We briefly explain each component in **figure 1.9** as follows:

1.4.1.1 Virtual Network Functions (VNFs)

A network function refers to the functional component of network infrastructure that provides a well-defined functional behaviour and external interfaces. Some examples of network functions are DHCP servers, firewalls, NAT and gateways.

1.4.1.2 NFV Infrastructure (NFVI)

NFVI is the infrastructure platform over which the VNFs are deployed. Specifically, NFVI is the collection of the physical and software resources.

- a. **Hypervisors** Hypervisors provide the abstraction of virtual resources over physical hardware for VNFs to run on. Also, hypervisors setup the virtualization layer that offers virtual machines

and virtual networks over physical resources. Hypervisors provide a logical slicing of actual network infrastructure into virtual networks for efficient management.

1.4.1.3 NFV Management and Orchestration (MANO)

According to the ETSI MANO framework [15], NFV MANO provides the framework for managing and orchestrating all infrastructure resources. Specifically, it provides virtual machines, configures the VMs and the physical infrastructure and manages of physical resources for VMs. The MANO contains three functional blocks:

- a. **NFV orchestrator** The NFV orchestrator handles on-boarding new network services and VNFs, manages global resources and validates, authorizes NFVI resource requests.
- b. **VNF manager** The VNF manager manages the life-cycle of VNF instances and coordinates the configuration and event reporting between NFVI and network management software.
- c. **Virtual infrastructure manager** The virtual infrastructure manager controls and manages the compute, storage and network resources of NFVI.

1.5 Software Defined Network (SDN)

SDN is an approach that brings intelligence and flexible programmable 5G networks capable of orchestrating and controlling applications/services in more fine-grained manner. The Open Network Foundation (ONF) [17] is a non-profit consortium dedicated to the development, standardization and commercialization of SDN. The ONF gives a definition of SDN as follows: “the physical separation of the network control plane from the data plane” [16]. This separation results into flexibility and centralized control with a global view of the entire network. It also provides capabilities of responding rapidly to changing network conditions, business, market and end user needs.

1.5.1 SDN architecture

Based on the Open Network Foundation (ONF), a high-level SDN architecture is presented, which is composed of three main planes, including data plane, control plane and application plane. The architectural components of each plane and their interactions are shown in **figure 1.10**.

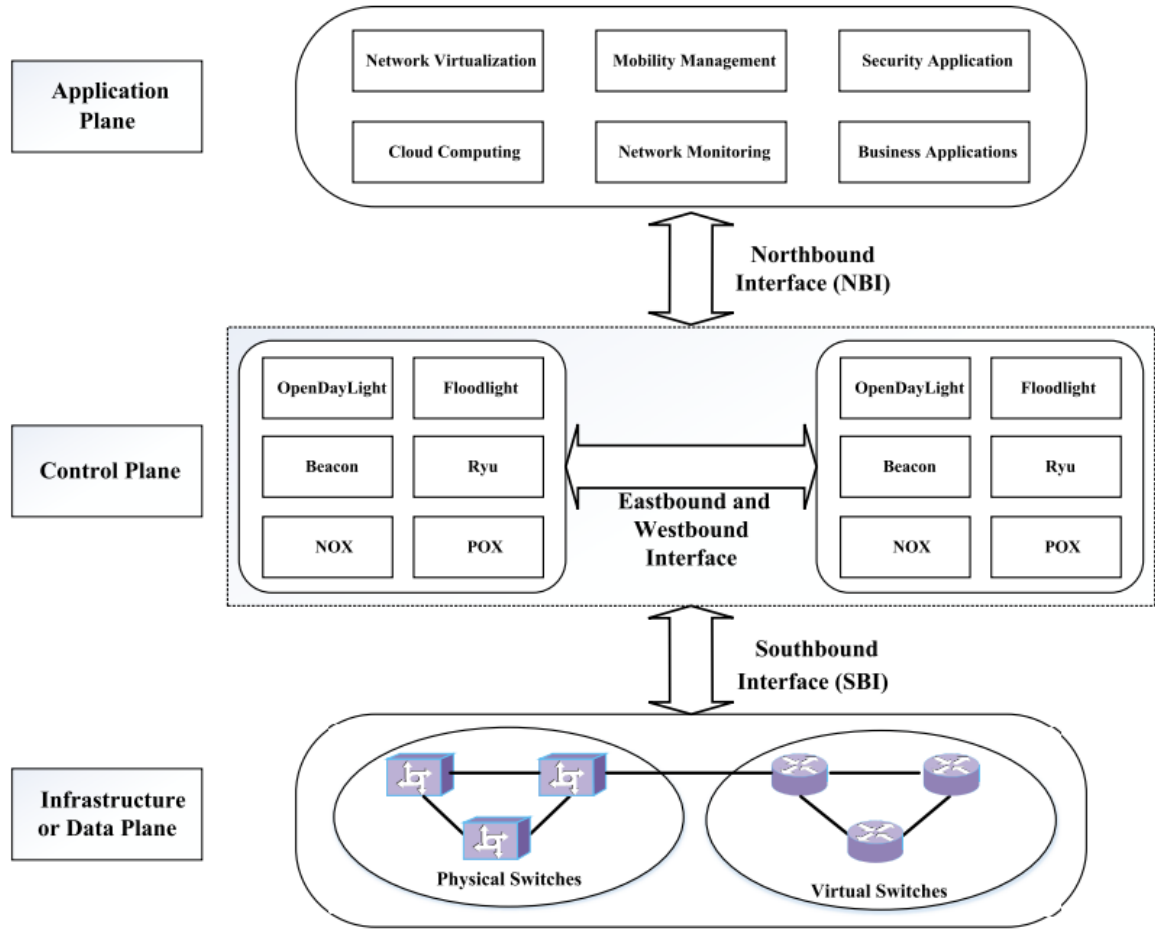


Figure 1.10: ONF SDN reference architecture [22].

In the following, we will give a detailed representation of these three planes and their interactions:

1.5.1.1 Data plane

The data plane also known as infrastructure plane, is the lowest layer in SDN architecture. This plane is comprised of forwarding devices including physical switches and virtual switches. Virtual switches are software-based switches, which can run on common operating systems such as Linux. Open vSwitch, Indigo and Pantou are three implementations of logical switches.

1.5.1.2 Control plane

The control plane is the “brain” of SDN systems, which can program network resources, update forwarding rules dynamically and make network administration flexible and agile. The main component of control plane is the logically centralized controller, which controls the communication between forwarding devices and applications. The controller exposes and abstracts network state information of the data plane to the application plane. Additionally, the controller provides essential functionalities that most of network applications need, such as shortest path routing, network topology storage,

device configuration and state information notifications. There are many controller architectures, such as POX, Ryu and Open Daylight.

Three communication interfaces allow the controllers to interact:

- a. **Northbound** The northbound API interface on the controller enables applications and the overall management system to program the network and request services from it. This application tier often includes global automation and data management applications, as well as providing basic network functions such as data path computation, routing and security.
- b. **Southbound APIs** Though not explicitly required by SDN, OpenFlow is a protocol often used as the southbound API that defines a set of open commands for data forwarding. These commands allow routers to discover the networks topology and define the behavior of physical and virtual switches, based on application requests sent via the northbound APIs.
- c. **The eastbound/westbound** interfaces are used in the multi-controller SDN networks. When deploying SDN in large-scale networks where a vast amount of data flows needs to be processed, due to the limited processing capacity of one controller, the large-scale networks are always partitioned into several domains. Each domain has its own controller.

1.5.1.3 Application plane

The highest layer in the SDN architecture, which composed of business applications. These applications can provide new services and perform management and optimization. In general, the applications can obtain the required network state information through controllers. Based on the received information and business requirements, the applications can implement the control logic to change network behaviours.

1.6 Service Function Chain (SFC)

Service function chaining is defined as an ordered sequence of VNFs and subsequent steering of flows through them to provide end-to-end services. SFC using SDN and NFV, facilitates implementation of 5G network slicing. An example of an SFC request, as shown in **figure 1.11** consists of the client (source), set of VNFs (firewall, Deep Packet Inspection, proxy,...) in a particular order and a server (destination).

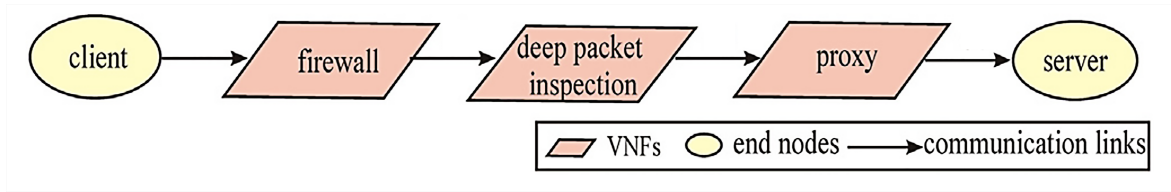


Figure 1.11: SFC request example [11].

1.6.1 Static service function chaining

In traditional service chaining, network functions are implemented as hardware middle boxes, and all are physically connected. Firewall (FW), Network Address Translation (NAT), Intrusion Prevention System (IPS), ...are examples of middle boxes used by network operators. In the static type every packet or flow will have to pass through the chain, although some requests need only a subset of these network services.

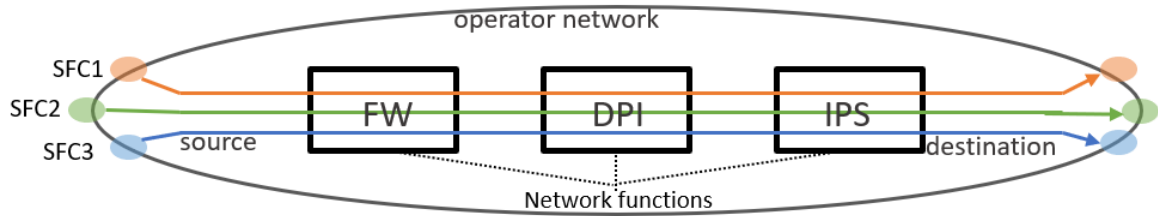


Figure 1.12: SFC Static [20].

As shown in **figure 1.12**, the source wants to send the request to destination and needs specific network function. But in static service chaining, traffic must pass through the entire network functions regardless of the need.

The following are the limitations of this approach according to [20]:

- Every device should have enough capacity to handle the full traffic.
- It is not possible to apply only desired network functions based on specific flow.
- CAPEX and OPEX cost due to purchasing new hardware devices if the existing topology are not able to fulfill it.
- The network devices must be physically connected and manually configured by network operators which may lead to inconsistent configuration.

1.6.2 Dynamic service function chaining

In dynamic service chaining, SDN and NFV replace traditional Network Function with Virtual Network Function and allow dynamic service chaining. In dynamic service chaining, the traffic needs

to be steered only through desired network functions according to specific flow requirements. SDN controller can create chains dynamically and forward traffic intelligently to a particular network function based on the label such as VLAN, source MAC address, network service header (NSH). This type of chain is called a software control service chain.

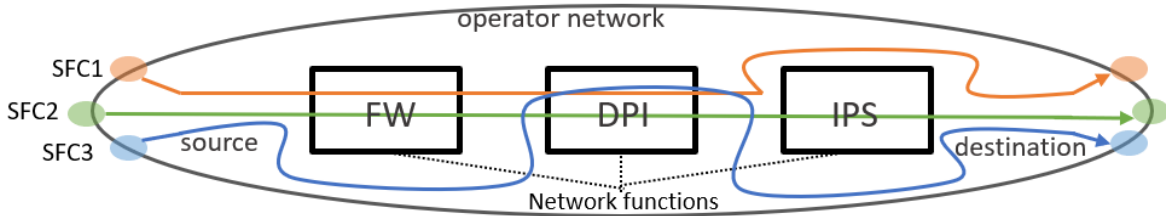


Figure 1.13: SFC dynamic [20].

As shown in **figure 1.13**, SFC1 needs only FW and DPI. The SDN controller creates a service function chain for SFC1: “FW, DPI” in which traffic will pass through FW and DPI. The controller will create separate SFC of SFC2: “FW, DPI, and IPS” in which traffic will pass through all the network functions. The SFC3 will pass through the only DPI using chain: “DPI”.

Based on [11] the following are the advantages of dynamic SFC:

- Service function chaining with SDN/NFV provides greater flexibility for end-to-end service provisioning.
- Reduces capital, operational cost as controller steers the traffic to only essential network function and eliminates over-provisioning of the network.
- SDN provides scalable, dynamic, flexible and automatic service function chaining.

1.6.3 Service function chaining architecture

The delivery of end-to-end services to the user depends on a series of network functions. The network functions such as FW, IDS and Web Proxy are sequentially processed in a chain called SFC. Internet Engineering Task Force (IETF) [19] has developed SFC architecture, and it defines all the components of SFC. SFC architecture consists of Service Classification Function (SCF), Service Function (SF), Service Function Forwarder (SFF), SFC enabled domain and SFC proxy, these components communicate with each other with the help of SFC Encapsulation. **figure 1.14** represent the IETF SDN based SFC architecture it defines it as a two-layered architecture consisting of data plane and control plane components.

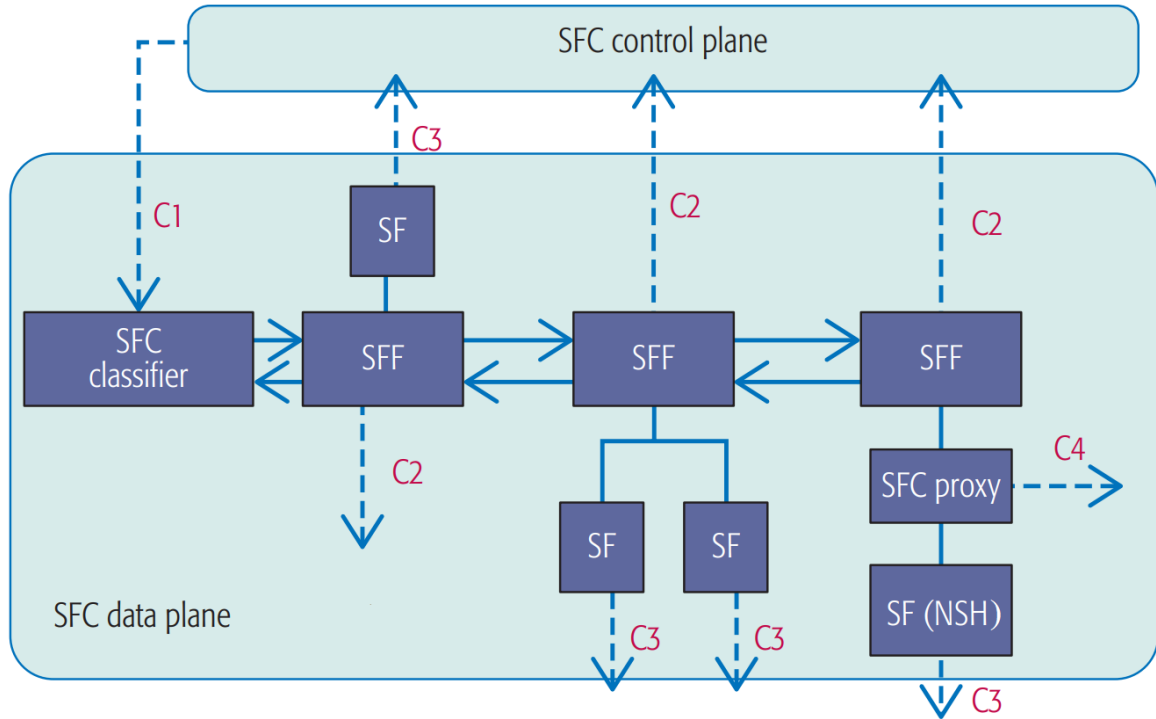


Figure 1.14: IETF SFC reference architecture [16].

1.6.3.1 SFC data plane component

The following elements are presented in the SFC data plane component in **figure 1.14**:

- a. **SFC classifier** This component is responsible for the classification of data. When a packet enters into the network, SCF classifies and match it with available policies and then chooses the appropriate SFC.
- b. **Service Function (SF)** SF is responsible for performing a particular network function. SF is a logical or virtual component used to give specific treatment to a packet. There may be multiple instances of the same network function that can be present in the SFC enabled domain. Network functions can be one of this two types:
 - If the data sent to SF contains SFC encapsulation, then it is called SF aware.
 - If the data sent to SF does not include SFC encapsulation, then it is called SF unaware.
- c. **Service Function Forwarder (SFF)** It is responsible for forwarding the traffic from one SF to another SF or SFF, according to the attached SFC encapsulation.
- d. **Service Function Chain (SFC)** SFC is an abstract view of SF that is applied to the packet resulting from SCF.
- e. **Service Function Path (SFP)** The actual path that consists of a number of SFFs and SFs to steer the traffic, is called SFP.

- f. **Service Function Chaining Proxy (SFC-Proxy)** SFC proxy inserts and removes SFC encapsulation for unaware SFs.
- g. **Service Function NHS** The network service header (NSH) provides an identifier that is used to forward the packet to particular SFF and SF. It is used for SFP identification.

1.6.3.2 SFC control plane component

The primary responsibility of the SFC control plane component is the management and controlling of SFC, management of SFs, the mapping of an abstract view of SFC to actual SFP and inserting rules into SFF components of data plane. It is responsible for dynamically changing the SFP if any SF or link is overloaded or inactive due to an error. It is also used for administrating and controlling SFC data plane components. The SFC control plane components interact with the SFC data plane components via four reference interfaces as **figure 1.14** shows. The first interface C1 is responsible for pushing the SFC classification rules defined by the SFC control plane into the SFC classifiers. The SFFs report the connectivity status of their attached SFs to the SFC control plane. Interface C3 is between the NSH SFs and the SFC control plane. It is used to collect some packet-processing statistics (SFs' load update) from the SFs. For NSH SFs, a SFC proxy is provided for collecting statistics (SF processing latency and workload) and transmitting this information over the C4 interface to the SFC control plane. The SFC control plane uses these statistics (received through interfaces C2, C3, and C4) to dynamically adjust the SFPs.

1.6.4 SDN/NFV architecture for SFC deployment

The limitations of the static SFC model can be eliminated by combining key technologies of SDN and NFV. The central controller of SDN architecture has a global view of the whole network topology. The controller will update the rules into the flow table of the switch, thus providing centralized control of VNFs.

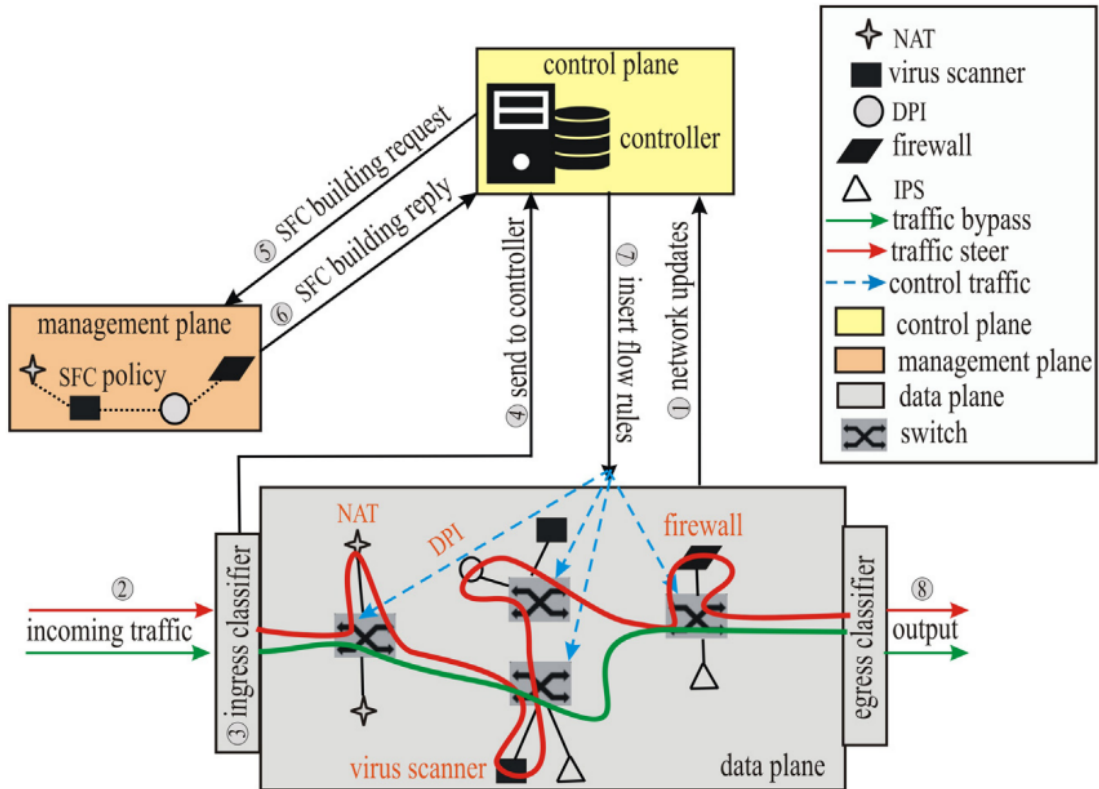


Figure 1.15: SDN/NFV based dynamic Service Function Chaining [11].

The SDN/NFV based architecture for SFC consists of three types of components, namely Orchestration Plane, Control Plane, and Data Plane, as shown in **figure 1.15**. The main responsibility of the Orchestration Plane is to build SFC strategies to control the global network according to different network traffic or user demands. The SDN controller adds flow rules into the OpenFlow table of the switch to orchestrate different VNFs. According to a particular SFC strategy, the SDN controller performs mapping of VNFs and virtual links onto a substrate network and form an SFP. The switches/routers and NFV platforms reside in the data plane and are responsible for the flow of traffic and service processing.

1.7 Conclusion

5G technology is being adopted as a global standard and promises high bandwidth, low latency, improved reliability, increased availability and uniform user experience. NFV and SDN are complementary technologies that help overcome architectural challenges in deployment of 5G by providing capabilities such as network slicing. An important factor in ensuring QoS delivery to users in these networks is the deployment of dynamic SFCs. In the next chapter we will see the different types of machine learning, especially Deep Reinforcement Learning with its application in networking and related work.

Chapter 2

Machine Learning and Networking

2.1 Introduction

Machine Learning (ML) is everywhere, from medical diagnosis based on image recognition to navigation for self-driving cars. ML has been evolving as a discipline to the point that it currently allows networks to learn and extract knowledge by interacting with data.

Driven by the demand to accommodate today's growing mobile traffic, 5G is designed to be a key enabler and a leading infrastructure provider in the information and communication technology industry by supporting a variety of forthcoming services with diverse requirements.

In SDN/NFV-enabled networks, Service Function Chains (SFCs) has become a popular networking service paradigm define as a set of ordered or partially ordered VNFs. The SFC deployment is the problem of choosing a set of optimal locations for a VNF and chain them according to the order of service request.

This chapter is organized as follows: we start with the definition of machine learning and its categories: supervised learning, unsupervised learning and reinforcement learning. After that, we will walk in detail through Deep Reinforcement Learning. Then, we introduced machine learning applications in networking, especially in 5G networks. Finally, we present a related work analysis by selecting the works that are most closely related to our project.

2.2 Machine Learning

Machine learning addresses the question of how to build computers that improve automatically through experience. It is one of today's most rapidly growing technical field, lying at the intersection of computer science and statistics and at the core of artificial intelligence and data science.

2.2.1 Machine Learning Types

ML approaches are classified into three primary categories based on how the learning is achieved: Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL). Each ML category can be divided into several sub-classes dealing with very specific algorithms.

2.2.1.1 Supervised Learning

Supervised learning (SL) uses labelled training datasets to create models that map inputs to corresponding outputs. Typically, this approach is used to solve classification and regression problems that pertain to predicting discrete or continuous valued outputs, respectively. For example, a classification problem can be to identify an attack as either denial of service (DoS), root-to-local (R2L), user-to-root (U2R), or probing. A regression problem can be to predict the time of future attacks [27].

Figure 2.1 shows that, SL model based-on the input labelled data will predict the output.

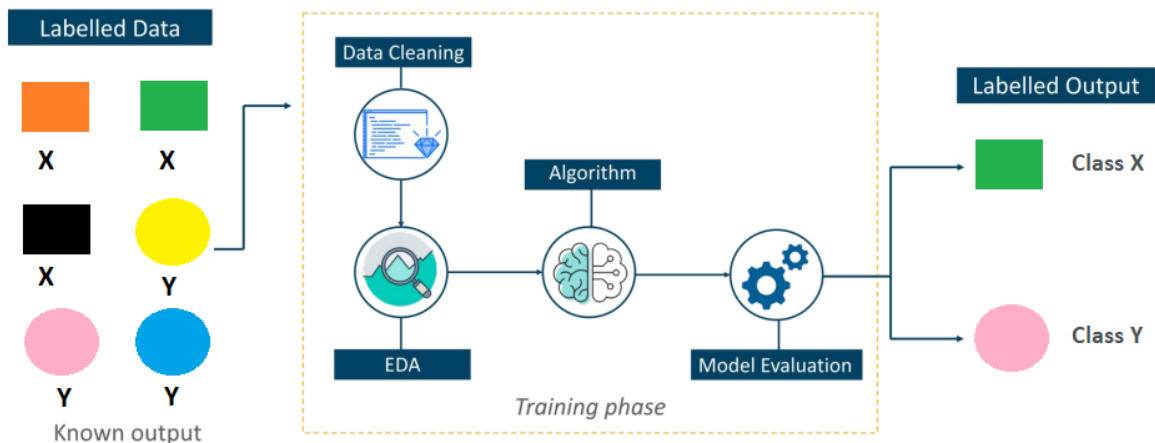


Figure 2.1: Supervised Learning [25].

Example of Supervised Learning algorithms:

- Linear Regression.
- Lasso and Ridge Regression.
- Naive Bayesian (NB).
- Support vector machine (SVM).
- etc...

2.2.1.2 Unsupervised Learning

Unsupervised learning (UL) uses unlabelled training datasets to create models that find dominant structure or patterns in the data. This approach is appropriate for clustering, outlier's detection and density estimation problems. For example, the clustering problem can pertain to grouping different instances of attacks based on their similarities [27].

Figure 2.2 shows that, UL model based-on the input unlabelled data will predict the output as clusters formed by feature similarity.

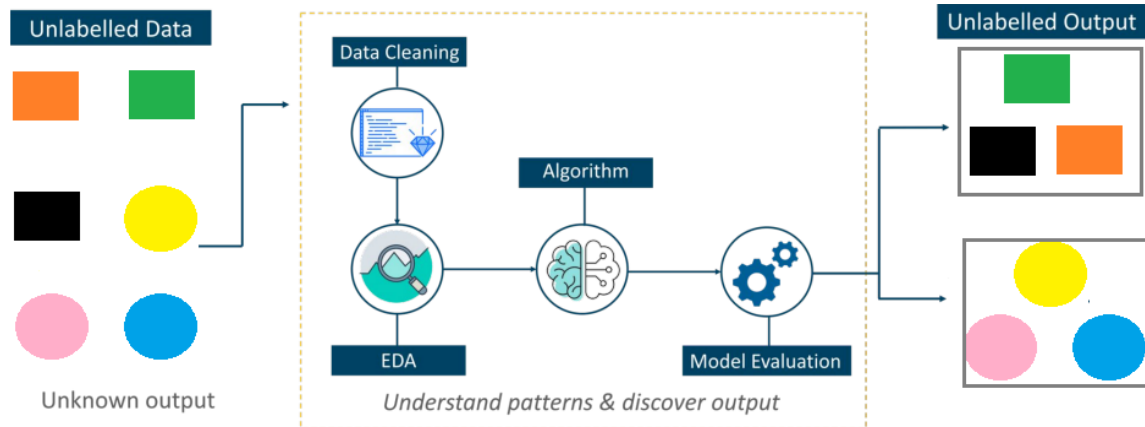


Figure 2.2: Unsupervised Learning [25].

Example of Unsupervised Learning algorithms:

- K-Means.
- Fuzzy-C-Means.
- Latent Semantic Analysis (LSA).
- Principal Component Analysis (PCA).
- etc...

2.2.1.3 Reinforcement Learning

Reinforcement Learning (RL) is an iterative process that uses the feedback from the environment to learn the correct sequence of actions to maximize a cumulative reward [27]. **Figure 2.3** shows that at time t , the agent is in state s_t and decides to perform an action a_t , at the next time step $t + 1$, it arrives in the state s_{t+1} and obtains the reward r_t . The goal of the agent is to maximize the reward obtained on the long term.

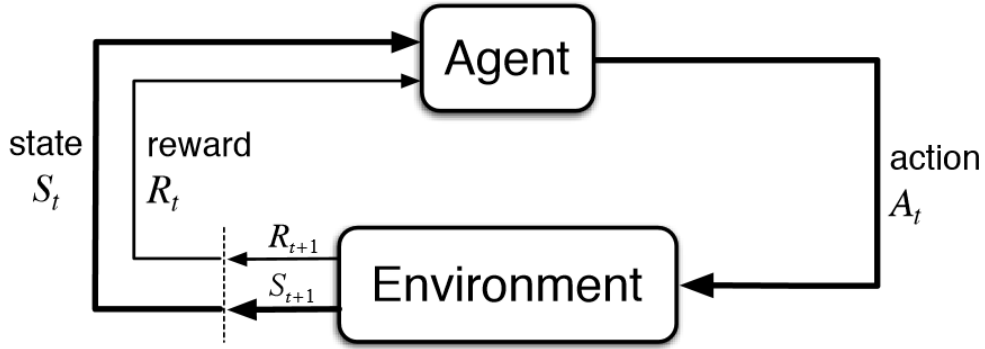


Figure 2.3: Interaction between an agent and its environment [25].

Agent represents the “solution”, which is a computer program with a single role of making decisions (actions) to solve complex decision-making problems.

Environment is the representation of a “problem”, which is everything that comes after the decision of the Agent. The environment responds with the consequences of those actions, which are observations or states, and rewards.

Reinforcement learning problems can be modeled as Markov Decision Processes (MDP), in [2] authors defined it by five quantities:

- A state space S of states s .
- An action space A of actions a .
- P the transition probability function with $P(s', r | s, a)$ the probability of transition from state s to state s_{t+1} under action a while receiving reward r . With \mathcal{P} the symbol of probability:

$$P(s', r | s, a) = \mathcal{P}[s_{t+1} = s', r_{t+1} = r | s_t = s, a_t = a] \quad (2.1)$$

- R the reward function which gives the expected value of the next reward r_{t+1} when taking the action a on state s :

$$R(s, a) = E[r_{t+1} | s_t = s, a_t = a] = \sum_{r \in R} r \sum_{s' \in S} P(s', r | s, a) \quad (2.2)$$

- $\gamma \in [0, 1]$ is the discount factor for future rewards. It specifies the extent to which future rewards impact on the outcome of the current action.

All states transition have the Markovian property: given the current state s and action a , the next state s_{t+1} is conditionally independent of all the previous states and actions.

Example of Reinforcement Learning algorithms:

- Asynchronous Advantage Actor Critic (A3C).

- Asynchronous Actor Critic (A2C).
- Q-learning.
- Soft-Actor-Critic (SAC).
- etc...

2.2.2 Difference between Machine Learning types

Figure 2.4 represents the difference of the input, output and the training goal between supervised learning, unsupervised learning and reinforcement learning




Supervised Learning	Unsupervised Learning	Reinforcement Learning
Data: (x, y) x is data, y is label	Data: x x is data, no labels!	Data: state-action pairs
Goal: Learn function to map $x \rightarrow y$	Goal: Learn underlying structure	Goal: Maximize future rewards over many time steps
Apple example:  This thing is an apple.	Apple example:  This thing is like the other thing.	Apple example:  Eat this thing because it will keep you alive.

Figure 2.4: Difference between SL, UL and RL [43].

- Reinforcement Learning vs Supervised Learning** Let's understand the difference between supervised and reinforcement learning with an example. Imagine we want to train a model to play chess using supervised learning. In this case, we will train the model to learn using a training dataset that includes all the moves a player can make in each state, along with labels indicating whether it is a good move or not. Whereas in the case of RL, our agent will not be given any sort of training data, instead, we just provide a reward to the agent for each action it performs. Then, the agent will learn by interacting with the environment and it will choose its actions based on the reward it gets.
- Reinforcement Learning vs Unsupervised Learning** Similar to supervised learning, in unsupervised learning, we train the model based on the training data. But in the case of unsupervised learning, the training data does not contain any labels and this leads to a common misconception that RL is a kind of unsupervised learning due we don't have labels as input data

but it is not. In unsupervised learning, the model learns the hidden structure in the input data. Whereas, in RL, the model learns by maximizing the reward.

2.3 Deep Reinforcement Learning

We obtain Deep Reinforcement Learning (DRL) methods by combined Reinforcement Learning (RL) with Deep Neural Networks (DNN) to overcome the limitations of RL in complex environments with large state spaces or high computation requirements. DNN used to approximate any of the following components of RL: value function $V^*(s; \theta)$ or $Q^*(a, s; \theta)$ and policy $\pi^*(a | s; \theta)$ [18]. Here, the parameters θ are the weights in deep neural networks. DRL have the same interaction between agent and environment in RL plus the use of DNN as shown in **figure 2.5**.

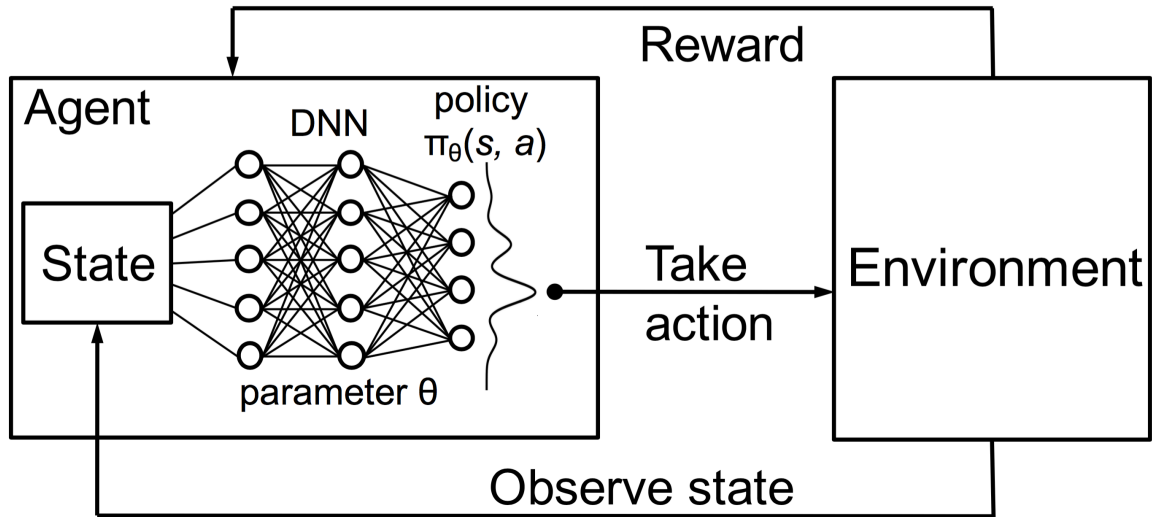


Figure 2.5: Interaction between an DRL agent and its environment [24].

2.3.1 Policy and Value function

By interacting with its environment, the agent determines which actions produce the greatest reward and uses this experience to improve its performance on future trials. The agent's main objective is therefore to maximise the total amount of reward received (good actions are reinforced). To this end, the agent's behaviour is determined by a policy π . It provides a direction on the action to take in a certain state. $\pi(a | s)$ gives the probability of taking action a in state s [28].

$$\pi(a | s) \in [0, 1] \quad (2.3)$$

$$\pi(a | s) = P(a_t = a | s_t = s) \quad (2.4)$$

The return G_t is a weighted sum of the future rewards starting from time t :

$$G_t = \sum_{k=0}^T \gamma^k r_{t+k+1} \quad (2.5)$$

Authors in [28] represent the return equation by (2.5) where $0 < \gamma < 1$ is the discount rate and r_t represents the reward obtained during the transition from s_t to s_{t+1} .

By setting the γ parameter to a value smaller than 1, we ensure that the further into the future a potential reward is the less impact it has on the G_t return.

If the task is episodic T is finite, the trajectories ends after a finite number of transitions k , but if the task is continuing $T = \infty$, trajectories have no end.

To choose the direction offering the best reward, a value function is associated $V(s)$ to each state. It predicts the expected value of the future rewards related to a state. The bigger the value function for a state, the better the state is. The value function $V^\pi(s)$ is the expected return starting with state s by following policy π [28].

$$V^\pi(s) = E_\pi[G_t | s_t = s] \quad (2.6)$$

The action-value (or Q-value) of a state action pair is the expected return starting with state s and first performing action a , before following policy π [28].

$$Q^\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a] \quad (2.7)$$

By following the target policy π , the state-value can be rewritten using the probability distribution over the possible actions and the Q-values [28].

$$V^\pi(s) = \sum_{a \in A} \pi(a, s) \cdot Q^\pi(s, a) \quad (2.8)$$

The optimal policy π^* achieves the optimal value functions: it maximises the expected cumulative reward. The optimal policy has a corresponding state-value function [28].

$$V^\pi(s)^* = \max_\pi V^\pi(s) \quad (2.9)$$

$$Q^\pi(s, a)^* = \max_\pi Q^\pi(s, a) \quad (2.10)$$

2.3.2 Exploration vs Exploitation

To find the optimal policy π^* and to be able to exploit it, the agent has to first explore the different states with every actions. However, this is not a good method. Indeed, it does not allow for scaling

when the number of states and actions are too large [2].

A simple and very common method is the ϵ -greedy policy. This method allows to control the exploration rate in relation to the exploitation rate. The value $\epsilon \in [0, 1]$ is the probability for the agent to choose a random action at each step: it is the exploration probability. Similarly, $1 - \epsilon$ is the probability of exploitation: the probability of following the policy.

The value ϵ can be a fixed or varied value, a commonly used method is the decay, the value of epsilon is close to 1 at the start of the algorithm to encourage a strong exploration at the beginning. Epsilon then decreases with each iteration to reach a minimum value (down to 0) to encourage exploitation. A good configuration of the decay allows not to fall in a local optimal at the beginning and to converge faster at the end of the training of the agent.

2.3.3 Deep Q-learning

Deep Q-learning Network (DQN) makes use of a deep neural network to approximate the Q-value function for potentially high-dimensional or continuous state-space problems. The state is given as the input and the Q-value of all possible actions is generated as the output of the neural network. The Q-value Q_{θ_t} is a vector represented by θ the first neural network named Q-network. A second neural network $\bar{\theta}$ called the Target network (usually a copy of the first Q-network) is used to calculate a target Q-value "y" (the value of the best possible choice in the target network). The target is used to train the network and compute the loss function [2]:

$$y_t = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \bar{\theta}_t) \quad (2.11)$$

The loss function is [2]:

$$L = y_t - Q(s_t, a; \theta_t)^2 \quad (2.12)$$

The aim of the learning algorithm is not only to maximise the reward, it is above all to learn, and therefore to be able to predict the reward. As the training progresses, the value of the sum of the rewards of each episode should increase while the value of the loss function should decrease and converge to a minimum value.

2.3.3.1 Deep Q-learning algorithm

Algorithm 1 as presented in [10], shows how the model learns the optimal policy π^* , it require the selection of DNN hyper-parameter $\bar{\theta}$ and θ , the discount factor γ , probability of chosen random action ϵ , the experience replay memory D used for store the MDP transitions and mini-batch m used for train the Q-network and update the weights. At *line 2* we create two Neural Networks the second used for avoid divergence by reducing correlation between $Q(s, a; \theta)$ and target $\bar{Q}(s_{t+1}, a; \bar{\theta})$. For

each step a random action with ϵ will be taken for exploration and $1 - \epsilon$ for maximize the rewards by chosen optimal action, after chosen an action the agent observes the reward and the next state. At *line 8* the previous experience will be stored in D , a mini-batch m will be randomly sampled from D which eliminate the correlation to avoid over-fitting. At *line 11* compute the target y_j for sample j , at *line 13* update the weights and apply gradient descent to minimize the squared loss $(y_j - Q_j)^2$. The target model will be updated periodically after C time steps as *line 14* shows.

Algorithm 1: DQN algorithm

```

1 Initialize:  $\gamma \in [0, 1], \epsilon \in [0, 1]$ 
2 Create: Q-network model with weight  $\theta$ , Q-target model with weight  $\bar{\theta}$ 
3 Initialize:  $\theta, \bar{\theta}$  and replay memory  $D$ 
4 Observe initial state  $s_t$ 
5 for  $step \leftarrow 1$  to  $T$  do
6     |
7     | select :  $a_t = \begin{cases} a_t \text{ random action} & \text{with probability } \epsilon \\ \operatorname{argmax}_a Q(s_t, a_{t+1}; \theta) & \text{with probability } \epsilon - 1 \end{cases}$ 
8     | Observe reward  $r_t$  and next state  $s_{t+1}$ 
9     | Add experience to  $D$ 
10    | Sample random mini-batch  $m$  from  $D$ 
11    | for  $j \leftarrow 1$  to  $m$  do
12    |     |
13    |     | set :  $y_j = \begin{cases} r_j & \text{if } t + 1 = T \\ r_j + \bar{Q}(s_{j+1}, a_{j+1}; \bar{\theta}) & \text{otherwise} \end{cases}$ 
14    |     | Perform a gradient descent step on  $(y_j - Q_j)^2$  with respect to the weights  $\theta$ .
15    |     | end
16    |     | Every  $C$  steps reset  $\bar{Q} = Q$ 
17    | end

```

Figure 2.6 summarises the operation of the DQN architecture. The DQN is trained in several steps, it goes through a sequence of operations at each time step. First, the agent selects an ϵ -greedy action from the current state, executes it in the environment which returns the reward and the next state. This operation is saved in the experience buffer. A random batch of samples is then formed by recent and older samples. This batch of training data is given as inputs to both networks. The Q-network takes the current state and action of each data sample and predicts the Q-value for that

action. The target network takes the next state of each data sample and predicts the best Q-value of all actions that can be taken from that state. The Q-value, the target Q-value and the observed reward of the data sample are used to calculate the loss to train the Q-network. The processing is repeated for the next time steps. After T time steps, the weights of the Q-network are copied to the target network. The iteration continues until the training ends.

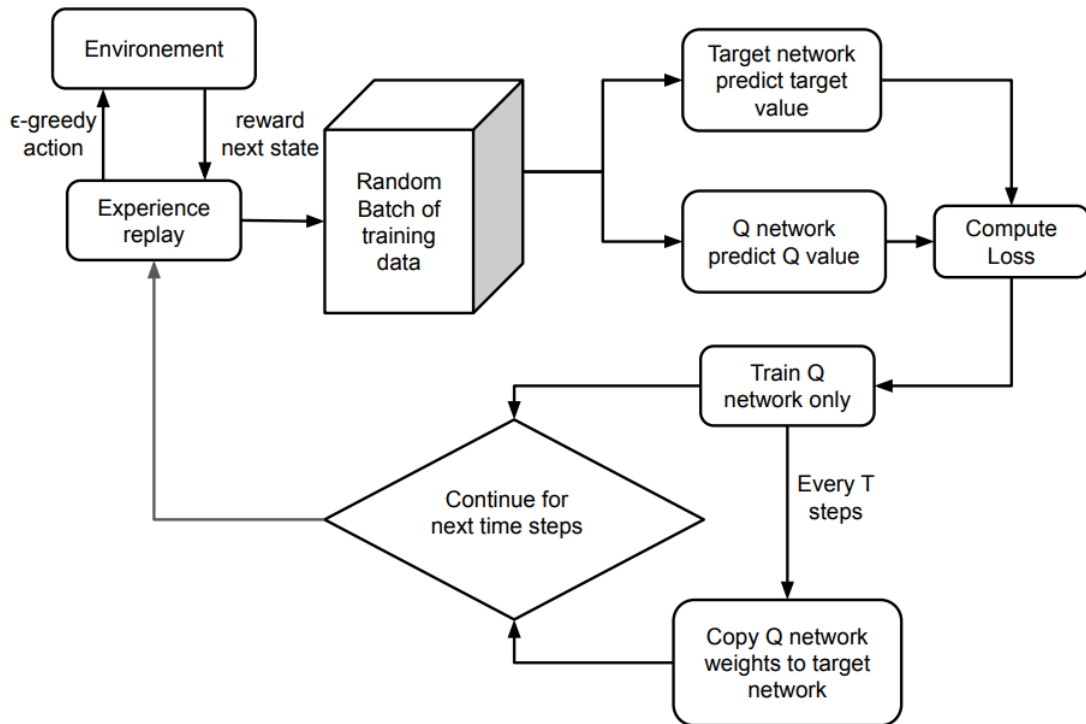


Figure 2.6: Deep Q Network operation [2].

2.4 Machine Learning in Networking

Machine learning (ML) has recently been applied to solve complex problems in many fields, including finance, health care and business. ML algorithms can offer computational models that can solve complex networking problems and consequently improve performance. ML is great for learning what normal network behavior looks like and highlighting anomalies relative to it. This understanding drives the utility of machine learning in networking in four areas, as detailed in [27]:

- a. **Performance management** Legacy network management systems equipped with Machine Learning can help both with moment-by-moment traffic management and with longer-range capacity planning and management. These tools can see if traffic is spiking in some places or failing to flow in others and they can direct automated or manual management responses. Existing efforts have concentrated on using ML for performance and traffic load prediction and quality of experience/ service (QoE/QoS) correlation.

- b. **Fault Management** Failure in networks is a norm rather than an exception and its impact can be quite costly. The slow reaction time and poor accuracy of traditional fault management techniques further increase this cost. This has motivated efforts that leveraged ML for proactive fault prediction. Additional works considered the usage of ML for fault localization and automated mitigation to minimize downtime and human intervention.
- c. **Configuration Management** As the network state is constantly changing, network managers find themselves constantly configuring the network to adapt to these changes, which is a cumbersome and error-prone process. ML can help automate this process by training models to identify optimal state-action pairs as the network behavior changes over time. A handful of works have showcased the benefits of ML for dynamic resource allocation and service configuration.
- d. **Security** Spotting anomalies in network behavior can help cybersecurity teams find everything from a compromised hardware node to an employee going rogue on the company network. Machine learning techniques have vastly improved the behavioral threat analytic space, as well as distributed denial-of-service detection.

Figure 2.7 shows the basic workflow for applying ML in networking, including problem formulation, data collection (offline and online), data analysis, model construction, deployment and inference and model validation.

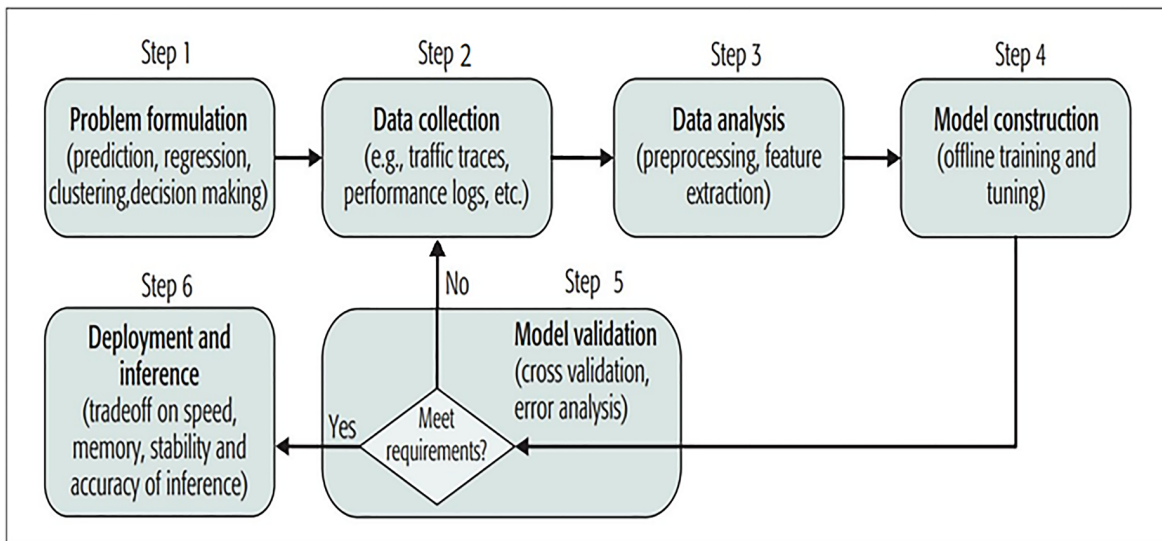


Figure 2.7: The typical workflow of machine learning for networking (MLN) [29].

Machine Learning strategies have a non-ignorable impact on modern attempts of the networking field. **Table 2.1** show how ML perform at each step of the ML in Networking (MLN) workflow

Networking application		Steps of MLN workflow					
Objectives	Specific works	Problem formulation	Data collection		Data analysis	Offline model construction	Deployment and online inference
			Offline	Online			
Traffic prediction	Traffic volume prediction	SL: prediction with Hidden-Markov Model (HMM)	Synthetic and real traffic traces with flow statistics	Only observe the flow statistics	The flow count and the traffic volume have significant correlation	Training model with Kernel Bayes Rule and Recurrent Neural Network with Long Short-Term Memory	Take flow statistics as input and obtain the output of the traffic volume
Resource management	Job scheduling	RL: decision making with DRL	Synthetic workload with different patterns is used for training	The real time resource demand of the arrival job	Action space is too large and may have conflicts between actions	Offline training to update the policy network	Directly schedule the arrival jobs with the trained model
Network adaptation	Routing strategy	SL: decision making with Deep Reinforcement Architecture	Traffic patterns labeling with routing paths computed by OSPF protocol	Online traffic patterns in each router	It is difficult to characterize the input and output patterns to reflect the dynamic nature of large-scale heterogeneous networks	Take the Layer-Wise training to initialize and the back-propagation process to fine-tune the DBA structure	Record and collect the traffic patterns in each router periodically and obtain the next routing nodes from the DBAs
Performance prediction	Throughput prediction	SL : prediction with HMM	Datasets of HTTP throughput measurement	Take users' session features as input	Sessions with similar features tend to behave in related pattern	Find set of critical features and learn a HMM for each cluster of similar sessions	A new session is mapped to the most similar session cluster and corresponding HMM are used to predict throughput
Configuration extrapolation	Cloud configurations extrapolation	SL: parameter searching with Bayesian optimization	Take performance under current configuration as model input	Large configuration space and heterogeneous applications		Take trials with different configurations and decide the next trial direction by Bayesian Optimization model.	

Table 2.1: Relationships between Network application and MLN workflow [29].

2.4.1 Special Considerations to deploy ML in Networking

Bringing ML into Networking production incurs a unique set of challenges that needs to be understood before starting any project or research. The most prevailing ones is:

2.4.1.1 The critical role of data

High-quality data is an essential piece in ML applications, and the type of data (labeled or unlabeled) is a key factor when deciding which type of learning to use, especially when it comes to deploying applications for 5G use cases.

2.4.1.2 Hyper-parameters selection

Most ML algorithms have values that are set before the training begins. These settings are called hyper-parameters because their choice influences the eventual parameters (the coefficients or weights) that are updated from the learning outcomes [31]. For instance, In the case of RL the values of number of averaged experiment trials, or the environmental characteristics are considered as the hyper-parameters that control the learning process.

2.4.1.3 Performance metrics

In order to determine how well an ML algorithm will work when deployed in a real scenario, it is important to measure its performance on unseen/unlabeled data. Generally, the performance measure is specific to the task being carried out by the system. For tasks such as classification the accuracy of the model is used as a measure of performance. Accuracy is defined as the percentage of samples for which the algorithm produces a correct output [33].

2.4.1.4 Privacy and security

The ability of ML to swiftly overcome to changing situations has enabled it to become a fundamental tool for computer security, intrusion detection and cyber physical attacks on mobile and wireless communications [34]. Ironically, that adaptability is also a vulnerability that may produce unexpected results in the network.

2.4.2 Machine Learning for 5G Network

There is a strong relation between ML algorithms and 5G requirements. The intent of these requirements is to ensure that 5G network guarantees more flexibility, security and reliability. Providing a variety of services and deployment scenarios for a wide range of environments. The main 5G requirements grouped into three generic communication services, **table 2.2** present it with the way of how ML can assist in reaching their demands.

Service	KPI	ML application
eMBB	Peak data rate	SVM: to classify channel state information and select the optimal antenna indices in MIMO [47]. DNN: for channel estimation and direction of arrival (DOV) estimation in MIMO [49].
	User Experience Data Rate	Supervised Classifier: to dynamically allocate network resources according to connectivity performance [45]. Q-Learning: to optimize the handover-decision in HetNets based on QoE [54].
	Spectrum Efficiency	Actor-Critic: to access the spectrum opportunistically to reduce intra/inter-tier interference [26]. Hierarchical Clustering: to detect faults and intrusions in the wireless spectrum [50].
	Area Traffic Capacity	Hierarchical Clustering: to detect faults and intrusions in the wireless spectrum [50]. Affinity Propagation Clustering: to manage network resources in ultra-dense small cells [53].
	Mobility	DRL: to determine the set of possible connecting neighbour nodes and configure caching parameters in joint V2V networks [52]. DNN: to predict and coordinate beam forming vectors at the BSs [48].
uRLLC	Latency	k-means Clustering: to assist in partitioning the data centre contents in blocks before storage, to reduce the data travel among distributed storage systems [51]. Probabilistic Learning: to adjust the TDD uplink-downlink in hybrid fiber-wireless network based on ongoing traffic conditions [46].
	Connection Density	Logistic Regression: to allocate frequency and bandwidth dynamically in dense small cell deployment [44]. Multi-armed Bandit: to increase the capacity by optimizing the handover process in HetNets [37].
	Network Energy Efficiency	Linear Regression: to predict and model energy availability, in order to define scheduling policies for harvesting wireless nodes [36]. Q-Learning: to model self-adaptive sleep-scheduling algorithms for wireless nodes and BSs [35].

Table 2.2: Brief summary of corresponding ML-driven approaches to cope with the demand of the 5G standards[34].

2.5 Related Works

Inspired by the success of machine learning in solving complicated control and decision making problems, RL-based approaches can be an option for solving resource and service management problems in 5G networks, which are characterized by temporal variation and stochasticity of service and resource availability as well as of system parameters and states. The SFC deployment, which is an NP-hard problem, [42] has become a hot spot and various solutions based on RL/DRL approaches have been proposed over the past few years, which allow network entities to learn and build knowledge about the networks and even make optimal decisions locally and independently. Some of these solutions are:

2.5.1 DRL for NFV-based Service Function Chaining in Multi-Service Networks

Authors in [1] propose NFVDeep, which is a DRL-based approach for SFC deployment to minimize the operational cost of NFV providers and maximize the total throughput requests of clients for the total profits of the NFV system. They use DRL because of its ability to deal with large network state spaces and real-time network state transitions, while Policy-Gradient (PG) shows good advantages in improving the training efficiency and convergence to optimality. To evaluate the performance, they compare their algorithm with the non-recursive greedy SFC placement (NGSP), where it preferentially deploys VNFs of SFCs at used nodes with a high resource utilization rate and can get a feasible result with $O(m \times n)$ time complexity. They also compare it with GPLL and Bayes (a Bayesian learning based approach). Results show that NFVdeep achieves the highest total rewards with 32.59% higher total throughput of accepted requests and a 33.29% lower operation cost on average.

2.5.2 Online Service Function Chain Deployment with DRL in 5G networks

Authors in [38] propose an SFC deployment model for 5G networks in the context of live streaming to overcome traditional content delivery network problems by improving VNFs usage, content delivery delay, throughput and operational costs, which are composed of data transmission costs and hosting costs. They use a DRL-based approach to achieve this objective by enhancing the exploration mechanism over a Dueling Double Deep Q-Network (DDQN) agent, where Double Deep Q-Learning helps to de-correlate the noise introduced by the parameters that approximate the function used to choose the best actions from the noise of the parameters of the approximator used to evaluate the choices, Dueling architectures have the ability to generalize learning in the presence of many similar-valued actions. which leads to the convergence of sub-optimal SFC deployment policies. They compared

their algorithm with the NFVDeep framework presented in [23] and they created three variants of it.

2.5.3 Adaptive Online SFC Deployment with Deep Reinforcement Learning

Authors in [39] present a Deep Reinforcement Learning (DRL) approach for the optimization of SFC deployment problems in multi-service networks with coexistence of background flows (flows without NFV/SFC) and SFC flows. The ultimate purpose is to achieve overall load balancing (in terms of both traffic link load and computing load of NFV nodes) and low maximum flow path delay. The proposed solution, Advantage Actor-Critic (A2C), has combined the strengths of deep neural networks and reinforcement learning models for better training. They implement the theoretical mixed-linear programming (MILP) model in order to evaluate the performance of the proposed scheme.

2.5.4 Q-Learning based SFC deployment on Edge Computing Environment

Authors in [30] use Reinforcement Learning based Q-Learning algorithm to find an optimal SCF deployment path in edge computing with limited computing and storage resources. Configuring SFC in edge computing environment comes with several challenges as the server resources limited in edge computing, it is important to allocate resource intelligently to maximize the utilization and minimize the latency. RL can provide optimal solution by exploring the environment without preparing and training large datasets, they use RL based Q-Learning algorithm to find optimal path for SFC deployment and because Q-Learning data structures are easy to modify, optimize and extend, also because Q-Learning algorithm reduce the training time and computational complexity compare to other approaches based on graph theory and linear programming. The authors evaluate their model by changing the length of SFC and with high and low resources.

2.5.5 RL based QoS/QoE-aware Service Function Chaining in Software-Driven 5G Slices

Authors in [41] proposes the reinforcement learning based QoS/QoE-aware service function chaining framework in 5G Network. For improving QoE and maintaining QoS metrics (such as bandwidth, delay, throughput, etc.). Compared with supervised learning, in that no prior extensive training dataset is required, Training datasets from real-world networks are hard to acquire. On the one hand, great efforts are required both in computing and storage to store operational statistics. The model trained by supervised learning can hardly reflect the dynamics of a continuously changing network environment. On the contrary, through the reward mechanism, reinforcement learning can

better adapt to environmental changes. They compare their algorithm with violent search, which guarantees the best service function chain with the highest QoE and random search, which gives a functionally feasible chain with minimal response time.

2.5.6 Discussion

After analysing the works [23][38][39][41][30] presented in **table 2.3** where the authors try to solve SFC deployment problem focusing on improving various metrics in different type of networks and often studying their reason of choosing DRL or RL approach and the arguments of choosing the metrics introduced, we get into that decision:

In our proposed model, we will use Software-Defined Networking (SDN) and Network Function Virtualization (NFV) which allow for flexible and cost effective for 5G networks.

We will use Markov Decision Process (MDP) for modulating the SFC deployment problem. Also, we will choose a DRL approach because traditional RL algorithms are tabular and evaluate the performance of action under a state with a Q-table. Since the dimension of the Q-table is finite, traditional RL is limited. Furthermore, RL has a weakness in making decisions in large environments with a wide range of possibilities. We will choose the DQN approach, which is a combination of both neural networks and Q-learning approaches, Deep Neural Network (DNN) is used to replace Q-table, DNN can build the relations among high dimensional states, actions and Q-value to offer powerful learning capacity to solve complex problems. DQN (Deep Q Network) uses the same RL agent, but instead of updating the Q-table, which is hard to search in environments with large state space, it applies a deep neural network as an approximation of the value-function.

The authors in [23] and [38] propose their models to minimize the operational cost of the provider and maximize throughput for customers. Furthermore, [38] try to balance VNFs usage between large numbers of requests. In [30] and [41] authors try to optimize SFC deployment by taking care of different QoS/QoE requirements, authors in [39] evaluate their model based on maximum link usage and VNFs usage. Based on their metrics, we will try to implement our model to provide load balancing in terms of link and node.

Papers	Infrastructure Network	Emerging technologies	IA approach	DRL approach	Metric of Evaluation
[2]	5G network	SDN/NFV	DRL	PG	operational cost and Throughput.
[38]	(core)			DDQN	VNFs usage, Throughput and Operational cost.
[39]	Multi-service networks (core)			A3C	Link usage.VNVs usage.
[41]	5G Network (core)		RL	DQN	QoS/QoE.
[30]	Edge Computing			Q-Learning	Optimal SFC path ,resource usage and latency.

Table 2.3: comparison between related works.

2.6 Conclusion

In this chapter, we have presented machine learning techniques focusing on Deep Reinforcement Learning and its algorithms. After that, we presented machine learning applications in networking, especially in 5G networks. Then, a comparative study of some recent related works was presented, where the authors tried to solve the SFC deployment problem with different mechanism. Finally, by taking care of different requirements and based on their work, we combined the advantages of each one and gave a brief essential description of our proposed model.

In the next chapter, we will propose a new model for the SFC deployment approach by presenting the general and detailed design of this proposed approach.

Chapter 3

Design

3.1 Introduction

In previous chapters, we presented the theoretical part of the basic concept used in these project and some related work, which helped us to propose a new model for SFC deployment.

In this chapter, we will present the most important task in the project, which is the design. We start with the system modulating by explaining the abstract definition of the network, the SFC request, the mathematical formulation of the problem, and the MDP component description. After that, we present the general and detailed architecture of the DRL-based approach for SFC deployment and give a description of the DRL-based SFC deployment process. Then, we present the state of the art for the compared algorithm RL-based approach. Finally, we end by giving an illustration of the RL-based SFC deployment process.

3.2 System Modeling

3.2.1 Network Definition

The physical network presented as a graph $G = (N, E)$, where N represent the set of nodes and E the set of links. $u, v \in N$ are two nodes, $uv \in E$ stands for the physical link connecting node u with v . The resource capacity of node $u \in N$ represent as C_u^r , The bandwidth capacity of link $uv \in E$ is denoted as C_{uv}^{bw} .

3.2.2 Service Function Chain Request

An SFC request consist of source, destination and series of different VNFs, we use $\bar{G} = (\bar{N}, \bar{E})$ to represent service function graph of SFC request, \bar{u} and \bar{v} stands for two hosted VNF nodes in \bar{N} and the link between them denoted as $\bar{u}\bar{v} \in \bar{E}$. Service function graph is a directed graph and

the link directions satisfy the order of VNFs in SFC request. The arriving requests denoted by $Req = \{SFC_{req_1}, SFC_{req_2}, \dots, SFC_{req_i}\}$, we use $SFC_{req} = \{vnf_1, vnf_2, \dots, vnf_k\}$ to represent the needed VNFs for an SFC request $SFC_{req} \in Req$, each SFC_{req} has its resource demand for each vnf_k : $D_{vnf_k}^r$ and traffic bandwidth requirements $D_{vnf_k, vnf_{k+1}}^{bw}$ which is the same between all VNFs (ie; for an $SFC_{req_1} = \{vnf_1, vnf_2, vnf_3, vnf_4\}$ bandwidth requirement between vnf_1 and vnf_2 = bandwidth requirement between vnf_2 and vnf_3 = bandwidth requirement between vnf_3 and vnf_4).

3.2.3 Problem Formulation

In this part we describe SFC deployment problem for SDN/NFV in 5G context. For an SFC request the demand bandwidth of link $\bar{u}\bar{v}$ must not exceed the bandwidth capacity of link $\bar{u}\bar{v}$:

$$D_{\bar{u}\bar{v}}^{bw} \leq C_{\bar{u}\bar{v}}^{bw}, \forall \bar{u}\bar{v} \in \bar{E}.$$

Also, the demand resource of node \bar{u} must not exceed the resource capacity of this node:

$$D_{\bar{u}}^r \leq C_{\bar{u}}^r, \forall \bar{u} \in \bar{N}.$$

In this study, we propose and implement two optimizing objectives, maximizing the remaining resource capacity in each node:

$$Max(R_{\bar{u}}^r).$$

where:

$$R_{\bar{u}}^r = C_{\bar{u}}^r - D_{\bar{u}}^r, \forall \bar{u} \in \bar{N}.$$

and minimizing the usage of link bandwidth of each SFC request:

$$Min(C_{\bar{u}\bar{v}}^{bw}).$$

3.2.4 Markov Decision Process (MDP) for SFC deployment

Since the SFC deployment problem is a continuous decision problem, to deal with network transition we formally present the MDP model, which is defined as $\langle S, A, R, P, \gamma \rangle$ where S is the set of states, A set of actions, R for reward function, P transition probability distribution and γ the discount factor.

3.2.4.1 States representation

For each state $s \in S$ we define it as vector $(C_j^{bw}, D_{j'}^{bw}, C_i^r, D_{i'}^r)$. where C_j^{bw} represent the bandwidth of all links where $C_j^{bw} = \{C_1^{bw}, C_2^{bw}, \dots, C_j^{bw}\}$, $\forall j \in \bar{E}$, while C_i^r is the resource capacity of all nodes where $C_i^r = \{C_1^r, C_2^r, \dots, C_i^r\}$, $\forall i \in \bar{N}$. $D_{j'}^{bw}, D_{i'}^r, \forall j' \in \bar{E}, \forall i' \in \bar{N}$ represent the demand of bandwidth and resource respectively.

3.2.4.2 Actions representation

$A = \{nId_1, nId_2, \dots, nId_i\}$, $\forall i \in N$ is the set of actions that agent could take, nId represent the node identifier where VNFs will be deployed.

3.2.4.3 Reward function

The reward is designed to encourage the agent to optimize two objectives, maximizing the remaining resource of nodes and minimizing the usage of links bandwidth for each SFC request, generally, we return positive reward (3.1) for placing each VNF in appropriate node successfully, negative reward (3.2) for punish failing actions and returning (3.3) for minimizing the maximum value of node and link load balancing for optimizing the two objectives this formula taken from [55]. The mathematical representation of reward function considering three case are:

$$R(s, a) = \begin{cases} 1 & \text{node selected with success.} & (3.1) \\ -1 & \text{node selected with failure.} & (3.2) \\ 0.5 * \max(C_u^r - D_u^r) + 0.5 * \max(C_{uv}^{bw} - D_{uv}^{bw}) & \text{SFC is deployed.} & (3.3) \end{cases}$$

3.2.4.4 State transition

MDP state transition defined as $P = (s_t, a_t, r_t, s_{t+1}) \forall s_t, s_{t+1} \in S, \forall a_t \in A, \forall r_t \in R$ and t is the time step, where s_t is the current network state, a_t is the taken action to place requested VNF in appropriate node, r_t is the given reward by taken action a_t in state s_t , s_{t+1} is the next network state.

3.3 General Architecture

The general architecture of our proposed approach is presented in **figure 3.1** it contains three principal parts which are:

Network topology it represents the physical description of the 5G network component (nodes and links) it grouped into 3-layer RAN, Core and internet.

Centralized Controller it controls the network by establish appropriate links between nodes, Also, it enables the network to be intelligently controlled by using centralized applications.

IA based decision-making module this is the brain of the network, requested by the controller to define the deployment rules.

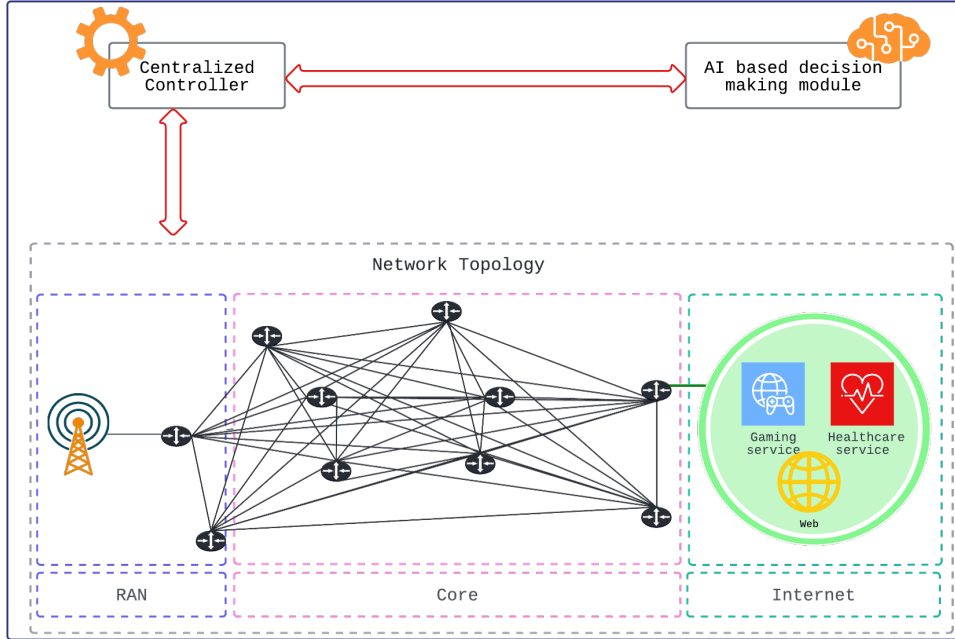


Figure 3.1: General architecture for the proposed model.

3.4 Detailed Architecture

Figure 3.2 is the detailed architecture of our proposed DRL approach for SFC deployment in 5G context which is a composition of two emerging technologies SDN and NFV each one grouped into many layers. To explain this architecture we will use this scenario: we have a source that need to access to specific application, the appropriate SFC is: $\{VNF_1, VNF_2, VNF_3, VNF_4\}$, the traffic will be steered from source to destination through the nodes that host these VNFs by forming VNF forwarding Graph(VNF-FG). In the beginning when the request arrive the controller will receive it and export the needed $VNFs$, after that the controller collect data about the state of the network and try to find the optimal policy which satisfy the best action could take for hosting the VNFs in appropriate nodes and chain them, all the work that the controller do is controlled by an software application in this case this application is the decision making system we have proposed.

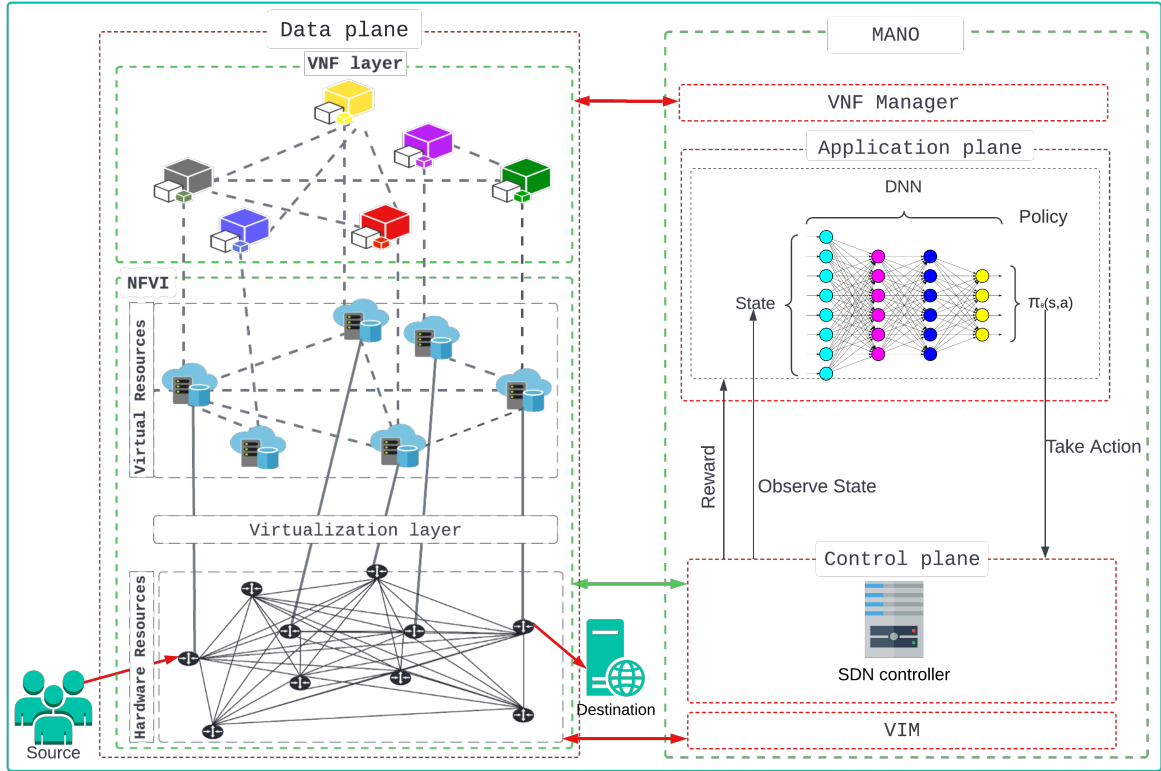


Figure 3.2: Detailed architecture.

3.4.1 Source and Destination nodes

The source will be placed in the Radio Access Network (RAN). It represents the SFC request provider, the request will reach its target, which represents the Internet by going through the core nodes (VNF nodes).

3.4.2 Network Function Virtualization

NF is a network function run on physical hardware like firewall, load balancer, Intrusion Detection System (IDS), etc. NFV is the virtual version of hardware devices of NF such as virtual firewall, virtual load balancer, virtual IDS, etc. As we presented in section (1.4.1) NFV architecture contain 3 layer which are: Network Function Virtualization Infrastructure (NFVI), NFV Management and Orchestration (MANO) and Virtual Network Function layer.

- **Network Function Virtualization Infrastructure (NFVI):** Is a combination of hardware resources (compute, storage, network) and virtualization software (hypervisor, container) to provide virtual resources (compute, storage, network), which build up the environment for where VNFs will be deployed, executed and managed. Virtualized Infrastructure Manager (VIM) is a part of NFV Management and Orchestration (MANO) which control and manage all the parts of NFVI. VIM responsible of allocation of virtual resources and the association

of physical to virtual resources. Also, responsible for VNF service chaining, by chain multiple VNFs together according to the order of the request we create Service Function Chain SFC.

- **Virtual Network Function layer:** VNF layer is where we find the VNFs software which executed on NFVI, Virtual Network Function Manager (NFVM) represent the key component of NFV MANO is responsible of life-cycle management of VNFs: instantiating, updating, termination, etc. Also, it determines how successfully VNFs in VNF layer are created and managed.
- **NFV Management and Orchestration (MANO):** it contain Virtual Network Function Manager (VNFM), Virtualized Infrastructure Manager (VIM) and the DRL proposed module.

3.4.3 Software Define Network

It applies radical change in the networks by separating the control plane from the data plane according to architecture presented in section (1.6.3) it contains 3 layers which are:

- **Application plane:** It's the brain at the back of the control plane. It represents the application that the controller calls to offer the best network control, it also contains the DRL model agent's decision-making system.
- **Control plane:** It is the most important part of SND. It represents the intelligence in the network. In our case, it contains the SDN controller.
- **Data plane:** It is the network topology that represents the environment for the DRL model. It contains the NFVI with its 3 sub-layers (hardware resources, virtualization layer, virtual resources) and the VNF layer.

3.4.4 Network topology

In our project, we have studied and proposed a Service Function chain deployment based DRL approach for **dfn-bwin** [56] SNDLib topology with 45 links and 10 nodes. It has high density of connections, medium scale and widely used in literature, as **figure 3.3** shows.

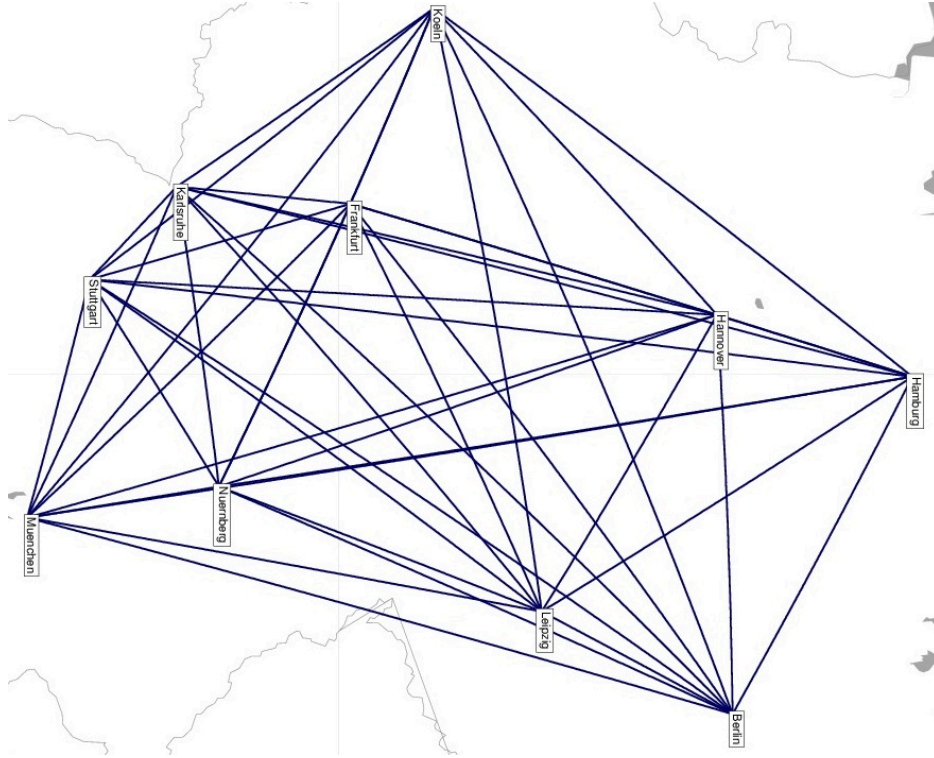


Figure 3.3: SNDlib dfn-bwin topology [56].

3.4.5 DRL model architecture

Reinforcement Learning is the origin of Deep Reinforcement learning, generally RL problems are solved by using Q-learning algorithms. As mentioned in chapter 2 (section 2.3), the agent interacts with its environment to maximize the return (the sum of future rewards). Q-learning maps action and state pairs to Q-values in a Q-table. However, in real world scenarios, there will be a huge number of states, which makes it computationally intractable to build a Q-table. We use an action-value function to overcome this limitation. Using an action-value function rather than a Q-table gives the same results as mapping action and state into Q-values as **figure 3.4** shows. Neural networks prove that they are excellent at modelling complex functions by combining them with Q-learning under the name of Deep Q-Learning, also called Deep Q-Network, to approximate action-value function. This function maps a state to the Q-values of all the actions that can be taken from that state.

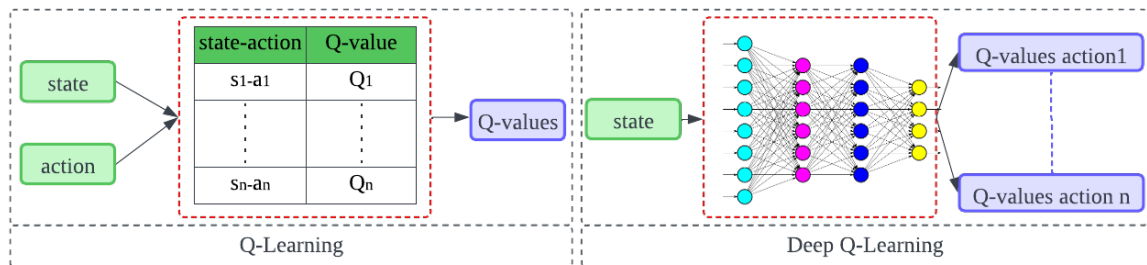


Figure 3.4: DQN vs Q-learning.

3.4.5.1 DQN Architecture Design

DRL approaches are classified into two types: off-policy, also called value-based approaches like DQN, and on-policy, known as policy-based approaches like the Policy Gradient Approach. In the SFC deployment problem, we have a large state-action space, for which the value-based approach Deep Q-Learning is adopted. While MDP will allow us to automatically characterize the network variations, we need an effective policy that can automatically take appropriate actions by placing VNFs in nodes and building the chain of an SFC in each state to achieve positive reward. The goal of planning an MDP is to maximize the expected future discounted reward when an agent chooses an action according to policy π in the environment. The action-value function is used to evaluate the policy. Its calculation is realized by an action-value function approximated using Q-network $Q(s, a; \theta)$ which is a Q-function parameterized by theta, the weight of the Deep Neural Network.

3.4.5.2 DQN component

The DQN algorithm contains many components that help it to achieve better results and higher performance, which are:

- a. **Experience replay** Experience replay is the memory that stores $\langle s, s_{t+1}, a, r \rangle$ tuples where s, s_{t+1} the current and the next state respectively, a for action and r reward, this tuples represent the training data which gather it by interacting with the environment. When training the neural network we take random batches from this memory to calculate the Q-values. **Algorithm 2** represent the pseudo code of how experience replay works, we start with creating the memory D , in *line* (3 – 5) for each time step we interact with the environment and collect MDPs transition then store it in D , after that we chose random samples for train the network.

Algorithm 2: Experience replay pseudo code

```
1 Create: memory  $D$ 
2 for  $t \leftarrow 1$  to  $T$  do
3   | Interact with the environment by chose random actions  $a_t$  for  $T$  time.
4   | Collect:  $tuple[t] = \langle s_t, s_{t+1}, a_t, r_t \rangle$ 
5   | Add:  $tuple[t]$  to  $D$ 
6 end
7 for  $m \leftarrow 1$  to  $size(batch\_memory)$  do
8   | Sample:  $tuple[m]$  from  $D$  to  $M$ 
9   | Train the neural network
10 end
```

- b. **ϵ -greedy method** With ϵ -greedy, the agent selects a random action with a fixed probability ϵ . **Algorithm 3** shows the steps of ϵ -greedy method. we initialize ϵ and random generated number p , if ϵ had a higher value than p a random action selected from action-space. Else, the action will be chose greedily according to the learned action-value $Q(s, a; \theta)$. By applying this method, we solve the dilemma of the exploration/exploitation trade-off. As we mentioned in section (2.3.2),

Exploration: maximize the probability of choosing this action if it gives a high reward.

Exploitation: gather more information and explore all possible new paths.

Algorithm 3: ϵ -greedy pseudo code

```

1 Initialize:  $\epsilon \in [0, 1], p \in [0, 1]$  randomly
2 if  $p < \epsilon$  then
3   | chose random action  $a$ 
4 else
5   | action  $a = \operatorname{argmax}_a Q(s_t, a_{t+1}; \theta)$ 
6 end

```

- c. **Target-network and Q-network** In DQN, we estimate target and Q-value-function by two different neural networks, target and Q-network. The target network updates their weights with the parameters of the Q-network. By giving current state s the Q-network calculate $Q(s, a; \theta)$ and the target network calculate $\bar{Q}(s_{t+1}, a; \bar{\theta})$.

3.4.5.3 DQN workflow

Figure 3.5 to represent the workflow of the Deep Q-learning model, we briefly explain it by splitting it into 5 steps as follows:

- **step 1:** Start by executing a few actions and adding tuples to the replay memory. The Q-network interacts with the environment and generates training samples. Experience replay selects an ϵ -greedy action from the current state, executes it and receive the reward and next state, this observation will be saved in replay memory which will be used as training data.
- **step 2:** We now start the training of the DQN model by selecting a random batch of samples from the training data and giving them to the two networks as inputs.
- **step 3:** The Q network takes the current state s_t from each tuple in sample batch and predicts the Q value $Q(s_t, a_t; \theta)$ for all actions a_t that could be taken, this is the predicted Q-Value. The Target network takes the next state s_{t+1} from the same tuple as input and predicts Q-value

$\bar{Q}(s_{t+1}, a_t; \bar{\theta})$ for all action a_t that could be taken, then select the best $\bar{Q}(s_{t+1}, a_t; \bar{\theta})$ and observe the reward r_t for this data sample.

- **step 4:** We will compute the loss by calculating the difference $(\bar{Q}(s_{t+1}, a_t; \bar{\theta}) - Q(s_t, a_t; \theta))^2$.
- **step 5:** Then we will back-propagate the loss and update the Q-network weights using gradient descent. And this end the DQN training process for one time step, the target-network is not trained, so no loss calculation and no back-propagation.

This process will be repeated for T time step, the Q-network learn and update the weights and target-network remain fixed. After T time step gone, we will copy the weights of Q-network to the target-network and this will help it to predict more accurate Q-values.

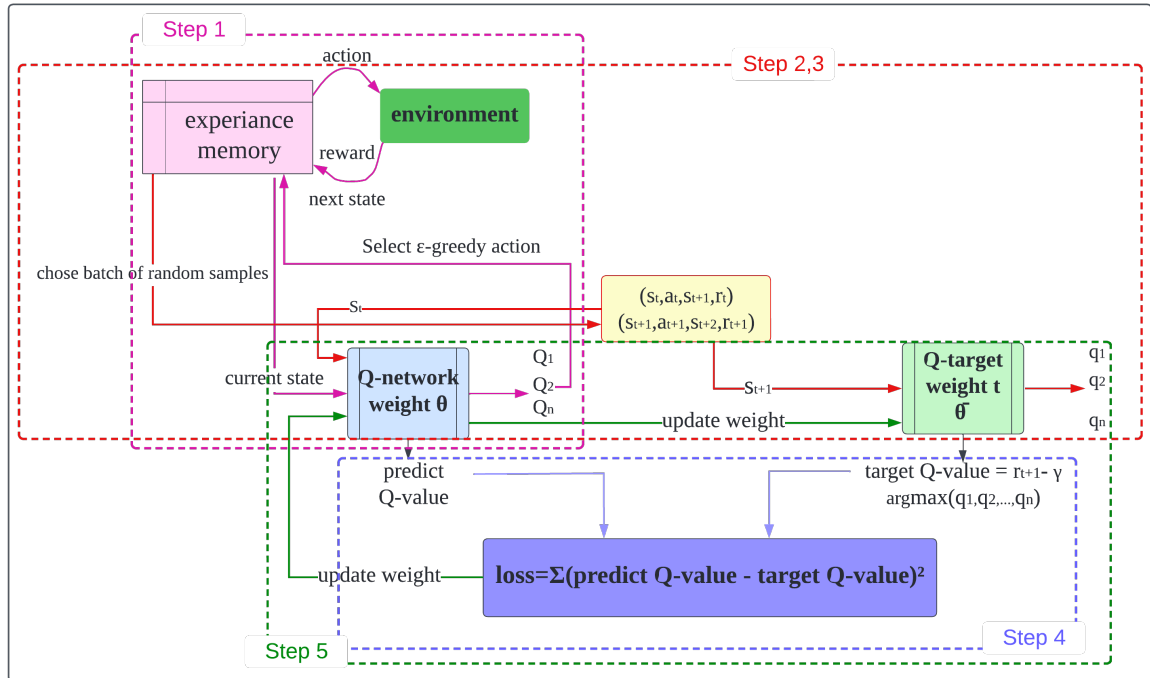


Figure 3.5: DQN workflow.

3.4.5.4 Q-network and target-network design

We can build DQN without target network but Q-network will do two passes one for predict Q-value for state s and the second for predict target-value for state s_{t+1} , this made a problem where the direction of prediction of target-value will be changed. By using a second neural network target network which does not get trained, we ensure that the target values remain stable for a short period. The Target-network is still a prediction network, so it needs to update its parameters to improve its outputs. After a period of time steps, its weights $\bar{\theta}$ will be updated by copying them from the Q-network weights θ . **figure 3.6** represent both DQN Deep neural networks.

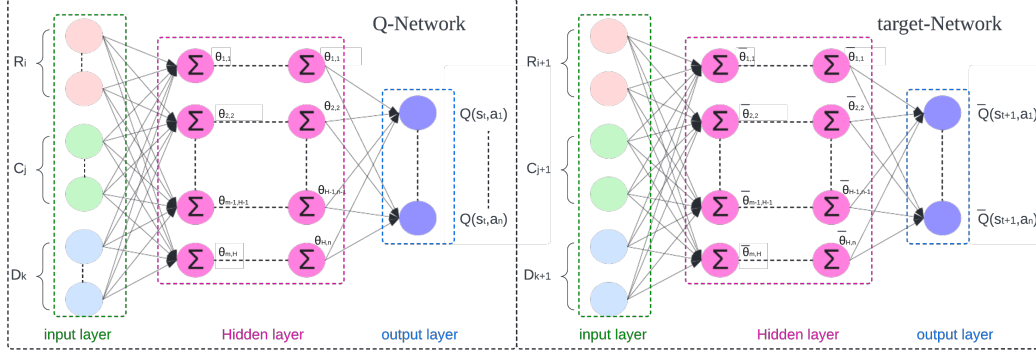


Figure 3.6: Q-network and target-network.

Each Deep Neural Network contains 3 main layers, As we mention before the target-network is the same as the Q-network so its neural network architecture will be the same, the three layers are:

- **Input layer:** The number of neurons in the input layer is fixed and corresponds to the state of the environment, which represent by 3 features R_i the remain resource in nodes, C_i the bandwidth capacity and D_i the VNF demands.
- **Hidden layer:** In general there will be one or more hidden layers in neural network architecture. the number of the layers and neurons are hyper-parameters of the architecture. $\theta_{X,Y}$ represent the weight of neuron where X is the number of origin neuron and Y is the number of destination neuron in each layer, the Σ symbol represent the activation function in our model, we will use ReLu.
- **Output layer:** The output layer represents the prediction of Q-value for each action that can be taken from the given state that was passed as input, for Q-network it represents $Q(s_t, a_t; \theta)$ where we pass state s_t , $\bar{Q}(s_{t+1}, a_t; \bar{\theta})$ for target-network after passing state s_{t+1}

3.4.5.5 SFC deployment approach

The DQN agent deal with SFC request in chronological order, where an MDP transition will happen if the request is deployed or rejected, **figure 3.7** present SFC deployment approach in time steps.

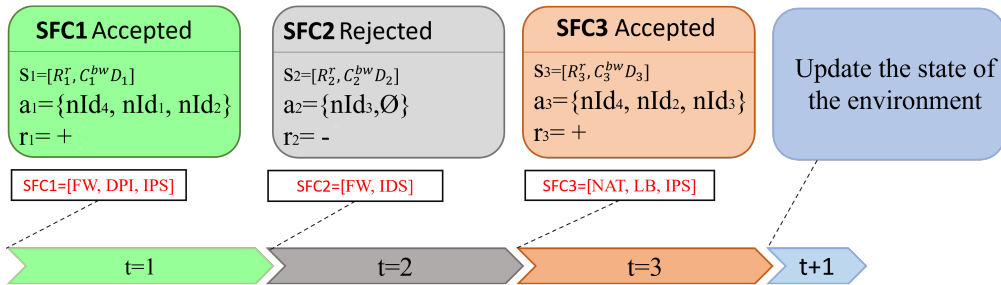


Figure 3.7: SFC deployment approach in time steps.

As illustrated in **figure 3.7** three SFC requests arrives in different time steps, $SFC_1 = \{FW, DPI, IPS\}$, $SFC_2 = \{FW, IDS\}$, $SFC_3 = \{NAT, LB, IPS\}$ where: firewall(FW) Deep packet inspection(DPI), Intrusion prevention system(IPS), Intrusion detection system(IDS), Load balancer(LB) and Network address resolution(NAT) represent the needed virtual network function in each request. In the beginning the agent will monitor all the network and collect the statistics about nodes resources and links bandwidth, Then receive the SFC requests based on arriving time, at time $t = 1$ the state is: $s_1 = [C_{uv \in \bar{E}}^{bw}, D_1^{bw}, C_{\bar{u} \in \bar{N}}^r, D_1^r]$ where $C_{uv \in \bar{E}}^{bw}$ represent all the network links bandwidth, $C_{\bar{u} \in \bar{N}}^r$ is the resource capacity of all nodes, D_1^{bw}, D_1^r is the request demand of bandwidth and resource. DQN agent will calculate the Q-value for each possible action $Q_1 = (s_1, a_i; \theta)$ where $a_i \in A$, A represent the possible actions that represent the appropriate node Ids nId_i for deploying each VNF in the SFC request. We assume that for SFC_1 the chosen action is $a_1 = \{nId_4, nId_1, nId_2\}$ which mean that: FW will be deployed in node that identified by nId_4 , DPI will be deployed in node that identified by nId_1 , IPS will be deployed in node that identified by nId_2 . SFC_1 deployed successfully at state s_1 by taken action a_1 which give the biggest value of $Q_1(s_1, a_1; \theta)$ and gets positive reward r_1 . After that, the model move to the next state s_2 , for SFC_2 the chosen action is $a_2 = \{nId_3, \phi\}$ because the second requested VNF IDS is not deployed which indicated by ϕ , a negative reward r_2 will be returned and the request will be rejected. At time t_3 , state s_3 , when the request demands satisfied for SFC_3 it will be deployed successfully with positive reward r_3 . Finally the model will pass into the next state.

3.4.5.6 DRL based approach for SFC deployment algorithm

Algorithm 4 shows how the model learns the optimal policy π^* for SFC deployment, it start with instantiating of environment (network topology) and DQN agent (brain) which require the selection of the hyper-parameter, the initialization of Q-network with θ and target-network with $\bar{\theta}$, the mini-batch M and the experience replay memory D . Starts by calling *reset* function presented in **algorithm 5** for initialize the environment nodes resources and links bandwidth. Then get SFC requests by calling *SFC_requests* function presented in **algorithm 6** by give it the number of SFCs requests and the size of each request. After that, for each *step* the agent get the network current *state* by calling *start* function presented in **algorithm 7** which parameterised by the demands of the SFC request *bandwidth* and *resource*. For each *vnf* in SFC request the agent will choose an action based on **algorithm 3** presented in section (3.4.5.2). At *line(12)* DQN agent call *step* function presented in **algorithm 8** by giving it the chosen *action*, it will indicate the deployment decision for the received SFC, accepted or rejected and returns *next_state*, *reward*, *terminal_state*, *terminal_state* flag will help the agent for determine if this state is the terminal state. The previous MDP transition will be stored in replay memory D , after that, we will update the *state* of the environment and test the

terminal state. A mini-batch M will be randomly sampled from D which eliminate the correlation to avoid over-fitting for train the Q-network and update the weights based on **algorithm 2**. The target-network will be updated periodically after C time steps as *line 21* shows.

Algorithm 4: DQN SFC deployment

```

1 Instantiate environment
2 Instantiate DQN agent
3 Initialize hyper-parameter
4 Initialize Q-network  $Q$  with weight  $\theta$ , Q-target  $\bar{Q}$  with weight  $\bar{\theta}$ 
5 Initialize batch_size  $M$  and replay memory  $D$ 
6 reset(); // initialize the environment
7  $D_k^{bw}, D_k^r \leftarrow SFC\_requests(number\_requests, sfc\_size)$ ; // get SFC requests
8 for  $step \leftarrow 1$  to  $STEPS$  do
9   state  $\leftarrow$  start( $D_k^{bw}, D_k^r$ ); // get the current state
10  for  $vnf \leftarrow 1$  to  $SFCvnfs$  do
11    action  $\leftarrow$  chose_action(state); // chose action based on  $\epsilon$  greedy method
12    next_state, reward, terminal_state  $\leftarrow$  step(action); // get the deployment
        decision
13    store_transition(state, action, reward, next_state); // store transition in D
14    state  $\leftarrow$  next state; // update the current state
15    if terminal_state then
16      break; // testing terminal state
17    end
18  end
19  sample randomly mini_batch  $M$  of transition from  $D$ 
20  Train the Q-network and update the weights.
21  Every  $C$  steps copy weight from  $Q$  to  $\bar{Q}$ 
22 end

```

- a. **Reset function** This function is responsible for the initialization of the environment by setting the network link bandwidth and node resources.

Algorithm 5: Reset function

```

1 Function reset():
2   Initialize_network_links_bandwidth()      // network links bandwidth  $BW_i$ 
3   Initialize_network_nodes_resources()      // network nodes resources  $COST_j$ 
4 End Function

```

- b. **SFC request function** Start by initializing the number of requests and each SFC request size, then creating and initializing four variables that set the min and max values for bandwidth and resource requirements. *SFC_request* function parameterized by the number of requests *number_requests* and SFC request size *sfc_size*, we set the bandwidth requirement for the sfc request as random number between B_{min} and B_{max} , after that, For each *vnf* in *sfc_size* we set the required resources presented as random number between $cost_{min}$ and $cost_{max}$. Finally, we will return the SFC request, which a vector of two values required link bandwidth and node resources.

Algorithm 6: SFC request function

```

1 initialize SFC requests number_requests
2 initialize SFC requests sfc_size
3 initialize  $B_{min}$  and  $B_{max}$  bandwidth requirement interval
4 initialize  $cost_{min}$  and  $cost_{max}$  resource requirement interval
5 Function SFC_requests(number_requests,sfc_size):
6    $D_k^{bw} \leftarrow random\_number(B_{min}, B_{max})$ 
7   for vnf  $\leftarrow 1$  to sfc_size do
8      $D_{vnf}^r \leftarrow random\_number(cost_{min}, cost_{max})$ 
9   end
10  return ( $D_k^{bw}, D_{sfc\_size}^r$ ) // return the bandwidth and resoures requirments
11 End Function

```

- c. **Start function** Each time step the agent calls this function, it returns the current state of the environment, which is a combination of: BW_i network links bandwidth, D_k^{bw} required bandwidth, $COST_j$ network node resources, D_k^r required resources.

Algorithm 7: Start function

```

1 Function start( $D_k^{bw}, D_k^r$ ):
2    $state \leftarrow BW_i, D_k^{bw}, COST_j, D_k^r$ 
3   return  $state$ 
4 End Function

```

- d. **Step function** This function is responsible for SFC deployment. It starts by initializing *terminal_state* flag by *False*, then, testing if the bandwidth and resources in the network can allocate the required bandwidth and resources. After that, it sets the *reward* of the transition and the *terminal_state* flag by *True* for the terminal state and updates the current state. Otherwise, the SFC request will be rejected for one of two reasons: the constraint of the required bandwidth or resources is not satisfied and set *terminal_state* flag *True* for the terminal state. Finally, return *state, reward, terminal_state*.

Algorithm 8: Step function

```

1 Function Step( $action$ ):
2    $terminal\_state \leftarrow False$ 
3   if  $D_{uv}^{bw} \leq C_{uv}^{bw}$  and  $D_u^r \leq C_u^r$  then
4     allocate  $D_{uv}^{bw}$  to  $C_{uv}^{bw}$  and  $D_u^r$  to  $C_u^r$ 
5      $reward \leftarrow r(state, action)$ 
6      $terminal\_state \leftarrow True$ 
7     update state
8   else
9     reject SFC
10     $terminal\_state \leftarrow True$ 
11  end
12  return  $state, reward, terminal\_state$ 
13 End Function

```

3.4.5.7 DRL based approach for SFC deployment process

Figure 3.8 represents a DRL-based SFC deployment process. It starts with the environment instantiating and initialization with the configuration parameters. The DQN agent then instantiates and initializes its hyperparameters. If a request is received, a resource test is launched to determine if the demand constraint (bandwidth and node resources) is satisfied, so the request will be deployed otherwise rejected. After that, the agent will allocate resources and bandwidth to SFC request based on their requirements. Finally, we test if this is the last received request for stop the process.

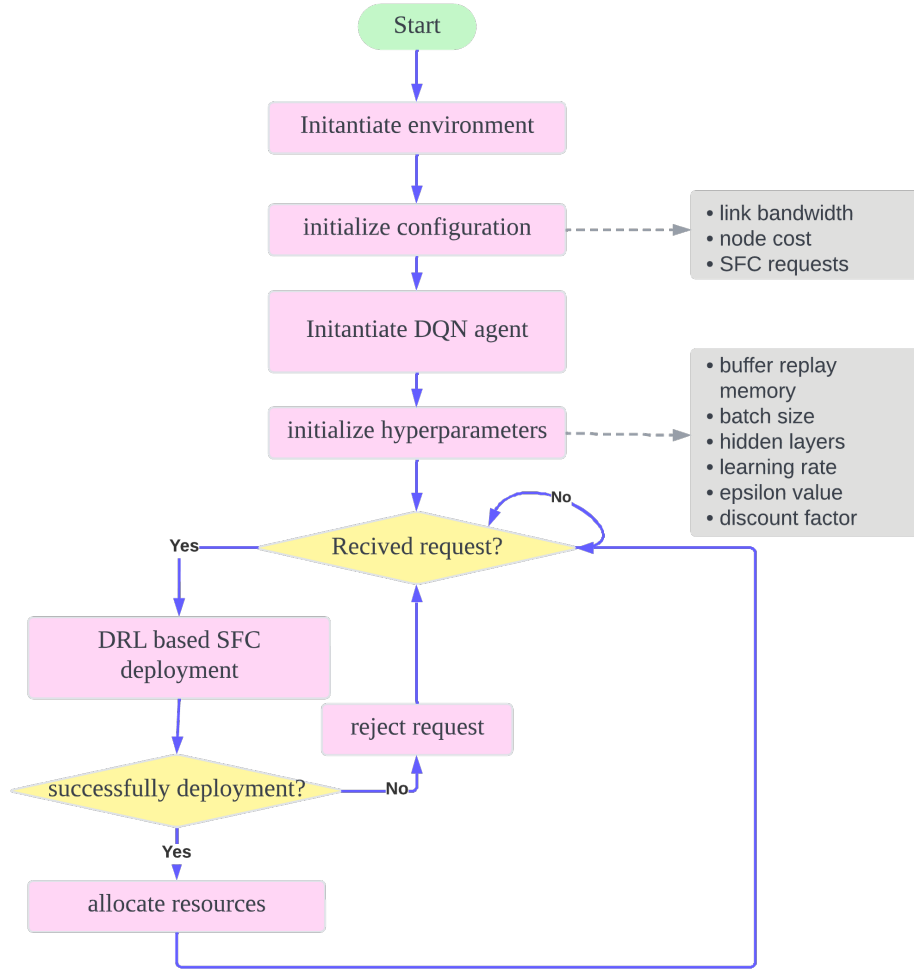


Figure 3.8: DRL based SFC deployment process.

3.5 Reinforcement Learning model architecture

We will compare our Deep Reinforcement Learning proposed approach Deep Q-learning with the Reinforcement learning approach Q-learning, which uses the same environment configuration as presented below:

3.5.1 RL based approach for SFC deployment algorithm

Q-learning is an off-policy algorithm that seeks to find the best action to take given the current state to maximize the future reward.

Algorithm 9 present the SFC deployment approach based Q-learning, start with the initialization of the environment and RL agent when q-learning is performed the agent initialize hyperparameters and create a q-table of *state, action* pairs with the q-values $Q(s, a)$, which will be updated after each step. This q-table used as a reference table by our agent to select the best action based on the q-value. It starts by, calling *reset* function presented in **algorithm 5**, then call SFC request generator

function *SFCrequests* explained in **algorithm 6**. For each time step the agent get the network state by calling *start* function in **algorithm 7** with giving it the appropriate parameters, After that, for each vnf in the SFC request the agent choose an action by following one of the two ways as **algorithm 3** presented in section (3.4.5.2) shows:

- **Exploration** By using the information we have available in the q-table and viewing all possible actions for a given state, the agent selects an action based on the maximum q-value of those actions.
- **Exploitation** The agent selects an action by acting randomly, which allows it to explore new states.

At *line(10)* the agent call *step* function presented in **algorithm 8** by giving it the chosen *action*, it will returns *next_state, reward, terminal_state*. The agent by interacting with the environment will update q-table by calling *agent_learn* function presented in **algorithm 10**, then the current state will be update and test if it is the terminal state for deploying the request. The agent will not learn after a single step, but eventually with enough exploring (steps) it will learn the optimal q-values $Q^*(s, a)$.

Algorithm 9: Q-learning SFC deployment

```
1 instantiate environment
2 instantiate RL agent
3 initialize q-table
4 reset(); // initialize the environment
5  $D_k^{bw}, D_k^r \leftarrow SFC\_requests(number\_requests, sfc\_size)$ ; // get sfc requests
6 for  $step \leftarrow 1$  to STEPS do
7     state  $\leftarrow start(D_k^{bw}, D_k^r)$ ; // get current state
8     for  $vnf \leftarrow 1$  to SFCvnfs do
9         action  $\leftarrow chose\_action(state)$ ; // chose action based on  $\epsilon$ greedy method
10        next_state, reward, terminal_state  $\leftarrow step(action)$ ; // deployment decision
11        agent_learn(state, action, reward, next_state); // agent learn & update q-table
12        state  $\leftarrow next\ state$ ; // update current state
13        if terminal_state then
14            break; // testing terminal state
15        end
16    end
17 end
```

- a. **Agent_learn function Algorithm 10** represent the function that enforce the agent to learn the optimal policy, we start with initializing the discount factor γ and the learning rate lr , the function parameterized by the previous MDP transition $state, action, reward, next_state$, at *line(3)* the agent predict the q_value $Q(state, action)$ if the current $state$ exist in the q_table , After that, the agent estimate the q_target $\bar{Q}(state, action)$ by add the reward to the multiplication of the maximum q_values and the discount factor γ , otherwise the agent update the q_table based on the difference between the discounted new q_values and the q_target.

Algorithm 10: agent_learn function

```

1 initialize gamma  $\gamma$  and learning rate  $lr$ 
2 Function agent_learn( $state, action, reward, next\_state$ ):
3    $Q(state, action) = q\_table[state, action]$  // predict Q-value from q-table
4   if not terminal state then
5      $\bar{Q}(state, action) \leftarrow reward + \gamma * \max(Q(state, action))$  // calculate
6     q-target
7   else
8      $\bar{Q}(state, action) \leftarrow reward$ 
9      $q\_table \leftarrow lr * (Q(state, action) - \bar{Q}(state, action))$  // update q-table
10  end
10 End Function

```

3.5.2 RL based approach for SFC deployment process

Figure 3.9 represents RL-based approach for the SFC deployment process. It is the same as the DRL-based approach, but here we use a Q-learning agent initialized by learning rate, epsilon value, and discount factor.

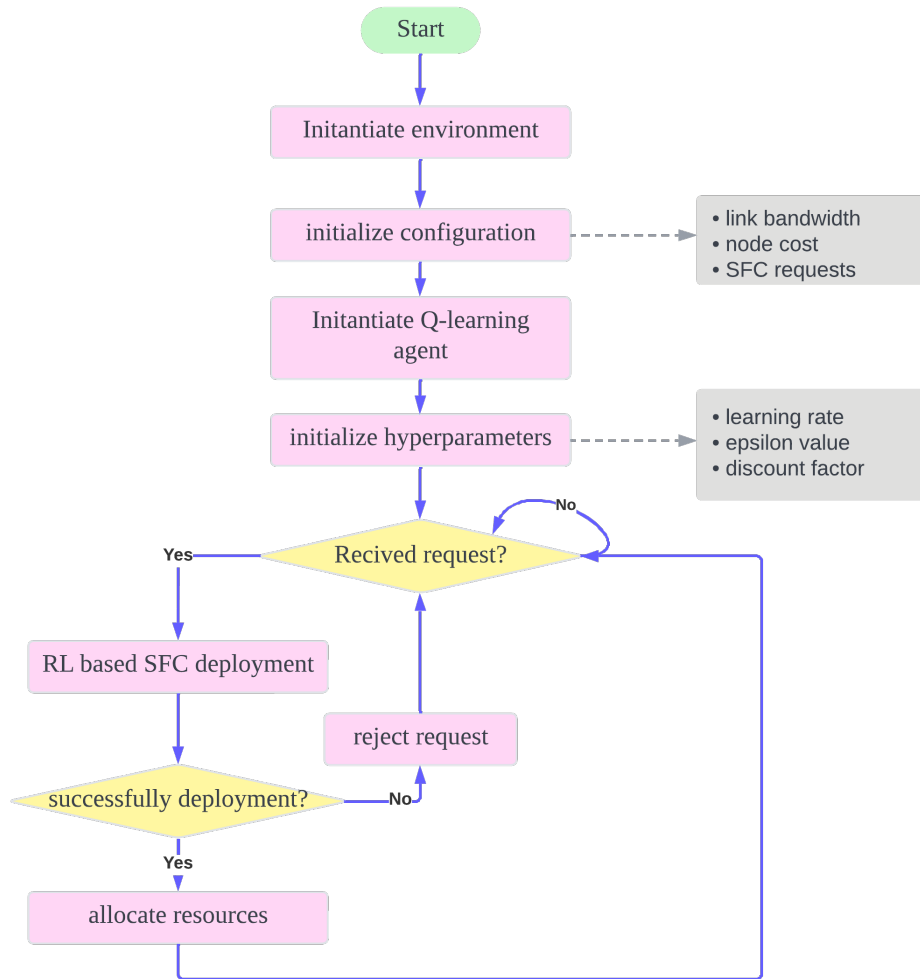


Figure 3.9: RL based SFC deployment process.

3.6 Conclusion

Through this chapter we present the global design of our system, starting by presenting the principle algorithms for SFC deployment based DRL and the algorithms for SFC deployment based RL, then we formulate the problem and present the general and the detailed architecture. In the next chapter we will see the experimental study after implement the presented algorithms and discuss the different results of this models.

Chapter 4

Experimental study and results

4.1 Introduction

In the previous chapter, we presented our DQN model and propose Q-learning approach for comparison. Furthermore, we have detailed the different design steps of our system to achieve these results. In order to implement our model, we used several development tools. This chapter is devoted to the presentation of the working environment, the programming language, and the tools we have used to build this system. Thereafter, we will explain the results of the proposed method as well as discuss it.

4.2 Development tools

We used the Python programming language for developing our system, with some libraries working on Deep Reinforcement Learning and Reinforcement Learning. We also use Jupyter Notebook to develop our model locally, and we use the internal cloud service Google Collaboratory. The different used tools are presented in **figure4.1**.

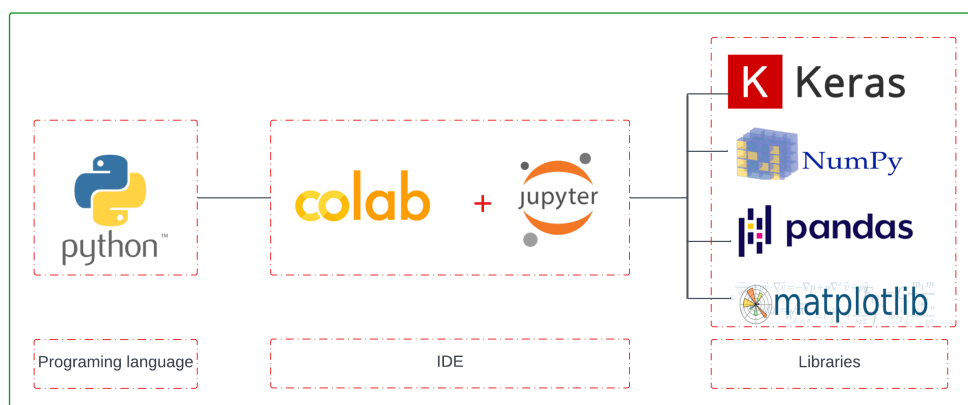


Figure 4.1: Used tools.

4.2.1 IDE (integrated development environment)

4.2.1.1 Google Colab

Is a free tool offered by Google that allows users to write and execute Python code in their web browsers. Colab is actually based on the Jupyter open source, and essentially allows us to create and share computation files without having to download or install anything. However, we might need to install some libraries in our environment during initialization [57].

4.2.1.2 Jupyter Notebook

Free software, open standards and web services for interactive computing across all programming languages, also known as IPython Notebook, which is an interactive computational environment, in which we can combine code execution, rich text, mathematics, plots, and rich media. It is an incredibly powerful tool for interactively developing and presenting data science projects [58].

4.2.2 Programming language

4.2.2.1 Python

Python is a widely used general-purpose, object-oriented, high-level programming language. It is one of the most popular coding languages due to its versatility. The language has a design that emphasizes code readability and supports multiple programming paradigms. Python includes high-level data structures, dynamic binding, dynamic typing, and some other features, making it suitable for complex application development. It is used in Artificial Intelligence, Machine Learning, Gaming, Product Development, Rapid Application Development, Testing, Automation, and more. Python is growing in popularity as the primary language for many applications [59].

4.2.3 libraries

The libraries are called at the beginning of the program to use their predefined functions.

4.2.3.1 Keras

Keras is a powerful and easy-to-use open source Python library for developing and evaluating Deep Learning models. It includes the efficient numerical computation libraries Theano and TensorFlow, allowing us to define and train neural network models in just a few lines of code [60].

4.2.3.2 Numpy

NumPy stands for Numerical Python, it is a Python open-source project, used for working with arrays. It has functions for working in the domains of linear algebra and Fourier transform [61].

4.2.3.3 Pandas

Pandas is an open source Python package that is most widely used for data science, data analysis, and machine learning tasks. It allows manipulation and analysis of data and proposes, in particular, data structures and numerical table manipulation operations [62].

4.2.3.4 Random

Random is an in-built module of Python that is used to generate random numbers. This module can be used to perform random actions such as generating random numbers, printing random values for a list or string, etc [63].

4.2.3.5 Matplotlib

Matplotlib is a comprehensive library for creating static and interactive visualizations in Python. It produces publication-quality figures in a variety of paper formats and interactive environments on all platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, web application servers, and various toolkits with graphical user interfaces [40].

4.2.3.6 Collections

The collection module in Python provides different types of containers. A container is an object that is used to store different objects and provides a way to access the contained objects and iterate over them. Some of the built-in containers are: tuple, list, dictionary, etc [32].

4.3 Implementation

The following steps present the implementations of our background Deep Reinforcement Learning model DQN and Reinforcement Learning model Q-learning, which were mentioned in the previous chapter:

4.3.1 Environment

The scenario proposed consists of the deployment of SFC requests comprised of ten nodes with 45 links and 200 different SFC requests for training, as **table 4.1** summarizes the chosen configuration:

Parameter	Value
Nodes	10
Links	45
SFC requests	200


Table 4.1: Environment configuration

4.3.1.1 Links

We use the link capacity offered by SNDlib Benchmark, **figure 4.2** presents the link configuration, where table 'SNDlib bandwidth data' represents the configuration taken from the SNDlib **dfn-bwin** topology and 'environment bandwidth data' represents the configuration used in our implementation, where we change the string values of *source* and *target* to numerical values to deal with them easily.

source	target	link capacity
Leipzig	Berlin	800
Leipzig	Koeln	800
Leipzig	Muenchen	800
Hamburg	Karlsruhe	800
Hamburg	Berlin	800
Hamburg	Nuernberg	800
Hamburg	Stuttgart	800
Hamburg	Muenchen	800
Hannover	Frankfurt	800
Hannover	Karlsruhe	800

SDNlib bandwidth data



source	target	link capacity
0	1	800
0	2	800
0	4	800
3	5	800
3	1	800
3	6	800
3	8	800
3	4	800
7	9	800
7	5	800

environment bandwidth data

Figure 4.2: SNDlib and environment link data.

4.3.1.2 Nodes

Because SNDlib does not offer resource capacity for nodes, we assume that each node has an capacity of 200 units.

4.3.1.3 SFC requests

We fix the SFC size by 5, where each SFC request contains 5 vnfs. The **figure 4.3** represents a sample of the used SFC requests in training phase, where *bw* represents the bandwidth requirement which measured by units. From *vnf0* to *vnf4*, represents the required resources for each vnf.

bw	vnf0	vnf1	vnf2	vnf3	vnf4
64	4	4	11	17	7
85	4	18	11	6	15
89	18	4	4	17	11
79	13	18	18	17	9
97	11	9	5	3	14
56	7	19	5	10	15
86	19	14	7	17	3
69	7	4	7	16	18
63	11	11	8	16	10
66	10	19	14	8	6
63	7	10	19	9	9
78	16	6	4	14	19
71	14	11	17	10	9
53	4	19	5	15	16

Figure 4.3: SFC requests sample.

4.3.2 Deep Q-learning Agent

This study is to understand how the DQN agent works and interact with its environment. We mainly focus on 3 different metrics, which are: the evolution of the total reward, resource consumption, and bandwidth consumption. The set of hyperparameters is presented in **table 4.2**.

Hyperparameter	Value
Replay buffer size	2000
mini batch size	64
Learning rate	0.001
Discount factor	0.95
Epsilon	1
Epsilon decay	0.996
Hidden layers (layer 1,layer 2)	(100,100)

Table 4.2: Hyperparameter configuration

4.3.2.1 DQN agent class

The DQN agent class contains many methods, which we will present separately as below:

- **Import libraries** first of all, we start by importing the different libraries that we need to implement our DQN agent, which are:

```

1 # Import the used libraries in DQN agent
2 import numpy as np
3 # Deque module provide fast and memory-efficient ways to append
4 from collections import deque
5 # Create sequence of layers
6 from keras.models import Sequential
7 # Type of layers in neural network Dense:fully connected NN
8 from keras.layers import Dense
9 # Optimal set of weight in NN
10 from keras.optimizers import Adam

```

Listing 4.1: Used libraries

We use the Numpy library for dealing with arrays. Also, we use Deque from collections, which provides a fast and memory-efficient way to append and pop operations from both ends of memory. Then we import from keras a Sequential model for the layer plan stack, Dense for the deeply connected neural network layer, and optimizer Adam, which is a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments.

- **Initialization** here, we initialize the hyperparameters as presented in **table 4.1**:

```

1 # Configuration parameters for the whole setup
2 def __init__(self, state_size, action_size):
3     self.state_size = state_size     # State size

```

```

4     self.action_size = action_size    # Action size
5     self.memory = deque(maxlen=2000) # Replay buffer size
6     self.gamma = 0.95                # Discount factor
7     self.epsilon = 1.0               # Epsilon greedy parameter
8     self.epsilon_min = 0.01          # Minimum epsilon greedy parameter
9     self.epsilon_decay = 0.996       # Epsilon greedy decay parameter
10    self.learning_rate = 0.001        # Learning rate
11    self.batch_size = 64              # Batch memory size

```

Listing 4.2: DQN agent initialization

- **Build model** the implementation of our model, is implemented by the `_build_model()` function:

```

1  def _build_model(self):
2      # Neural Net for Deep-Q learning Model
3      model = Sequential()
4      # Input layer
5      model.add(Dense(75, input_dim=self.state_size, activation='relu'))
6      # First hidden layer with 100 neuron
7      model.add(Dense(100, activation='relu'))
8      # Second hidden layer with 100 neuron
9      model.add(Dense(100, activation='relu'))
10     # Output layer
11     model.add(Dense(self.action_size, activation='linear'))
12     # defines the loss function and the optimizer
13     model.compile(loss='mse', optimizer=Adam(learning_rate=self.learning_rate))
14     return model

```

Listing 4.3: Build model

We build a sequential model with two hidden layers of 100 neurons for each one with an activation function Relu, where in the output layer we specify the activation function as Linear, which spits out the value that was given. Then we compile our model by specifying the loss Mean Square Error and optimizer Adam.

- **Store transition** is implemented under the name `memorize()` as below:

```

1  # Store previous MDP transition in replay buffer
2  def memorize(self, state, action, reward, next_state, done):
3      self.memory.append((state, action, reward, next_state, done))

```

Listing 4.4: Memorize function

In the `memorize` function we get the input data, which is the previous MDP transition, and store it in the replay buffer that we initialized before.

- **Chose action** We chose action based on the greedy method, as we explained in the previous chapter in section (3.4.5.2), ϵ -greedy method is implemented as *act()* function:

```
1 # Chose action based on epsilon greedy method
2 def act(self, state):
3     # initialize random variable x
4     x = np.random.rand()
5     if x <= self.epsilon:
6         # return random action from action size
7         return random.randrange(self.action_size)
8     # predict action based on current state
9     act_values = self.model.predict(state)
10    # returns action (node ID)
11    return np.argmax(act_values[0])
```

Listing 4.5: ϵ -greedy function

- **Agent learn** the mechanism of how DQN agent will learn is implemented below:

```
1 # replay represent How DQN agent learn?
2 def replay(self, batch_size):
3     # sample minibatch randomly
4     minibatch = random.sample(self.memory, batch_size)
5     for state, action, reward, next_state, done in minibatch:
6         # Q-value of target network = reward
7         target = reward
8         if not done:
9             # Q value of target network = reward + discount factor * expected future
            reward
10            target = (reward + self.gamma * np.amax(self.model.predict(next_state)[0])
11            )
12            # target_f network predict action
13            target_f = self.model.predict(state)
14            # Take best action
15            target_f[0][action] = target
16            # fit the model
17            self.model.fit(state, target_f, epochs=1, verbose=0)
18        if self.epsilon > self.epsilon_min:
19            # Decay probability of taking random action
20            self.epsilon *= self.epsilon_decay
```

Listing 4.6: DQN agent learning function

The agent will sample a minibatch of replay buffer. Then estimate the Q-value for the current state by target_f neural network and the next state by target neural network. Finally, decay the probability of chosen random action.

4.3.2.2 Training results

The simulation will run for 600 steps of training and 200 SFC request, with an evaluation performed every 100 steps. Then, all the information from the model is exported. We use the hyperparameters presented in **table 4.1** which can help achieve better results. A sample of SFC requests used in this phase is presented in **figure 4.3**.

- **Reward** We start by studying the evolution of the reward that is calculated by using the reward function presented in section (3.2.4.3), which minimizes the maximum value of both link and node load balancing for optimizing the two objectives of maximizing the remaining resources of nodes and minimizing the usage of link bandwidth. Now we are going to present the return, which is the sum of future discounted rewards calculated by using the following equation:

$$return = \sum_{step=1}^{600} reward \quad (4.1)$$

Then we calculate the average return to measure the performance of the agent by the following equation:

$$average_return = \frac{\sum_{step=1}^{600} reward}{number_of_reward} \quad (4.2)$$

The results obtained from the DQN model during training are shown in **figure 4.4**:

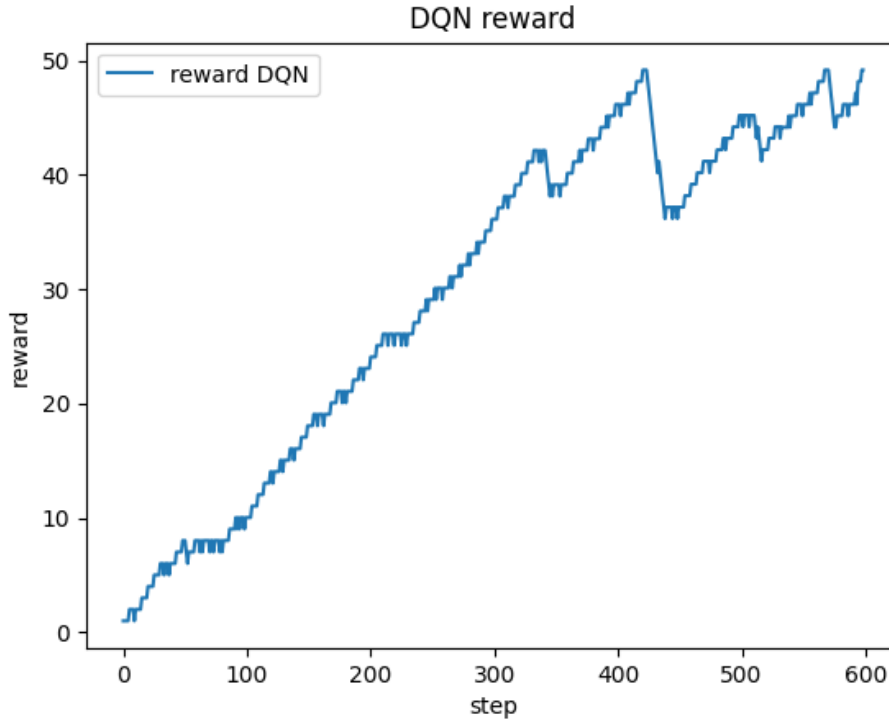


Figure 4.4: DQN training phase reward.

Reward discussion The total reward is increasing little by little while the policy is learning good behavior. It peaks at the highest value 49.17 at step 422, then it goes down a little bit until it reaches the value 36.17 at step 446, after that it continues to rise. That happens because of the nature of agents that learn by training and make mistakes. These mistakes may happen because the constraint of SFC requests is not satisfied. The total reward at step 600 is equal to 49.17. The percentage of average return equal to 8.2%.

4.3.3 Q-learning Agent

Q-learning is the second approach studied in this thesis. First, it will be implemented to see the impact of this approach on SFC deployment and compare that impact with the DQN impact based on evolution of reward, bandwidth consumption, resource consumption and average return rate. **Table 4.3** shows the set of hyperparameters for the Q-learning agent.

Hyperparameter	Value
Learning rate	0.01
Discount factor	0.9
Epsilon	0.9

Table 4.3: Hyperparameter configuration

4.3.3.1 Q-learning agent class

The Q-learning agent class contains many methods, which we will present separately as below:

- **Import libraries** first of all, we start by importing the different libraries that we need to implement our Q-learning agent, which are:

```
1 # Import the used libraries in Q-learning agent
2 import random
3 import numpy as np
4 import pandas as pd
```

Listing 4.7: Used libraries

We use Random library for generating random values, Numpy for dealing with arrays, and Pandas for data analysis and associated manipulation of tabular data in dataframes.

- **Initialization** here, we initialize the hyperparameters as presented in **table 4.3**:

```
1 # Configuration parameters for the whole setup
2 def __init__(self, actions, learning_rate=0.01, reward_decay=0.9, e_greedy=0.9):
3     self.actions = actions      # Action
4     self.lr = learning_rate     # Learning rate
5     self.gamma = reward_decay   # Epsilon greedy decay parameter
6     self.epsilon = e_greedy     # Epsilon greedy parameter
```

```
7 self.q_table = pd.DataFrame(columns=self.actions, dtype=np.float64) # initialize
  q_table
```

Listing 4.8: Q-learning agent initialization

We initialize the q-table for mapping q-values for each state-action pair.

- **Check state in q-table** *check_state_exist()* function is for testing if the given state exist in *q_table*:

```
1 # Test if the state exist in q-table
2 def check_state_exist(self, state):
3     if state not in self.q_table.index:
4         # append new state to q table
5         self.q_table = self.q_table.append(
6             pd.Series(
7                 [0]*len(self.actions),
8                 index=self.q_table.columns,
9                 name=state,))
```

Listing 4.9: *check_state_exist* function

This function is responsible for testing if the current state exists in the *q_table*. If it does not exist, it will create a item for each state-action pair and add it to *q_table*.

- **Chose action** is the ϵ -greedy method that presented in section (3.4.5.2):

```
1 # Chose action based on epsilon greedy method
2 def choose_action(self, state):
3     # Check if state exist in q-table
4     self.check_state_exist(state)
5     # Action selection
6     if np.random.uniform() < self.epsilon:
7         # choose best action
8         state_action = self.q_table.loc[state, :]
9     else:
10        # choose random action
11        action = np.random.choice(self.actions)
12    return action # return the chosen action
```

Listing 4.10: ϵ -greedy function

It starts by checking if the current state exists in *q_table*. After that, by choosing a random value, the agent chooses random action if the value is less than epsilon, otherwise it chooses the best action.

- **Agent learn** algorithm of which present how Q-learning agent will learn is implemented below:

```

1 # learn represent How Q-learning agent learn?
2 def learn(self, state, action, reward, next_state):
3     # Check if next_state exist in q-table
4     self.check_state_exist(next_state)
5     # Get the q-value of state action pair
6     q_predict = self.q_table.loc[state, action]
7     if next_state != 'terminal': # Next_state is not terminal state
8         # Calculate q_target value
9         q_target = reward + self.gamma * self.q_table.loc[next_state, :].max()
10    else: # Next_state is terminal state
11        q_target = reward
12    # Update q-table
13    self.q_table.loc[state, action] += self.lr * (q_target - q_predict)

```

Listing 4.11: Q-learning agent learning function

First of all, the agent tests if the given `next_state` exists in `q_table`. Then `q_predict` gets the `q_value` of the given state and action from `q_table`. After that, the agent checks if the `next_state` is not terminal state for predicts the maximum `q_value` for `q_target`; if yes, `q_target` gets the value of reward. Finally, update `q_table` by discount the difference between `q_target` and `q_predict`.

4.4 Results

The simulation will run for 380 steps for both approaches, DQN and Q-learning. After saving the DQN model every 100 steps, we choose 3 different models with a long time period between them. agent 1 taken in step 100, agent 2 in step 300 and agent 3 in step 600. The data used in the DQN evaluation phase is generated randomly and it will be used by all DQN agents and by the Q-learning. **Figure 4.5** shows a sample of the used data, where *bw* represents the bandwidth requirement for each request, while from *vnf0* to *vnf4* represents resource requirements.

bw	vnf0	vnf1	vnf2	vnf3	vnf4
250	6	7	10	9	11
276	6	6	5	5	14
186	4	13	6	6	10
283	12	10	4	8	10
211	12	5	13	14	12
285	13	12	12	11	11
176	4	12	12	8	3
192	7	13	3	9	14
173	5	3	11	10	3
182	4	6	5	11	12
200	10	7	12	9	13
168	13	5	13	3	14

Figure 4.5: SFC request used in evaluation.

4.4.1 DQN models evaluation results

In this section we are going to present the evaluation results of Deep Q-learning agents:

4.4.1.1 Reward

Figure 4.6 represents the total reward of the three DQN agents, where axis x represents the steps of training and axis y represents the value of the rewards.

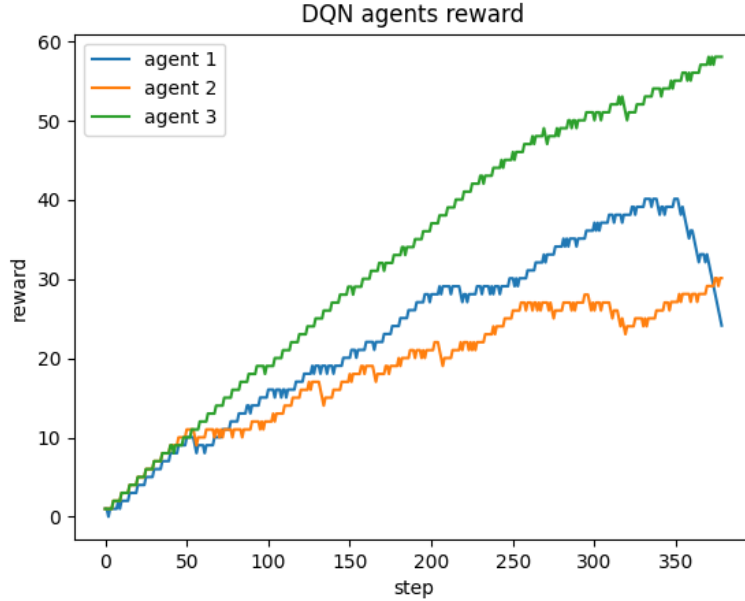


Figure 4.6: DQN agents reward.

Reward discussion We observe that from step 0 to 50 all the DQN agents increase together, with a negligible difference between them. After step 50, *agent1* and *agent2* rise very slowly compared with *agent3*, which rises very fast because it trained very well more than the other agents, where *agent1* has the shortest training period and *agent2* has the medium training period. At the end of this simulation, we note that *agent2* and *agent3* achieve the highest rewards 30.14 and 58.08, respectively, in the last step, while *agent1* achieves 40.14 in step 333, then decreases to the end because the remaining capacities do not satisfy the requested requirements. The obtained total reward at step 380 for *agent1* is : 24.14. While the average return percentages for *agent1*, *agent2*, and *agent3* are 6.35%, 7.93%, and 15.28%, respectively. Based on those results, we come to the conclusion that the third agent is the best trained once, whose behavior improves quite a lot.

4.4.1.2 Node consumption

The reward gives information about the performance of the agent, but it is necessary to look at the other metrics to know if the behavior is as desired. Now, we are going to study the evolution of

the nodes resource consumption, which is calculated by using the following equation:

$$\text{resource_consumption} = \frac{\sum \text{node_resources}}{\text{number_of_nodes}} \quad (4.3)$$

Figure 4.7 represents the total resource consumption of the three DQN agents, where axis x represents the steps of training and axis y represents the value of the resource consumption which is divided by 100 to make the curve readable.

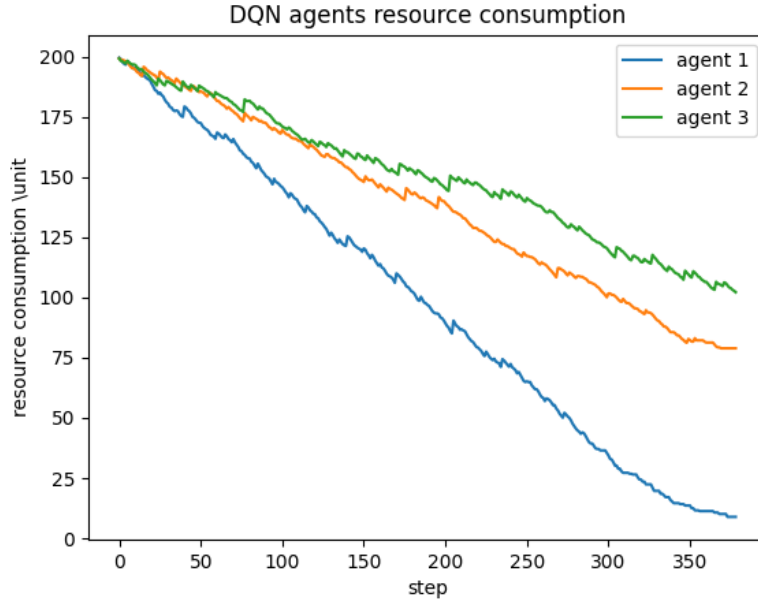


Figure 4.7: DQN agents resource consumption.

Resource consumption discussion

We see that the capacity is assigned according to the available resources in the environment. When the SFC request resources constraint is lower than the total capacity available in nodes, the resources are efficiently allocated to each SFC request. However, as the SFC request resources go up, the allocated capacity will increase, which will lead to a decrease in the nodes capacity. As shown in **figure 4.7**, the third DQN *agent3* still produces the best results, where in step 380 uses only half of the node capacity, where *agent2* uses a little less than half, and *agent1* produces the worst results, using almost all of the node capacity. As a result, the third *agent* outperformed the others and learned extremely well. We see that from step 0 to 25, all the DQN agents resource consumption decreases slowly. After that, we observe that *agent3* continues to decrease slowly, while *agent2* decreases a little bit faster than *agent3*, and *agent1* is the fastest one. At the end of simulation, we observe that at step 380, the remaining resources in the environment of *agent3* are 102 unit, which consumes 44% of the total environment resources, *agent2* is equal to 89 unit and consumes 55.5%, and *agent1* is equal to 9 unit and consumes 95.5%. Based on these results, we find that *agent1*, which has the shortest period

of training, does not know how to maximize his total reward, so he still makes mistakes and does not know how to deal with new different requests. He consumes all the resources of its environment and allocates resources without taking into consideration the objectives that we built him to optimize, and that leads to rejecting all the new arrival requests. While *agent2* achieves little different results than *agent3*, who achieves the best resource consumption due to the powerful training that he has taken.

4.4.1.3 Link consumption

After that, we are going to study the evolution of the links bandwidth consumption, which is calculated by using the following equation:

$$bandwidth_consumption = \frac{\sum link}{number_of_links} \quad (4.4)$$

Figure 4.8 represents the total bandwidth consumption of the three DQN agents, where axis x represents the steps of training and axis y represents the value of the bandwidth consumption which is divided by 100 to make the curve readable.

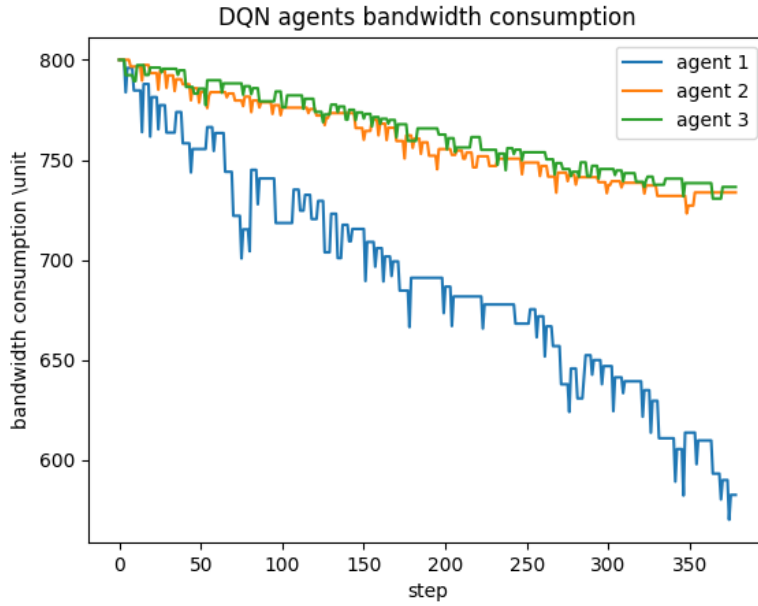


Figure 4.8: DQN agents bandwidth consumption.

Bandwidth consumption discussion We see that the bandwidth capacity is assigned according to the offered bandwidth in the environment. When the SFC request bandwidth constraint is lower than the available bandwidth, the bandwidth is efficiently distributed and allocated to each SFC request. However, as the SFC request bandwidth requirements go up, the allocated bandwidth will increase, which will lead to a decrease in the link capacity. As we see in **figure 4.8** for the first 25

steps, all the agents consumes slightly the same bandwidth. After that, we observe a convergence of consumption between *agent2* and *agent3* with a little difference, but *agent1* decreases rapidly, where he consumes bandwidth much faster than the others. By the end of the simulation, we find that the third DQN agent still achieves the highest results, where in step 380 only 8% of the total bandwidth capacity is used, while *agent2* uses 8.4%, but *agent1* gives the worst result, which uses 27.3%. Based on that, the third agent achieved the best performance and trained very well.

4.4.2 DQN vs Q-learning comparison results

In our work, we will evaluate two approaches: DQN based DRL and Q-learning based RL. For Q-learning, it is necessary to tune the hyperparameters configuration to optimize the learning process, for which we use hyperparameters presented in **table4.3**. After training the DQN model and implementing Q-learning, we are going to compare their results, the simulation were made in the same environment condition with the same number of SFC requests (100).

4.4.2.1 Reward

Figure 4.9 represent the evolution of total reward for both Deep Q-learning agent and Q-learning agent where axis x represents the steps of evaluation and axis y represents the value of the total reward.

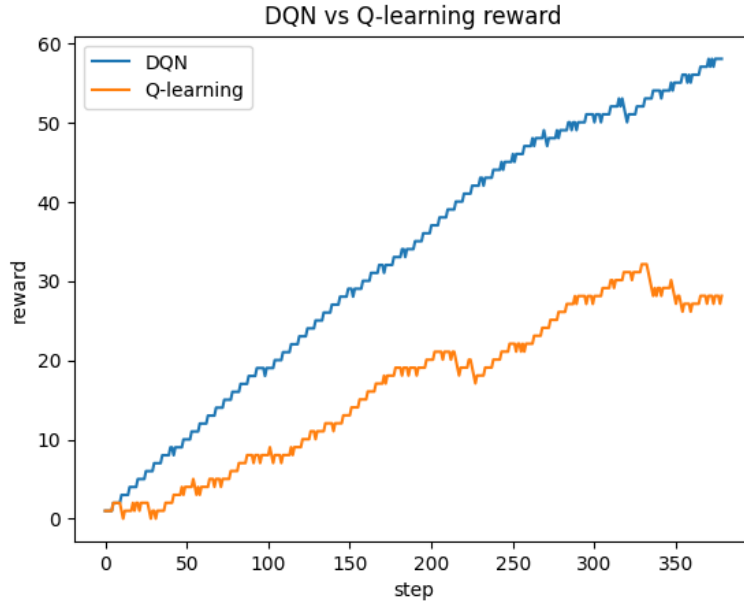


Figure 4.9: DQN vs Q-learning reward.

Reward discussion Here, we focus on the behavior of the policy obtained in each approach, **figure 4.9** shows that the reward of the DQN agent is much higher than the Q-learning agent because the DQN uses Deep Neural Network as a function approximator, which makes him learn

much faster than the approach based q-table. The return of both approaches is: DQN 15.28% and Q-learning 7.40%. The highest reward for DQN is 58.08, achieved in step 380. For Q-learning it is 32.14, achieved in step 332. In the step 380 Q-learning decrease till achieve 28.15 because the DQN agent benefits from neural network advantages, which give him the ability to work with insufficient knowledge.

4.4.2.2 Resource consumption

Now, we will compare the resource consumption of both approaches. The resource consumption for Deep Q-learning agent and the Q-learning agent is presented in **figure 4.10**, where axis x represents evaluation steps and axis y represents resource consumption which is divided by 100 to make the curve readable.

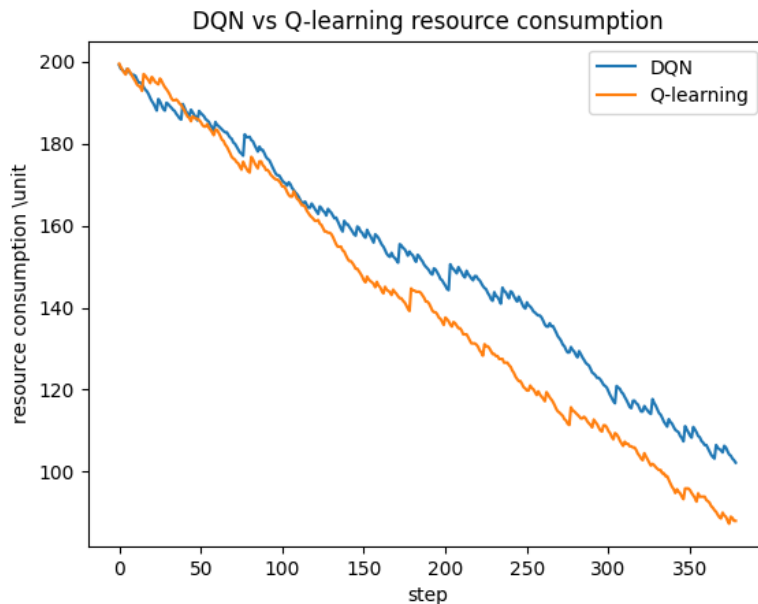


Figure 4.10: DQN vs Q-learning resource consumption.

Resource consumption discussion From the beginning to step 40, the Q-learning agent consumed fewer resources than the DQN agent. From 40 to 56, the consumption is slightly similar between them, with total resource consumption equal to 8% for DQN and 8.5% for Q-learning. After step 56, we observe that the DQN decreases more slowly than Q-learning, where the DQN agent consumes only 44% of the total resources at step 380 and the Q-learning agent consumes 56% of the total resources at the same step. Although the difference between DQN and Q-learning resource consumption not much difference, it has a non-negligible effect on the remaining resources in the environment. These results demonstrate that DQN interacts with the environment more effectively than Q-learning.

4.4.2.3 Bandwidth consumption

Finally, we will compare both approaches in term of bandwidth consumption. **Figure 4.11** shows how DQN and Q-learning agents consume environment bandwidth, with time steps represented by axis x and bandwidth consumption represented by axis y which is divided by 100 to make the curve readable..

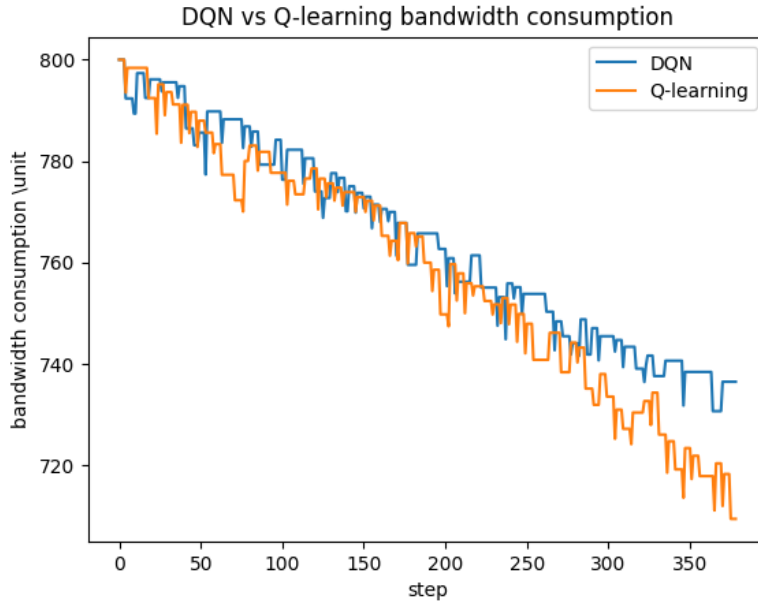


Figure 4.11: DQN vs Q-learning bandwidth consumption.

Bandwidth consumption discussion From the first step to 275, we observe that both agents decrease continuously together, where the DQN agent consumes 6.8% of the total bandwidth and the Q-learning agent consumes 7.8% of the total bandwidth in the environment. After that step, we notice that the DQN agent decreases slowly, which means that he consumes bandwidth in an effective way, whereas the Q-learning agent decreases a little bit faster than DQN, which means the way he consumes bandwidth doesn't give good results as the DQN mechanism. We find that, at step 380, the DQN agent consumes 8%, while Q-learning consumes 11.4% of the total bandwidth.

4.5 Final discussion

Table 4.4 represents the final comparison between all agents in terms of return, resource and bandwidth consumption.

model	return	Resource consumption	Bandwidth consumption
DQN <i>agent1</i>	6.35%	95.5%	27.3%
DQN <i>agent2</i>	7.93%	55.5%	8.4%
DQN <i>agent3</i>	15.28%	44%	8%
Q-learning	7.4%	56%	11.4%

Table 4.4: Comparison between all models

Based on the results presented above, we conclude that DQN *agent3* approach outperforms other agents in solving SFC deployment problem by achieving the best return while consuming the least amount of resource and bandwidth. The second agents who gives acceptable results *agent2* and Q-learning agent which achieves a slightly difference than *agent2*, while *agent1* remains at the bottom of the list with its poor performance.

4.6 Conclusion

In this last chapter, we presented the different software tools that we used for the implementation of our project. Next, we described the implementation of our approaches, starting from the steps of configuring the environment, building DQN and Q-learning algorithms and selecting different hyperparameters that are included in the training of the model and validating our model. At the end, we discussed the different results obtained, which show the performance of our model compared with the RL-based approach.

General conclusion

Traditional networks have reached their maximal capabilities. The increase in throughput and the number of users cannot be met by simple hardware evolution. The management of networks must also evolve. This evolution began with the introduction of Software Defined Network (SDN) and Network Function Virtualization (NFV). These two paradigms enable programmable networks, resource abstraction and enabling automatic management. 5G increases the need for network management evolution and innovation. It imposes severe constraints on latency, mobility, throughput, energy efficiency, security, and, most importantly, resource utilization. The infrastructure of 5G networks is also evolving, but this must be coupled with a change in the network management approaches. The great heterogeneity of use cases and service requirements in 5G networks led to the introduction of the Service Function Chain (SFC), this new paradigm involves the use of the capabilities supplied by SDN and NFV to create virtual network services. With the new technologies come new algorithms with many objectives will be proposed.

Artificial Intelligence not only provides the ability for our devices to perceive, reason, and act intuitively, but also changes how we approach and solve technical challenges. The advent of 5G is introducing new challenges for network service providers, where they face a problem in deploying network services intelligently with a guarantee of using the minimum resources of nodes and links. In this thesis, we propose a Deep Reinforcement learning model Deep Q-learning (DQN) for solving the SFC deployment problem in 5G networks. We focus on minimizing the use of node resources and link bandwidth. We start by implementing our environment where the DQN agent will learn how to deploy SFC requests based on its requirements and the network state. We also propose a Reinforcement Learning approach (Q-learning) that helps us evaluate the performance of our DQN model. We can say that both approaches succeeded in solving the problem successfully, but with different performances, where the experimental results show that the DQN agent achieves 15.28% of the average return while Q-learning achieves 7.4%, which confirms that adopting the DRL approach gives great potential in solving networking problems, especially with the presence of neural networks. Although our proposed DRL approach can achieve good results, there are still some shortcomings

that need to be resolved. Firstly, there is no free simulator for 5G networks that support SDN, NFV and SFC, to assess how the network will behave for deploying SFC requests using both approaches. Secondly, the parameters of neural networks and Deep Reinforcement Learning, including hidden layers, neuron number, batch size, learning rate and other hyperparameters, are designed through manual choice, by doing several tests and varying the different model hyperparameters. Finally, we can't guarantee that this performance is the best.

Future development

In this thesis, we have seen the use of DRL-based approach for SFC deployment in order to evaluate bandwidth and resource consumption on 5G networks, but there are some enhancements that can be done to gain more performance:

- Optimize the used hyperparameters of the agent and also the environment configuration.
- Apply the DQN model in a network emulator environment like (Mininet or Openmano) to achieve the experimental results in line with the actual conditions.

Bibliography

- [1] Series, M. (2015). IMT Vision–Framework and overall objectives of the future development of IMT for 2020 and beyond. Recommendation ITU, 2083, 21.
- [2] Gausseran, A. (2021). Optimization algorithms for network slicing for 5G (Doctoral dissertation, Université Côte d’Azur).
- [3] Mobile industry approves completion of standalone release 15 5G specifications. (2018,). ICT Monitor Worldwide.
- [4] Navarro-Ortiz, J., Romero-Diaz, P., Sendra, S., Ameigeiras, P., Ramos-Munoz, J. J., Lopez-Soler, J. M. (2020). A survey on 5G usage scenarios and traffic models. *IEEE Communications Surveys and Tutorials*, 22(2), 905-929. <https://doi.org/10.1109/COMST.2020.2971781>
- [5] Gupta, A., Jha, R. K. (2015). A survey of 5G network: Architecture and emerging technologies. *IEEE Access*, 3, 1206-1232. <https://doi.org/10.1109/ACCESS.2015.2461602>
- [6] Medhat, A. M., Taleb, T., Elmangoush, A., Carella, G. A., Covaci, S., Magedanz, T. (2017). Service function chaining in next generation networks: State of the art and research challenges. *IEEE Communications Magazine*, 55(2), 216-223. <https://doi.org/10.1109/MCOM.2016.1600219RP>
- [7] Basavarajappa, V. (2018). A Proposal of Antenna Topologies for 5G Communication Systems (Doctoral dissertation, universidad de cantabria).
- [8] Chen, S., Zhao, J. (2014). The requirements, challenges, and technologies for 5G of terrestrial mobile telecommunication. *IEEE Communications Magazine*, 52(5), 36-43. <https://doi.org/10.1109/MCOM.2014.6815891>
- [9] Ordonez-Lucena, J., Ameigeiras, P., Lopez, D., Ramos-Munoz, J. J., Lorca, J., Folgueira, J. (2017). Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges. *IEEE Communications Magazine*, 55(5), 80-87. <https://doi.org/10.1109/MCOM.2017.1600935>
- [10] Fan, J., Wang, Z., Xie, Y., Yang, Z. (2019). A theoretical analysis of deep Q-learning.

- [11] Barakabitze, A. A., Ahmad, A., Mijumbi, R., Hines, A. (2020;2019;). 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks* (Amsterdam, Netherlands : 1999), 167, 106984. <https://doi.org/10.1016/j.comnet.2019.106984>
- [12] European Telecommunications Standards Institute. (2014). Network functions virtualisation (NFV); architectural framework.
- [13] Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., De Turck, F., Boutaba, R. (2016;2015;). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 18(1), 236-262. <https://doi.org/10.1109/COMST.2015.2477041>
- [14] Best practices to accelerate 5g base station deployment: Massive mimo. Available online: <https://www.qorvo.com/design-hub/blog/best-practices-to-accelerate-5g-base-station-deployment>. (accessed on 20/03/2022)
- [15] European Telecommunications Standards Institute. (2014). Network functions virtualisation (NFV); management and orchestration.
- [16] Sezer, S., Scott-Hayward, S., Chouhan, P. K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., Rao, N. (2013). Are we ready for SDN? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7), 36-43. <https://doi.org/10.1109/MCOM.2013.6553676>
- [17] Open Networking Foundation. Available online: <https://www.opennetworking.org/>. (accessed on 08/01/2022)
- [18] 5g bytes: Beamforming explained, <https://spectrum.ieee.org/5g-bytes-beamforming-explained> . (accessed on 09/01/2022)
- [19] Ahmadi, S. (2019). 5G NR: Architecture, technology, implementation, and operation of 3GPP new radio standards. <https://doi.org/10.1016/C2016-0-04944-6>
- [20] Yan, W., Zhu, K., Zhang, L., Su, S. (2017). Efficient dynamic service function chain combination of network function virtualization. Paper presented at the 163-168. <https://doi.org/10.1109/ICDCSW.2017.39>
- [21] Ghosh, A., Thomas, T. A., Cudak, M. C., Ratasuk, R., Moorut, P., Vook, F. W., Rappaport, T. S., MacCartney, G. R., Sun, S., Nie, S. (2014). Millimeter-wave enhanced local area systems: A high-data-rate approach for future wireless networks. *IEEE Journal on Selected Areas in Communications*, 32(6), 1152-1163. <https://doi.org/10.1109/JSAC.2014.2328111>

- [22] Schaller, S., Hood, D. (2017). Software defined networking architecture standardization. *Computer Standards and Interfaces*, 54, 197-202. <https://doi.org/10.1016/j.csi.2017.01.005>
- [23] Xiao, Y., Zhang, Q., Liu, F., Wang, J., Zhao, M., Zhang, Z., Zhang, J. (2019). NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning. Paper presented at the 1-10. <https://doi.org/10.1145/3326285.3329056>
- [24] Koo, J., Mendiratta, V. B., Rahman, M. R., Walid, A. (2019). Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics. Paper presented at the <https://doi.org/10.23919/CNSM46954.2019.9012702>
- [25] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1), 1-99. <https://doi.org/10.1186/s13174-018-0087-2>
- [26] Alnwaimi, G., Vahid, S., Moessner, K. (2015). Dynamic heterogeneous learning games for opportunistic access in LTE-based Macro/Femtocell deployments. *IEEE Transactions on Wireless Communications*, 14(4), 2294-2308. <https://doi.org/10.1109/TWC.2014.2384510>
- [27] Ayoubi, S., Limam, N., Salahuddin, M. A., Shahriar, N., Boutaba, R., Estrada-Solano, F., Caicedo, O. M. (2018). Machine learning for cognitive network management. *IEEE Communications Magazine*, 56(1), 158-165. <https://doi.org/10.1109/MCOM.2018.1700560>
- [28] Arulkumaran, K., Deisenroth, M. P., Brundage, M., Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6), 26-38. <https://doi.org/10.1109/MSP.2017.2743240>
- [29] Wang, M., Cui, Y., Wang, X., Xiao, S., Jiang, J. (2018;2017;). Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2), 92-99. <https://doi.org/10.1109/MNET.2017.1700200>
- [30] Pandey, S., Hong, J. W., Yoo, J. (2020). Q-learning based SFC deployment on edge computing environment. Paper presented at the 220-226. <https://doi.org/10.23919/APNOMS50412.2020.9236981>
- [31] Fourati, H., Maaloul, R., Chaari, L. (2020;2021;). A survey of 5G network systems: Challenges and machine learning approaches. *International Journal of Machine Learning and Cybernetics*, 12(2), 385-431. <https://doi.org/10.1007/s13042-020-01178-4>
- [32] Python Collections Module. Available online: <https://www.geeksforgeeks.org/python-collections-module/> (accessed on 05/06/2022)

- [33] Chang, Z., Lei, L., Zhou, Z., Mao, S., Ristaniemi, T. (2018). Learn to cache: Machine learning for network edge caching in the big data era. *IEEE Wireless Communications*, 25(3), 28-35. <https://doi.org/10.1109/MWC.2018.1700317>
- [34] Morocho-Cayamcela, M. E., Lee, H., Lim, W. (2019). Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions. *IEEE Access*, 7, 137184-137206. <https://doi.org/10.1109/ACCESS.2019.2942390>
- [35] Ye, D., Au, M. Z. (2018). A self-adaptive Sleep/Wake-up scheduling approach for wireless sensor networks. *IEEE Transactions on Cybernetics*, 48(3), 979-992. <https://doi.org/10.1109/TCYB.2017.2669996>
- [36] Azmat, F., Chen, Y., Stocks, N. (2016). Predictive modelling of RF energy for wireless powered communications. *IEEE Communications Letters*, 20(1), 173-176. <https://doi.org/10.1109/LCOMM.2015.2497306>
- [37] Dhahri, C., Ohtsuki, T. (2012). Q-learning cell selection for femtocell networks: Single- and multi-user case. Paper presented at the 4975-4980. <https://doi.org/10.1109/GLocom.2012.6503908>
- [38] Moreno, J. F. C., Sattler, R., Caulier Cisterna, R. P., Celsi, L. R., Rodríguez, A. S., Mecella, M. (2021). Online service function chain deployment for live-streaming in virtualized content delivery networks: A deep reinforcement learning approach. *Future Internet*, 13(11), 278. <https://doi.org/10.3390/fi13110278>
- [39] Ning, Z., Wang, N., Tafazolli, R. (2020). Deep reinforcement learning for NFV-based service function chaining in multi-service networks : Invited paper. Paper presented at the , 2020-<https://doi.org/10.1109/HPSR48589.2020.9098994>
- [40] Matplotlib: Visualization with Python. Available online: <https://matplotlib.org/> (accessed on 05/06/2022 at 17:35)
- [41] Chen, X., Li, Z., Zhang, Y., Long, R., Yu, H., Du, X., Guizani, M. (2018). Reinforcement learning-based QoS/QoE-aware service function chaining in software-driven 5G slices. *Transactions on Emerging Telecommunications Technologies*, 29(11), e3477-n/a. <https://doi.org/10.1002/ett.3477>
- [42] Tomassilli, A., Giroire, F., Huin, N., Perennes, S. (2018). Provably efficient algorithms for placement of service function chains with ordering constraints. Paper presented at the , 2018-774-782. <https://doi.org/10.1109/INFOCOM.2018.8486275>
- [43] Roderick, M., MacGlashan, J., Tellex, S. (2017). Implementing the deep Q-network.

- [44] Bojović, B., Meshkova, E., Baldo, N., Riihijärvi, J., Petrova, M. (2016). Machine learning-based dynamic frequency and bandwidth allocation in self-organized LTE dense small cell deployments. *EURASIP Journal on Wireless Communications and Networking*, 2016(1), 1-16. <https://doi.org/10.1186/s13638-016-0679-0>
- [45] Martin, A., Egana, J., Florez, J., Montalban, J., Olaizola, I. G., Quartulli, M., Viola, R., Zorrilla, M. (2018). Network resource allocation system for QoE-aware delivery of media services in 5G networks. *IEEE Transactions on Broadcasting*, 64(2), 561-574. <https://doi.org/10.1109/TBC.2018.2828608>
- [46] Sarigiannidis, P., Sarigiannidis, A., Moscholios, I., Zwierzykowski, P. (2017). DIANA: A machine learning mechanism for adjusting the TDD uplink-downlink configuration in XG-PON-LTE systems. *Mobile Information Systems*, 2017, 1-15. <https://doi.org/10.1155/2017/8198017>
- [47] Sanchez-Fernandez, M., de-Prado-Cumplido, M., Arenas-Garcia, J., Perez-Cruz, F. (2004). SVM multiregression for nonlinear channel estimation in multiple-input multiple-output systems. *IEEE Transactions on Signal Processing*, 52(8), 2298-2307. <https://doi.org/10.1109/TSP.2004.831028>
- [48] Alkhateeb, A., Alex, S., Varkey, P., Li, Y., Qu, Q., Tujkovic, D. (2018). Deep learning coordinated beamforming for highly-mobile millimeter wave systems. *IEEE Access*, 6, 37328-37348. <https://doi.org/10.1109/ACCESS.2018.2850226>
- [49] Huang, H., Yang, J., Huang, H., Song, Y., Gui, G. (2018). Deep learning for super-resolution channel estimation and DOA estimation based massive MIMO system. *IEEE Transactions on Vehicular Technology*, 67(9), 8549-8560. <https://doi.org/10.1109/TVT.2018.2851783>
- [50] Parwez, M. S., Rawat, D. B., Garuba, M. (2017). Big data analytics for user-activity analysis and user-anomaly detection in mobile wireless network. *IEEE Transactions on Industrial Informatics*, 13(4), 2058-2065. <https://doi.org/10.1109/TII.2017.2650206>
- [51] Liao, Z., Zhang, R., He, S., Zeng, D., Wang, J., Kim, H. (2019). Deep learning-based data storage for low latency in data center networks. *IEEE Access*, 7, 26411-26417. <https://doi.org/10.1109/ACCESS.2019.2901742>
- [52] Tan, L. T., Hu, R. Q. (2018). Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 67(11), 10190-10203. <https://doi.org/10.1109/TVT.2018.2867191>

- [53] Wang, L., Cheng, S. (2019;2018;). Data-driven resource management for ultra-dense small cells: An affinity propagation clustering approach. IEEE Transactions on Network Science and Engineering, 6(3), 267-279. <https://doi.org/10.1109/TNSE.2018.2842113>
- [54] Tabrizi, H., Farhadi, G., Cioffi, J. (2012). Dynamic handoff decision in heterogeneous wireless systems: Q-learning approach. Paper presented at the 3217-3222. <https://doi.org/10.1109/ICC.2012.6364194>
- [55] Yang, Y., Meng, X., Kang, Q., Zhao, W. (2021). Reliable service function chain deployment method based on traffic optimization. Xi Tong Gong Cheng Yu Dian Zi Ji Shu, 43(10), 3017-3025. <https://doi.org/10.12305/j.issn.1001-506X.2021.10.38>
- [56] Orłowski, S., Wessälly, R., Pióro, M., Tomaszewski, A. (2010;2009;). SNDlib 1.0-survivable network design library. Networks, 55(3), 276-286. <https://doi.org/10.1002/net.20371>
- [57] Google Colaboratory. Available online: <https://research.google.com/colaboratory/faq.html> (accessed on 05/06/2022)
- [58] What is the Jupyter Notebook?. Available online: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html (accessed on 05/06/2022)
- [59] What is Python? Executive Summary?. Available online:<https://www.python.org/doc/essays/blurb/> (accessed on 05/06/2022)
- [60] About Keras. Available online: <https://keras.io/about/> (accessed on 05/06/2022)
- [61] What is numpy?. Available online: <https://numpy.org/doc/stable/user/whatisnumpy.html> (accessed on 05/06/2022)
- [62] pandas - Python Data Analysis Library. Available online: <https://pandas.pydata.org/> (accessed on 05/06/2022)
- [63] Python Random Module . Available online: <https://www.geeksforgeeks.org/python-random-module/> (accessed on 05/06/2022)