Optimising Resource Allocation for Computational Offloading in a Mobile Edge Environment

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy (PhD)

in

Department of Electronic and Computer Engineering

College of Engineering, Design and Physical Sciences

Brunel University London

by

Sarfraz Hussain

July 2024

Dedication

To everyone who supported me throughout this process and to my beautiful, loving partner Dimitra without whom, this would not have been possible. And to Professor Li, the man, the myth, the legend.

Declaration of Authorship

The work detailed in this thesis has not been previously submitted for a degree in this university or at any other unless otherwise referenced. It is the author's own work.

Acknowledgement

First and foremost, I am immensely grateful to Prof. Maozhen Li, my esteemed supervisor and mentor. Prof. Li's exceptional expertise in the field of 5G MEC and his unwavering commitment to my academic growth have been instrumental throughout this journey. His invaluable guidance, patience, and profound insights have not only shaped the direction of this research but have also instilled in me a deep passion for exploring the intricacies of RL resource allocation. I am truly indebted to Prof. Li for his tireless support, encouragement, and belief in my capabilities. I would also like to thank Dr. Zobaa for his support throughout the appeals process and reassurance.

Furthermore, I would like to extend my heartfelt appreciation to my partner, Dimitra Avramidou, whose unwavering support and understanding have been the cornerstone of my perseverance. Dimitra's patience during the long hours of research and writing have been my source of strength. Her love, care, and understanding have kept me motivated and balanced throughout this challenging process. I am forever grateful to Dimitra for being my rock and for standing by my side unconditionally despite the prolonged of this research.

Abstract

With the recent albeit limited rollout of the fifth generation of communications, alongside the widespread adoption of open-source networking solutions based on SDN and NFV technologies, opportunities to define the architecture of 5G over its lifetime have become a hot topic in the industry, both professionally and academically. Despite noticeable advances in bandwidth, services planned to be integrated deep within the architecture of 5G technologies such as Mobile Edge Computing are emerging.

The successful allocation of resources is a pivotal component upon which effective, latency-sensitive handling of data will build on to enhance the future of communication. This research makes three significant contributions to the field of Multi-access Edge Computing (MEC). Firstly, it involves testing and validating various network simulation software to identify the most effective tools for simulating MEC environments. The efficiency of these simulators is evaluated to ensure they accurately replicate real-life network scenarios, which is crucial for constructing precise algorithms and determining simulation parameters.

Secondly, the study implements a single-layer reinforcement learning (RL) algorithm within the orchestration module of the simulator to optimize network resource allocation. The goal of the algorithm is to reduce latency and task failure rates while increasing efficiency. The RL algorithm is benchmarked against traditional methods like Round Robin and Greedy algorithms, demonstrating significant improvements in network service levels and task success rates.

Lastly, the research develops a multi-layer reinforcement learning algorithm based on the initial single-layer approach. This advanced algorithm incorporates replay memory and approximate Q functions within a neural network, addressing various stages of network infrastructure and leveraging previously generated Q tables. These enhancements ensure more efficient and effective network management in MEC environments.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Nomenclature

4IR	4th Industrial Revolution
5G	5th Generation of Communication
AI	Artificial Intelligence
API	Application Programming Interface BYO
AR	Augmented Reality
ASP	Authorised Service Provider
BYO	Bring Your Own
BSS	Business support system
CAPEX	Capital Expenditure
CCM	Client Connection Manager
CFS	Customer Facing Service
CI/CD	Continuous Integration/Continuous Delivery
CoT	Cloud of Things
CP	Control Plane
CPE	Customer Premises Equipment
CRAN	Cloud Radio Access Network
D2D	Device-To-Device
DASH	Dynamic Adaptive Streaming over HTTP
DISBAT	Display and Battery Device
DN	Domain Name
DNS	Domain Name System
DSL	Digital Subscriber Line
DSRC	Digital Short-Range Communications
DT	Data Tunnel
E2E	End-To-End
EAB	Edge Accelerated Browser
EPC	Evolved Packet Core
EPG	Electronic Programme Guide
GPS	Global Positioning System
GPRS	General Packet Radio Service

GW	Gateway
HTTP	Hyper Text Transfer Protocol
HW	Hardware
ICE	Integrated Communications Environment
IGMP	Internet Group Multicast Protocol
IM	Instant Messaging
IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
IT	Information Technology
IXP	Internet Exchange Point
LAN	Local Area Network
LI	Lawful Interception
LOS	Line of Sight
LTE	Long Term Evolution (4g)
LTE	Long Term Evolution
MAC	Media Access Control
MADP	Multiple Access Data Proxy
MAMS	Multiple Access Management Services
MANO	Management and Orchestration
MBS	Macro Base Station(s)
ME	Mobile Equipment
MEAO	MEC Application Orchestrator
MEC	Mobile Edge Computing
MIMO	Massive Input Massive Output
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MNO	Mobile Network Operator
MPEG	Moving Pictures Experts Group
MTC	Machine Type Communication

NCM	Network Connection Manager
NEF	Network Exposure Function
NFV	Network Function Virtualisation
NFVO	Network Function Virtualisation Orchestrator
NOMA	Non-Orthogonal Multiple Access
NTP	Network Time Protocol
OPEX	Operating Expenditure
OSS	Operations Sub-System
OTT	Over-The-Top
PBX	Private Branch Exchange
PCC	Policy Control and Charging
PCF	Policy Control Function
PER	Packet Error Rate
PHY	Physical (Layer)
PIM	Protocol-Independent-Multicast
POX	Plain Old XML
PSS	Packet Switched Streaming Service
PTP	Precision Time Protocol
QCI	Quality Class Indicator
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RAP	Random Access Preamble
RAT	Radio Access Tower
RD	Retained Data
REST	Representational State Transfer
RL	Reinforced Learning
RNC	Radio Network Controller
RNI	Radio Network Information
SAND	Server and Network assisted DASH
SAR	Specific Absorption Rate
SBS	Small Base Station(s)

SDN	Software Defined Networking
SIC	Successive Interference Cancellation
SLA	Service Level Agreement
SMF	Session Management Function
SMS	Short Message Service
SPID	Subscriber Profile ID
SRS	Sounding Reference Signal
STB	Set Top Box
TCP	Transmission Control Protocol
TEID	Tunnel Endpoint ID
TEO	Third-party Edge Owner
TV	Television
UE	User Equipment
UP	User Plane
UPF	User Plane Function
UTC	Universal Time (Co-ordinated)
UX	User Experience
V2X	Vehicle-to-Everything
VEC	Vehicular Edge Computing
VLC	Visible Light Communication
VM	Virtual Machine
VNF	Virtual Network Function
VOD	Video on Demand
VR	Virtual Reality
WLAN	Wireless Local Area Network
WCET	Worst-case execution Time

Chapter 1 Introduction

According to the 3GPP release documentation, the full evolution of 4G communication took almost 11 years to reach its potential, a potential that most urban populations in developed countries are yet to experience [1]. Naturally, we must learn from previous endeavours and speed up the process of evolution. Therefore, to ensure rapid deployment of the future of communication, steps must be taken to improve the process of upgrading the infrastructure.

To this end, many corporate giants in the industry have already begun research and development within the sector, leveraging upcoming advancements such SDN and NFV. Companies such as Nokia, Huawei and NTT Docomo have begun developing technologies that will allow for extended service offerings and ‘evolved networking’ [2], [3]. With the plethora of services due to be introduced, innovation must accompany technology to ensure that demands of UEs are met.

The 5th Generation of communication is looming overhead, with an initial expected rollout date already elapsed according to the 3GPP [4]. Along with improvements to latency [5] and speed requirements, 5G will bring with it a whole host of new technologies that will further enhance supplier’s abilities to offer consumers greater usability and a wider range of services[6]. Smartphone and desktop applications have shifted over to the cloud in recent years, and the number of application service providers considering cloud only hosting of centralised software has increased exponentially, access to the cloud will need to be optimised to guarantee a rich experience therefore, close collaboration between network operators, applications and content providers is of utmost importance [7].

Unfortunately, this heavy reliance on links to centralised locations offering cloud services inevitably creates bottlenecks, hindering the overall experience of services and applications, despite the efforts to upgrade existing links. The

expected growth has been further accelerated by the unprecedented pandemic [8], where global growth from the previous year has been 60%.

One of the sub-technologies garnering a wide interest is the concept of Edge Computing [9] which attempts to alleviate the heavy burden being placed on backhaul links, circumventing network congestion and reducing service latency [10]. This thesis will attempt to explore the variety of uses that Edge Computing offers and the possibilities of future developments. The shift to Industry 4.0 and Web 2.0 requires greater utilization of communications technologies [11] and in turn, this also increases expectations and server load.

Through leverage of Edge Computing (EC), we find that systems are not only more connected, but more powerful than ever. The offering of offloaded computation enabled by MEC can offer smaller devices the ability to communicate and operate at a greater length. They can offer increased power, limited only by the latency between the nearest point of connection.

Networking performance of said devices are key indicators in their usability, particularly where visual interaction of the real world with machines is concerned. The slightest amount of latency can result in loss of QoS or render some applications and processes invalid, e.g., robot-assisted remote surgery [12]. To this end, the use of offloaded computation can enable applications and services within devices not able to offer such applications on their own. Such an approach also further future-proofs devices already in circulation as well as offering updates that can increase rather than decrease performance over time with modular upgrade to the infrastructure that the computational model relies on.

This in turn can enable image processing, whether AI assisted or not, within micro devices, effectively evolving a personal communications/wearable device into actuators and sensors that simply relay information so data can be processed externally. Latency requirements differ across applications, as the speeds of networks have increased, so have the plethora of applications

available for users, TABLE 1.1 contains some of the latency requirements for a variety of different application purposes:

TABLE 1.1
APPLICATION LATENCY REQUIREMENTS

Wearable devices	Latency	Capacity	Reliability	Possible specifications
AR/VR helmets	High	High	Low	mm-wave cellular, WLAN
Smartphones/tablets	Medium	Med-High	Medium	LTE, Bluetooth, mm-wave, cellular and WLAN
Medical Sensors	High	Low	High	LTE, Bluetooth
Smart watch/glasses	Medium	Medium	Low	LTE, Bluetooth
Smart clothing/shoes	Low	Low	Low	Bluetooth, ZigBee

The second major concern in said devices, is that of power. Despite the tremendous advances in processing capabilities in CPUs and GPUs, advances in battery life have not shown such progression.

Whether due to the lack of material development or due to health and safety concerns, hardware manufacturers are wary of the amount of power that can be condensed into a battery and subsequently used in mobile devices, various manufacturers have experimented with the technology, but the risks can damage a company's reputation hence, manufacturers, often, proceed with caution.

This is somewhat offset by emerging techniques such as system-on-chip and system-in-package, which offer scaled down PCBs making it possible to creatively implement silicon within a range of hardware. Unfortunately, due to the variety of hardware being introduced such as wearables, drones etc., real estate (i.e., physical size) within the hardware is almost always limited and battery power is often relative to the size of the device and the time of release [12].

This is prominent in the wearables market where bulky devices are often shunned by prospective markets and doomed to failure, but also carries on into other hardware sectors, bulky simply is not to the average consumers taste any longer [12].

Although EC offers up a host of benefits to industries and consumers alike, cost, density and health implications are a factor that can raise some concern. Architectural and implementational costs must be appealing to a business to ensure that the cost of hardware and therefore profitability is manageable. Thus, ensuring the prospect of competition within industrial and commercial application, subsequently leading to innovation across the sector.

This is apparent when we compare the adoption rate of computation within industry spanning over the last 5 decades, now there are virtually no corporations that do not utilise the internet or the use of a computer in some form or another whereas in the past, industries wary of technology were at the mercy of manual methods, creating redundancy and opening the door for human error prone processes.

The application of EC applies to both fixed line and mobile communications technologies, but in turn, enables a variety of software solutions that were not previously possible. However, despite the obvious benefits, as always, there are some obstacles in the adoption of new technology including those from a social-economic standpoint.

Controversy within the launch window of 5G portrayed harmful effects of mm-Wave technology as being above the SAR and potentially harmful to human beings due to containing increased PD and ensuring that OPEX and CAPEX requirements are met [12]. Research within a contained environment, however, is very different to real-life scenario testing and to fully understand the implications of such advancements and mitigate risks, further study is required within the field.

Typically, to conduct this type of research, most solutions involve simulations of real-world scenarios within an open-source or academic development plan. A simulator is designed to conduct early research subsequently being released into the wild as an open-source project which is then contributed towards by numerous developers around the world and eventually developed enough to somewhat simulate a real-world scenario.

Consequently, many cloud simulation applications are currently in circulation, giving us the opportunity to adequately test the performance of networks within cloud environments. On the other hand, many weaknesses have been identified with cloud environments such as inevitable variable network environments and the number of hops that applications must traverse, as well as single-point of-failure. There are also numerous security concerns to take into consideration when storing all customer data within a cloud-hosted internet-based platform [13].

There is also unease regarding bandwidth availability, as anyone familiar with IoT, Big Data and machine learning surely knows, the devices on the internet are expected to multiply significantly and the underlying network must evolve to ensure that it is capable of smooth functionality meeting strict availability requirements under these conditions.

As observed with the lack of forethought in IPv4 address space, it is pivotal to ensure that 5G will meet user demands and have expansibility options available once demand has exceeded capacity.

The goal of this research is to devise an optimisation algorithm, implemented in software form, to find the ideal solution of resource allocation according to the requirements outlined by a variety of different tasks with their respective resource requirements. This will be achieved by determining the optimal solution based on Figure 2.2. The intention is to use state-of-the-art techniques such as Reinforcement Learning (RL) to determine the best possible solution to allocate resources and enable a host of applications servicing a multitude of tenants on an Edge Computing server.

- To get involved in the emerging MEC technology.
- To create an intelligent orchestration method of resource allocation within MEC environment
- Multi-layer Reinforcement Learning application within MEC infrastructure

I extend the previous contribution by developing a multi-layer reinforcement learning application. This involves implementing a more sophisticated model that can optimize resource allocation across multiple layers of the MEC infrastructure, considering various factors and constraints using approximate Q-Learning and Experience Replay.

1.1 Motivation

Despite increased advancements in RAN and Cloud Computation, there are still many unsolved mysteries when it comes to supplemental computation for mobile and IoT devices that can solve challenges in a low-latency environment. Addressing these challenges requires a comprehensive resource allocation mechanism that can efficiently distribute resources, reducing network load and task failure rates based on latency and time to compute whilst taking into consideration energy usage constraints and ensuring the efficiency of the architecture.

Although addressing all the above will inevitably take time and research, utilising technologies such as AI or RL within this context can greatly reduce the aforementioned challenges.

Thus, the motivations for this research are formulated as follows:

- To get involved in the emerging MEC technology
- To create an intelligent orchestration method of resource allocation within MEC environment
- Multi-layer Reinforcement Learning application within MEC infrastructure

1.2 Methodology

This research utilizes an experimental method to model and simulate enhanced RL algorithms, aiming to improve resource allocation by employing several different AI techniques to optimize network efficiency for MEC environments using a comprehensive open-source simulation software, PureEdgeSim.

The goals within the scope of this research are to simulate a larger environment that expands outside the realms of pure IoT/IIoT and to attempt to tackle consumer-facing networks within a metropolitan area. Thus, ensuring the correct simulation parameters were paramount to valid results.

A sophisticated model was built for an RL algorithm that optimises resource allocation within a simulated Edge Computation network hosting a multitude of devices including health, IoT, end-user and vehicle using a multi-stage learning strategy, considering various factors and constraints using approximate Q-Learning and Experience Replay.

The model pre-emptively and continuously adapts to the network, ensuring the best QoS for users and efficient utilisation of networked resources. Measured results include task failure rate, task initialisation time and energy consumption. The tiers are divided into the following stages: WAN aware RL, LAN aware RL and Edge-aware RL. In each stage, limited parameters were provided, including historical data of utilization. Different algorithms were employed by the three tiers based on contextual data deemed most beneficial to each respective tier, thereby preventing unnecessary data transmission among linked tiers.

Finally, the algorithm is be compared with other solutions and critiqued on its efficiency. Results are analysed, and an evaluation presented. The research work includes a detailed literature review, giving background of the topic of MEC and its contextual application within larger network environments. This combined with the results of the simulation show a promising outlook on how multi-layer and tiered AI algorithms can promises enhancement of orchestration methods.

1.3 Major Contributions to Knowledge

The goal of this thesis is to prove the application of RL within resource allocation in MEC. Existing academic efforts fail to explore the depths of the topic, whilst disregarding the dynamic network conditions of a RAN. We will use several techniques to simulate a 5G network realistically and monitor how resources can be efficiently allocated whilst considering the range of applications that MEC will be able to service.

The three major contributions of this research are:

- **Testing and validation of current available simulations for an MEC environment**:

  This thesis compares the numerous network simulation software to discover the optimal solution for simulating MECs. Each simulation software is measured in efficiency for the various modules incorporated within the simulator to ensure that accurate simulation results are returned, like life implementation. Not only is this step vital to ensuring the validity of the research conducted over the course of this thesis, but it also is an important first step to determining how the algorithms will be constructed and variables to take into consideration when deciding simulation parameters. Validity of simulation software is assured through testing of the integrity of simulation software and accountability for the various aspects of the network infrastructure.

- **Single Layer Reinforcement Learning application for MEC resource allocation**:

  A RL algorithm is used to allocate network resources within the orchestration module of the simulator, the goal of the algorithm is to decrease latency whilst enhancing efficiency and reduce task failure rate due to network usage. Numerous factors are taken into consideration such as ensuring that offloading mechanisms are utilising the available networks to an optimal degree and ensuring that task failure rates are reduced to the lowest amount possible. The algorithm is compared with traditional algorithms such as Round Robin as well as Greedy and results are produced and evaluated. The initial algorithm greatly enhances allocation

efficiency of MEC networks, ensuring that networks can offer improved service levels with lower task failure rates.

- **Multi-layer Reinforcement Learning application within MEC infrastructure:**
  A multi-layer algorithm is designed from the initial algorithm, considering various stages of network infrastructure as well as previously generated Q tables and replay memory, the algorithm is then enhanced with replay memory and approximate Q function embedded within the neural network.

In addition to academic contributions, the outlined research has significant non-academic impacts.

Firstly, the focus on refining and enhancing edge computation allocation within the industry aligns with the development of 5G solutions, leading to improved industry standards and solutions.

Secondly, advancements in communication technology resulting from this research benefit various sectors and contribute to technological progress.

Thirdly, enabling a multitude of XaaS (Anything as a Service) offerings by offloading computational power enhances the capabilities of mobile user devices, promoting performance and longevity.

Lastly, allowing IoT devices to preserve precious battery life by offloading computational tasks that adhere to both Quality of Service (QoS) and differentiated service models supports a wide range of services from smart cities to video streaming for end users.

1.4 Thesis Structure

The description below outlines the overall structure of the thesis and the purpose of each chapter:

1.5 Summary

This chapter provides an introduction of the topic and direction of the thesis, covering the motivations and goals behind the research and presenting the methodology that was used to cover the concept of MEC. An outline of the thesis is also present that visualises how the thesis will proceed henceforth.

Chapter 2 Literature Review

2.1 Concept

Numerous gaps were found within the literature, indicated by academics currently in the field as well as several articles and thesis written at the time[13], [14].

The challenges presented below had not yet been addressed from the time of discovery and even now, to the point of submission, research in the area is lacking despite recent efforts to produce simulations, albeit their limited nature. Gaps in the literature included but were not limited to:

- Lack of solidified specifications.
- The secretive nature of industry due to the emerging nature of the technology which can hinder innovation due to lack of collaboration.
- Lack of defined goals – MEC.
- Lack of industrial examples.
- Lack of resource scheduling techniques.
- Backwards compatibility needs to be investigated, to ease the transition of older network type to 5G.
- Broader definition of consumer-based services that will be available with the integration of MEC (what does it offer consumers).
- Security concerns over cross-network architecture.
- Conformation of services provided to local government bodies.

Outside of academia, there are also concerns with industrial implementation and one of the greatest factors is cost. Despite the numerous advantages of MEC, there is a great amount of risk present to Mobile Network Operators (MNOs). On the other hand, poorly planned infrastructure with retrospective upgrades can significantly increase costs in any industry [15]. Contextual awareness and current events indicate that because of the pandemic, more network traffic than ever before has increased the demand for future-ready

infrastructure[16], which can be observed in Figure 2.1 displaying a clear upward trend in network data usage.

In this context, optimisation of the networking infrastructure would not be limited to the software elements, as it is also possible to explore how optimisation techniques can lead to optimal placement of RATs to ensure that coverage is optimal, particularly where large language models can be integrated to assist in understanding the changes within society and pre-emptively predict the placement of infrastructure.

Figure 2.1. Global mobile network data traffic and growth

As the implementation on the technology is still in its infancy, the issues are critical, but many of them stem from previous generations of communication that must be tackled. Thankfully, experience from previous generations gives us foresight into some of the obstacles that must be acknowledged and resolved before a standard can be fully realized despite current efforts by the standardisation group ETSI [4].

With the expected launch of the first iteration of 5G in late 2018/early 2019 along with the introduction of release 16 (3GPP), major corporations (Huawei, Vodafone, Nokia, General Electric etc) with stakes in MEC technology began developing proof of concept and technical specifications available for both network providers and content developers to ensure timely release of services to recover the research and development efforts and investments of the technology as soon as possible.

Some efforts have already been realized [17] within testing environments however, due to the upgrades that are promised with 5G, this technology is expected to be a large contributor to the future of networking, where devices can be untethered and lightweight, integrating only the use of a mobile networking card and subscription to a service.

This approach will also split the market again, to devices that are extremely cheap to buy however require a subscription that can vary depending on the needs of a user allowing for larger varieties of products to be offered.

Industry standards must also conform to end user requirements as well as developers and service providers creating an Integrated Communications Environment (**ICE**).

Thus, to future proof this research and maintain its validity across both industry and academia, following a set of best practises across disciplines is necessary and pre-emptive research is necessary for achieving such a task. Observing design principles, such as those outlined in Figure 2.2.

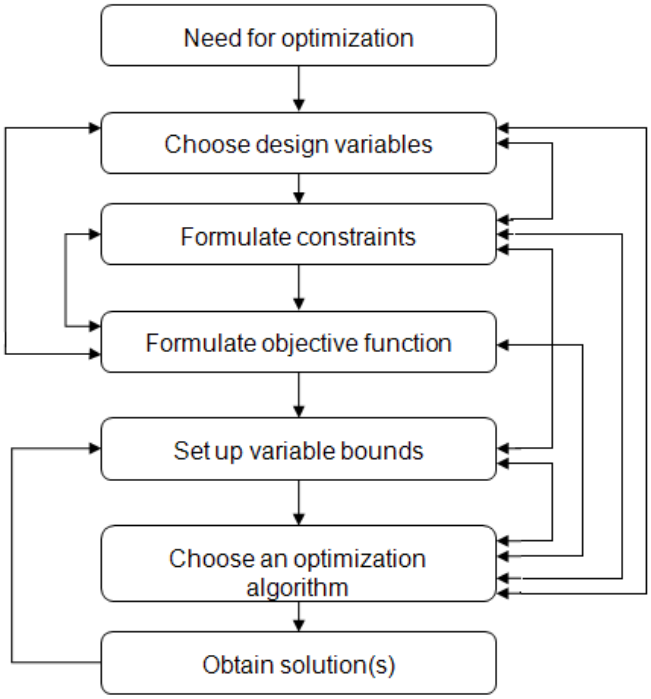


Figure 2.2. Algorithm creation flowchart

2.2 Literature identification

Ensuring selection of appropriate literature was crucial in the success of the project therefore, a method was devised and employed to ensure validity and impact of the existing literature, helping in efficient identification of appropriate sources, and helping to determine research questions within scope of the project by ensuring that requirements are met and procedures followed such as algorithm creation as observed in Figure 2.2.

In this case, the Chambers Dictionary is used to define research as: a careful search; a systematic investigation towards increasing the sum of knowledge in a particular field of interest. With systematic being the key term here, I hoped to use my accumulated knowledge and experience within academia and industry to ensure that my approach was extensible, so that I could update the thesis over the duration of my PhD without formulating conclusions that were subject to change, but also ensure that the journey of the technology was accurately recorded over the course of the research to give us a better of how technologies mature with implementation, analogous to the construct of a large

building site, I was sure that despite the meticulous planning involved, some retrospective changes would have to be made.

Due to the extensive amount of literature related to the research topic, it was important to find an effective method to filter the information required to ensure relevancy as well as direction. The approach used to uncover the most relevant information for this research was based on the “subject pearl growing” [18] style which identifies subjects and keywords within an electronic database including citations within the selected piece. This method identifies all relevant literature that was used to inspire and create the key literature found to be most valid during any research, leading to an enhanced understanding of the subject. The literature was then sorted based on relevancy and impact within the field, to ensure validity.

The databases used were IEEE, Google Scholar, SAGE, Emerald Fulltext, Elsevier, EBSCOhost and JSTOR with a focus on IEEE as the main source of information regarding academic writing on the subject. Additional sources utilised for discovery and analysis of key literature were online industrial newsletters, websites containing insight from industry insiders and academic blogs.

A Systematic review process was used, implementing colour coding for various aspects of a text and their contribution towards my research, by determining the sections that extracts of the text related most to therefore, eliminating the need for re-visiting previously read articles and having the ability to identify key elements of a text with greater efficiency and speed.

To ensure up-to-date knowledge of any advances in the field, notifications were set upon the IEEE mobile application with relevant keywords which were also amended over the duration of the thesis with the formulation of new technologies and services that accompanied MEC. It was vital to ensure that changes in specifications and revisions as well as any new academic sources were incorporated. To this end, key words, found in TABLE 2.1 were recorded

from the establishment of the thesis until the completion of the research, keeping up with requirements of validity and reliability of the thesis and contribution towards academia and industry. Some other assistive technologies included within the composition of this research were Mendeley Desktop and Web importer for storing academic sources and referencing purposes and versioning control, cross-referencing and bibliography implemented with MS Word.

TABLE 2.1
KEY SEARCH TERMS

Key Search Terms	Source	From Date	To Date	Ongoing Research Status
Fog	IEEE, Google Scholar, JSTOR	21/10/18	1/3/19	Discontinued
Smart Cities	IEEE, Google Scholar, JSTOR	21/10/18	1/3/19	Discontinued
IoT	IEEE, Google Scholar, JSTOR	21/10/18	1/3/19	Discontinued
(Offloaded)Computing	IEEE, Google Scholar, JSTOR	21/10/18	Present	Active
Cloud	IEEE, Google Scholar, JSTOR	21/10/18	1/3/19	Temporarily suspended
Computation	IEEE, Google Scholar, JSTOR	18/11/18	1/3/19	Temporarily suspended
(Cache) Allocation	IEEE, Google Scholar, JSTOR	18/11/18	Present	Active
SDN	IEEE, Google Scholar, JSTOR	18/11/18	Present	Active
NFV	IEEE, Google Scholar, JSTOR	18/11/18	Present	Active
Edge	IEEE, Google Scholar, JSTOR	2/1/18	Present	Active
EC	IEEE, Google Scholar, JSTOR	2/1/18	Present	Active
MEC	IEEE, Google Scholar, JSTOR	2/1/18	Present	Active
5G	IEEE, Google Scholar, JSTOR	2/1/18	Present	Active
Orchestration	IEEE, Google Scholar, JSTOR	3/6/18	Present	Active

From the conception of this research, as with any venture in cutting-edge technologies I am sure, the solutions proposed by researchers have mutated and evolved to consider their peers' solution offerings and combining them to present the best solution. This section aims to select those key pieces of literature that contributed heavily to the final formulation of this research.

We explore the most important works' available in this subject area and analyse them deeply to give us a better understanding of how to approach the problems this research is aimed at tackling.

Key literature was identified with the table above, those that contained 3 or more of the key search terms were marked as 'Highly Valuable' and scrutinised closely, each piece of literature and source was analysed and graded from 1 (considerable value) – 5 (highly valuable) indicating its value for the research. The concept of subject pearl growing was then applied to those that were found highly valuable to ensure that any contributing literature/references and subsequently concepts were not neglected. This approach ensured validity of the research and reliability, as it assessed each piece of literature individually and attempted to learn from those found highly valuable by assessing the source and direction of the researcher as well as any influence they may have come across over the course of their research.

Virtualisation efforts were first introduced in computing architecture to provide scalability and intelligence, supporting a more agile platform. It has since grown to be an integral part of OS and dedicated platform providers alike [19] reporting figures such as 73% reductions in time spent on routine administrative tasks within IT. The ability to automate is one of the key offerings of virtualisation but businesses often employ virtualised technologies to 'dramatically lower costs' [20]. VMware, a key operator in this industry, claims that 92% of executives plan to increase their virtualisation efforts to benefit from the added security, agility, and implementation of computational tasks within businesses. The benefits of virtualisation, however, extend far beyond business applications as we will discuss below.

Orchestration, in its technical application, refers to effective management of resources using a dedicated resource to handle incoming and outgoing requests to applications, network slices etc. Using an orchestration module (software based), dedicated resources can be utilised to ensure that requests made by an application are being handled in the most optimised way, with some level of QoS, thanks to optimisation algorithms that can be implemented as well as integrated with ML. Academic efforts to implement optimised orchestration within virtualised applications across various sectors have exponentially increased in popularity, thus portraying the rapid development prospects of virtualisation techniques [21], [10], [22], [23].

2.3 Edge Computing, Mobile Edge Computing and Cloud Computing

Despite its introduction in several forms over the previous few years, EC and MEC has persisted as a concept that has the potential to be a fundamental cog in the communications architecture. The technology is diverse in its application, but it is safe to say that the core concepts have remained largely the same and loosely encompass Distributed Computing.

The idea is to disperse larger-scale computation closer to where it is required, i.e., closer to user entities, to enable applications to offload computational tasks and enable a plethora of applications that would normally not be feasible with local computation, i.e. the EU device. In turn, support for larger-scale cloudlet computing becomes a reality for latency-sensitive applications and settings.

It offers the ability to vend computational services, data storage and other hardware needs of individuals and enterprises. Like any business, the cost of the hardware and implementation of the architecture is eventually planned to be offset by the number of customers that are available for service provider. As the product grows in service offerings and coverage, the customer base increases, eventually turning the loss made by network providers on implementation into a profit accumulated from thousands of subscribers.

Figure 2.3 can help visualise the focus in the subject area where the number of publications, particularly those containing the keywords 'Mobile Edge Computing', are weighed against the publication's year of the academic journal article, and we observe that they are exponentially increasing year by year.

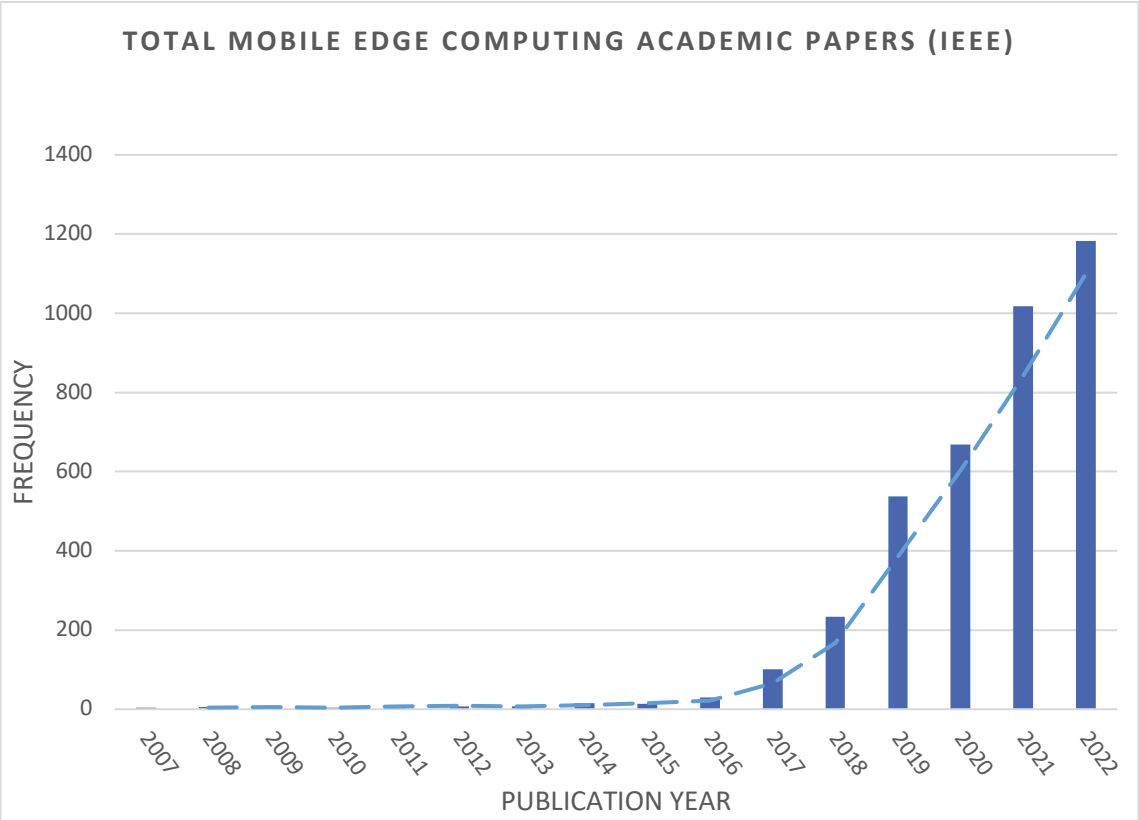


Figure 2.3. Plot of Mobile Edge Computing journal papers

The prospects of the technology were discovered to be of great importance due to the explosive growth of cloud technology, steadily becoming one of the dominating factors in modern technology today.

Corporate giants like Google, Amazon and Microsoft started to focus on '***x*** as a Service' technologies that would allow greater monetisation of services and product offerings but also prevent piracy, offering legitimate consumers constant updates and everlasting marketing from the company said software was purchased from. The success and demand for larger cloud provider services then branched into a range of products such as computation, storage, and networking.

According to [24], Edge Computing is an umbrella term used to refer to a family of relevant technologies such as Cloudlet, Mobile Cloud Computing, Fog Computing and Multi-Access Edge Computing, synonymous with Mobile Edge Computing and mobile cloud systems (aka MEC)[25].

In addition, a concept called Industrial Edge Computing (IEC) also exists offering a central solution for devices to work in tandem and improve inter-communication[26]. Each of the technologies are based on the engineering principle that computational resources are better located on the edge where demand is physically present, and network traffic can be circumvented [24].

The authors of [27] state that Edge Computing is the control and management of a standalone end-point device individually or through a set of software functions in the fog domain. Bearing in mind that each of the technologies produce their own set of advantages and are distinctly different in application. When discussing vertical scalability, Cloud Computing can provide access from virtually anywhere providing that the end-user has an internet connection.

Our concern with this research is on the applications that require low-latency support that Cloud Computing does not offer due to the variable hops that are required to reach the centralised server.

Cloud computing offered a host of advantages and benefits, especially for smaller corporations that required larger computational tasks that they may have lacked the budget to host in-house paving the way for SaaS solutions [28]. Despite the abundance of cloud computing services available, there were a few issues present that limited the applications that could be offered to users.

Network congestion

- Most IoT devices function from a sensor, actuator mechanisms where they are expected to make actionable decisions in near real-time for some applications, severely limited by the number of hops that data must traverse[29]

Latency

- A great limiting factor to real-time applications, where human responses are measured and required such as cloud-gaming services or remote operation of robots/vehicles.

Performance

- Reliance on the network can be severely restrict the QoS provided by cloud application providers [29]

Vendor Lock-in

- Though the uptake of virtualisation within the network has become commonplace, proprietary technologies still exist which can limit flexibility and further increase costs.

Limited Resource Control

- Cloud computing often offers limited control over the underlying hardware and software which can introduce issues such as resource contention and noisy neighbour problems [29]

One of the mainstream applications of the technology was cloud gaming, introduced back in 2004 within Japan, even though average speeds were 861 kbps at the time [30].

Naturally, the lack of a steady connection and preposterous amounts of latency meant that the movement failed at the time, yet the industry persisted with several attempts at cloud gaming up until the very present time with companies like Google offering products like Google Stadia. Currently, efforts persist within the Cloud gaming industry including offerings such as GeForce Now by GPU manufacturer Nvidia and Boosteroid.

The difference between conventional cloud computing and the, then radical, edge computing concepts were the fact that conventional cloud computation occurs when a client requests a list of VMs to be created within a datacentre which are then assigned numerous tasks to run. The tasks themselves are considered as a bunch of operations, subsequently keeping the VMs busy for

hours, making the flow of operation like renting VMs for a limited time until a certain task has been completed.

The most notable difference is that cloud computing utilises the resources a lot quicker and within shorter bursts of time akin to service providers in mobile networks using OFDM on a receiving end of the transmitter [24], users also had access to computational power far beyond that of which was cost-efficient locally, giving vendors viable market opportunity.

To tackle some of the limitations mentioned above, researchers deployed various systems and methodologies for traditional cloud systems including middleware solutions such as Hadoop, Apache HAMA, StorkCloud, etc. [25].

The demand for Edge Computation emerged with the increase in popularity of low-latency applications [13]. Service providers then began looking towards implementation within their frameworks and architectures to meet user requirements.

Applications of the technology vary in use but used in conjunction with the correct infrastructure, near real-time processing of information as well as inter-communication and limited local awareness are some of the possibilities on offer. The industry was made aware of applications that would require rapid communication with edge nodes for devices that were limited in computational power.

These devices could then make decisions on how to prioritise local data computation against data which could be offloaded to the nearest edge node which could provide extended computational power, autonomous vehicles are said to be one of the use cases in this scenario.

Services like Netflix then began utilising Edge Computing on a smaller scale, using local caching to serve metropolitan areas of larger scales with their own dedicated servers offering the capability of storing and retaining product

offerings (in this case movies/shows) that a particular locale may be interested in. Introduction of a different multiplexing method was expected; however, some small changes were also predicted/requested from the RAT. Namely, the inclusion of edge servers within micro base stations to offer services for a small cell of users. Some of these future use cases can be found in TABLE 2.2:

TABLE 2.2
EDGE USE CASES

Use Case	Requirements
Autonomous Vehicles	Self-driving vehicles need the ability to communicate with servers without traversing many hops along the network to provide real-time processing of data.
Industrial Automation	Edge computation offers the ability to create machines that can sense, detect, and learn things without having to be programmed.
Augmented Reality and Virtual Reality	Can be used to train employees, help those less able and visualise new concepts.
Retail	Attract customers using technological implementation of AR/VR/MR technologies.
Connected Homes and Offices	Due to the centralised nature of the smart home systems, tasks take some seconds to happen as opposed to utilising edge computing where tasks would be in near real-time.
Predictive Maintenance	Edge computing can help detect machine faults within an industrial setting, allowing maintenance of expensive machinery circumventing repair and replacement.
Video Monitoring	Video cameras can collect a vast amount of data within a very short amount of time, especially when you consider facial monitoring and motion detection which requires further computation and data storage.
Software Defined Networking	Even SDN requires local computation to determine best routing path, as the decisions are made on a Virtual Machine that hosts the SDN controller, computing abilities are a requirement
Blockchain	Blockchain technology requires ledgers to be stored within each device that can host the ledger. For the blockchain application to function, computation is required.
Fog Computing	Fog computing uses edge devices that connect to a distributed computing model, these models can harness underused cycles across the edge and then continue to the cloud (zdnet edge computing uses).

Each of the applications rely on low latency and transfer of high data rates, paving the way for further opportunities and enhanced applications. Further use cases involve situations where devices in a particular locale are contributing towards the same goals, and an edge orchestrator can signal each connected device to process a chunk of information which can then be consolidated.

Use cases such as these are particularly helpful for academic efforts where a group can leverage the combined processing power of all connected devices, including the edge node, to achieve a set of objectives or simulate an experiment with greater efficiency than when assigning a singular device.

The concept itself is recognised as one of the key emerging technologies alongside NFV and SDN but can utilise both to offer enhanced services, aiding to advance the mobile broadband network into a programmable world and contributing to satisfying the demanding requirements of 5G in terms of expected throughput, latency, scalability and automation [13].

MEC itself is based heavily on NFV, the primary difference however, is the fact that the MEC framework enables applications to run at the edge of the network whereas NFV is focused on network functions. Thus, each concept can be used synonymously to aid and abet the other within the infrastructure subsidising both OpEx and CapEx for organisations.

The introduction of MEC opens services to end-users as well as adjacent industries giving them the ability to deliver their mission-critical applications over a mobile network. Opening a host of fresh business opportunities and new use cases across multiple sectors.

Thus, standardising the technology before implementation will benefit both the industry and end consumers greatly as an open programmable interface will encourage co-operation between network providers further subsidising costs and ensuring that a vast number of customers of mobile operators can be

served [13]. Use cases for Edge can branch in several sectors, in technical and business usage as well as others, some examples can be found in Figure 2.4.

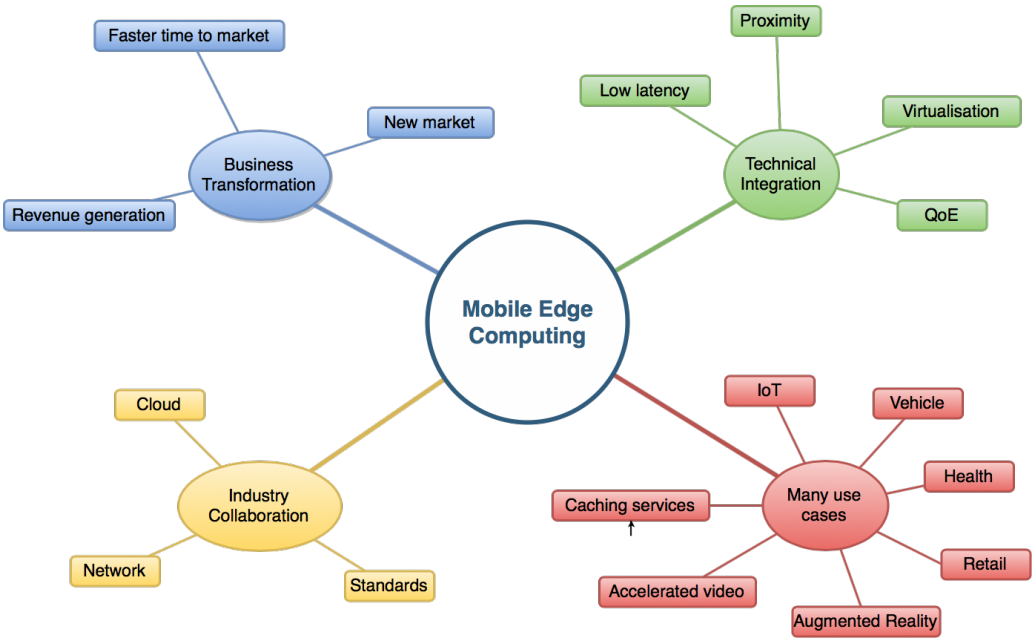


Figure 2.4. Web diagram of use cases in Edge Computing

2.3.1 MEC Infrastructure and Architecture

Taking into consideration that latency is one of the most crucial offerings of 5G, it is pivotal to ensure that MEC is correctly implemented within the network architecture to offer maximum benefit.

To this end, it is also important for the scope of this research to understand where MEC fits in within the infrastructure and the reasons therein.

As this research is dedicated to the implementation of MEC within a 5G environment, we will review some of the unique elements and attributes that a 5G Mobile Network Environment (MNE) will provide access to, allowing us to determine the difference in energy costs as well as impact on performance that a 5G MNE will bring.

2.3.2 Specifications in MEC

Notable key literature that required analysis was the official ETSI documentation on Mobile Edge Computing, formulated because of numerous conferences and found on etsi.org. The material was split, referring primarily to Multi-access Edge Computing (MEC) use cases and requirements, frameworks, and general principles for MEC Service APIs [31], [32].

Introduced to ETSI in February 2016, a portion of the ETSI website is dedicated to what ETSI refers to as Multi-Access Edge Computing, effectively incorporating the use of both RAN and WANs. The concept is defined as offering application developers and content providers cloud-computing capabilities and IT service environments at the edge of the network where the environments are characterized by ultra-low-latency, due to being located at the network edge, and high bandwidth as well as real-time access to radio network information and services towards mobile subscribers, enterprises, and vertical segments.

This section will highlight the standardisation efforts made by ETSI and 3GPP, we will then cross-reference the changes with Key Literature found within this chapter to make observations. Indeed, we must expect that some of the literature identified over the course of the research itself will inevitably help to shape the standards set out by both ETSI and 3GPP. Respectively, I will also attempt to critically de-construct the arguments as well as implementations set out by any identified key literature over the course of this research. To this end, I will begin with the deconstruction of the ETSI use cases and requirements [33] as we can safely assume that they will lay out the foundations and frameworks for any standards that are established.

Dissemination of the specifications of MEC will also allow us to further validate our simulation software to ensure the expectations of standards are met, further validating our research. The specifications define the architectural framework, application enablement framework and platform components for

MEC deployments which in turn help outline the key functional components, their interactions, and the standardised interfaces and APIs that enable communication between MEC mechanisms for security, resource management, and interoperability in MEC environments.

The key components tackled within the specification are as follow:

- ETSI MEC Architecture: ETSI has defined a high-level MEC architecture that outlines the main functional components and their interactions. This architecture provides a standardized framework for implementing MEC solutions and enables interoperability among different vendors.
- MEC Application Enablement Framework: ETSI specifies a framework for developing MEC applications, which are software applications that can leverage the MEC infrastructure. This framework defines interfaces and APIs for communication between applications and the underlying MEC platform.
- MEC Platform: ETSI defines the MEC platform as the infrastructure that hosts MEC applications. It includes various components such as MEC hosts, MEC services, and MEC system management functions. The MEC platform provides compute, storage, and networking capabilities at the edge of the network.
- MEC Service APIs: ETSI has standardized a set of APIs that enable MEC applications to interact with the MEC platform. These APIs cover functions such as location, mobility management, radio network information, context information, and media services. By using these APIs, MEC applications can access real-time network and context information, enabling them to make intelligent decisions and deliver low-latency services.
- MEC Security: ETSI specifications address the security aspects of MEC deployments. They define security mechanisms and guidelines for protecting MEC infrastructure and applications. This includes authentication and authorization mechanisms, secure communication protocols, and threat mitigation strategies.
- MEC Resource Management: ETSI provides specifications for managing the resources in the MEC environment. This includes resource

discovery, allocation, and optimization mechanisms to efficiently utilize the available compute, storage, and network resources. It also covers aspects like load balancing, scaling, and lifecycle management of MEC applications.

- Interoperability and Standardization: ETSI emphasizes interoperability and standardization in MEC deployments. By defining common interfaces, protocols, and data models, ETSI enables different MEC components and solutions from multiple vendors to work together seamlessly. This promotes a vibrant MEC ecosystem and facilitates the development of innovative edge applications.

For this research, we want to ensure that our simulation environments as well as any experimentation conducted is as true as possible to the expectations of ETSI, we will have to bear in mind that the novel nature of the research alongside the young, yet growing interest of the subject matter may result in some requirements not being met.

In true ETSI fashion, specification sheets are continuously evolving and branching out to include other subdomains of the technology. To ensure that our simulator continued to meet the requirements as outlined by ETSI, we will continue to monitor these changes, and this chapter will be updated to include them.

After exploration of current requirements expected by the technology, early implementations, either in simulation form or conceptual stages of physical form were located on academic sites. Below, we will attempt to summarise some of the most prominent features of the specifications released by ETSI for MEC.

ETSI GS MEC 002 (2018-10) Use Cases and Requirements

The standards defined by ETSI in the aforementioned documents were considered during the formulation of methods and acquisition of results portrayed in this thesis. The goal was to ensure that any methods devised

conformed to industry standards and academic standards to enhance contribution.

ETSI GS MEC 002 contains a variety of use cases, outlining some of the finer details and potential products that will result from Mobile Edge Computing. Numerous key figures are presented detailing subscriber-based routing as well as video content delivery. ETSI's vision differs from solutions currently in place for services such as Netflix and Amazon by taking into consideration the 5GCoreConnect.

Several future developments will be enabled/enhanced with the inclusion of Edge Computing, including but not limited to; Autonomous Vehicles, Large Sensor Networks, VR, AR, MR, Connected Cities, Edge-node caching etc. [1]. The main benefit of Edge Computing is the fact that information can be both centralized and stored locally, mechanisms for preliminary processing of information can be implemented within the edge node, filtering unnecessary information provided by sensors and only synchronised at scheduled periods. This removes the redundancy of having constant communication with a server many hops away in turn decreasing latency.

This also decreases the amount of unnecessary information to be processed before actuators can be given instructions on how to react after the information provided by the sensors has been processed.

This approach is of particular interest to companies like General Electrical, who have large scale sensor networks within their energy grids which must be carefully monitored, response times to emergencies and accidents is significantly decreased as the information can be monitored through a central location, but an edge node can act according to information within a shorter period of time, mitigating risks [2].

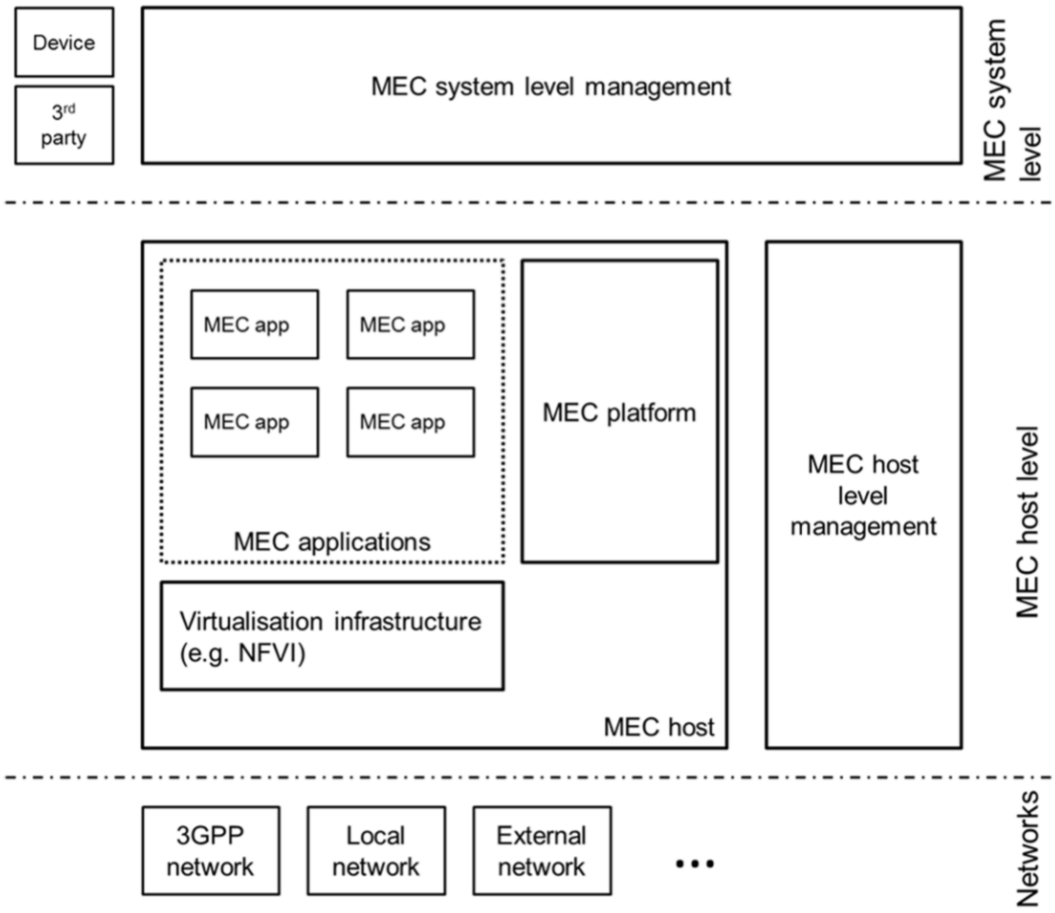


Figure 2.5. MEC Framework architecture

Framework standards mentioned in ETSI MEC 003 [34] outline some of the basic requirements regarding the expected framework of EC integration within a variety of networks including wired and RAN. In a nutshell the MEC host is expected to operate at a layer above the underlying network and the first port of call for an end device will be to communicate with MEC system level management before being directed towards the MEC host level.

Figure 2.5 portrays the basic architectural requirements of an MEC implementation. The MEC Host Level layer contains a host and its own orchestrator, which can be either a physical or virtual entity to provide computing and storage resources at the edge of the network. MEC applications can be deployed and executed at the host level, using the available resources

to deliver EC services. Its primary purpose is to provide management and utilisation of resources at its own individual level.

On the other hand, the MEC system level refers to the overall MEC infrastructure that encompasses multiple hosts. As observed in Figure 2.6, the architecture utilises elements of existing VNF and VMs to host several applications on integrated or embedded CPUs and other computational hardware to host several applications which can then be used to serve users. In this case, the system level management is used to allocate resources and direct users to appropriate services based on requirements and needs.

A variant architecture for MEC is also included which further elaborates on functions expected to run on an MEC device, incorporating the use of Orchestrators in both MEC applications and VNFs to enhance the utilisation of resources and cater to user service requirements more comprehensively.

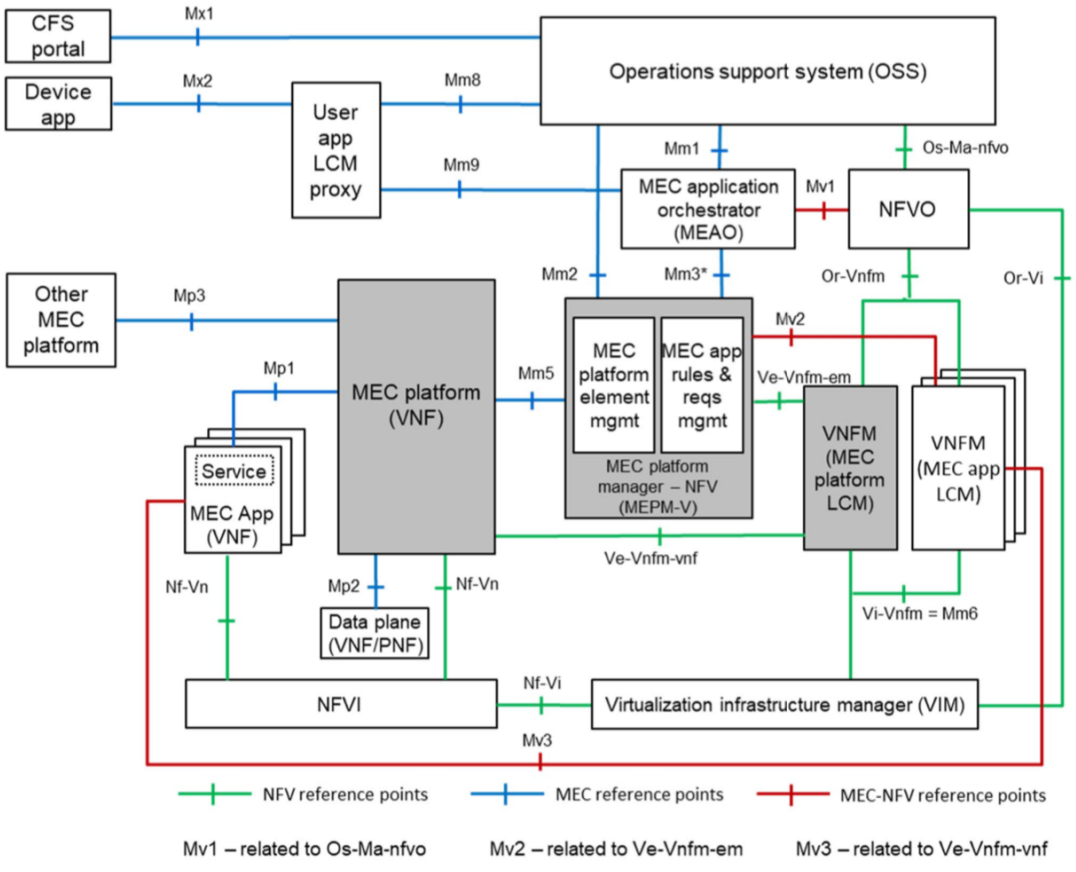


Figure 2.6. VNF Architecture (as proposed by ETSI)

Figure 2.6 also portrays the underlying functions expected to appear in MEC including color-coded relationships, it also proposes an architecture, seen in TABLE 2.3 [34], where the multi-access edge system will be deployed within an NFV environment as a VNF [34]. To implement this architecture, several orchestrator functions are strategically placed throughout the framework. A User is referred to the operations support system (OSS), the request is then parsed through the MEC application orchestrator (MEAO) and the NFVO.

TABLE 2.3
MEC COMPONENTS

Element	Definition
MEC Host	Contains the MEC platform and virtualisation infrastructure, provides compute, storage, and network resources for the MEC applications. Includes a data plane that executes traffic rules received by the MEC platform and routes traffic among applications and various network functions.
MEC Platform	Responsible for offering an environment where MEC applications can discover, advertise, consume, and offer MEC services. Receives traffic rules from the MEC platform manager, applications, and services, subsequently instructing the data plane to act on the provided instructions.
MEC Application	Run as virtual machines atop the virtualisation infrastructure provided by the MEC host, offering the ability to interact with MEC platform to consume and provide MEC services. MEC applications can also interact with MEC platform in some cases to provide support procedures related to application lifecycles.
MAEO	Multi-access edge orchestrator provides core functionality of the MEC system level management. It is responsible for maintaining an overall view of the MEC system, providing a platform for application packages, selecting appropriate MEC host(s) for application instantiation based no hardware constraints and triggering applications.
OSS	Operating Support System receives requests via the CFS portal and device applications and acts according to instructions. Can also receive and act on requests for relocation of applications between external clouds and MEC systems.
User application lifecycle management proxy	A UA is an MEC application instantiated in the MEC system to respond to a user request via application running on a user device.

ETSI GS MEC 009 V2.1.1 (2019-01) General principles for MEC Service APIs

The general principles conference paper, ETSI GS MEC 009, delves into further detail regarding communication protocols and other guidelines industry and developers should follow to meet the requirements of standardisation as defined by ETSI. The standards share many similarities with existing TCP framework between Representational State Transfer (REST) clients and REST servers. Communication of API is handled using HTTP protocols and methods, utilising HTTP headers and simplifying communication between server and client.

An important point to be noted here is that subscription types and statuses can be updated and posted. This also assists in centralization as a component in a distributed system must keep all involved components informed of any changes of state within a component at any given time.

All HTTP functionality is not necessarily included, rather a subset is used for simplified commands such as the POST method, 204 NO CONTENT method etc. Lists are represented in JavaScript Object Notation (JSON).

Further detail is included on how naming conventions should be handled as well as error codes however, in this research this will not be explored in depth. It will be touched upon further in the research to ensure that any development work conducted conforms to industry expectations to future-proof the research rather than backdating to meet requirements and standards for any further work conducted.

ETSI GS MEC 003 V2.2.1 (2020-12)

The final version of the reference architecture and framework explored for this research was the most recent iteration of the framework. The expected model of the framework itself largely retains the same expectations where Figure 2.5 portrays the structure. The framework is grouped into system level, host level and network level entities. New clauses introduced to the document highlight

the addition of new architectural assumptions; the MEC is deployed as a VNF, MEC applications appear as VNFs towards the ETSI NFV MANO components, the Virtualisation infrastructure is deployed as an NFVI and is managed by a VIM as defined by ETSI GS NFV 002, the MEC Platform Manager (MEPM) is replaced by a MEC Platform Manager.

NFV (MEPM-V) that delegates the VNF lifecycle management to one or more VNF managers (VNFM), and the MEC Orchestrator (MEO) is replaced by a MEC Application Orchestrator (MEAO) that relies on the NFV Orchestrator (NFVO) for resources orchestration and for orchestration of the set of MEC application VNFs as one or more NFV Network Services (NSs).

In summary, the MEC functionality is expected to be entirely virtualised is nature, including the orchestration methods however, they are now further divided in sub-components to ease organisation and implementation.

2.4 SDN and NFV

2.4.1 SDN

Originating from OpenFlow, SDN was introduced as a framework or a set of solutions for enhancing networking speeds as well as offering a standardised method of future-proofing networks for greater efficiency, programmability, centralisation, and open-standards [35].

Explosive growth in the use of online services and net based application has exponentially increased network traffic and demands. Network operators need to focus on minimising costs whilst increasing efficiency. In addition, MNOs are also focusing on increased efforts to implement the technology and offload to non-dedicated hardware managed systems thus enabling virtual network functions such as firewalls and load balancers to be software oriented [36].

Unfortunately, legacy network architectures and their management tools were not designed to cope with such highly elastic demand which ultimately limits

the operator's ability to cost-effectively respond to the scale, performance, and user experience requirements of today's dynamic environments, or to roll out differentiated services.

It is a framework that reduces dependency on underlying hardware, offering separate control and data planes, which allows for greater control and manipulation of the routing protocols and all packet forwarding functions. This means that for hardware life cycles have been drastically improved, where clients can now purchase a service that provides constant updates to the firmware and software rather than purchasing new hardware after longer periods of time.

This allows for the rapid amplification in the time it takes to apply network upgrades, both for intranets and the internet. Companies can purchase solutions knowing that they will not be outdated in performance within a short time frame, which in turn allows for a greater range of services to be offered.

5G will be leveraging SDN technologies within the network backbone to maximize network performance and by doing so, it will enable many more services that the network providers can then offer to the user.

MEC will be using the extremely low latencies offered by 5G services to provide an unprecedented network experience that can be utilised by many different industries and sectors around the world including but not limited to, public and private health services, emergency services, disaster control, city-wide management services, local governments etc.

For the reasons previously mentioned, research is well under way to provide the best foundations for 5G communication and accelerate deployment. The following issues have yet to be tackled in the context of MEC and integration of Edge Computing within the wider network environment [7].

A useful distinction between the technologies can be found in Figure 2.7, indicating the different uses for each respective technology as well as the relationship between them[27] as well as the features they share.

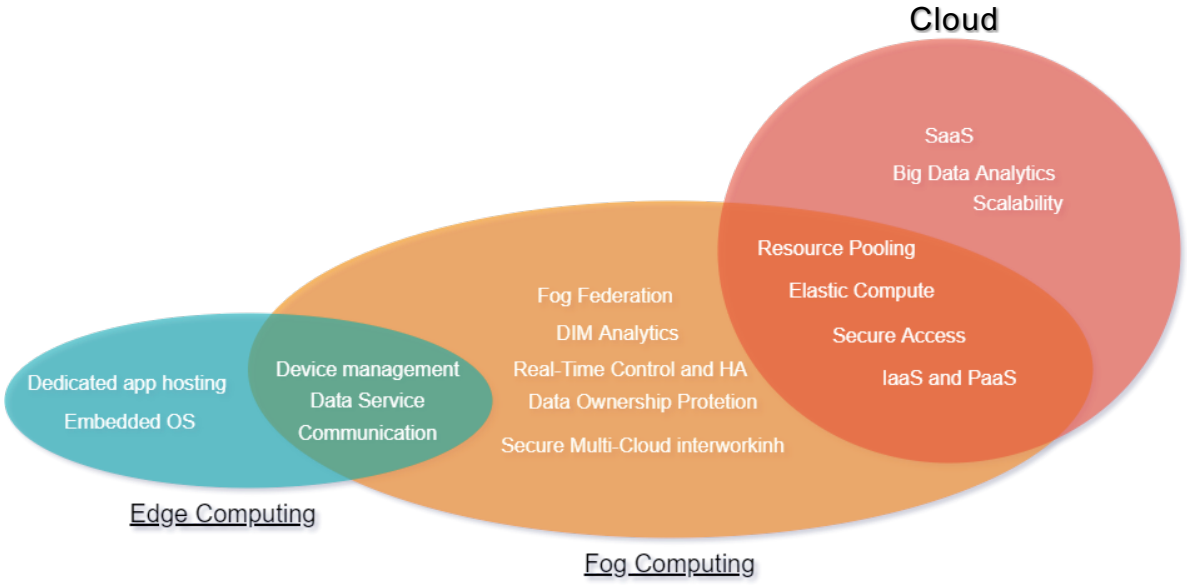


Figure 2.7. Features shared between Edge, Fog, and Cloud computing

2.4.2 NFV

NFV also plays a critical role in enabling said technologies, virtualising network functions brings greater control to network operators and industries alike, paving way for 5G networking and improved network management. Some of the features expected with the utilisation of NFV include agnostic network access, mobile edge computing, and 5G network slicing where grouped subscriber or machine-to-machine (M2M) and IoT devices are service by separate, virtualised core networks.

5G will merge IT and Cloud into mobile core networks methods for accessing subscriber information efficiently via the use of edge networks to reduce data latency and intelligently place network resources for reduced backhaul.

This will be achieved by placing subscriber profiles as close as possible to the user, thus making services and profiles almost as mobile as the subscriber and device [37]. Various key components function as one to enable these services such as established anchor points that can be defined for a specific network

slice of common subscriber's devices. Subsequently, anchor switches provide network traffic routing between SDN anchor points [37].

2.5 Reinforcement Learning and Deep Reinforcement Learning

As we attempt to explore the most effective solution to optimise resource allocation within an EC environment, it is important to understand the most effective techniques being utilised today.

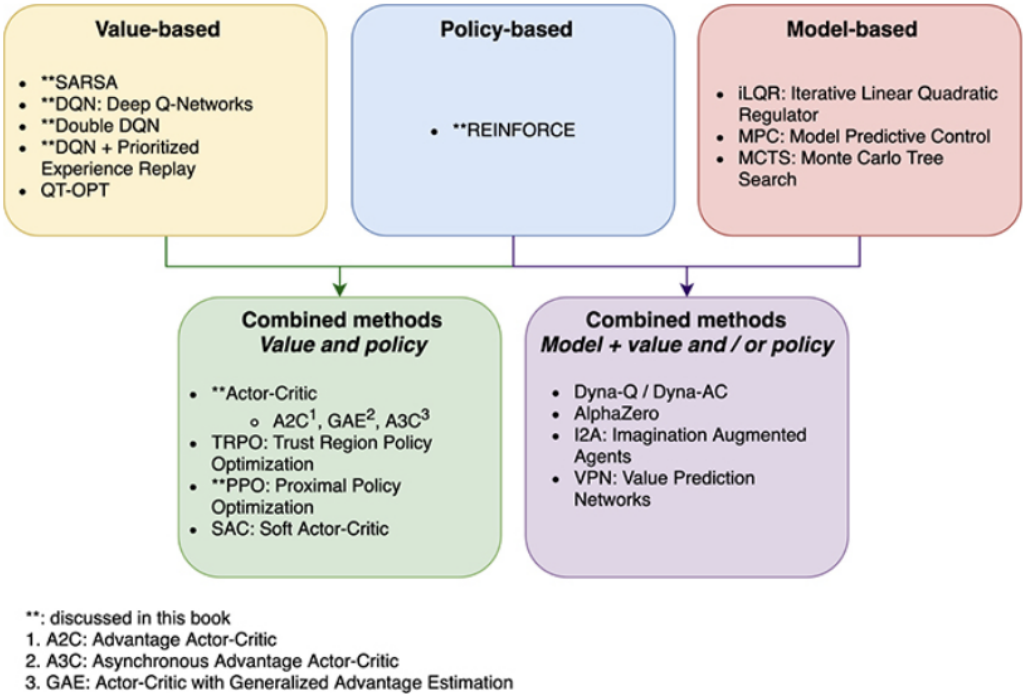


Figure 2.8. Deep reinforcement learning algorithm families

One such instance is RL, which provides software-defined agents the ability to learn the best actions possible in virtual environments to attain their goals. This is achieved by uniting function approximation and target optimisation, mapping state-action pairs to expected rewards and aiming to achieve the highest reward to the desired objective[38] [39].

In RL, a learning agent is not told which actions to take but instead, attempts to yield the highest reward, according to [40], 'trial-and-error search and delayed reward' are the two most important distinguishing features of reinforcement learning, more examples of which are depicted in Figure 2.8.

Reinforcement learning distinguishes itself from both, supervised and unsupervised learning as it does not conform to requiring a dataset of trained examples before the learning process begins (supervised learning), nor does it adhere to finding structure hidden in collections of unlabelled data (unsupervised learning).

Appropriate to the problem at hand, reinforcement learning tackles issues where a problem is interactive, and the environment and data is constantly evolving. This kind of learning appropriately fits the issues presented with ongoing optimisation solving and provides the algorithm with the tools necessary to continuously respond to changes within the environment and optimise for them [41].

The concept of Reinforcement Learning combines artificial neural networks with a reinforcement learning architecture to enable software-defined agents to learn the best approach to take in a virtual environment to achieve set goals [39].

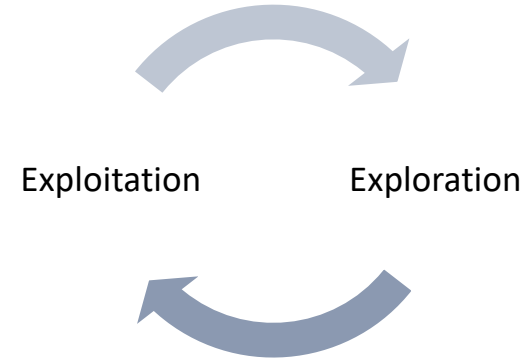


Figure 2.9. RL Trade-off

As observed in the popular Netflix documentary about achievements and breakthroughs in AI, neural networks can combine computer vision, machine translation and time series prediction whilst utilising reinforcement learning to enable algorithms that can achieve superhuman abilities and outperform even the best of human abilities in complex games like Go[42], where complex

decision-making skills and considering all future moves are vital to ensure victory.

RL algorithms start off with minimum configuration, essentially a blank state, and continue to develop the best approach, incorporating deep neural networks, by using a reward-based points system to encourage what is defined as a good decision and penalise what is defined as a bad one as shown in Figure 2.9.

According to [39], there are numerous industrial applications for the technology which are already being used in industrial operations and supply chains to optimise factories and warehouse logistics.

Arising from the human interaction with the physical world, RL can be analogised as an infant using trial and error to understand the implications of their actions within the natural world, using a learning approach to understand good practises within the environment, with no explicit teacher present to teach right from wrong [40].

Unfortunately, this does present a trade-off between exploration and exploitation, where the learning agent inevitably fails as it must explore a variety of approaches and actions before it can begin to favour those which yield the best reward.

It is emphasised in [40], that RL is different from ‘unsupervised learning’, which usually consists of finding structure hidden in collections of unlabelled data. RL instead, attempts to maximise the reward signal instead of attempting to find hidden structures, therefore one the primary drawbacks for RL is that it can take a significantly long period of time to train an algorithm from the beginning [43], take for example the case of learning an Atari game, where an RL agent can take the equivalent of weeks of playing to reach the performance of a human counterpart that has played only 15 minutes. Naturally, in the context of this research.

Any undue delays within formulating the best approach in resource allocation would be highly unwelcome as efficiency and speed are highly important factors when optimisation is an objective.

2.5.1 Machine Learning

Considered a sub-field of AI, Machine Learning (ML) is one of the leading types of AI, it is designed around the question of how to develop software agents that improve automatically with experience, enabling more accurate prediction of results without specific programming and is broken down into the following categories found in TABLE 2.4 [44]:

TABLE 2.4
THE THREE TYPES OF MACHINE LEARNING

Type of ML	Description
Supervised Learning	Learns from a set of training data of labelled examples provided by a domain expert who has the role of an external supervisor in the learning process. Learns by using the provided data set to generalise responses to cases not included in the training set.
Unsupervised Learning	Learns by finding hidden patterns and knowledge in a dataset without an external supervisory application present.
Reinforcement Learning	Learns by using guided methods defined by a specific objective that the application requires to meet an end. It's analogous to how a child learns using a trial-and-error method by using their observational skills by attempting and subsequently observing the consequences of each action. A grading/points-based system is then used to score a positive/negative action for subsequent responses.

Efforts in Machine Learning have increased significantly, particularly in infrastructure and back-end implementations such as MEC, SDN and NFV [45]. Some common algorithms that define a prediction model are Decision tree,

Neural networks, Gaussian Processes, Hidden Markov models, Dynamic Bayesian network, etc. that are discussed within the literature review sections of this research.

2.5.2 Bellman Equation

The Bellman equation, Eq. (2.1), is a fundamental concept in the field of dynamic programming and reinforcement learning. It provides a recursive relationship for calculating the optimal value function in a Markov decision process (MDP).

The Bellman equation captures the principle of optimality, stating that the optimal value of a state can be expressed in terms of the optimal values of its successor states. By iteratively applying the Bellman equation to all states in the MDP, the optimal value function can be computed.

$$V(s) = \max_{a}(R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s') \qquad (2.1)$$

TABLE 2.5
DENOTATIONS FOR BELLMAN EQUATION

Denotation	Description
$\boldsymbol{V(s)}$	The value of being in a particular state
$\boldsymbol{s}$	A particular state
$\boldsymbol{a}$	An action
$\boldsymbol{s'}$	Next state (from s)
$\boldsymbol{\gamma}$	Discount factor
$\boldsymbol{R(s,a)}$	Reward function

The Bellman equation is a foundational concept used in various algorithms, such as value iteration and policy iteration, to solve MDPs and determine the optimal policy. It provides a mathematical framework for reasoning about

decision-making problems under uncertainty and forms the basis for many reinforcement learning algorithms.

2.5.3 Markov Decision Process

Reinforcement learning algorithms are typically modelled as Markov Decision Process (MDP), a mathematical framework based off the Markov Chain, for describing stochastic control processes.

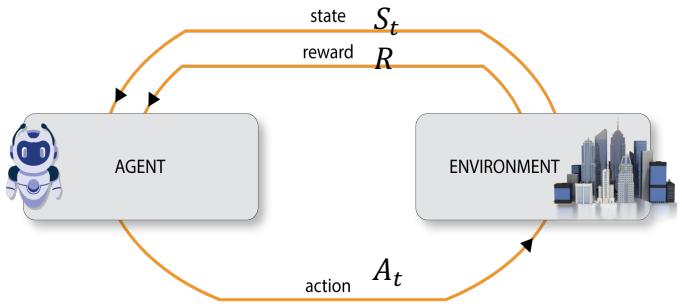


Figure 2.10. The Markov Decision Process

The goal, as outlined in Figure 2.10. Mathematically, a Markov decision process can be defined as the 4-tuple $M = \langle S, A, P, R \rangle$ with the following attributes, further elaborated in TABLE 2.6:

TABLE 2.6
MDP VARIABLE TABLE

Denotations	
$\boldsymbol{S}$	The State Space, a finite set of all possible states of the system.
$\boldsymbol{A}$	The Action Space, a finite set of actions that the agent can perform.
$\boldsymbol{A_s}$	The set of actions available from the state s.
$\boldsymbol{P}$	The set of transition probabilities from one state to another for any given action
$\boldsymbol{P_a(s, s')}$	The probability to go to state s' from state s by action a.
$\boldsymbol{R}$	The reward function to determine the value of the immediate reward obtained after transition from state s to state s by action a, denoted by $R_a(s, s')$.
$\boldsymbol{\pi}$	Policy, a mapping from the state space to the probabilities of choosing a different state.

From state s_t at time slot t, the agent will select the action a_t to change state s_{t+1} according to π to receive reward r_{t+1}. The goal of the MDP is to find the optimal policy $\pi*$ to maximise the cumulative R. The cumulative reward of state s_t can be defined as the sum of the geometrically discounted future state rewards using the γ factor $(0 \leq \gamma \leq 1)$ as defined below in the general RL equation(2.2) [46]:

$$R_t = \sum_{t=0}^{\infty} \gamma^t r\big(x(t), a(t)\big) \qquad (2.2)$$

TABLE 2.7
RL EQUATION DENOTATION TABLE

Denotation	
$\boldsymbol{\gamma}$	Discount factor
$\boldsymbol{r}$	Reward
$\boldsymbol{x}$	Task
$\boldsymbol{a}$	Agent
$\boldsymbol{t}$	Timeslot

2.5.4 Dynamic Programming (DP)

DP is a technique or solving complex problems by breaking it up into simple subproblems and computing then subsequently storing the solutions. If a subproblem re-occurs, the stored solution is used.

Policy iteration and value iteration are both dynamic programming methods used to solve Markov Decision Processes (MDPs) to find the optimal policy. While they both aim to achieve the same goal, they differ in their approaches and computational processes.

Value Iteration

Begin with a random value function and then optimise it iteratively.

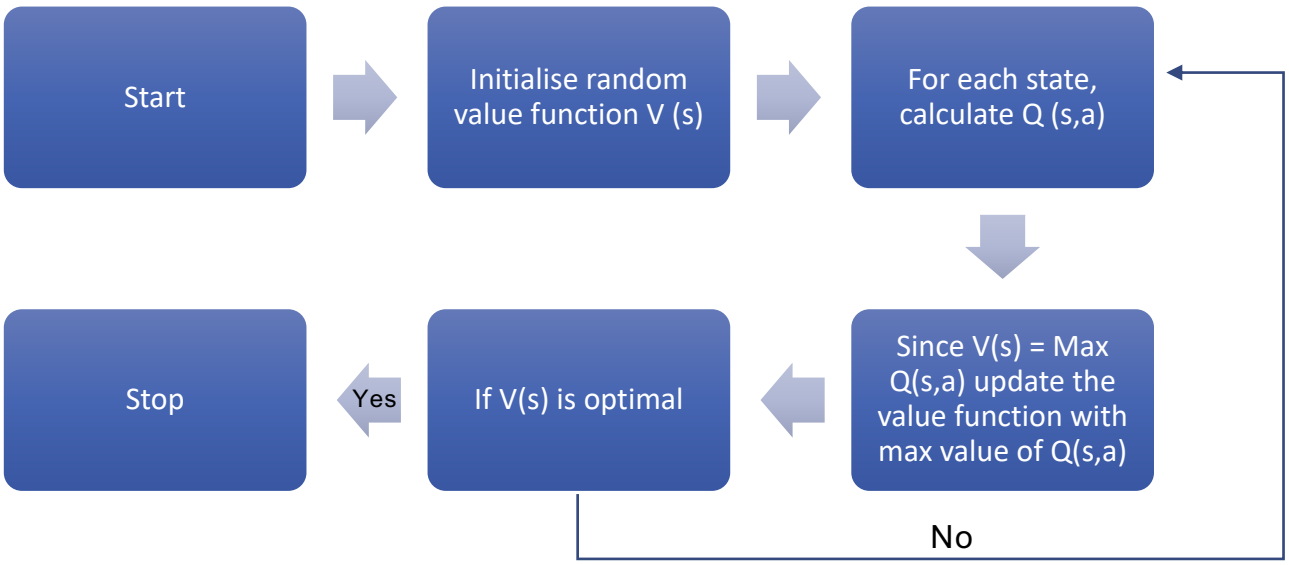


Figure 2.11. Value iteration diagram

Breakdown of the value iteration algorithm as seen above in Figure 2.11:

- **Initialization**: Initialize the value function $V(s)$ arbitrarily for all states $s \in S$. Commonly, $V(s)$ is initialized to zero for all states.
- **Iterative Update**: For each state s, update the value function using the Bellman equation:

$$V_{k+1}(s) = \max_{a} \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V_{k(s')} \right] \quad (2.3)$$

Here, $V_k(s)$ is the value function at iteration k, $V_{k+1}(s)$ is the updated value function and γ is the discount factor, as seen in the general policy iteration Eq. 2.3.

- **Convergence Check**: Check if the value function has converged, i.e., if the maximum change in the value function across all states is less than a small threshold θ. If not, repeat iterative update.

- **Policy Extraction**: Once the value function converges, extract the optimal policy $\pi*$ by choosing the action that maximizes the expected value as seen in Policy Extraction Eq. 2.4:

$$\pi^{*(s)} = \arg \max_{a} \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma V(s)] \quad (2.4)$$

Policy Iteration

Decide which actions the agents need to take or initialise first, create a value table according to policy viewed in Figure 2.12.

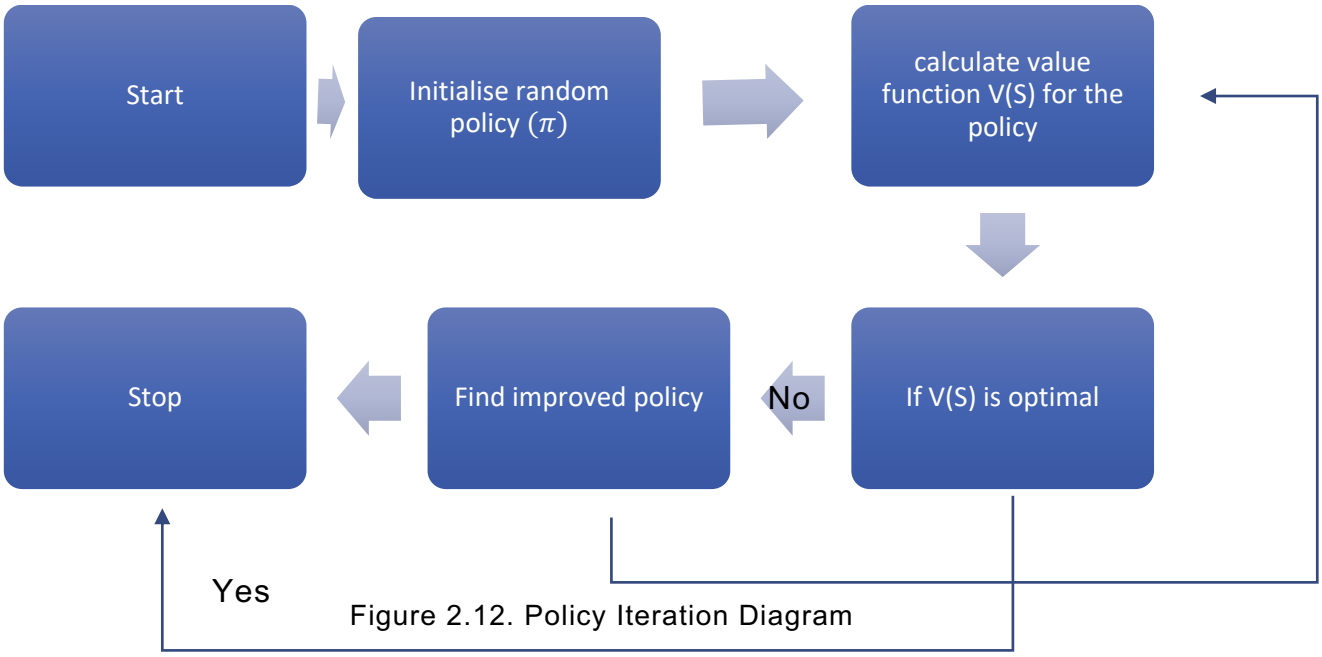


Figure 2.12. Policy Iteration Diagram

The data fed to the algorithm can be either continuous or discrete. This research can benefit from the culmination of both data types and use the FDT logic to return discrete variables or perform more finite data analysis from continuous data monitoring.

The added complexity of monitoring a continuous data stream would increase the time taken to implement and perform tasks in our implementation of the optimisation algorithm but deciding where and when to use the different types

of data was imperative to ensure that maximum operational efficiency of the final algorithm.

As observed in [47], an FDT model uses classification of data to label a category of data giving the algorithm a chance to interpret the data using real-world variables and effectively reducing the dataset into types as allocated by the FDT.

TABLE 2.8
ADVANTAGES OF CONTINUOUS VS DISCRETE DATA

Continuous Data	Discrete Data
Inferences can be made with few data points – valid analysis can be performed with small samples.	More data points (a larger sample) needed to make an equivalent inference.
Smaller samples are usually less expensive to gather.	Larger samples are usually more expensive to gather
High sensitivity (how close to or far from a target)	Low sensitivity (good/bad, pass/fail)
Variety of analysis options that can offer insight into the sources of variation	Limited options for analysis, with little indication of sources of variation

As observed in TABLE 2.8, continuous data monitoring has numerous advantages over its counterpart and provides detailed insight and can prove beneficial for industrial applications as demonstrated by the comparison presented in Figure 2.15.

Despite the obvious advantages of using a continuous model, more data will inevitably impact the efficiency of the algorithm and impede its ability to make rapid decisions whilst taxing the computational resources more. Therefore, a balance between the use of continuous and discrete data must be used where a centralised controller that can host and monitor the continuous data, portrayed in Figure 2.14, to provide a base or template for the RL algorithm would be the best approach to use in this case.

Using a system that can leverage the greater amount of detail provided by the continuous data to supply finite control over the algorithm would give the

algorithm the ability to rapidly take actions from discrete data (as shown in Figure 2.13) received in real-time, adjusting to the networks needs dynamically whilst always learning from the continuous data to enhance efficiency of the RL model and use transfer learning to share this across all nodes giving them the opportunity to benefit from the detailed insights provided.

Placement of the respective data models would be crucial in visualising and developing the ideal algorithm therefore, planning was crucial at this stage to understand the process of how it would be implemented.

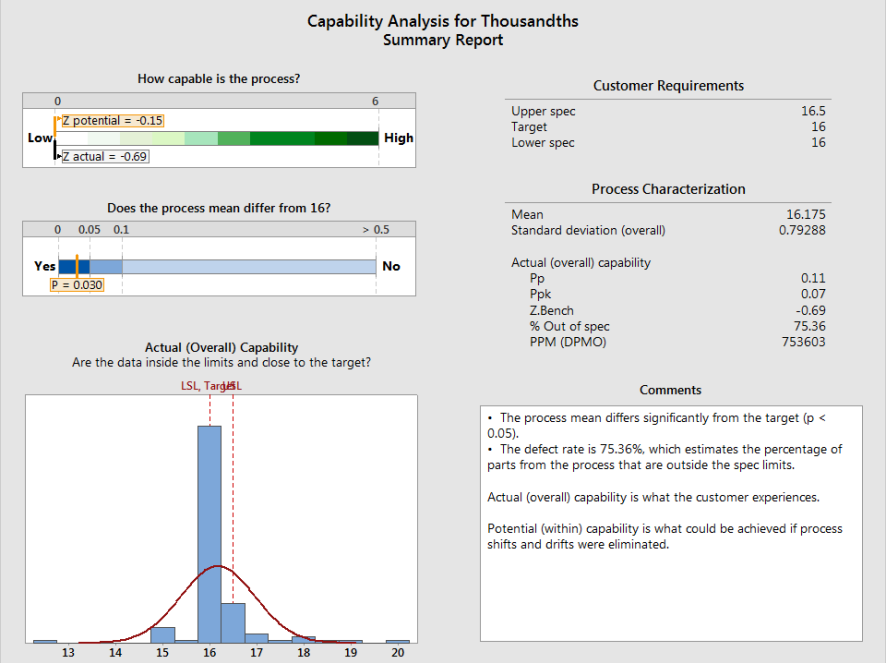


Figure 2.14. Example of continuous data

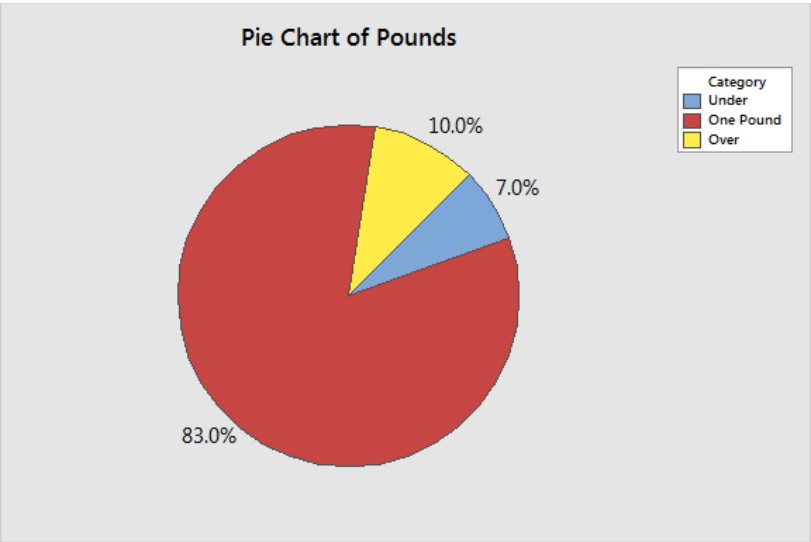
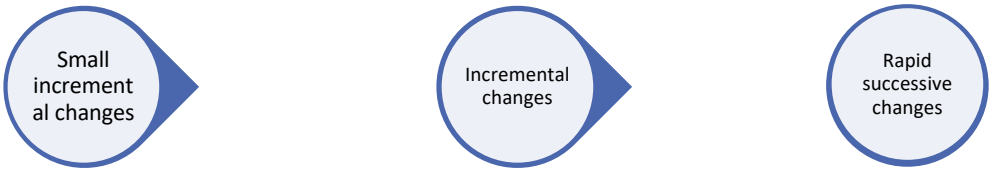


Figure 2.13. Example of Discrete Data



	Non-rapid turnover	Macro parameters	Rapid turnover
Examples	Link speed Routing tables Other network conditions	Closest serving cloud/exchange for service types	Fuzzification results # of devices connected # of services active
Description	Includes parameters that require a lower refresh rate and will generally be retained longer	Parameters that will be occasionally re-assessed to ensure that the algorithm is functioning at its optimum	Parameters that will require constant redress to ensure that algorithm is optimal

Figure 2.15. Algorithm planning stage one

2.5.5 Fuzzy Logic

Despite its introduction, or at least conception by Prof. Lotfi Zadeh in 1965 [48], [49], the concept of Fuzzy Logic did not gain traction within academia until 1993. The professor himself mentions that acceptance of his theory took many years to come to fruition and be recognised in academia. As observed in [48], one the earliest works based on fuzzy logic, there were barely a few references, all based on similar timespans to cite from.

In fact, we find that fuzzy logic and its application in both ML and AI efforts started to gain traction in the late 2000's (based of IEEE database search ranging from earliest publication date to current point in time). It is visibly clear that the concept of Fuzzy Logic is beginning to play its role in modern

distributed computing, AI and Machine Learning alike effectively tripling the popularity of the concept.

The intention of Fuzzy Logic is to model logical reasoning with imprecise statements using vague statements to provide a determinable outcome that a machine can then easily classify [50]. It uses a repository or set of many-valued logics and then stipulates the truth value of logically compounded propositions. Its core functionality is based on the concept of the *'gray area'* between the black and white or binary true and false values where 1 represents 'Completely true' and 0 represents 'Completely false' mapped on the Universe of Discourse, a set of entities where variables may range.

Fuzzy logic applies in a variety of engineering efforts, including systems in aerospace, civil, automotive, support, chemical, natural language processing, and modern control systems. Its goal is to mimic how a human being would make decisions, at computational speeds (much faster), providing the ability to use Fuzzy Logic in Neural Networks. Creating further distinction and classification between simply true and simply false gives the machine the ability to lean towards better decision making whilst learning from an ever-increasing dataset. An interesting comparison found in [51], details the difference between probability and Fuzzy Logic found in TABLE 2.9:

TABLE 2.9
FUZZY LOGIC VS PROBABILITY

Fuzzy Logic	Probability
The goal of Fuzzy Logic is to try and capture the essential concept of vagueness	Probability is associated with events and not facts, and those events will either occur or not
Fuzzy Logic captures the meaning of partial truth	Probability theory captures partial knowledge
Fuzzy Logic takes truth degree as a mathematical basis	Probability is a mathematical model of ignorance

A clearer example, stated by Shlomo Zilberstein [51], 'Fuzzy logic is a technique for representing and manipulating uncertain information. In the more traditional propositional logic, each fact or proposition, such as 'it will rain tomorrow,' must be either true or false.

Like probability theory, fuzzy logic attaches numeric values between 0 and 1 to each proposition to represent uncertainty. But whereas probability theory measures how likely the proposition is to be correct, fuzzy logic measures the degree to which the proposition is correct, as noted in TABLE 2.9 FUZZY LOGIC VS PROBABILITY.

The architecture, shown in Figure 2.16, has the following format to 'fuzzify' crisp inputs to an output format that can follow conventional rules, giving AI, or other applications, logical reasoning according to rules applied to parameters [52].

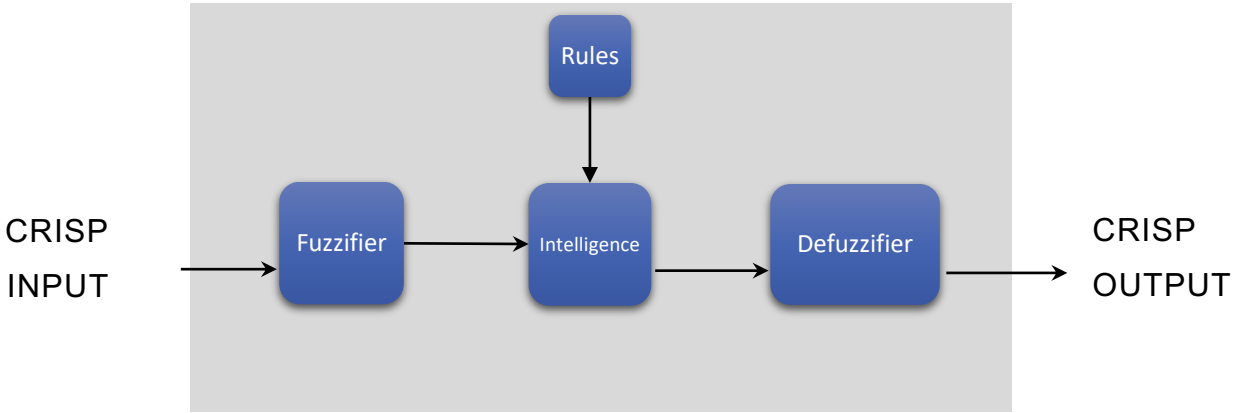


Figure 2.16. Fuzzy logic architecture

Therefore, we can conclude that Fuzzy sets are used to gain *degrees of truth* in a variable data set within a Universe of Discourse. The outputted data can be implemented within AI and ML applications alike to provide learning mechanisms within applications for perpetual improvement. Unfortunately, Fuzzy Logic does come with some disadvantages such as:

- Output data is not always accurate as the results are perceived based on assumptions.
- Fuzzy Logic systems cannot recognise ML & neural network type patterns.
- Setting *exact* fuzzy rules and membership functions is inherently difficult.
- Often confused with probability theory

It does, however, suit our application needs perfectly. Particularly when used for orchestrated learning methods, as we will find when dissecting the simulation software used in this research. It will ultimately provide an automated management system using the intuition of a real-world administrator, letting the network adapt dynamically to user's needs [24].

Fuzzy sets are the sets of information within a universe of discourse that do not have a defined membership property, whereas classical sets have a binary membership value of either 1 or 0 [53]. Denoted by μA(x), Eq. 2.5 and 2.6 compare the difference between a classic set membership and Fuzzy Set:

$$\chi_A(x) = \begin{cases} 1, & x = A, \\ 0, & x \neq A. \end{cases} \quad (2.5)$$

$$\mu_A : X \rightarrow [0,1] \quad (2.6)$$

The aforementioned equations portray the difference between the binary relationships in classic set membership where there is no middle ground, against the non-binary relationship that can be utilised with a varying degree of membership.

Defining fuzzy variables occurs in the declared .**fcl** file, which the simulator can then process to ensure flexibility within the process. This helps to make a more readable format for the end-user as well as apply dynamic changes according to the policy parameters. Rules can be applied using Fuzzy Logic in

the following manner, which in turn makes the process human friendly and easier to read as seen in TABLE 2.10:

TABLE 2.10
TABLE OF RULES EXAMPLE FOR 'FUZZIFICATION'

Rule No.	Rule
Rule 1	IF wan IS low AND tasklength IS low AND destinationUsage IS low AND delay IS low THEN offload IS edge;
Rule 2	IF wan IS low AND tasklength IS low AND destinationUsage IS low AND delay IS medium THEN offload IS edge;
Rule N	…

For our use case, fuzzification values as declared in the .fcl file can be iteratively updated on simulation completion, thereby enhancing our initial state. Said values can also be communicated across to a cloud orchestration function to dedicate learning tasks acquired from local MEC hosts and adapt to wider use case scenarios where similar environments can be provided with initial values acquired from live usage of the system.

2.6 Resource allocation with Reinforcement Learning

2.6.1 A Q Learning approach

One of the more recent works closely related to the research proposed can be found in Robles-Enciso, et al. [54]. Despite the focus on Task Assignment Problem (TAP), [54] propose an RL-ML enabled technique that gives edge agents the ability to query an upper-level agent to increase contextual knowledge of the network environment to the actors involved, thus routing the task in a more efficient manner and meeting the QoS requirements of the individual tasks more accurately.

Several basic algorithms are implemented alongside a single layer and multi-layer Q-learning algorithm to test the performance of RL with near-optimal greedy algorithms, tasks are also offloaded to neighbouring devices for fog offloading purposes within M2M environments. To address the TAP, [54] divide

the main system into three separate layers proposed with an intermittent connection and dynamic positioning. It is also proposed that the given layer has the least latency but also the least computational capacity.

The subsequent layer is referred to as the Fog layer which offers both intermediate latency and computational capacity, and finally the cloud layer which offers the greatest computational capacity alongside the most latency. Task characteristics are outlined with the capabilities of the simulator to define the parameters of each task as well as the remaining characters for the network nodes to ensure a controlled environment during simulation[47].

A device is given several options to allocate a task including local execution ($a = 0$), send it to an adjacent node ($a = 1$), the fog layer ($a = 2$) or the cloud layer ($a = 3$). A breakdown is visible in TABLE 2.11. An important point to be noted here is that [54] do not consider a distributed computing cloud computing model at the penultimate layer, therefore the cloud layer which consists of a single device is the final call for a task to be executed. Additionally, a penalty cost δ is allocated to ($a = -1$) should a task fail, alluding to the following segmented function as shown in Eq. 2.7:

$$C_{d,k}(a) = \begin{cases} C_{d,k}^{l} \; if \; a = 0 \\ C_{d,k}^{m} \; if \; a = 1 \\ C_{d,k}^{f} \; if \; a = 2 \\ C_{d,k}^{c} \; if \; a = 3 \\ \delta_d \; if \; a = -1 \end{cases} \quad (2.7)$$

TABLE 2.11
TOTAL COSTS FOR OFFLOADING TASKS

Device	Link	Cost
Local	N/A	$C_{d,k}^{l} = L_{d,k}^{l} + \beta E_{d,k}^{l}$
Edge	Wireless	$C_{d,k}^{m} = L_{d,k}^{m} + \beta E_{d,k}^{m}$
Fog	Wired	$C_{d,k}^{f} = L_{d,k}^{f} + \beta E_{d,k}^{f}$
Cloud	Wired	$C_{d,k}^{c} = L_{d,k}^{c} + \beta E_{d,k}^{c}$

When a task is offloaded, the propagation times as well as latency and the respective energy consumption is accounted for with wireless links to ensure all costs are considered. The greedy algorithm is implemented as a control method.

The computational complexity is $O(|D_n| + |D_f| + |D_c|)$ respectively, where $d =$ Device where the task originates, $D_n =$ Nearest Edge Device Set, $D_f =$ Fog server device set, $D_c =$ Cloud device set, accordant tasks are denoted as t. Ultimately, the heuristic value of each device is calculated using the following formula where β is defined as a weighting parameter to regulate the trade-off between latency and consumption:

$$min = \omega_x \times \frac{t_{mips} \cdot d_{tr}}{d_{mips}} \times \phi d_{cpu} \quad (2.8)$$

TABLE 2.12
DENOTATIONS FOR HUERISTIC DEVICE VALUES

Denotations	
$\boldsymbol{\omega_x}$	Different weighting constant for each set (ω_n, ω_f and ω_c)
$\boldsymbol{\phi}$	Trade off constant between actual CPU usage and running tasks

The algorithm is constructed to offload the task assignment decision according to the greedy policy as outlined, subsequently, two other algorithms are constructed, a single layer reinforcement learning algorithm and a multi-layer reinforcement learning algorithm based off the Markov Decision Process (MDP) that we will explore further later over the course of this research.

The authors state their intention to use the Q-Learning algorithm, which focuses on the optimisation of the action-value function (Q) using an iterative update based on previous values and temporal difference as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \, max_{a'} \, Q(s_{t+1}, \, a') - Q(s_t, a_t)] \quad (2.9)$$

TABLE 2.13
DENOTATIONS TABLE FOR Q-LEARNING ALGORITHM

Denotations	
s_t	Current state
a_t	Current action
γ	Discount factor
a'	Learning rate $= \{0 \rightarrow 1\}$

Ultimately, a piecewise function is defined as where the second function indicates the action to take in the event of a task failure, multiplying the defined penalty by the given task's parameters:

$$C_t = \begin{cases} (T_{end}^t - T_{start}^t) + \beta T_{energy}^t & T_{end}^t - T_{start}^t < T_{dl}^t \\ \delta \cdot \left((T_{end}^t - T_{start}^t) + \beta T_{energy}^t\right) & \text{otherwise} \end{cases} \quad (2.10)$$

Finally, a multi-layer algorithm is introduced where offloading decisions made can also be transferred across to the overarching Fog/Mist layer, where available computational power is greater. Using the previous algorithm introduced[54], each device works independently using aggregated global information with local information to make allocation decisions, however, the biased view of the environment and lack of knowledge in the early stages of Q Table generation was displaying lower performance in complex situations.

To address this shortcoming, the following approach is proposed for a multi-layered algorithm that provides contextual global knowledge to the secondary layer with the following structure:

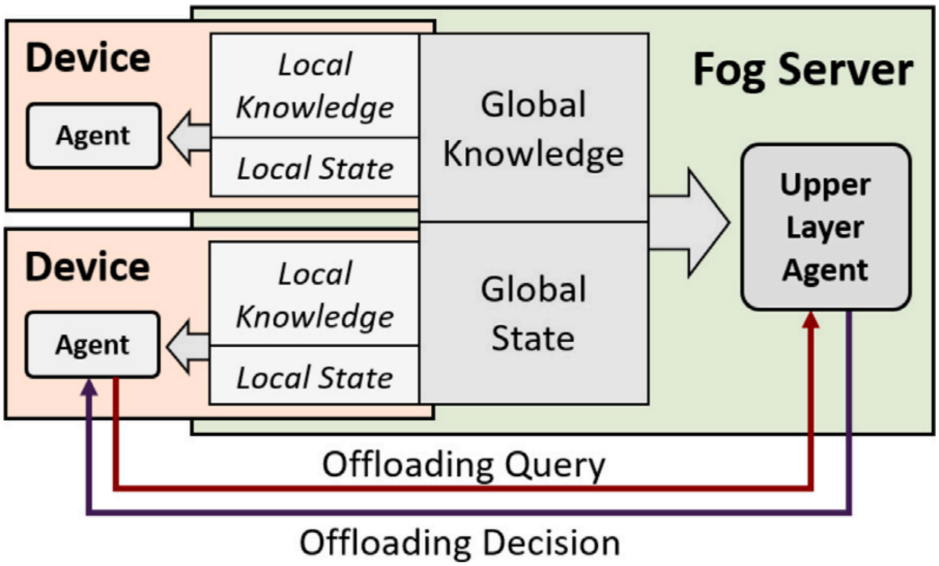


Figure 2.17. Multi-layer offloading query process

An additional action $(A \cup \{4\})$ is introduced which denotes the offloading process for a query in $A \in A$. As a result, each device manages its own independent Q-Table which is locally trained and the upper layer agent will subsequently be able to take advantage of the interactions with devices to update and maintain a 'global state' as highlighted in Figure 2.17.

Results gathered portray that the multi-layered algorithm improves energy consumption when compared to the control greedy algorithm whilst increasing performance across the board including adding significantly to the success rate of tasks completed when compared to the fixed and stable behaviour due to the lack of dynamic components. As a result, the authors find that as the device density increases, the multi-layered approach improves performance.

A Deep Q-network is proposed by Xiong, et al. [55] where multiple replay memories are applied to enhance the learning process on each iteration of the algorithm. Furthermore, action spaces are separated into two subspaces to reduce the action space size. Their implementation is tested within a simulation environment where an MEC system is deployed at the base station in a single-cell cellular network as can be viewed in Figure 2.18.

Resources are allocated to items in the job queue at each time slice, the actions taken by the host are then passed on to the host level management where a reinforcement learning algorithm can refine the allocation process and assign rewards using the DQN algorithm devices by the team. This research is limited to a single mobile-edge application within an industrial IoT environment to optimise low-latency performance of resource allocation.

IoT devices are randomly deployed and are tasked to upload sensor data to the network over M2M connectivity, where a single application is deployed for simplicities sake, and the tests are focused on resource allocation.

Their research is based off [46], where Q learning is used for resource allocation and tackling the offloading decision problem in an IoT edge network. [56], where DQN-based strategic computation offloading algorithm for MEC environment is used to minimise the long-term weighted sum of execution delay.

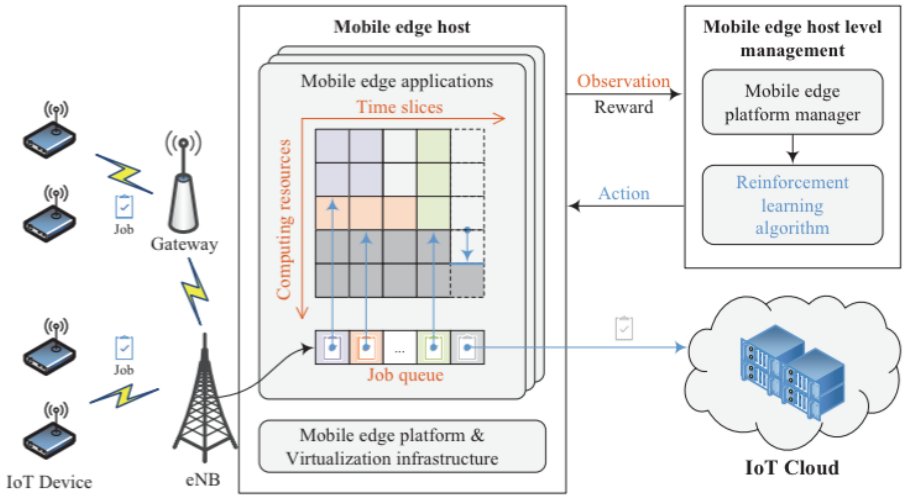


Figure 2.18. System model

To better understand resource allocation, it is imperative to comprehend how tasks are queued at the orchestrator, as observed in Figure 2.19, provided by Xiong, et al. explains how time-based resource allocation can be portrayed in diagram coupled with its denotations.

To ensure minimum latency, Xiong, et al. propose dividing the action space in two, and only observing earlier-arriving jobs to decrease long waiting times for job scheduling derived from early research within resource management with deep reinforcement learning [57].

It is further clarified that the allocation of computing resources C is formulated as a $n_c \times n_t$ matrix, where n_c is the number of total computing resources on the mobile edge host and n_t portrays the number of time slices in the sliding window. Incidentally, each row of C represents a computing resource $c_{i_c}, i_c \in \{1,2,\dots,n_c\}$ scheduled for allocation starting from the current time slice and looking ahead n_t time slices into the future.

Additionally, each computing unit can hold a value of $\{-1,\ 0, 1\}$ representing three different states: unavailable, available, or allocated. States are updated accordingly when computing units are requested by the job waiting in the queue.

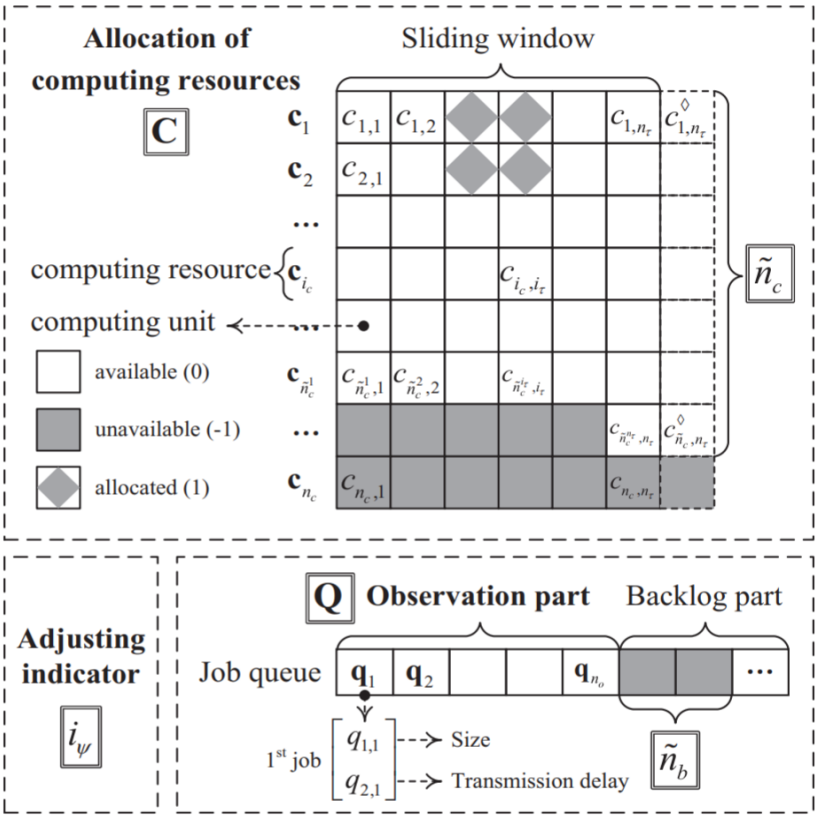


Figure 2.19. Illustration of the state $S = (C, Q, i_\psi, \tilde{n}_c, \tilde{n}_b)$

TABLE 2.14.
DENOTATION TABLE FOR FIGURE. 20

Denotations	
$\boldsymbol{C}$	Allocation of computing resources to jobs
$\boldsymbol{Q}$	Observation part of job queue
$\boldsymbol{i_{\psi}}$	Adjusting indicator
$\widetilde{\boldsymbol{n}}_{\boldsymbol{c}}$	Number of computing resources requested in the next timestep
$\widetilde{\boldsymbol{n}}_{\boldsymbol{b}}$	Number of jobs in the backlog part of job queue

To efficiently identify the multitude of use case scenarios within their simulation software, [58] compare the use of ML in VEC against a multitude of different competitor algorithms such as SMA, MAB and game-theory based vehicular edge orchestration.

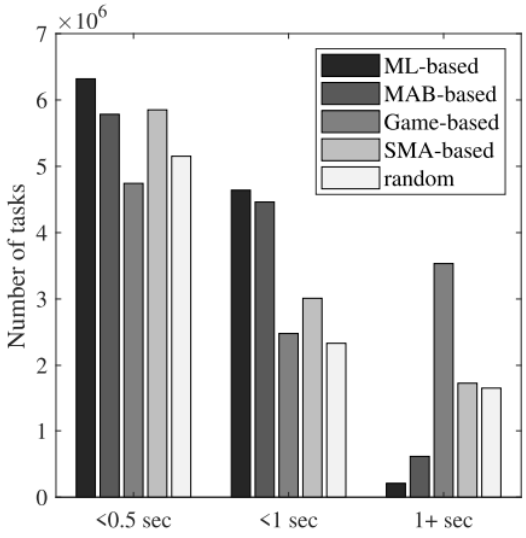


Figure 2.20. Histogram of service times

Tasks are offloaded to the target server in a probabilistic manner where the probability of selecting all targets is the same. Although the use case in [58] is applied in VEC, the usage of an ML based algorithm portrays improvement within orchestration with the utilisation of reinforcement learning, particularly when the service time of the task is greater than one second as can be viewed in Figure 2.20.

The efficiency of the algorithm increases as the ecpected service time increases over the number of tasks, particularly where service times are greater than a second. This effect in the TAP is commonly seen across a number of research efforts where ML is introduced to improve task orchestration and resource allocation[45], [57], [59].

The authors observe however, that the evaluator can be misled in delay-loss systems, so to ensure that an adequate service time is provided without loss of successful tasks, a quality of experience formula Eq.(2.11, is defined which considers both the service time and task loss. Thus, tasks that meet the service time and yet fail are not considered succesful.

$$QoE_i = \begin{cases} 0, & \text{if } T_i \geq 2R_i \\ \left(1 - \frac{T_i - R_i}{R_i}\right).(1 - S_i), & \text{if } R_i < T_i < 2R_i \\ 1, & \text{if } T_i \leq R_i \end{cases} \quad (2.11)$$

One of the notable works that aided in the earlier direction of my research was [60], my initial approach was to discover literature that had utilised the simulator that I intended to use for research purposes so that I could further understand it's implementation and limitations.

This led me to Zhang, et al.'s proposed research using a double deep Q-Learning model applied in EdgeCloudSim and focusing heavily on energy-efficient scheduling. Unfortunately, I was yet to understand the heavy implications of energy-efficiency when integrating EC into the 5G network architecture, however, it was apparent that this study was of vital importance when considering the implications of efficient scheduling.

Numerous academic articles as well as research has been conducted utilising EdgeCloudSim as a framework for further testing of algorithmic implementation in various modules as included in EdgeCloudSim.

Literature to be noted in this case is the aforementioned Deep Q-Learning by Zhang, et al. [3] which attempts to implement an energy-efficient solution for resource allocation utilising deep learning. Despite the research's emphasis on energy consumption, the authors attempt to implement an energy-efficient algorithm that utilises less power when a task is allocated by measuring frequency and voltage using DVFS as proposed in the following diagram by Zhang, et al.[60]:

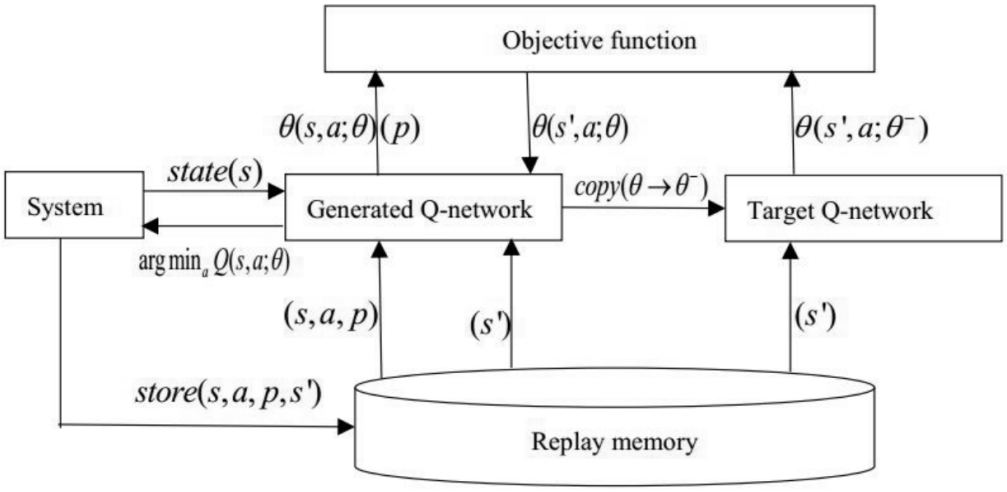


Figure 2.21. Learning scheme for double deep q-learning model

The proposed involves creating and testing a learning algorithm that stores and reacts to every action processed and carried out, it then optimises the task for efficient energy consumption, utilising less power as a result and using 2%-2.4% less energy. This improved efficiency can be applied to always-on IoT devices or indeed, as in this case, be used to prolong battery life for user devices.

The premises of the research begin with the analysis and exploration of how computation is completed, Zhang, et al. propose that within a lab environment, the following portrays the energy consumption of a single end-user 'labtop' [60].

These figures, though general in nature, must be taken as an approximation, despite the article being dated 2018, power efficiency within the ECE industry

as well as adoption of more efficient technologies such as SSD, as well as omission of the RAM modules within a device prove that our authors have not taken all aspects of the energy consumption in a single device, into consideration.

As this was one of the earlier academic articles approached over the course of this research, it was apparent that energy-efficiency was to become an integral part of this research over its duration. The use of DVFS algorithms suggested that the goal was to optimise for energy-efficiency rather than a scalable network environment. During their testing, Zhang, et al. broke down the key components that utilise power within a device as seen in Figure 2.22.

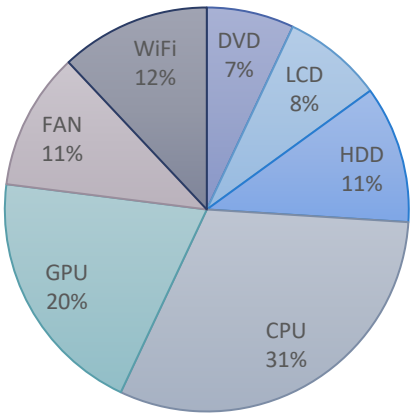


Figure 2.22. General power consumption of a 'labtop'

Research aims are to tackle limitations in energy consumption on both user and client ends and it is indicated that serious implications apply when edge servers reach capacity and begin to fail tasks, especially in government or time-sensitive applications[60]. However, when considering that 5G will offer the ability to produce layers that differentiated services can operate on, some of the main concerns posed in the research are alleviated [61].

The algorithm proposed in this research is being applied at the infrastructure level as seen in although not explored within the scope of their research, Zhang, et al. indicate the identification of QoS requirements posted by UEs using EC, such as worst-case execution times WCET, and subsequently

completing the tasks using an energy-efficient approach. The research is built on existing literature but uses a combined learning model that overcomes limitations within discrete system states, allowing for continuous adaptation of the network model without having to expand the Q-table size to recompute optimum performance values.

To formulate the algorithm, Zhang, et al. use the following approach to define the task and energy models [60]. To keep this research on track, the energy model, including CMOS based processor energy consumption of 70nm technology will be intentionally omitted, primarily to avoid constrained variables and secondly, at the time of writing, current-gen technology utilises a 14nm process and power-consumption values have changed significantly. This does not however, undermine the research presented by Zhang, et al. as MNOs do not upgrade their infrastructure yearly, but as this research is exploring architecture that is yet to be installed/leveraged, it is safe to assume that it won't be done on outdated technology [62].

The variables collected from the models discussed above are supplied as inputs to the Q-Learning model, where double deep Q-learning method is applied to the variables to generate two Q-network models: the generated Q-network $\big(Q(s,a:\theta)\big)$and the target Q-network $\big(Q(s,a:\theta^{-})\big)$. A rectified linear units (ReLU) function is used as the activation function as it is more efficient for gradient propagation.

2.6.2 Cache allocation and Computational offloading

In their research, Ndikumana et, al. present two scenario's that they wish to tackle; drones used in large swarms to broadcast professional sport activities, and send live stream videos and medical imaging, where edge-hosted resources allow rapid access [63]. Their proposed model suggests a lone MEC server, restricted from other network resources, which must maximise resource utilisation in a contained network environment, akin to models within an industrial setting aim to optimise usage without relying on additional, external resources.

The system model used [63], consisted of MEC servers located at the RAT's which are then linked to each other and finally synchronised to a centralised data centre. They also suggest that each MEC hosts a RAT (Resource Allocation Table). Each table stores information of resources available to the architecture, and continuously update at regular time intervals.

This static approach works well where workloads and expectations of streams of data are predictable within the scope of its applications. In the case scenario described by [63], the key factor in the research is that the Edge Servers collaborate to process requests and update one another using RATs (Resource Allocation Tables). The weighted payment framework allows the MNO to further classify requests from the UE by exchanging updated RAT at Δ_t.

This works well for organisations and establishments where repeatability of tasks is greater however, does not fit a dynamic metropolitan model that can serve as underlying architecture for a 5G environment offered to the public. Its implementation is also limited by the complexity of the model and becomes unsuitable for true optimisation where constraints are not only limited to cache size and computational resources.

A further point to be noted is the redundancy introduced in line 2 of the algorithm where values are initialised upon each update, no doubt to pave way for convergence [64] and remove unpredictability from the results of the algorithm however, use of such techniques may introduce latency if each update calls for re-initialisation of values and losing key information that could assist with a ML approach coupled with continuous data acquisition methods [45].

One means to circumvent this loss of data, could be to find the difference between the values and log said information, forming a trend chart over time allowing the system to pre-emptively estimate network conditions. [65], where such techniques are employed, employ the use of such a predictive model within their IDTM algorithm using augmented backlogs as an input variable,

giving the system the ability to pre-empt the backlog queue, offering to solve for a worst-case scenario.

To facilitate this methodology, an anomaly detection mechanism, amongst other challenges, must also be overcome to ensure that the system is not acting on false or anomalous figures [66] introducing complexity and in turn, latency into the functionality. The approach taken by [63] also lacks the advantages gained from concepts such as Reinforcement Learning (RL) providing MNOs perpetual systems that require less maintenance and management and encourage the network to learn from past interactions to ensure that the network is always operating at its peak potential [67].

The methods outlined by [63] and [68] both function on the same underlying principle, neighbouring nodes adapt to frequently requested content according to location and attempt to duplicate content fewer hops away from the UE. This in turn leads to reduced delay when accessing content, and reduced traffic load on the network [68].

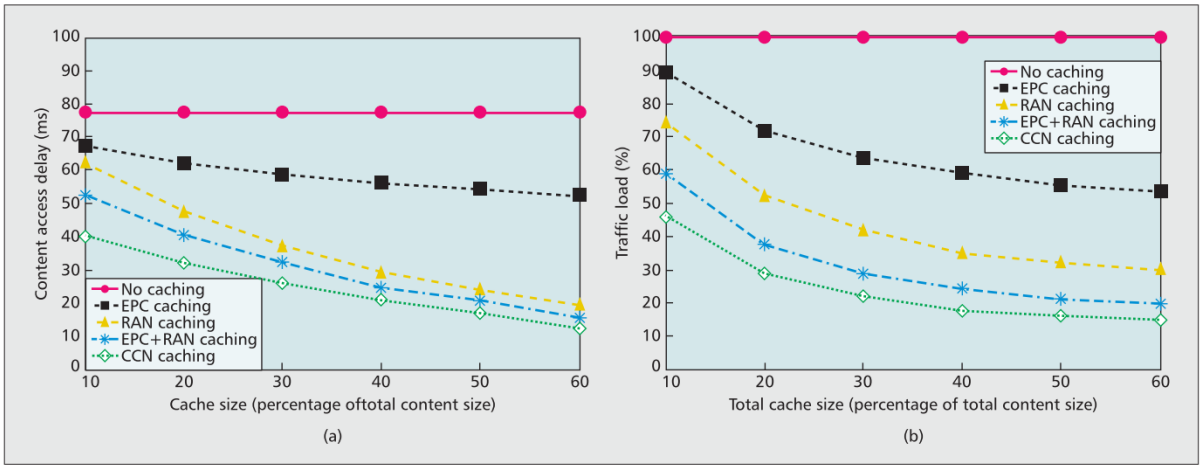


Figure 2.23. Caching types compared [68]

The method used by [63] works well but not as intended by the research. Despite the advantages offered when resources are cached, the goal of MEC is to offload computation to servers a single hop away to service low-latency applications and turnover data in rapid succession.

This doesn't mean however, that the authors of [63] were wrong in their approach, collaborative caching has contributed to QoS offered by many 'as-a-service' companies including Netflix [69]. In fact, according to [68], caching in both UMTS and LTE networks has been proven to reduce mobile traffic by one to two thirds, comparisons of which can be seen in Figure 2.23, where cache types are compared in different locations in the network, and CCN caching reduces network load and drastically increases performance with greater utilisation whilst reducing delays.

2.6.3 Stochastic Gradient Descent

The authors of [70] explicitly and comprehensively tackle the application of ML within dynamic resource allocation, also bearing into account the impact of the learning algorithm within the simulated delay. Of further interest, is the fact that the author's make it a point to question the efficiency and performance of the algorithm as a standalone ML algorithm as opposed to solely when it is applied within context.

This approach is particularly useful as within any realm, there is always room for improvement and the authors acknowledge that the quality of each component involved, directly affects the quality of the outcome. Opting to use a supervised learning method, a Stochastic Gradient Descent (SGD), used for large-scale and sparse machine learning problems, is implemented on the edge server, as found below:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) \qquad (2.12)$$

$$w := \mathrm{w} - \eta\nabla Q(\omega) = \omega - \frac{\eta}{n}\sum_{i=1}^{n} \nabla Q_i(\omega) \qquad (2.13)$$

SGD is primarily an optimisation technique used to train models where training steps (also known as learning rate in machine learning) are dictated by η, and steps can be limited to avoid taking into consideration the entire dataset therefore, decreasing the time taken for the algorithm to produce a response in exchange for accuracy.

To avoid looping through the same data, a *shuffle* of the dataset is introduced within the first stage of the algorithm, and the algorithm is repeated to provide a convergence to produce the output.

The network model used to test the efficiency of the algorithm consists of a single edge server located at P which is in turn connected to sensors(*K)* and the model used is stochastic in nature, leveraging the use of Lyapunov optimisation without assuming any prior knowledge of data.

2.7 Summary

Determining the correct algorithm was of vital importance in the formulation of this research. We explored the use of Reinforcement Learning (RL) to provide and subsequent key concepts.

This chapter examined key academic efforts that were closely analysed during this research. These efforts encompassed foundational discoveries and technologies that underpinned the intelligence of current technologies, as well as potential enhancements to the network infrastructure and hardware we used in our daily lives.

The chapter also delved into pertinent literature, past and present, to guide the research direction and the selection of the most suitable technologies for our optimization algorithm.

Chapter 3 Simulator Comparison and Design

The following chapter contains detailed comparison of the simulators shortlisted for this research and concludes with the selection of the final simulator used for the implementation of the algorithm formulated over the course of this research.

3.1 Introduction

As proposed by [71], the use of a simulator was the most advantageous approach to take for the purpose of this research, namely due to the high costs involved that would stop most students in their tracks, and the convenience of conducting real-time tests with rapid turnover on any modifications that may be required. Thus, ensuring that the research had a good starting point meant that several simulation software had to be compared of which one would emerge the victor in operability and academic viability.

Advantages of simulation include:

Controlled

- Ensuring that our study does risk or damage any live/active systems.

Reproducible

- Where we can easily reproduce our scenarios and conduct them under the same conditions.

Cost-effective

- Ensuring that our study is not costly to conduct and therefore can be refined as many times as required. Additionally, no proprietary hardware is required.

Time efficient

- Simulation times can be increased to multiples of real-time implementation, providing results over longer time periods without the wait.

Flexible

- Can adapt to numerous testing scenarios rapidly.

TABLE 3.1.
DETAILED FEATURE LIST OF SIMULATION SOFTWARE

Software:	Features:
Network Simulation 3	Mature application capable of simulating numerous network environments Advantages: • Free for research and educational use • Aligned with the simulation needs of modern development
	Disadvantages: o Code based development, which in turn makes it harder to visualize a network and implement changes
CloudSim	Provides a generalized and extensible framework to simulate cloud-based environments. Advantages: • Provides a generalized and extensible framework that enables seamless modelling, simulation and experimentation of Cloud computing infrastructures and application services
	Disadvantages: o Limited set of tools and applications
iFogsim	Designed to simulate Fog and Edge environments and created as an enhancement of CloudSim. Advantages: • Specialized for modelling fog environments and for evaluation of resource management and scheduling policies
	Disadvantages: o Complex interface and clunky controls
GNS3	Network simulation tool that provides a Graphical User Interface. Advantages: • Allows for quick modification of network elements
	Disadvantages: o Low integration with other applications
Qualnet	Network simulation tool with a GUI. Advantages: • Comprehensive modelling and simulation of real-world scenarios • Both wired and wireless networks can be easily built using GUI
	Disadvantages: o Not an open-source software o Paid license

Due to the ever-increasing popularity of the subject, possibly owed to the limited roll-out of 5G, a number of scholars, hobbyists and even some corporations have been working on or funding open-source initiatives for

developers around the globe to contribute to while enhancing their own research or meeting their ends (no doubt to monetise on them once the time comes) [72].

Several companies began and continue development of open-source network simulation software, all which lack Edge Computing features but are mature in their network simulation capabilities. Many of the simulators mentioned in TABLE 3.1 have a GUI along with their CLI to accommodate ease of use. Unfortunately, for the purpose of this research however, they do not support or offer dedicated cloud-based simulation.

After the initial intention of utilising GNS3 for simulating an edge computing environment, further research revealed that a purpose made simulator had been created by researchers from Bogazici University, Istanbul [73]. This was acknowledged to be an ideal simulator to build upon for the purpose of my research by the review conducted by [11].

The works conducted by Svorobej, et al. compare 7 different edge/fog simulation tools, their synonymous nature paves way for comparison between the tools as fog computing was the term originally coined by Cisco for cloud computation at the edge of the network [11].

Svorobej, et al. found that EdgeCloudSim contains the most accurate simulation tools for recreation of an MEC environment yet lacks scalability. Naturally, over the course of the literature and due to the open-source nature of the software created by Sonmez, et al. [73] the simulator is constantly being improved and contributed to by researchers.

Thus, in its current state, scalability features have been implemented allowing for the test of several hundred devices in a simulated edge environment where each UE simulated by the software is allocated a random task that is offloaded to the network edge [73]. In an update to my research, a simulator based on the initial findings of [73] was subsequently released[74] which will be discussed later in this research.

TABLE 3.2.
FURTHER SIMULATION SOFTWARE EFFORTS

Simulator	Features	Interface	License
Antidote (NRE Labs)	Combines a network emulator with a presentation framework	CLI and GUI	Open source
Cloonix	Simple network emulator with FTP abilities, can spawn QEMU-KVM	CLI	Open source
CORE	Network emulator and virtualization platform	CLI	Open source
EVE-NG	Network emulation platform with GUI	GUI	Open source
IMUNES	Integrated Multiprotocol Network Emulator/Simulator	CLI and GUI	Open source
Kathara	Lightweight network emulation tool	CLI	Open source
Mininet	Emulator for rapid prototyping of SDN networks	CLI	Open source
NS-3	Discrete-event network simulator	CLI	Open source
VNX and VNUML	Virtual Network experimentation and Network User Mode Linux	CLI	Open source
Containernet	Fork of mininet, allows for the use of docker containers	CLI	Open source
Knet network simulator	Uses docker containers to build network nodes and Open vSwitch to create switches. Has a CLI and web interface and incorporates SDN	CLI	Open source

Educational Network Simulator	Simple network simulator designed for educating young adults		Open source
Labtainers	Uses Docker containers and has many prepared labs for cybersecurity	CLI	Open source
Cnet network simulator	Enables development and experimentation with a variety of network protocols like WAN, LAN, and WLAN	CLI	Open beta
NetMirage	Allows real time code testing for IP applications	CLI	Open beta
ESCAPE (Extensible Service ChAin Prototyping Environment)	Supports development of several parts of the service chaining architecture including VNF	CLI	Open source
OMNeT++	Discrete event simulator and INET Framework simulator for both wireless and mobile networks	CLI	Open source
Netsim	Network simulator for Rust programmers	CLI	Licensed
VIMINAL	Linux based network simulator	CLI	Open source
5GPy	Python based 5G simulator with Fog Computing	CLI	Open source

TABLE 3.2 outlines several efforts to simulate containerised network applications. These were worth a mention due to the sudden surge in industry efforts to implement VNFs into application development. There are several advantages to this approach, not only does it simplify repeatability and modification of VMs that can be scaled easily, but it also allows software updates in a shorter time span. Applications such as Docker and Kubernetes have become accepted in wide stream web-applications and SaaS services [75], [76], [77], [78].

As highlighted above, the purpose of this research was to use a dynamic approach in the discovery of research software, where a testing scenario was devised but did not rely heavily on the simulation software being used. By using an approach akin to SDN and VNF concepts of decoupling [79] the variables to be tested from the simulation software, I was given the freedom to implement my approach on the highest bidder, or in this case the best simulator, available at the time of testing.

To achieve this, I ensured that I had devised a testing scenario that would largely remain unchanged over the course of this thesis until I was ready to discover results. Instead, my time and efforts were spent on researching the literature on my subject area.

This approach gave me the opportunity to discover improved simulation software along the course of this research, which in turn were implemented within the thesis after simple and repeatable tests were conducted on said simulators.

Unfortunately, this approach did also have drawbacks, namely that I would have to test every simulator and compare their features to find what would work best for this research project.

3.2 Requirements for MEC and Simulator Selection

3.2.1 MEC Requirements Review

To form the simulator selection, requirements must first be reviewed for MEC:

- Low/Intermittent connectivity
    - Bandwidth and associated high cost of transferring data to the cloud
    - Low latency, such as closed loop interaction between machine insights and actuators
    - Immediacy of analysis
    - Access to temporal data for real-time analytics
    - Compliance, regulation, or cyber security constraints

- Business Implications
- Predictive Maintenance
    - Reducing costs
    - Security assurance
    - Product-to-service extension (new revenue streams)
- Energy efficient management
    - Lower energy consumption
    - Lower maintenance costs
    - Higher reliability
- Smart Manufacturing
    - Increased customer demands mean product service life is dramatically reduced
- Customization of production modes
- Small quantity and multi-batch modes are beginning to replace high-volume manufacturing
- Flexible device replacement
    - Flexible adjustments to production plan
    - Rapid deployment of new processes and models

3.2.2 Simulator Selection

TABLE 3.3 highlights open-source software that portrays good examples for the framework of this research and can be manipulated to provide the groundwork to create the proposed application capable of meeting the requirements.

Further research conducted over the course of the thesis, paved way for the discovery of more suitable applications that were in development stages with incremental updates being introduced to add new features.

TABLE 3.3.

COMPARING FOG AND EDGE SIMULATORS

Attributes	FogNetSim++	iFogSim	FogTorchII	EdgeCloudSim	IOTSim	EmuFog	Fogbed	5GPy	PureEdgeSim
Computing paradigm	Fog computing (general)	Fog computing (general)	Fog computing (general)	Edge computing (IoT)	Edge computing (IoT)	Fog computing (general)	Fog computing (general)	5G Simulation PHY layer	Edge Computing(IoT)
Infrastructure and network level modelling	Distributed data centres Sensors Fog nodes Broker Network links Delay Handovers Bandwidth	Cloud data centres Sensors Actuators Fog devices Network links Delay Network usage Energy consumption	Latency Bandwidth	Cloud data centres Network links Edge servers WLAN and LAN delay Bandwidth	Cloud data centre Latency Bandwidth	Network links Fog nodes Routers	Virtual nodes Switches Instance API Network links	PHY layer simulation Modular Fronthaul communication support Fog and Cloud modules	Cloud data centres Network links Edge servers WLAN and LAN delay Bandwidth Enhanced Orchestration
Application-level modelling	Fog network	Data stream Stream-processing	Fog applications	Mobile edge	IoT	Fog	Fog network	Fog, Cloud	Mobile, Cloud, Edge, Fog, Mist
Resource management modelling	Resource consumption (RAM and CPU)	Resource consumption Power consumption Allocation policies	Resource consumption (RAM and CPU)	Resource consumption (RAM and CPU) Failure due to mobility	Resource consumption (RAM, CPU, and storage)	Workload	Resource consumption (RAM and CPU) Bandwidth Workload	Full infrastructure including power simulation	Ram, CPU, Energy, Mobility, Network conditions
Mobility	Yes	No	No	Yes	No	No	No	Yes	Yes
Scalability	Yes	No	No	No	Yes (MadReduce)	No	No	Yes	Yes
Date released	Jan 2019	Oct 2016	Apr 2018	Sep 2018	Feb 2016	Sep 2017	Nov 2016	2020	2019 - current
Forked	3	83	3	127	Not available	7	2	2	60
Last updated	Jan 2019	May 2017	Apr 2018	Nov 2020	Not available	Sep 2020	Nov 2018	Apr 2020	December 2022

3.3 CloudSim

As previously discussed, the introduction of any new software technologies brings with it a host of simulators that can help to alleviate issues before real-world implementation and deployment of said technology. Good engineering practises dictate that a better understanding of a new solution can lead to rapid evolution of technology and within networking applications, costs of deployment can escalate quickly.

Simulation software can help encourage better understanding and implementation of new solutions by giving software engineers the ability to simulate their solution within a containerised environment designed to mimic real-world implementation dependant on the amount of detail engineers wish to determine. One such simulation solution developed to understand the implementation of cloud technologies is CloudSim; a framework for modelling and simulation of cloud computing infrastructures and services [80].

An initiative going as far back as 2002, CloudSim is an open-source cloud environment simulation tool, formerly known as GRIDS Lab, developed by the School of Computing and Information Systems, University of Melbourne, Australia.

It has since been used for several academic articles and publications, research efforts etc. which have undoubtedly led to numerous industrial implementations over the years including sponsors the likes of Microsoft, Samsung, Huawei, Lockheed Martin, Sun Microsystems, European Union etc. It has also led to a few *forks* such as one of the simulation software's that will be discussed in-depth in later chapters.

In more recent and ongoing efforts, there have also been attempts to utilise the simulator to tackle the ongoing COVID-19 pandemic [81] using Fog Computing indicating the versatility and applicability of the technology as well as its current and future relevance.

The main functionalities, as listed on the website dedicated to the project are:

- Support for modelling and simulation of large-scale Cloud computing data centres
- Support for modelling and simulation of virtualized server hosts, with customizable policies for provisioning host resources to virtual machines
- Support for modelling and simulation of application containers
- Support for modelling and simulation of energy-aware computational resources
- Support for modelling and simulation of data-center network topologies and message-passing applications
- Support for modelling and simulation of federated clouds
- Support for dynamic insertion of simulation elements, stop and resume of simulation.
- Support for user-defined policies for allocation of hosts to virtual machines and policies for allocation of host resources to virtual machines

Some of the additional characteristics, as indicated by [82] are as follow:

- Developed in Java, a widely used programming language
- Open source in nature, enabling contributions from the developer community
- The first open-source specialised cloud simulation framework
- Provides great flexibility to create simulated scenarios, where each scenario is modelled using Java

CloudSims biggest attraction, however, lies in its potential for extensibility which has not gone unnoticed by researchers around the globe who have successfully implemented it into academic projects with varying subjects that may require simulation of cloud computing scenarios and subsequent technologies such as Mist, Edge, and Fog.

Due to the modular nature of the library, it can be implemented in any way that the user chooses, where the developer is free to call upon libraries contained

therein and extend them to their hearts content which can be observed in [83], [84], [22], [47]. Due to its early introduction, it has also become one of the academic go-to simulation software for research purposes and still retains funding from large technology corporations.

3.4 Simulation candidate 1: EdgeCloudSim

After analysis of the available simulators, EdgeCloudSim was one of the candidates shortlisted as one of the finalists, with good feedback from the community as well as some validation provided by members of the community and the developer. To simplify matters, we will only be analysing the two most suitable simulation software's in-depth.

Originally based off the CloudSim [4] simulation tool, which is currently on its 5th iteration, EdgeCloudSim uses the framework from CloudSim and implements a 3D Edge simulation model implementing use of wireless technologies that are currently not offered by CloudSim including WLAN and WAN, mobile nodes and mobility support and realistic Virtual Machine implementation.

The researchers for [4] decided to base their simulator on CloudSim due to its modular nature and its simplistic development when compared to other simulators or network modelling tools.

One the of key functions of EdgeCloudSim is the multi-tier approach for scenarios where multiple Edge servers can be run in coordination with upper layer cloud solutions.

To offer this functionality, EdgeCloudSim uses an orchestrator module to simulate orchestration actions of assigned tasks and actions which arise in Edge Computing scenarios.

The researchers then leveraged the extensive code base for modelling of computational tasks which has long been established and regularly contributed to, by developers of CloudSim due to its open-source nature.

3.4.1 Hierarchy and Design

Whilst developing EdgeCloudSim, the authors considered multiple approaches to designing the multi-tiered simulation environment. One of which was to utilise Hu et, al [5] optimisation method where mobile users offload computing.

Access point and remote clouds utilise a heuristic algorithm by considering both the communication and computation resources and thereby taking into consideration of the a RAN environment as a whole rather than considering only the data or computation oriented process and disregarding the mobile communications which will inevitably play a role, offering the possibility of further enhancements by working alongside the Edge computational model as opposed to being present as a separate entity [5].

The application is designed in the following order with each subset of applications functioning within their own designated layer and modules working with the following relationships as seen below [4]:

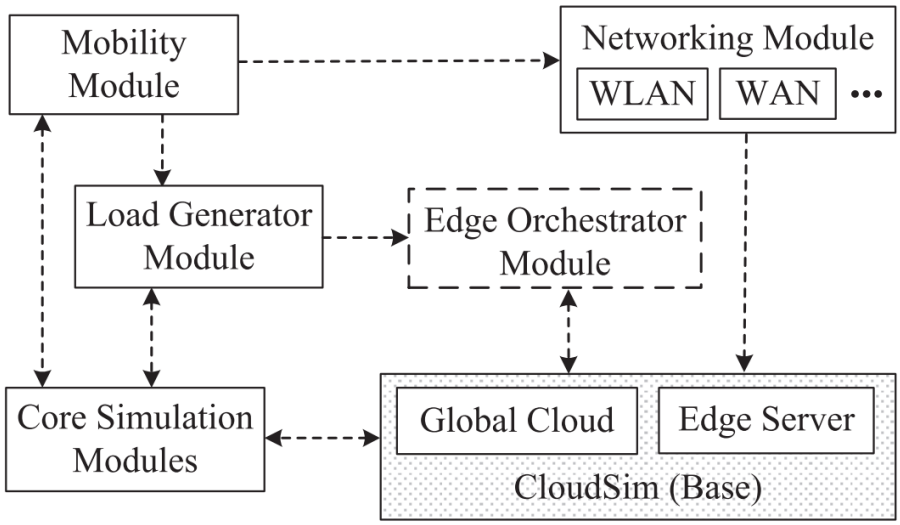


Figure 3.1. Relationship between EdgeCloudSim modules

Figure 3.1 outlines the relationship between the modules implemented by [73] utilising the CloudSim library as a base. Custom modules are introduced using CloudSims Global Cloud and Edge Serer classes as a base to implement a

custom Edge Orchestration as well as user mobility modules to produce the simulation output.

The remaining modules customise their CloudSim counterparts to further enhance the specific application of EdgeCloudSim as an Edge Simulation tool, utilising modified modules that can implement core functionality of edge simulation and orchestration. The authors also introduce a custom networking module that gives access to both WLAN and WAN link with customisation options for the user to shape the speed and bandwidth of said links to the simulation requirements.

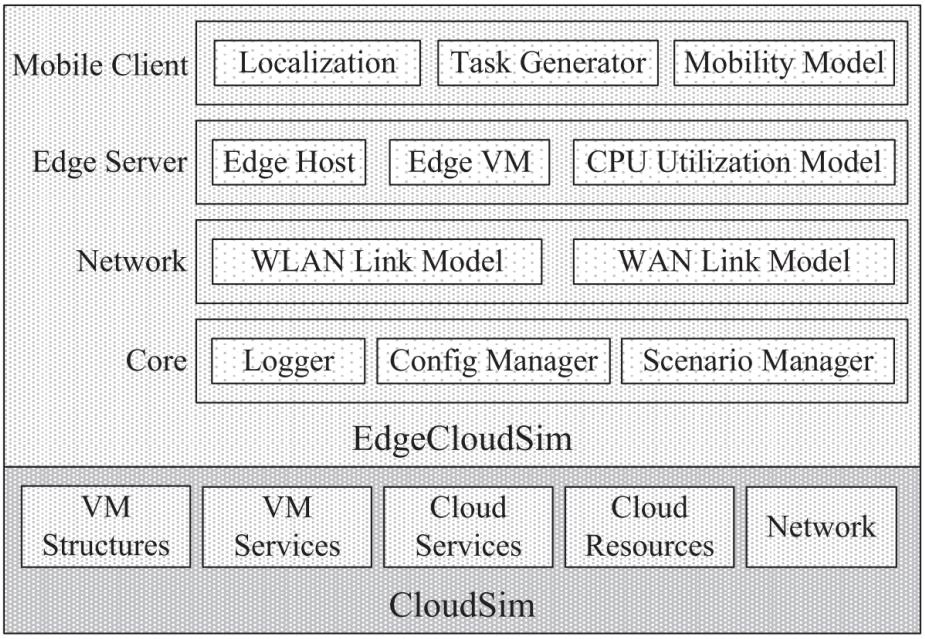


Figure 3.2. EdgeCloudSim layered architecture

Furthermore, Figure 3.2 portrays the various modules position within the simulator's architecture and the specific layer that they reside in. The mature modules from CloudSim supply the VM structures and services as well as define the network parameters to ensure that a simulation environment is as accurate as possible. EdgeCloudSim then supplies dedicated modules that tackle our particular use case scenario, giving users the ability to provide custom models for any module and fine-tuning their unique simulation environment.

The primary objective when designing the java-based simulator was to provide extensibility, therefore the authors created something called a scenario factory, a module within the simulator that orchestrates other modules to interact and have awareness of other modules as observed in Figure 3.2, orchestrates tasks to be sent to the VM for computational offloading.
The following modules are responsible for the tasks as detailed in TABLE 3.4:

TABLE 3.4
TABLE OF MODULES IN EDGECLOUDSIM

Module	Task
Core simulation module	Responsible for loading and running the entire application using the scenarios from the combination of XML and Java files. It also offers the ability to log and store results of the simulations.
Networking Module	Handles transmission delay and both WLAN and WAN scenarios, an improvement to the solution offered by CloudSim, as there were only static options available which were fixed for all users, the network link module makes a more accurate calculation based on several variables introducing dynamic scenarios and allows users to introduce their own network behaviours.
Edge Orchestrator Module	The edge orchestrator is akin to the backbone of the system as it directs and manages how tasks are handled by the Edge system. The authors claim that the orchestrator is very basic however, it allows for a more complex system to be integrated, extending functionality and optimising efficiency.
Mobility Module	The mobility module tracks the movement of the user within a 3-dimensional space using *x* and *y* co-ordinates. A hash table is used to store user co-ordinates where locations are dynamically updated.
Load Generator Module	The load generator module allocates tasks to the Edge Server which are then managed by the Edge Orchestrator and subsequently remaining modules. It utilises a Poisson distribution to assign tasks to various aspects of the edge server. The load generator coupled with the Mobility module are the main components as they provide input to other components.

To ease configuration and optimise task-flow for simulation purposes, an XML configuration file is used that can be changed on the fly to introduce any changes in the simulation. The parameters available in the XML file can manipulate usage for the four different types of applications available including variables such as delay sensitivity, data upload/download rates and individual VM utilisation per application. Other XML files can manipulate and modify the

amount of computing power available to the VM's available on the network edge.

These can range from multi-core processors to single-core devices, which can help find the most cost-effective solution to apply network resources. Further thought is also required when applying said solutions, as populated metropolitan areas are bound to request more resources than their counterparts based in the countryside, which would inevitably serve fewer users.

The greatest difference between CloudSim [84] and EdgeCloudSim [73] is the modular structure that EdgeCloudSim uses as well as the implementation of network modelling, device mobility modelling and a realistic load generator. EdgeCloudSim also supports multi-tier scenarios that can accurately determine the kind of workload that an Edge Server can expect [83].

As an extension of CloudSim, EdgeCloudSim was developed to implement features found in Edge Computing and make use of the architecture of Cloud Simulation that had already been provided by CloudSim.

A key omission from EdgeCloudSim was its lack of Mist computing features and for the purpose of this research, its lack of cellular access network model. The author(s) also acknowledged that Task Migration amongst the Edge or Cloud VMs, as previously observed in [63] was lacking, therefore a single identifiable MEC Server was responsible to manage as many tasks as possible whilst conforming to bandwidth and latency restrictions.

As the research was conducted in 2017, when the technology as well as the specifications were still in their infancy, the mobility model used by [73] was simplistic in its implementation which is also visualised during the simulation and can be seen in Figure 3.3.

The blue dots indicate end users moving around within the simulated cell space and the simulator accurately reproduces attenuation to ensure a realistic simulation by leveraging some of the developed class modules from CloudSim.

When compared to [74], which dynamically generates UEs and defines their position using x and y co-ordinates within a defined coverage area as observed below. Mist computing simulation was also added in [74] as well as an example file dedicated to comparing mist, edge, and cloud computing capabilities.

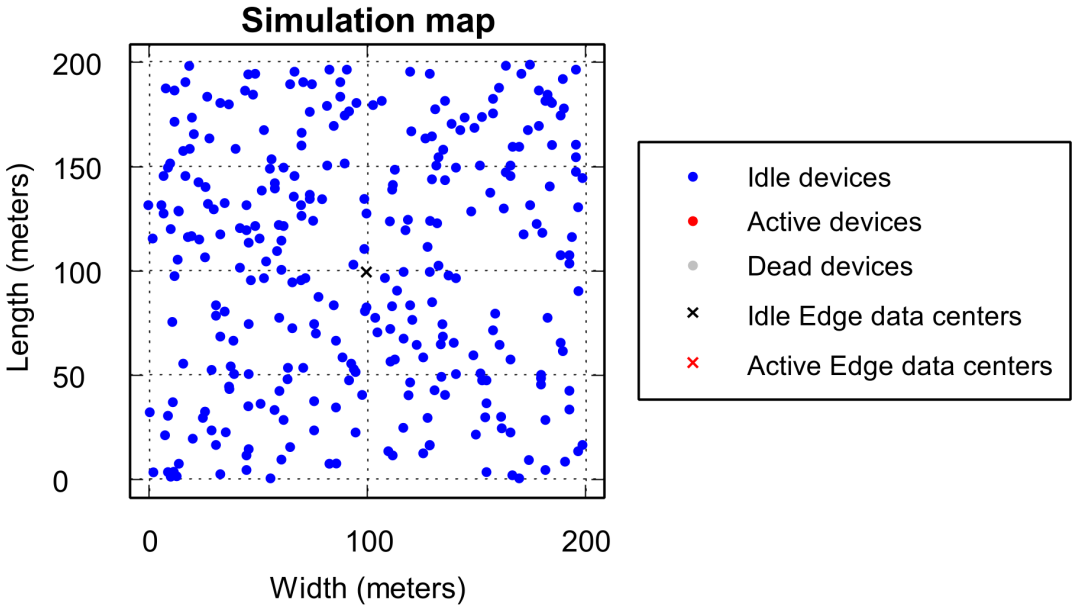


Figure 3.3. Mobility simulation map

The software class hierarchy can be seen in Figure 3.4, which details how the different classes within the simulator work together to provide the output.

The MainApp Class calls upon ScenarioFactory which in turn ties into several subclasses to gather variables from XML files which are used to organise and declare different environmental variables such as Edge Device specifications as well as Client Device specifications.

The simulator itself leans heavily on the CloudSim library, often utilising functions, and methods on underlying layers to provide key functionality.

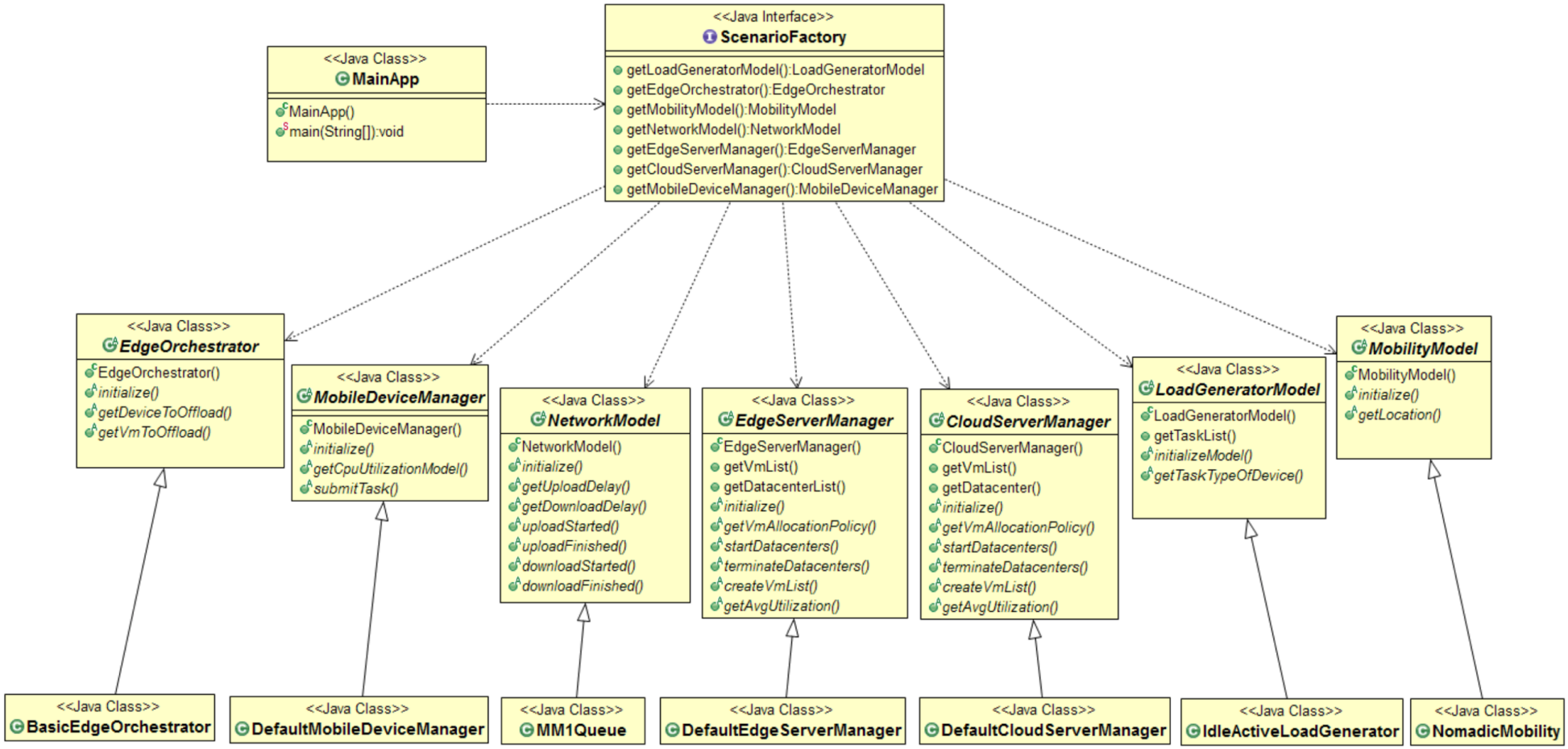


Figure 3.4. Java Class Hierarchy of EdgeCloudSim app

3.4.2 Modules

The simulator consists of several modules that work in conjunction to provide simulated network events which are as outlined below:

MAIN APP

The main application (MainApp) class in the model is contains and references various variables that draw dynamic variables such as directories and configuration files from other documents. It invokes the use of other supporting classes to start the simulation with the parameters required and produces the SimLogger data to the console.

SCENARIO FACTORY

The scenario factory invokes the remaining classes that can then use the configuration files in XML format to apply settings and resources for VM, UE's and Cloud applications. The scenario factory is responsible for invoking classes from CloudSim and instantiating them to the simulator.

EDGE ORCHESTRATOR

The edge orchestrator module administrates the system, it is responsible to assign and allocate incoming requests using a probabilistic approach. The edge orchestrator module deploys a WLAN located in a stable physical place with a predetermined wireless coverage area.

MOBILE DEVICE MANAGER

The MobileDeviceManager class invokes the CpuUtilisationModel. Its role is to ensure and simulate the mobile devices being simulated within the software, which are each connected to the network and portrayed as using one of the applications that have their own utilisation requirements.

NETWORK MODEL

The NetworkModel class is responsible for providing the number of mobile devices as well as the simulation scenario.

EDGE SERVER MANAGER

The EdgeServerManager contains specifications for the Edge Servers including their detail hardware specifications. Modifying this file can reflect on the simulation.

CLOUD SERVER MANAGER

Contains the parameters of the Cloud server should they need modification to match today's standards.

TASK GENERATOR MODULE

The load generator module uses an XML configuration file modifying variables found throughout the code to simplify access and enable quick scenario deployment. Load distribution is handled using a Poisson distribution, a traffic model that is most used in the communications industry, that distributes computational traffic to the Virtual Machines, after tasks have been allocated the by end user devices. The tasks that can be created by the simulation are unlimited in amount and in typical usage, it takes 20 minutes to create and simulate 50k tasks using a core i7-5600u processor and 8GB DDR3 RAM [83]. Offloaded tasks technique:

$$\mathrm{P}(X = x) = \frac{\lambda^x e^{-\lambda}}{x!} \qquad (3.1)$$

The default approach of the task generator is to use poisson distribution as seen above in Eq. (3.1 where λ is the mean number of occurrences in the interval and e denotes the Euler constant.

A generic model is applied for the purpose of the research, portraying the simple requirements of the system without any complex functionality, but providing a strong foundation of what researchers and the industry can further enhance. In essence, the module attempts to recreate a Wireless Local Area Network (WLAN) environment as opposed to a Mobile Radio Access Network

(MRAN) environment, and therefore discounts propagation of radio waves in long distance communication along with accompanying delays incurred and other variables that need to be taken into consideration.

The simulation environment used for testing purposes by the authors is one that reflects the state of a university campus, where users transition between a less frequented space (i.e., the administration building) to a greater frequented space (i.e. the library). This is then applied to a two-tier architecture and the Edge Orchestrator Module is responsible for handover between the two spaces, considering network traffic load.

Application models

The load generator module uses a combination of scripted use-cases to simulate a variable environment for load testing purposes. There are four application classes used by EdgeCloudSim[85]; Augmented Reality, Health (automation, smart devices, monitoring), Compute Intensive (computationally heavy tasks, simulations, renders etc.) and Infotainment Applications. Each application class varies in its application of task arrival distribution, delay tolerance and task length.

There are several features that are missing from EdgeCloudSim including advanced and calculated cache allocation methods, despite this several researchers have attempted to tackle the issue albeit in theoretical form [63], [22], [60], [83].

Mobility Model

A simplistic nomadic mobility model is used to plot x and y coordinates of clients. This is periodically updated according to the dynamically managed hash table, by default, a nomadic mobility model is used.

Requested Features for EdgeCloudSim

Needed features found on their GitHub page [86] indicate that the simulator requires the following features:

- Mist computing features (executes task on mobile devices via ad-hoc networking)
- Incorporating cellular access network model into EdgeCloudSim (3G/4G/5G)
- Task migration among the Edge or Cloud VMs
- Energy consumption model for the mobile and edge devices as well as the cloud datacentres
- Adding probabilistic network failure model by considering the congestion or other parameters such as the distance between mobile devices and the Wi-Fi access point (geolocation or local based location profiling)
- Graphical User Interface to interact with the simulator and display network topology

Ensuring the optimal use of resources is vital to any project. To ensure that the thesis accounted for any changes within the simulation software that could produce clearer and better results, I was adamant on keeping up with any changes made to the simulator over the course of its existence. As the project had been made open source to allow for public contribution [86], I expected that frequent changes to the simulation software were to be likely and therefore, always referred to the GitHub page of the project to ensure that my version of the simulator was always up to date.

Despite the increase of academic and industrial research efforts within MEC over the course of this thesis, I found that only minor changes were being implemented and those were usually orchestrated by the original author(s) of the software itself. This does not rule out the usage of the simulator however, as over 15 academic articles from the years 2018 to 2019 utilised or cited the simulator as part of their research [22], [60], [73], [83], [87], [88], [24], [89], [11], [22], [74].

In most cases, [22], [60], [83], researchers opted to extend the base functionality provided by the simulator in efforts to cater the software to their

individual needs. One of the key requirements that the software lacks is the integration of an energy model, particularly because of the importance of Mobile Edge Computing within IoT environments [90] which can affect the battery life of sensors and actuators located at the network edge. Maximising battery life potential, or indeed utilising perpetual energy methods to constantly power sensors [91], [92], can ensure that vital Edge Devices (EDs) remain powered, whether that be using wireless power transfer [92] or using solar-powered methods [91].

EdgeCloudSim was initially uploaded on the 18th of Feb 2017 and seemingly updated until the 22nd of July 2019 with the following changelog [86]. Additional works include [93] who aim to address the shortcomings of both the mobility module and the load generator module.

Freymann, et al. [93] state that the mobility module is predetermined in nature, and therefore provides unrealistic results with predictable outcomes. Additionally, the load generator module initially creates the task queue as opposed to generating each task or a list of tasks progressively which can cause memory overloads.

3.4.3 Assumptions

Unfortunately, the main repository itself, apart from the academic research efforts made, lacks accessibility to those unfamiliar with some aspects of software development. When *forking* the repository or *cloning* from GitHub, it can prove tedious to run the included example applications depending on the IDE being used. Simulation software is often run within a containerised environment or an IDE. This has several advantages for anyone simply wishing to modify some parameters within four sample scenarios included within the simulation software.

As aforementioned, EdgeCloudSim relies heavily on the CloudSim library for key background components such as VM integration and network backbone integration. Due to the extensible framework however, new features can be

constantly added without reworking the foundation of the application itself. On October 31st 2020, a major update was released [94] implementing the use Machine Learning for Vehicular Edge Computing. Despite the initial functionality and the class hierarchy largely retaining their form, a major update in the form a machine-orchestration edge orchestration model was implemented for multi-tier multi-access to the Vehicular Edge Computing (VEC) architecture [58].

VEC, as noted by Sonmez, et al. is an emerging technology offering in-vehicle applications the ability offload computational tasks to an edge infrastructure that has been streamlined for vehicular applications [94], there is a similar possibility of a future where connected services offered to vehicles via Edge Computing may make on-board applications redundant [95]. Sonmez, et al. incorporate the use of a two-stage ML based orchestration method that can intelligently identify the ideal location to offload tasks to maximise success.

The first stage is set to predict the chances of success, and the second stage predicts the expected service time. Their model is simulated in EdgeCloudSim, and findings published [94]

3.4.4 Validation

To ensure the integrity of the simulator and its results, a simple test of validation to confirm that the individual components react as expected, was held. The first validation method was to ensure that the simulator was acquiring the correct properties as defined by the XML file for its properties, a test simulation was then conducted, and results plotted for comparison and analysis purposes.

3.5 Simulation candidate 2: PureEdgeSim

Like its predecessor, [73], PureEdgeSim is an event-driven simulator based on a variant, or independent fork, of CloudSim called CloudSim Plus taking advantage of features including native support for discrete events simulation used during the communication between its components [74].

Scouring the documentation of CloudSim Plus explains that despite its close following of the original CloudSim, CloudSim Plus has a plethora of additional features that were a cornerstone in the development of PureEdgeSim.

Comprehensive documentation is included with PureEdgeSim that outlines the details of the simulator and its respective modules as well as core functionality. Each modules core functionality and extensibility is explained in detail by the author as well as intricate instructions on how to create a new example by extending its mainApplication class.

Despite the advantages of CloudSim, the authors of [82] decided that they wanted to pursue their own take on the popular simulation software due to the following shortcomings:

- Limited documentation.
- Amount of duplicated code that jeopardised maintainability, extensibility, and testing.
- Absence of functional/integration tests to ensure simulators correctness and validity.
- Absence of design patterns.
- Lack of conformance to some software engineering practises and recommendations such as SOLID principles.
- Lack of a more organised package structure paving the way for a better understanding and modularity of the project.
- Lack of a better class structure to allow third-party developers to implement missing features into the framework without needing to change core classes.

To tackle these issues, CloudSim Plus indicates that its main contributions are:

- Improved class hierarchy and code.
- Increased application of reusability principles.
- Overall review and improvement of code documentation.

- Re-structuring of project modules and packages to simplify usage and to improve separation of concerns (SoC) principles.
- Addition of integration tests to cover overall simulation scenarios.
- Completely new set of features described in detail on the official web site.

Like [80], the authors of [82] also continued to make the project an open-source initiative and there is still an active interest in the project. This observation further pertains that comprehensive and detailed documentation efforts within open-source software can help ensure the project remains valid and actively contributed to over the course of its lifetime [82].

Several key changes in formatting as well as many improvements over its predecessor, including (i) Extensibility improvements, (ii) Reduced code duplication, (iii) Tests and code coverage. In addition, extra features were implemented such as (i) Dynamic arrival of cloudlets and VMs, (ii) New datacentre brokers, (iii) Re-engineered network module and new set of interfaces, (iv) Event listeners, (v) Builder classes, (vi) Integration tests, (vii) Software design quality metrics [82].

3.5.1 Hierarchy and Design

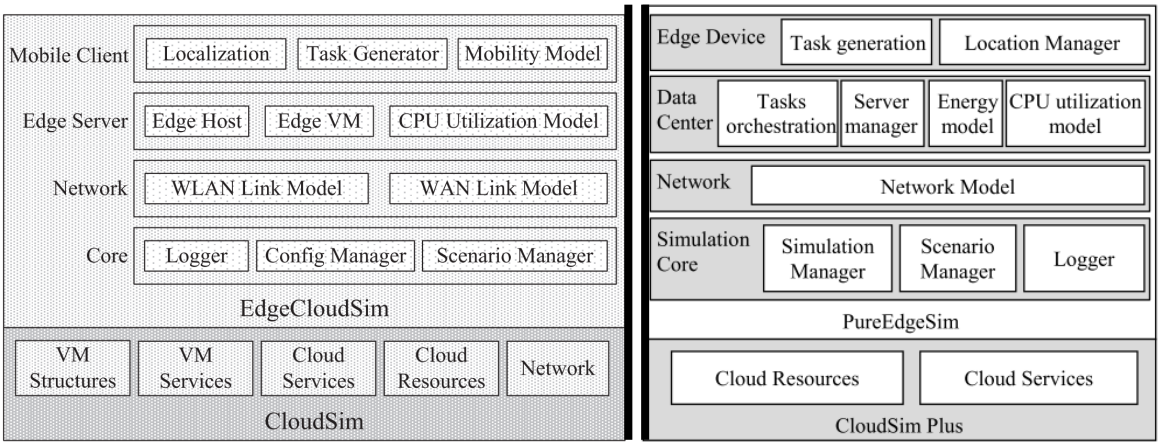


Figure 3.5. Side-by-side comparison of layered architecture

Figure 3.5 portrays the differences in architecture between the two-simulation software, a notable difference here is the custom Network Model used by

CloudSim Plus. CloudSim Plus drops usage of the heavy modules to ensure a more streamlined simulation process without drastic overheads, more suitable for academic purposes where many use cases can be observed without as much computational investment.

The author in this case has created a custom network model rather than relying on the libraries network model. Upon closer inspection, the network module utilised in PureEdgeSim contains integration of energy consumption monitoring but lacks WLAN integration, as the link in question over this research does not really on a WLAN connection, it is safe to say omitting that module should not be a cause for too much concern.

The lack of WLAN in this case does not affect us greatly as our emphasis is on resource allocation within the MEC environment. The integration of CloudSim Plus however, included major performance enhancements over task scheduling but most importantly, the use of multi-threading. Additionally, CloudSim Plus builds upon the modular nature of the original, giving academics and researchers the chance to easily implement more complex custom modules and algorithms for testing.

3.5.2 Modules

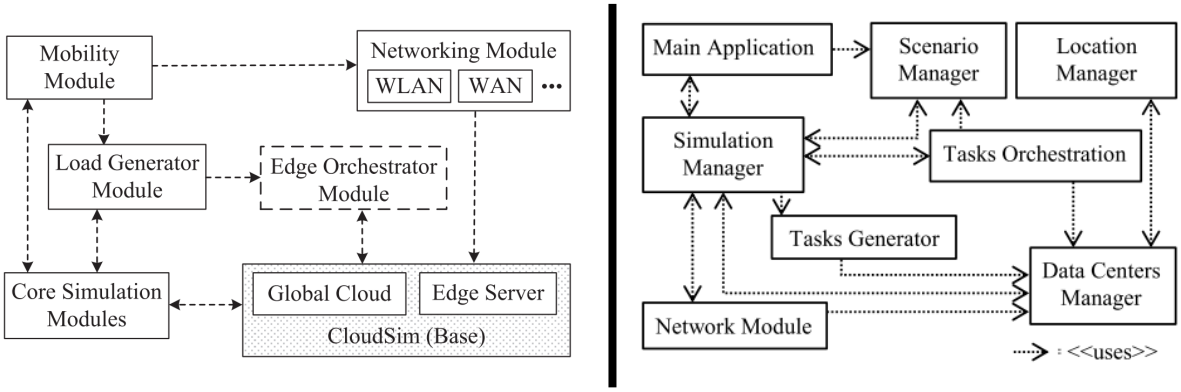


Figure 3.6. Relationship between EdgeCloudSim and PureEdgeSim

Figure 3.6 shows us the difference between how the modules inter-operate within the two respective simulation environments. To improve performance, it

is noted that PureEdgeSim sheds the use of many core libraries of CloudSim, instead relying on the most important to keep performance overheads low and improve simulation speed whilst maintaining the required accuracy.

The simulator consists of several modules that work in conjunction to provide simulated network events which are as outlined below:

SIMULATION MANAGER

The Simulation manager module represents the core of the simulation by co-ordinating tasks between all the modules involved within the simulator. It also handles the interfacing between CloudSim Plus and the PureEdgeSim modules. It consists of two main classes: 1) Simulation Manager, responsible for initialising the simulation, starting it, and scheduling its end. It also schedules the tasks generation, displays the simulation progress, and prints the results. 2) Simulation Logger, computes the results and displays them at the end of every iteration, subsequently saving them in a CSV format.

DATA CENTRES MANAGER

The data centres manager module extends CloudSim Plus by extending the Datacentre simple class. It contains the properties of edge devices and their corresponding methods such as location, mobility, energy source and the battery capacity. It also contains the Server Manager class which generates the different servers and devices, their hosts, and their virtual machines according to the configuration files modified by the user.

TASK GENERATOR

This module is responsible for generating tasks eventually processing by allocated computational resources. A collection of three example applications are used for reference; e-health, smart-home, and augmented-reality. Each application has specific parameters which can be modified by editing the configuration files. The example provided in [74] and therefore suitable for demonstration purposes.

LOCATION MANAGER

This module enables mobility by assigning users with x and y co-ordinates which can then be manipulated according to the mobility model the user wishes to utilise. This is particularly useful when simulating environments such as connected vehicles, each vehicle can be assigned an initial location and then modified over the course of the simulation.

NETWORK MODULE

Consists primarily of the network model. The author in this case created a custom module to consider the network load at each instant of the simulation and changes the allocated bandwidth for each task being transferred according. In essence, it improves upon the standard network module included with CloudSim Plus is its ability to adapt to network bandwidth availability and adjust the amount of bandwidth available per user/application. It also bears network load into account. A load balancer sample has not been included in the simulation.

TASK ORCHESTRATION MODULE

Like the previous module, a custom Task Orchestration Module has been designed for simulation purposes. The Orchestrator contained in the module has the task of the decision maker.

It decides whether to offload the task or execute it locally and where to offload it depending on the algorithm in question and the architecture that is used. It also can be extended by the user utilising the Orchestrator class. Several classes of tasks are available within the simulator which have been generified to test the performance of the simulator.

Here, we have not manipulated the task creation process but there are some assumptions that have been made such as the size of the data and the number of instructions required to complete the given task.

SCENARIO MANAGER

The scenario manager module is responsible for loading parameters from the custom configuration files within the simulator and parsing them to be used by the Tasks Generator module. It also loads the cloud data centres; the fog data centres and the edge device characteristics that are used by the Data centres manager module.

Finally, it loads the network settings used by the Network Module, the architectures and algorithms used by the task's orchestration module and other simulation parameters including simulation delay, log parameters, etc. required by the Simulation manager. It consists of two important classes: the File parser and the Simulation parameters.

Simulation Input Parameters

As with the previous simulator, [47] have made it relatively simple to input parameters for the simulation using an XML file where variables are contained and extracted to specify to the simulator the various parameters used to run the simulation shown in TABLE 3.5:

TABLE 3.5
SIMULATION PARAMETERS, PUREEDGESIM

Parameter	Description
Simulation_time	The simulation duration (in minutes)
Initialisation_time	The time required to generate the different simulation entities
Parallel_simulation	Enable or disable parallel simulations
Update_interval	The interval between simulation events (in seconds)
Pause_length	The pause between iterations (in seconds)
Display_real_time_charts	To display or not the simulation results in real-time
Auto_close_real_time_charts	Auto close real-time charts after the end of each iteration
Charts_update_interval	The interval of refreshing real-time charts (in seconds)
Save_charts	Save charts in ".png" format
Wait_for_all_tasks	Wait until all tasks get executed or stop the simulation on time (when the simulation time set by the user finishes)
Save_log_file	Save the log file
Clear_output_folder	Delete the output folder at the beginning of each simulation
Deep_log_enabled	Enable deep logging

Location Manager Parameters	
Length	The simulation area length (in meters)
Width	The simulation area width (in meters)
Edge_range	The transmission range of Edge devices (in meters)
Fog_coverage	The radius of area covered by each Fog server (in meters)
Speed	The speed of mobile devices (in meters/second)
The servers manager settings	
Min_number_of_Edge_devices	The number of Edge devices at the beginning of the simulation
Max_number_of_Edge_devices	The number of Edge devices at the end of the simulation
Edge_device_counter_size	The growth rate in the number of devices between iterations
The network model settings	
Wlan_bandwidth	The local area network bandwidth (in Mbps)
Wan_bandwidth	The backhaul network bandwidth (in Mbps)
Wan_propogation_delay	The propagation delay (when sending data/task to the Cloud) (in seconds)
Network_update_interval	The network model refresh interval (in seconds)
The tasks orchestrator settings	
Enable_registry	If enabled, a container will be pulled from the registry (by default) before the execution of the offloaded task
Containers_deployment	The containers deployment strategy that is defined by the user
Enable_orchestrators	Deploy the orchestrator to a physical device, If disabled, each device will orchestrate his tasks.
Deploy_orchestrator	To deploy the orchestrator to the Cloud, Fog, or any custom location
Tasks_generation_rate	The number of tasks generated by each device every minute
Orchestration_architectures	The computing paradigms that will be used
Orchestration algorithms	The orchestration algorithms that will be used/evaluated
Energy model parameters	
Consumed_energy_per_bit	The energy consumed when transferring 1 bit (in wh)
Amplifier_dissipation_free_space	The energy consumed by the amplifier in free space channel (in wh)
Ampilifier_dissipation_multipath	The energy consumed by the amplified in a multipath channel (in wh)

The above simulation parameters can be set before starting the simulation, a build takes places using the newly provided simulation parameters and the

simulation is then run according to the parameters provided. This simple text entry method simplifies changing the parameters and simulating different scenarios without having to configure or modify the source-code of the simulation software.

As observed in TABLE 3.5, many of the parameters provided rely on smaller modules, hence giving a modular approach to simulation which in turn works out perfectly for the scope of this research as the core focus is to improve the orchestration of the edge server.

Output Format

A key reason to transition to PureEdgeSim was the availability of pre-formatted files that contained the simulation logs in a CSV format. Additionally, during simulation, the authors of [74] have scripted in a graphical real-time analysis of the simulator running as seen in Figure 3.7, indicating the kind of data that will be received as a result, a log of the output can be seen in TABLE 3.6:

TABLE 3.6
EXAMPLE CONSOLE LOG OF PUREEDGESIM

Timestamp	Simulation Time	Source	Message
2020/11/03 17:51:53	0 (s)	ServersManager	Datacenters and devices were generated
2020/11/03 17:51:53	0 (s)	SimulationManager	Orchestration algorithm= ROUND_ROBIN - Architecture= ALL - number of edge devices= 100
2020/11/03 17:51:53	0 (s)	SimulationManager	Simulation: 1, iteration: 4

PureEdgeSim also contains real-time monitoring on the simulations progress, giving researchers the ability to view a particular simulation play out in real-time as seen in Figure 3.7, though this may be somewhat limited to non-parallel execution.

Execution times were also drastically improved in version 5.2, due to some major changes made by the author to accommodate for greater tasks to be scheduled on a greater number of devices[47].

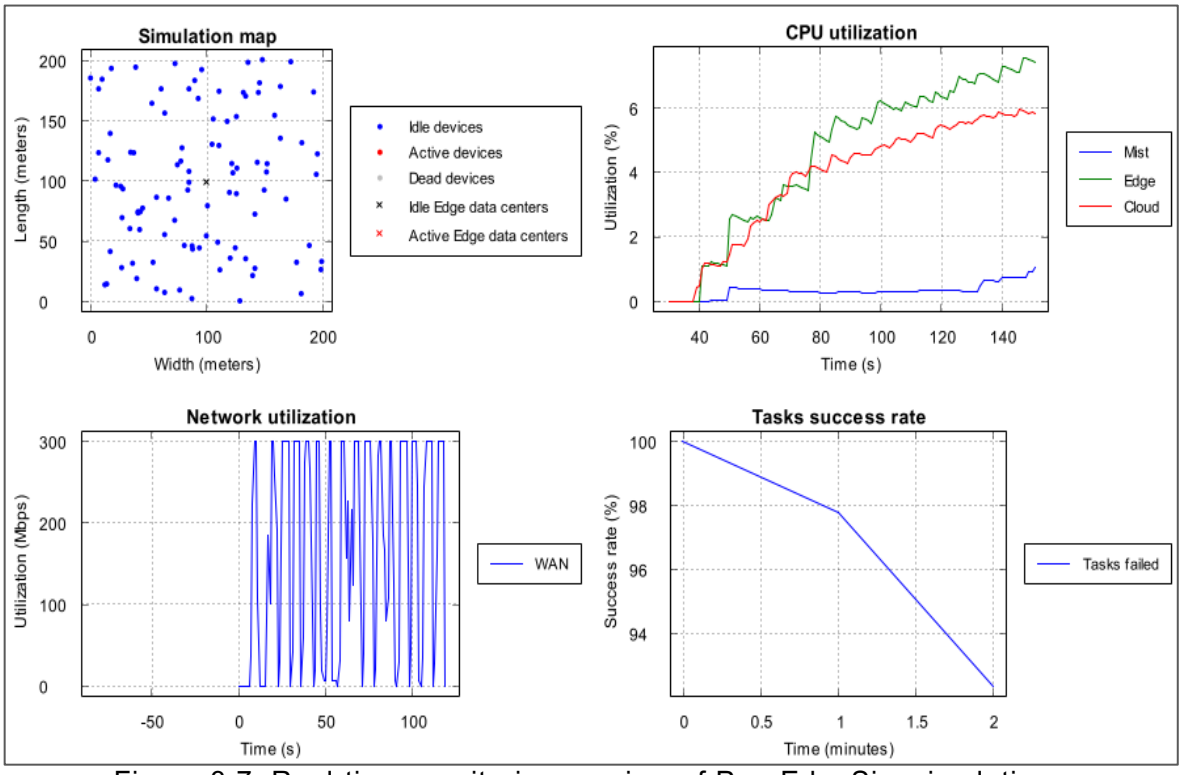


Figure 3.7. Real-time monitoring preview of PureEdgeSim simulation

3.5.3 Assumptions

Default modules include assumed and static parameters for the models, as seen in TABLE 3.7, but it must also be considered that the extensibility of PureEdgeSim allows for integration of custom modules to almost any aspect, allowing the replication of real-world event simulation using custom mobility and attenuation models.

TABLE 3.7
PUREEDGESIM ASSUMPTIONS

ASSUMPTIONS	
COMPUTING RESOURCES	End devices are assumed to have CPI, memory, storage and depletable batteries to process data and execute applications
NETWORKING CONNECTIVITY	We assume that all discoverable devices are connected to the network and are requesting resources in some form or manner

APPLICATION EXECUTION	Assumption that applications can be executed on edge devise, and that the application execution time and resource usage can be modelled.
WORKLOAD GENERATION	Assume that we can generate the workload according to the given distribution model, which helps to simulate the behaviour of the edge computing system under various conditions.
ENERGY CONSUMPTION	We assume that the energy consumption of devices is proportional to the amount of computation and data transmission performed by the device.
TASK SCHEDULING	Assume that tasks can be scheduled to edge devices based on different task categories as well as criteria such as proximity to the source, resource availability and energy consumption.
MOBILITY	We assume the position of the mobile devices as well as their travel trajectory.

3.5.4 Validation

Initial tests were run using [47] to ensure simulator validity. Both simulators came pre-installed with basic algorithms such as Round Robin and Random task allocation. The results produced were equivalent however, PureEdgeSim showed increased simulation speed of 50%.

3.6 The Final Selection

Based on the comparisons conducted above, it was clear that the more modern simulator with improved extensibility and greater performance was the final choice. When given similar scenarios, PureEdgeSim greatly outperformed EdgeCloudSim in performance and accuracy whilst further allowing for greater customisation. Though initial set-up required some configuration with PureEdgeSim due to the use of Maven, the result paid off in ease of use and drastically improved overall simulation times, allowing more simulations to be run.

Additionally, the ease of use when translating output data to a more usable format proves formidable for EdgeCloudSim, as the process is entirely manual in nature and the application requires that log files are extracted and imported into an external CSV/log parser before any meaningful conclusions can be

drawn from the data generated. Simulation times are severely lower, where a sample scenario containing 200 devices takes approximately 5 minutes on the same hardware as opposed to < 10 seconds on PureEdgeSim.

Some of the sample modules within the simulation includes a test use case of a Fuzzy Logic Orchestration algorithm, the results of which can be viewed in

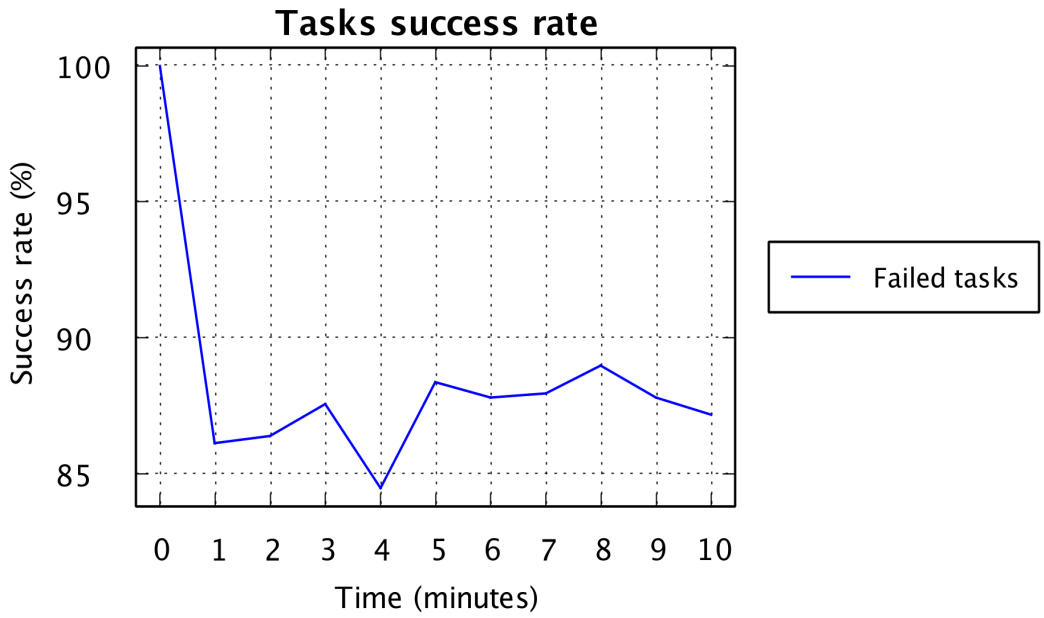


Figure 3.8. PureEdgeSim Fuzzy Logic

Figure 3.8 where the task success rate can be seen fluctuating over the course of the simulation.

Additionally, building on the research of [47] implementing the reinforcement learning algorithm using some of the fundamentals provided ensured a smoother transition process where we were able to test algorithms more efficiently.

Ultimately, the simulators used in this project were academic in nature, therefore no GUIs were provided with either of them, so knowledge of Java was required to implement our algorithms.

Despite this, the ease of use offered by PureEdgeSim showed that it was the most suitable candidate for the purposes of this research and hence, it was ultimately chosen as the simulator that we would go ahead with. Parameterisation using XML files is used to hard code most of the environment variables within the simulator increasing ease of use.

3.7 Final Test: PureEdgeSim

To initially test our simulator, we will use the predefined examples in PureEdgeSim to run an algorithm based on Fuzzy Decision Tree from [47] This example is provided within the library of PureEdgeSim and can be for comparison purposes when testing our proposed algorithm.

The example provided within the simulation uses a 3-tier classification approach for the data output from the Fuzzy Logic generator, namely the data is classified within either ‘low’, ‘high’ or ‘medium’ settings.

Although fewer tiers in this system helps to enhance the speed of the algorithm, there is a trade-off between optimum performance and speed which will unfortunately require a trial-and-error approach however, my theory is that giving the system classification and applying Fuzzy Logic within the learning process will help to improve overall performance of the systematic approach. Transfer learning will prove to be a key concept within the design of this algorithm as communication between edge-nodes will be particularly useful for providing an awareness about the network environment to the nodes. To enable a transfer of what has been learnt by the Edge Network (EN), the system will attempt to use dynamic variables acquired through each Edge Node and communicate parameters on a sporadic basis.

Retaining our simulation parameters as outlined above, a test was conducted to ensure to determine the performance of competing algorithms as devised by

[47] to sample the application on install and to generate control data for our manipulated parameters.

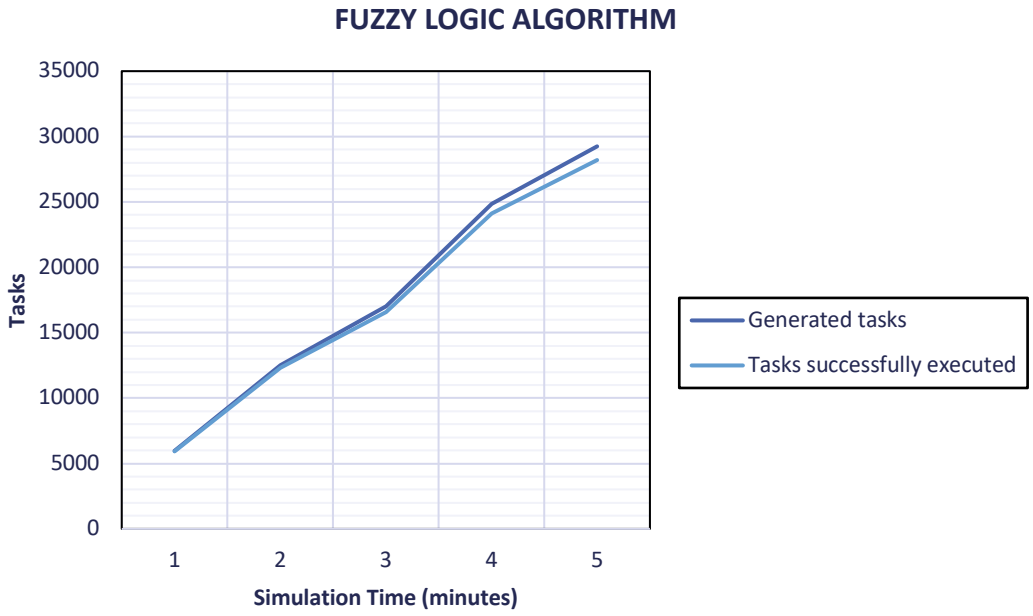


Figure 3.9. Generated against successful tasks (Fuzzy Logic)

Figure 3.9 and Figure 3.10 portray the effectiveness of utilising a Fuzzy Logic approach included as one of the examples within the simulator. It applies a simple approach to dictating variables based on input variables, applying vague classification within a group of strict variables.

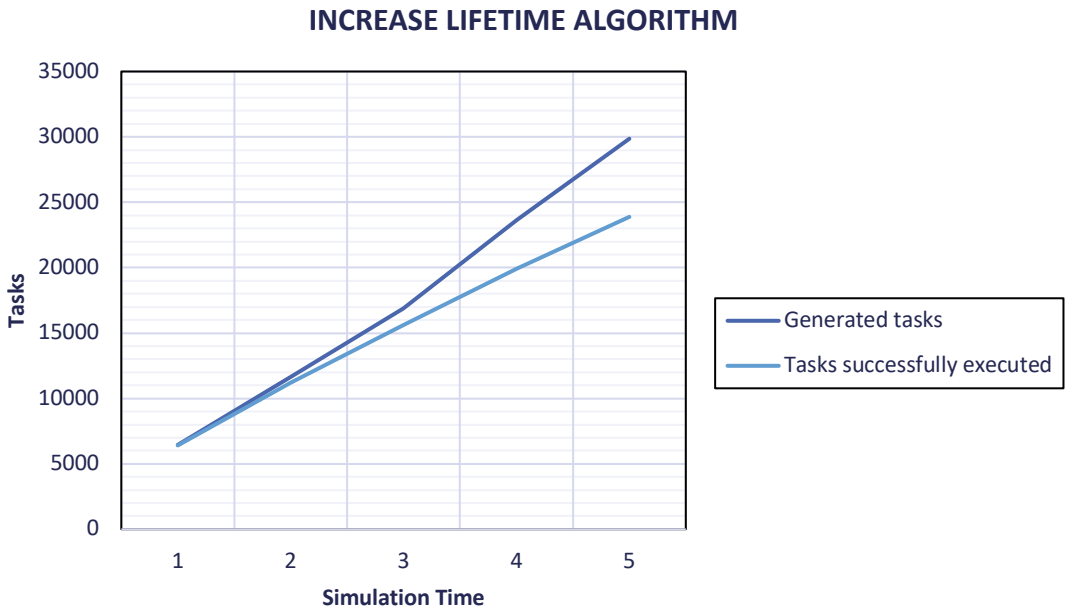


Figure 3.10. Generated against successful tasks (Increase Lifetime)

The Fuzzification process in the supplied use case is broken into two, where one is responsible for the output RULES and the other for determining the classification of variables within the fuzzy scope. The above approach is touched upon in [74] where the authors simulate a Fuzzy Decision Tree and *State of the Art* approach that offers improved functionality using Deep Reinforcement Learning and Transfer Learning.

As seen above however, the number of successful tasks completed by the simulator using a Fuzzy logic approach improves drastically, resulting in a much lower failure rate. The module in this case is implemented based of research conducted by the authors of [96], a library that implements Fuzzy Logic, fittingly called jFuzzyLogic.jar, is simply integrated and referenced within the code and provides most of the functionality.

Parameters are then passed on to the simulation in the Orchestration module, taking advantage of the Fuzzy approach, and then utilised to gain better efficiency in the simulation.

The figures shown above can be easily acquired running one of the examples provided within the simulation bundle but are restricted in the sense that they only compare to one other algorithm which is the increase lifetime.

The Fuzzy Logic simulation also uses increase lifetime algorithm as a base, but instead feeds it fuzzy variables which in turn improves system performance. In a similar fashion, over the course of this research, we wish to utilise the fuzzy output of the fuzzification process to give the algorithm the ability to 'understand' the rough definition the action that it will be taking to improve the optimisation process.

3.8 Summary

The advantages of using a simulator verified by the community meant that we could focus on the research with tested and validated contributing factors to recreate an accurate testing environment, rather than theoretical implementation of the devised algorithm.

A comparison of simulation software led to PureEdgeSim being the right selection for the purpose of this research. Although the frameworks are based on similar architecture, PureEdgeSims purpose made nature for Edge networks and increased efficiency of up to 40% as well as increased support for granular control made it the better choice.

It also came with well written examples incorporating fuzzy logic as orchestration algorithms, helping to understand how modules work in conjunction and providing a strong foundational basis to build the improved algorithm.

Chapter 4 Simulator Setup and Single Layer RL

4.1 Introduction

In this chapter, we will develop and utilise an ε-greedy Q Learning algorithm to optimise resource allocation in edge networks, testing in our selected simulator, PureEdgeSim. To ensure comprehensive coverage of all aspects within the simulator, each module was individually chosen and dissected, applying real-world scenarios, and ensuring that each element was carefully thought through before an allocation algorithm could be designed.

Firstly, we will go through the various modules involved that will ensure our simulation will cover the major areas that need to be addressed over the course of this research. This will include real-world figure implementation of hardware capabilities and estimation of population and their mobility over time.
The modules that require addressing in the simulator will be portrayed one-by-one, exploring in-depth analysis of each one and its role within the model.

TABLE 4.1
SIMULATION PARAMETERS

Simulation Parameters	Value
Simulation duration	30 mins
Min number of edge devices	100
Max number of edge devices	1000
Simulation area	2000×2000
Edge and fog bandwidth	1300 Mbps
Cloud bandwidth	10000 Mbps
Edge devices range	200

TABLE 4.1 contains some of the most crucial simulation parameters that define the control of the research. Over the course of the remaining chapters, we will

consistently retain the parameters to ensure that simulations are comparable in respect to each simulation run.

4.2 Simulation Environment

4.2.1 Task Modelling and Classification

Additional metadata was implemented in the tasks following the changes made by [54] to increase network awareness of the of tasks requested by the device d. Ensuring these parameters were accessible by the system was paramount to the algorithm recognising the requirements of the task and thus the capabilities required by the chosen offloading device to ensure adequate QoS.

The comprehensive task creation module tracks the size of each task created during the course of the simulation run, values from the respective tasks are then used to populate QTables to feed the RL algorithm. The parameters that are tracked from each task can be found in TABLE 4.2:

$$T^{d,k} = \{ start, end, dl, size, mi \} \quad (4.1)$$

TABLE 4.2
TASK PARAMETER DENOTATION

Parameter	Variable
Task Start Time	start
Task End Time	end
Deadline for task	dl
Incoming data size	size
Instructions needed to process the task	mi

Introduction of the metadata also made it possible to further identify the tasks to supplement the reinforcement learning algorithm with provided parameters. Each round of RL introduced metadata that provided contextual information of rewards and priority to each task. This made recognition of previously allocated

rewards and priorities identifiable by the algorithm to accurately construct Q Tables and implement RL.

As the Q-learning algorithm requires a finite state space, discretising the continuous values is necessary, thus we use Fuzzy Logic to define the following membership functions; **taskLength, taskMaxLatency, localCPU, localMIPS, avgEdgeCPU and avgCloudCPU** in the following manner in piecewise linear functions using Centre of Gravity (COG) method [97]:

taskLength:

1. Low:

$$\mu_{low}(x) = \begin{cases} 1 & if\ x \leq 0 \\ 1 - \dfrac{x}{20000} & if\ 0 < x \leq 20000 \\ 0 & if\ x > 20000 \end{cases} \quad (4.2)$$

Remaining membership functions can be found in Appendix A that define membership logic retained throughout all algorithms implemented in Chapters 4 and 5.

4.2.2 Mobility Modelling

For the most accurate results, a mobility model was required to ensure that client devices are not simply static within the cell.

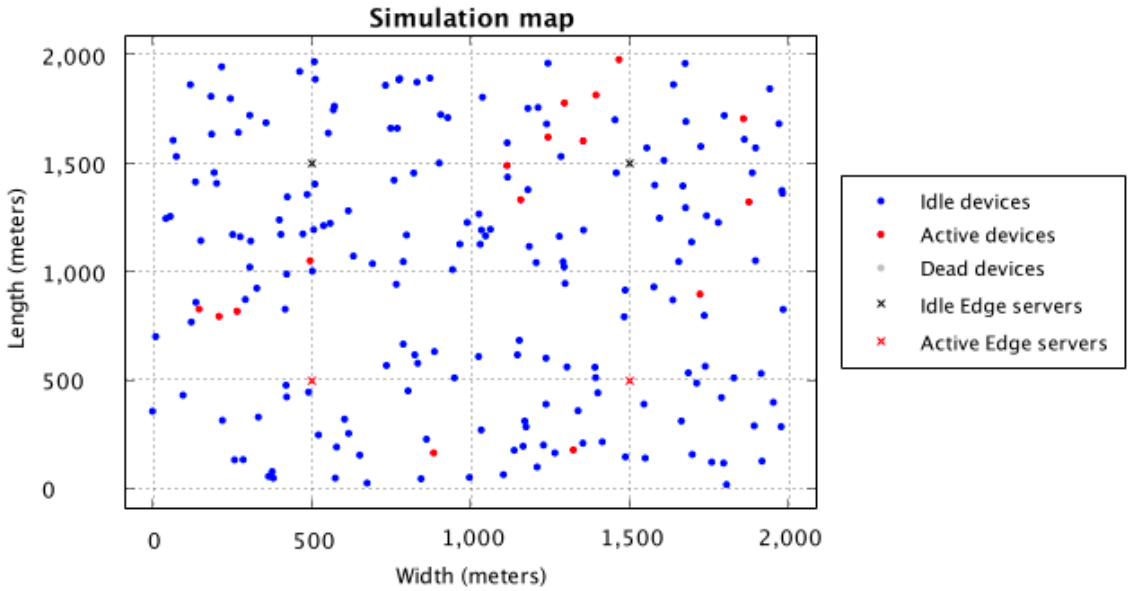


Figure 4.1 Sample real-time mobility view

The mobility model included in the simulator addresses the user device's mobility within a period and how that affects network conditions. From our knowledge in networking and routing tables, we infer that once a user is out of range from its nearest access node, a handover is conducted to the next node that the user is within range of. Figure 4.1, depicts how mobility simulation is visualised over the course of simulation time, where users are travelling around the cell space.

Often, this means that the node must re-establish its routing table. The method applies to any type of connected device that uses an AP to gain access to a wider network, including the RF domain.

The handover process ensures a smooth, and often unnoticeable transition of a user from one access point to another within a given domain, removing the need for re-authentication should any be required.

Continual adaptation of the network is paramount to a good user experience. Although it must be heralded that mobility is only of the reasons that a network should be continuously adapting to its network environment, other variables to consider are battery life, the disconnection of a device.

For example, an access point that is based in the city centre for public use must perform drastically different to an access point located within an industrial site as the industrial site is likely to have repeat visitors from the employees that work there, therefore to ensure optimisation, a network administrator or an orchestration algorithm must be optimised to continuously authenticate and de-authenticate the gradual change of users as opposed to the public access point which must be dynamic at all times.

Performance expectations of an access point contained within a company, however, are very different those within the public domain, users in a company expect rapid access to higher bandwidth services such as teleconferencing or uploading large files whereas those within a public domain wouldn't have the

same expectations of a public hotspot (though that doesn't mean they wouldn't appreciate them).

It is to be understood that 5G aims to tackle these real-world issues by offering low-latency, high-bandwidth access to all users but an orchestration algorithm must be dynamic and adaptive to its network environment as these conditions are subject to change in the blink of an eye.

Despite the setback on the data model obtained, the visualised scenario was still viable and would be the one used, namely, a dense metropolitan area providing 5G connectivity for a vast number of users within a limited space within rapid handover expectations.

Due to its growing popularity, and the interesting challenge it would pose, I believed that the city centre or high-street would be the ideal situation to base my simulation in however, this would also cater for another specific type of application, autonomous vehicular communication[98]. Introducing this application within the mobility module would mean that the algorithm designed had to be adaptive to the network environment, and inter-communicate with peer nodes rapidly.

This also meant taking into consideration the toll on computation in short bursts from node-to-node within a limited area however, the amount of space the simulation would take place in was now to be much larger than initially expected.

For the extents and purposes of this research, the default mobility model was lightly customised to ensure that simulated devices were randomly moved around the environment.

4.2.3 Network Modelling

The network will simulate a 5G environment as much as possible. This will include integration of the specifications devised by ETSI, including those that

may have been released post-simulator. Any Classes or Modules that may need adding as a result, will be devised, and developed to ensure that the simulator meets the needs of the research.

Each sample application simulated different environments using different variables. For this thesis, the most relevant was sample_app4 which used fuzzy logic on the Edge Orchestrator module to allocate resources where they were most required.

The weighting system was applied on the Edge Orchestration module to emulate a ‘human’ approach to administration and allocation of tasks [85]. There is, however, a limitation to the approach used by [73] which co-ordinated tasks using the three-tiered fuzzy logic approach. The orchestrator in their case is aware of the network resources available as well as the status of the edge server, yet a protocol of discovery has not been outlined.

The final tier is the global cloud server that can then be communicated with to back information up as required. Despite this being the most-recognised approach, the system must also co-ordinate the backup process.

Additionally, to ensure that no resources are wasted, edge servers must manage downtimes on uplink and downlink to ensure that no redundancy is introduced to the system, making it both energy and cost-efficient.

Not omitting the fact that certain applications leverage the use of the GPU which can significantly impact some applications that we will be addressing such as VR/AR applications, we must also consider the implication on battery life when both CPU and GPU are engaged.

The two branches are generated in conducting this research are tasks offloaded to conserve battery life for the UE which can handle the task in question, employing a heuristic approach we can assume that there will be certain classifications of devices that will be looking to offload tasks based on lack of resources and latency requirements.

4.2.4 Propagation Modelling

Weighting parameter

To fine-tune trade-off value between latency and power consumption, we use β as a constant throughout all cost calculation formulas.

Local

Given the task parameters, and device specifications, provided $T_{mi}^{d,k} \leq D_{mips}^{d}$, the given task can be processed locally, with the latency L^{l} defined as the following:

$$L_{d,k}^{l} = \frac{T_{mi}^{d,k}}{D_{mips}^{d}}$$

Additionally, we define the power consumption, $E_{d,k}^{l}$ as:

$$E_{d,k}^{l} = T_{mi}^{d,k} \cdot D_{ee}^{d}$$

The combined cost, $C_{d,k}^{l}$ is therefore defined as, where β is a weighting parameter the regulated trade-off between latency and consumption:

$$C_{d,k}^{l} = L_{d,k}^{l} \cdot \beta E_{d,k}^{l} \qquad (4.3)$$

Edge Processing

If task requirements exceed local computational specifications, the task is subsequently offloaded to be handled by the appropriate network device. To achieve this, task data must also be sent to the edge device, therefore total costs must take into consideration propagation of task data, we first define the latency $L_{d,k}^{m}$:

$$L_{d,k}^{m} = \frac{T_{size}^{d,k}}{R_{d,d'}} + \frac{T_{mi}^{d,k}}{D_{mips}^{d'}}$$

Where $R_{d,d'}$ is the maximum bit rate of the network between devices d and d' according to the Shannon-Hartley theorem. The channel noise is assumed to be white Gaussian noise with its variance as σ^2 and signal power S depends on path loss propagation model.

$$R_{d,d'} = D_{eb}^{d} \log_2\left(1 + \frac{S}{\sigma^2}\right)$$

Energy consumption is also impacted by the cost of transmitting data between device d to device d':

$$E_{d,k}^{m} = T_{size}^{d,k} \cdot D_{te}^{d} + T_{mi}^{d,k} \cdot D_{ee}^{d'}$$

Ultimately, the total cost for energy is the weighted sum the processing time $L_{d,k}^{m}$ and power consumption E^{m}:

$$C_{d,k}^{m} = L_{d,k}^{m} + \beta E_{d,k}^{m} \qquad (4.4)$$

Fog Processing

As above, we model the processing time, $L_{d,k}^{f}$:

$$L_{d,k}^{f} = \frac{T_{size}^{d,k}}{R_{d,f}} + \frac{T_{mi}^{d,k}}{D_{mips}^{f}}$$

Where R_f^d is defined as:

$$R_{d,f} = D_{fb}^{d} \log_2(1 + \frac{S}{\sigma^2})$$

Additionally, we model the energy consumption $E_{d,k}^{f}$:

$$E_{d,k}^{f} = T_{size}^{d,k} \cdot D_{te}^{d} + T_{mi}^{d,k} \cdot D_{ee}^{f}$$

Finally, the total cost for fog is defined as:

$$C_{d,k}^{f} = L_{d,k}^{f} + \beta E_{d,k}^{f} \quad (4.5)$$

Cloud Processing

If task parameters exceed local, edge and fog computation, devices can also offload tasks for cloud processing. Once again, we establish the latency taking propagation into consideration:

$$L_{d,k}^{c} = \frac{T_{size}^{d,k}}{R_{d,c}} + \frac{T_{mi}^{d,k}}{D_{mips}^{c}}$$

$$R_{d,c} = D_{cb}^{d} \log_2\left(1 + \frac{S}{\sigma^2}\right)$$

The energy consumption of task k of device d then depends on the cost of processing at cloud server c and the cost of transmitting the data from device d to the cloud server c:

$$E_{d,k}^{c} = T_{size}^{d,k} \cdot D_{te}^{d} + T_{mi}^{d,k} \cdot D_{ee}^{c}$$

And total cloud processing cost is:

$$C_{d,k}^{c} = L_{d,k}^{c} + \beta E_{d,k}^{c} \quad (4.6)$$

Figure 4.2. Device service request classification

As portrayed by Figure 4.2, we can classify UE by their service request types, assuming a future where display and battery (**DisBat**) devices can leverage 5G networking capabilities to give MNOs the chance to offer Mobile Devices as a Service with differentiating service plans and SLAs according to the requirements of the user. For now, unfortunately, we can assume that IoT sensors and devices can occupy the 'display and networking capabilities' section.

Differentiating service types using flags or identifiers will help MNOs achieve SLAs and help any learning models implemented on the MEC servers to respond with greater speed and accuracy. Each class of device will be able to identify it is needs and requirements according to the application request and determine its urgency. The request can then be actioned according to its requirements in latency, battery life implication, as well as other variables.

Processing time is a factor that must be heavily considered, especially when the cost of integrating MEC Servers within 5G is considered. The figure below details the average processing times when an application is processed solely on the network edge as opposed to a mobile device, Figure 4.3, where 200 devices were simulated for 30 minutes, and processing time recorded to test the initial run of the simulation.

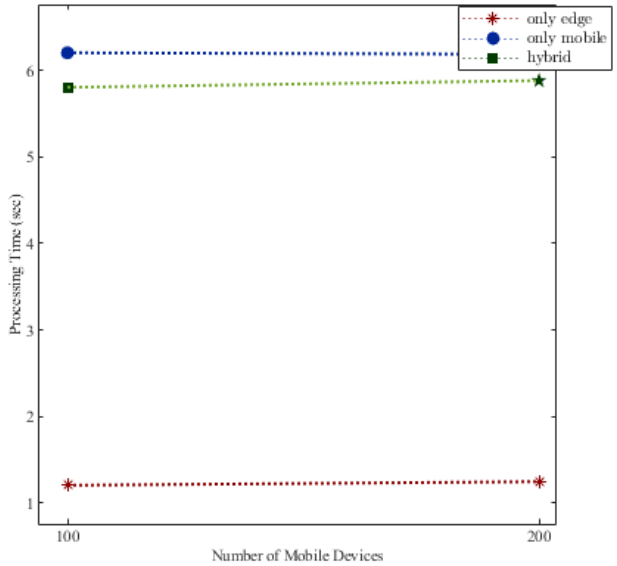


Figure 4.3. Processing time comparison example

Figure 4.3 plots our control simulation, the processing time of a mixture of computational tasks against number of mobile devices, operating solely on the

mobile device, solely on the edge device and using a hybrid method of both. The processing time of tasks is several times lower when the edge device is utilised to accomplish the task. The above plot was a quick simulation run, with up to a maximum of 200 mobile devices. The plot portrays an average of all application use case scenarios.

Theoretically, it makes perfect sense that the edge device would handle tasks at a blinding rate when compared to its counterpart. Unfortunately, the issue presents itself when other factors are considered such as network latency to return data back to the user. Despite the apparent benefit of being able to handle more intensive tasks that the mobile device may be incapable of handling, other advantages of offloading tasks include less consumption of battery life, less heat generated by the UE, faster response times for various applications and greater mobility options for UE owners.

Parameters for Edge Devices

To ensure that the tests were regulated, an end user was defined with the following specifications using average device specifications from the year 2023, the device specifications listed below do not consolidate laptops, TVs, and any other smart devices, they are strictly limited to smartphones accessible globally alongside average expected specifications [99], [100].

The specifications applied for consumer devices have been outlined in

Figure 4.4 portrays the device distribution over all simulation runs, each device has its own subset of applications that can be assigned with their own QoS.

TABLE 4.3 and Figure 4.4, where the distribution of the various devices we will be simulating is portrayed alongside the parameters assigned to each device, the device parameters specifically indicate computational power of user end devices.

Characteristics of devices within the simulator are defined by the following vector:

$$D^d = \{ cpu, tr, mips, ee, te, eb, fb, cb \}$$

Where cpu is the current % of CPU usage, tr is the number of tasks being executed, $mips$ is the maximum computational capacity, ee is the energy consumption per million instructions, te is the energy consumption per transmitted bit, eb is the bandwidth of the edge device in bits per second (bps), fb is the bandwidth of the fog device in bps and cb is the bandwidth of the cloud in bps.

Figure 4.4 portrays the device distribution over all simulation runs, each device has its own subset of applications that can be assigned with their own QoS.

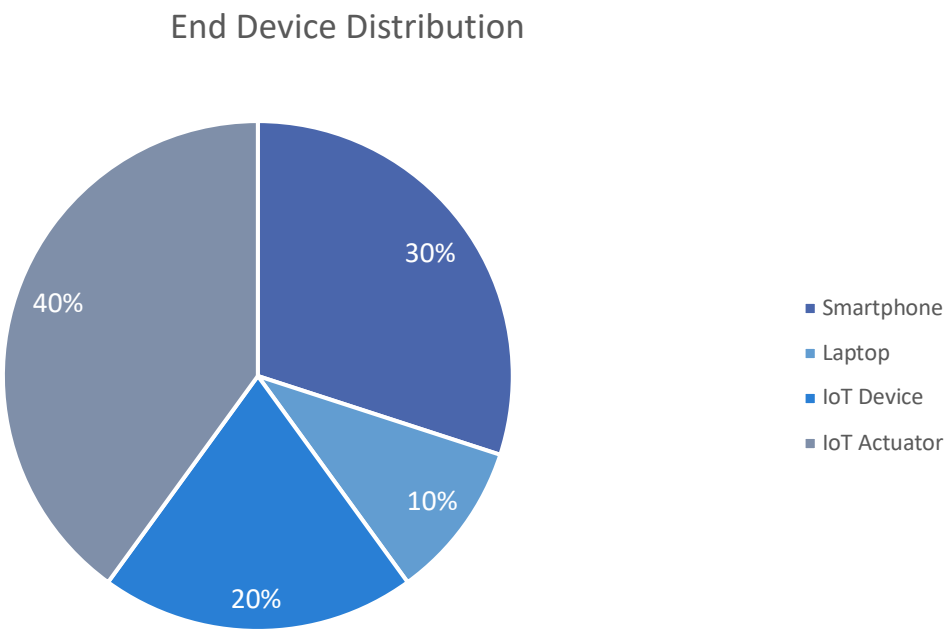


Figure 4.4. Device Distribution

TABLE 4.3
END USER DEVICE SPECIFICATIONS

User End Device Specifications				
	SMARTPHONE	LAPTOP	IoT DEVICE	IoT ACTUATOR
Processor	8 cores	8 cores		2
MIPS	25K	110K	16K	0
Storage	64GB	1TB		N/A
RAM	4GB	8GB		N/A
Battery Life	3,000 Mah	5,620 Mah		N/A
Mobility	True	True	False	False

4.2.5 Edge Orchestration

The orchestrator within the simulator is responsible for the effective allocation of resources and judging where the appropriate resources must be allocated. In this case, the simulator uses the fixed parameters as input variables to decide accordingly, how resources should be applied. To correctly insert our algorithm within the simulation software, the orchestrator itself will be manipulated to respond to the needs of the network environment dynamically, using the RL algorithm devised over the course of this research.

4.2.6 Simulation Architecture

A unique approach studied for optimisation policies, found in [101], detailed a method of implementing multiple stages in a deep learning algorithm for optimal control which reflected what I intended to do with my algorithm. Hypothetically, giving the system an overall understanding within a centralised location would mean operating in a distributed manner, hence introducing latency, not of data concerned being transferred, but of computational parameters exchanged between the central and edge servers.

Although this would provide the opportunity of greater computational resources allocated to solve the optimisation algorithm of a control network, the number of requests being served may introduce unnecessary complexities in computing the best approach for resource allocation. Decoupling the involvement of a central system, however, would give granular control to each sub-network, thus using [101]'s multiple stage approach to have differing action spaces, environmental model dynamics and reward signals.

The approach illustrated in Figure 4.5 portrays the movement of the environmental variables such as model dynamics, as communicated across the network at sporadic periods. This would enable the core network to not only understand the environment, but also predict usage shifts across all connected edge nodes and could be adapted to multiple stages dependant on network hierarchy.

Thus, computational time would be reduced, as the model can predict the best optimisation algorithm according to data shifts and given enough learning data, and supply control parameters to the entire network.

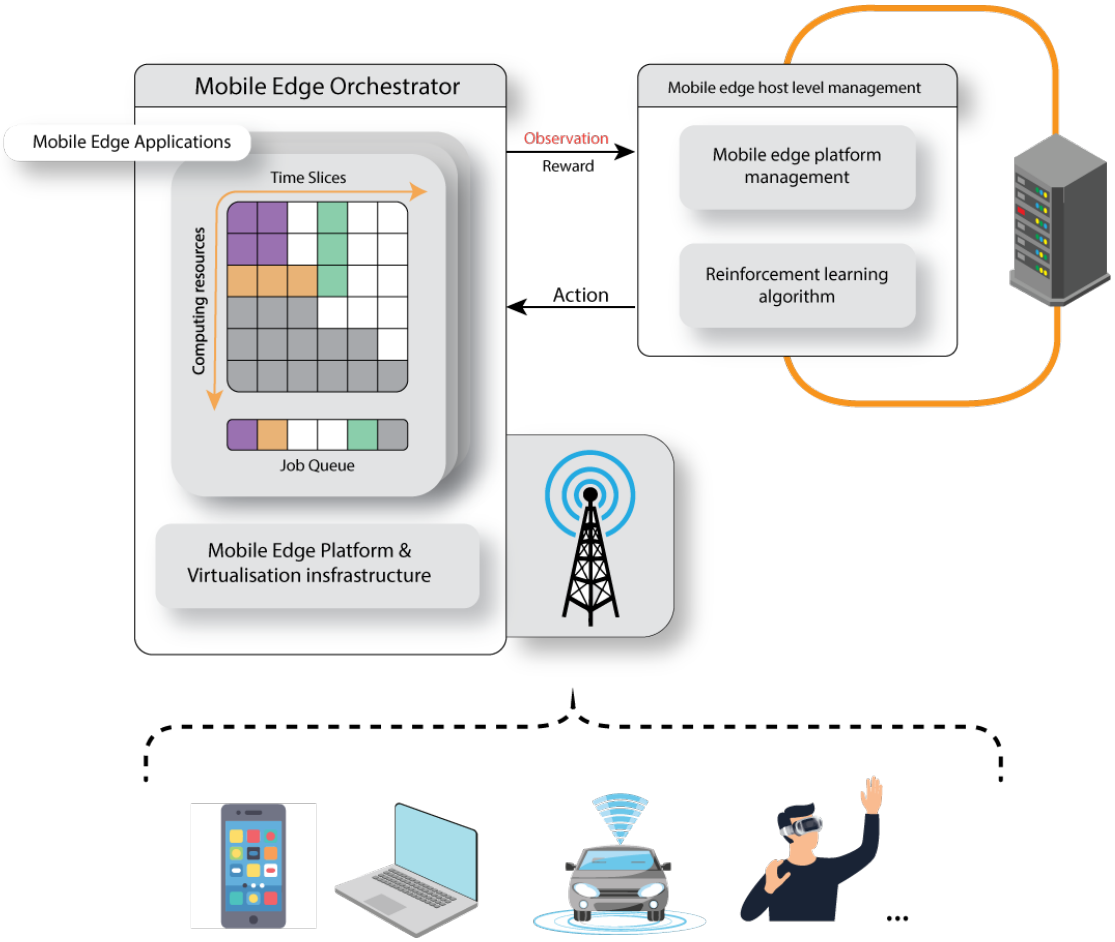


Figure 4.5. Orchestration Outline

Having browsed the literature, the best approach for the algorithm was to introduce a Fuzzy Logic approach which could then be interpreted by Deep Reinforcement Learning to optimise approach and further elaborate from non-definitive output of the Fuzzification process.

This meant first running a pass of Fuzzification, supplying observations to the Deep reinforcement learning algorithm and analysing output using a reward and punishment system to ensure that the system continued to function optimally under any changes in the network environment after a short learning process. Unfortunately, as someone well versed with the concepts being used in this research would immediately realise, this meant that each pass of the

algorithm would have to re-acquire all parameters such as network environment should those be included within the scope, introducing unnecessary re-calculation time, and lengthening the process.

As introducing unnecessary redundancy within the network optimisation algorithm may be the opposite of the goal of this research, it was probably a wise decision to establish some limitations on the frequency that parameters were refreshed within the algorithm depending on a layered approach that could prioritise each parameter contained within on the urgency and impact on the algorithm to ensure rapid return of learning results. This would in turn give some structure to the algorithm and ensure that response times were not affected.

This approach however meant that some definitive limitations had to be set within the Deep Reinforcement Learning approach to ensure all parameters were not established from the beginning but rather took a layered approach. This approach could then be readdressed taking all factors into consideration according to the historical time data that the algorithm could acquire, allocating re-assessment time of the entire algorithm according to when the best time/down-time of the network occurred.

As planning was required in understanding the direction and intention of the optimisation algorithm the initial plan of the layered approach was to ensure that the layered approach was divided amongst tasks that needed periodical refresh times.

Figure 2.15 was then devised as a preliminary approach. The final approach had to be compatible with end-goal of reduced latency therefore it was imperative to the research that algorithm calculation, or any variables output by the RL method were not introducing any unnecessary latency into the network. Categorising the initial variables was just the first stage however, this also meant that multiple algorithms would have to be put into effect and run at variable intervals to check the consistency of their counterparts.

The goal is to create an optimisation algorithm, therefore, considering the Bellman equation, classified as a functional equation, and solving it means finding the unknown function **V**. The formula below corresponds to discrete-time optimisation problems, namely, those that have a set value rather than a complex number. It is also referred to as the basic building block of solving reinforcement learning by some authors [102]. The goal of the function is to find the optimum value function to yield optimal rates, decreased energy usage balanced with least task failures.

4.2.7 Simulation Hardware

For transparency purposes, I have outlined the device that I used for simulation purposes and will be keeping a log of simulation times as well as other factors that portray my findings. The device used to run the simulations has the following specifications as found in TABLE 4.4:

TABLE 4.4
SIMULATION DEVICE SPECIFICATIONS

	Devices used to run simulations
Model	MacBook Pro
OS	MacOS Sonoma 14.5
SYSTEM TYPE	ARM 64
IDE	Visual Studio Code
IDE dependencies	jFuzzyLogic.jar
CPU	M2 Max
CPU Cores	12
GPU	M2 Max 38 Core
RAM	32GB DDDR5 UNIFIED
Storage	1TB SSD

For simplification purposes, I have omitted the operational frequency as well as dynamic frequencies of the CPU as well as the operational speeds of other components such as the SSD and RAM. I will also avoid monitoring the CPU and GPU temperatures whilst running simulations as I believe that they portray the functionality/optimisation of the simulator rather than having any effect on the results gathered. These details are not being recorded as they will have little to no consequence on real-world implementation of any optimisation algorithms used or discovered over the course of the research.

Although it must be acknowledged that CPUs and GPUs approach tasks in an assumed manner (even with equivalent specifications). The only change I believe that will occur on another device is that the time taken to complete simulation will be different and as I will be going through several simulations, this can only impact my research negatively.

4.2.8 Assumptions

There were numerous assumptions made over the course of this research as we could not account for every possible variable, however, to ensure that results were valid within the scope of the research and well-rounded, the following concrete assumptions were used as seen in TABLE 4.5:

TABLE 4.5
ASSUMPTIONS TABLE

ASSUMPTIONS	
COMPUTING RESOURCES	End devices are assumed to have CPU, RAM, storage and depletable batteries to process data and execute applications
NETWORKING CONNECTIVITY	We assume that all discoverable devices are connected to the network and are requesting resources in some form or manner
APPLICATION EXECUTION	Assumption that applications can be executed on edge devise, and that the application execution time and resource usage can be modelled.
WORKLOAD GENERATION	Assume that we can generate the workload according to the given distribution model, which helps to simulate the behaviour of the edge computing system under various conditions.
ENERGY CONSUMPTION	We assume that the energy consumption of devices is proportional to the amount of computation and data transmission performed by the device.
TASK SCHEDULING	Assume that tasks can be scheduled to edge devices based on different task categories as well as criteria such as proximity to the source, resource availability and energy consumption.
MOBILITY	We assume the position of the mobile devices as well as their travel trajectory.

4.3 Algorithm Design

The value function of the state s under policy π is defined as follows:

$$V^{\pi} = \mathbb{E}_{\pi}[R_t | s_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t = s\right] \quad (4.7)$$

The quality of action a under policy π is then expressed as the expected value of the cumulative reward starting from the state s, acting on a, and then following the policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_t |_{a_t = a}^{s_t = s}] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \Big|_{a_t = a}^{s_t = s}\right] \quad (4.8)$$

As aforementioned, the task of the MDP agent is to find the best possible policy by comparing value functions. π' is considered better than another policy π only if each state transition yields a higher value. Subsequently, the value and quality functions can be rewritten in accordance with the Bellman optimality equations as outlined above:

Value $$V^*(s) = \max_{\pi} V^{\pi}(s) = \max_{a} \sum_{s'} \mathcal{P}_a(s,s')[\mathcal{R}_a(s,s') + \gamma V^*(s')] \quad (4.9)$$

Quality $$\mathcal{Q}(s,a) = \max_{\pi} \mathcal{Q}^{\pi}(s,a) = \sum_{s'} \mathcal{P}_a(s,s')\left[\mathcal{R}_a(s,s') + \gamma \max_{a'} \mathcal{Q}^*(s',a')\right] \quad (4.10)$$

To ensure the correct balance of exploitation and exploration, off-policy techniques are used to separate the policy searched from the one used to make decisions, the learning process focuses on optimisation of the action-value function (Q) using an iterative update based on previous values and temporal difference.

$$\mathcal{Q}(s_t,a_t) = \mathcal{Q}(s_t,a_t) + \alpha\left[R_{t+1} + \gamma \max_{a} \mathcal{Q}(s_{t+1},a) - \mathcal{Q}(s_t,a_t)\right] \quad (4.11)$$

ALGORITHM 4.1

SINGLE LAYER RL

Single-layer ε-greedy Q-Learning Algorithm

Parameters: discount factor γ, learning rate α, exploration rate ε, penalty factor δ, query reward factor ρ, and query use penalty ω

```
begin
for each step t do
Observe actual state st
Determine feasible action set A' from A
sQuery ← false
e ← random number from [0,1]

    # Exploration vs. Exploitation
    if e < ε then
        at ← randomly select an action from A'
    else
        at ← arg min a∈A' Q(st, a)
    end

    # Offloading Decision
    if at is to offload to a fog server then
        isQuery ← true
        Send the offloading request to a fog server
        at ← get the fog server decision
    end

    Execute or send the offloading action at
    Wait for the task to be completed
    Observe new state st+1

    # Calculate Reward
    Calculate reward Ct using (15)

    # Apply Query Reward and Penalty
    if isQuery then
        Ct ← ρ · Ct
        Cqt ← ω · t · Ct

        # Update Q-value for Query Action (4)
        Update Q(st, 4) using (14) with Cqt

    # Update Q-value for the selected action at
    Update Q(st, at) according to (14) with Ct
end
```

4.4 Reinforcement Learning (single layer) results

Against our control algorithms, Round Robin and Random Allocation, we ran our initial ε-greedy Q-Learning Algorithm. The following results were obtained:

4.4.1 Task Success Rate

Using our control algorithms, random assignment and round robin as comparison, initially, task success starts similar to round robin and already provides a vast improvement over random assignment.

As devices increase however, we observe in Figure 4.6 that the efficiency rate of the ε-greedy Q-Learning Algorithm begins to ascend and shows signs of further improvement. Towards the end of our simulation run, with 1000 devices being serviced, we observe an almost 20% increase in efficiency with task success.

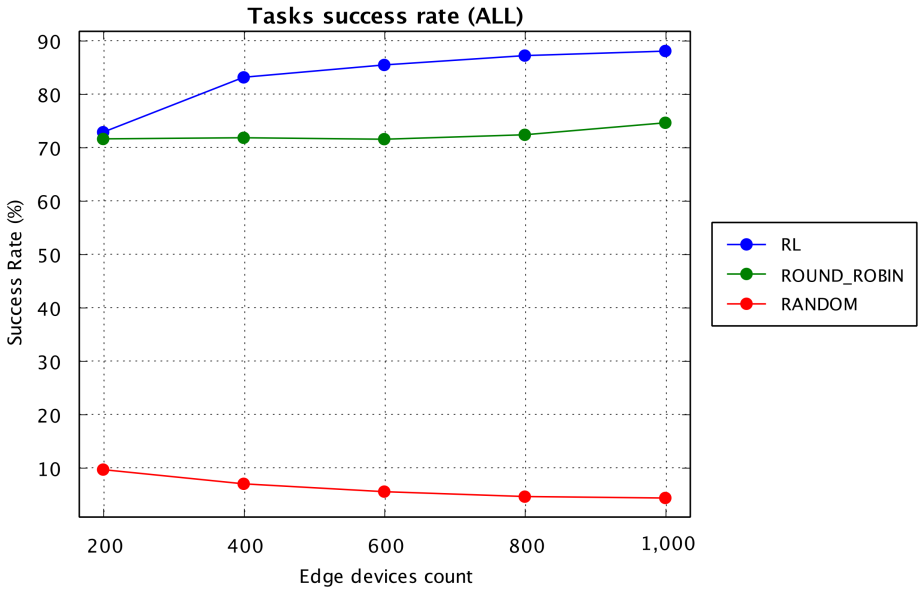


Figure 4.6. Task Success Rate Single Layer RL

Initially, Round Robin and RL begin at similar success rates, which eventually is superseded by the RL algorithm as it continues to develop intelligence and more efficiently allocates resources. The algorithm meets expectation and task success rate continues on an upward trend as the number of edge devices increases over iteration count.

4.4.2 Energy Usage

The large trade-off here comes with the computational power energy requirements made by the RL algorithm as seen above in Figure 4.7, where power usage is required to keep latency levels low and task success rate high. A drastic increase in energy consumption across the architecture is observed with the use case of 1000 devices on an untrained network.

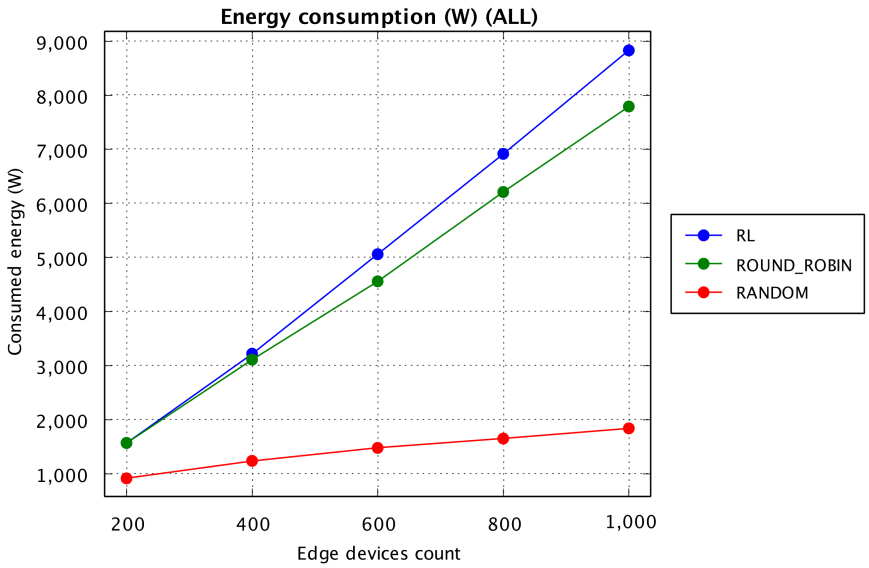


Figure 4.7. Energy Usage Single Layer RL

It is expected that RL will be slightly higher in energy usage, largely due to the computational requirements as the CPU load is increased when completing ML tasks. We see that task success rate in Random allocation is almost below 5% which is significantly lower than desired QoS rate. Due to constraints in computational capacity applied to the edge node, offloading decisions in the control algorithm RANDOM are handled immediately and primarily by Edge as opposed to being offloaded to the cloud as seen in Figure 4.8.

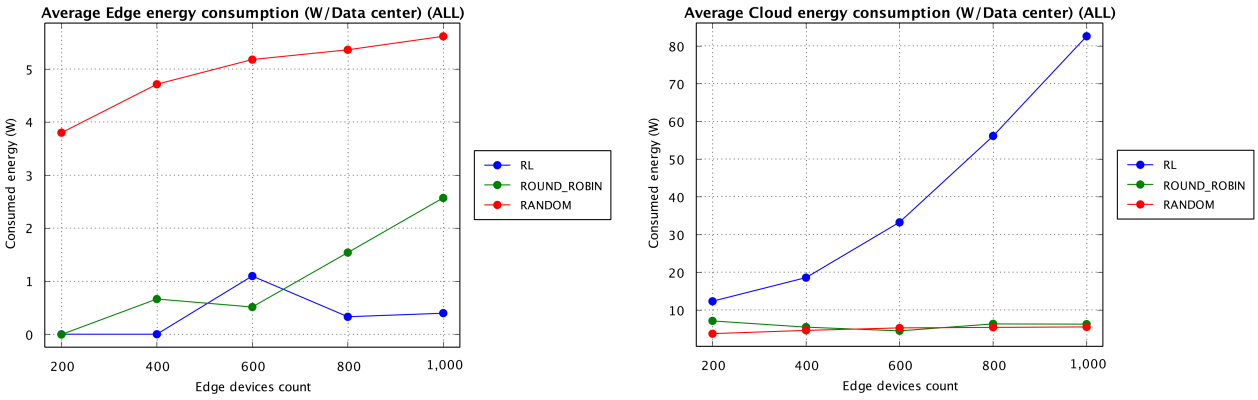


Figure 4.8. Comparison of Cloud vs Energy usage

4.4.3 Average CPU Usage, All Hierarchy

The energy usage as depicted in Figure 4.7 also correlated directly with the CPU utilisation of the RL tasks, however, the higher value in task success is worth the trade-off as a round robin requires an equal amount of energy usage. Despite the greater computational requirements of the RL algorithm, we observe that CPU utilisation across the entire hierarchy is equivalent when compared with the Round Robin algorithm as seen in Figure 4.9.

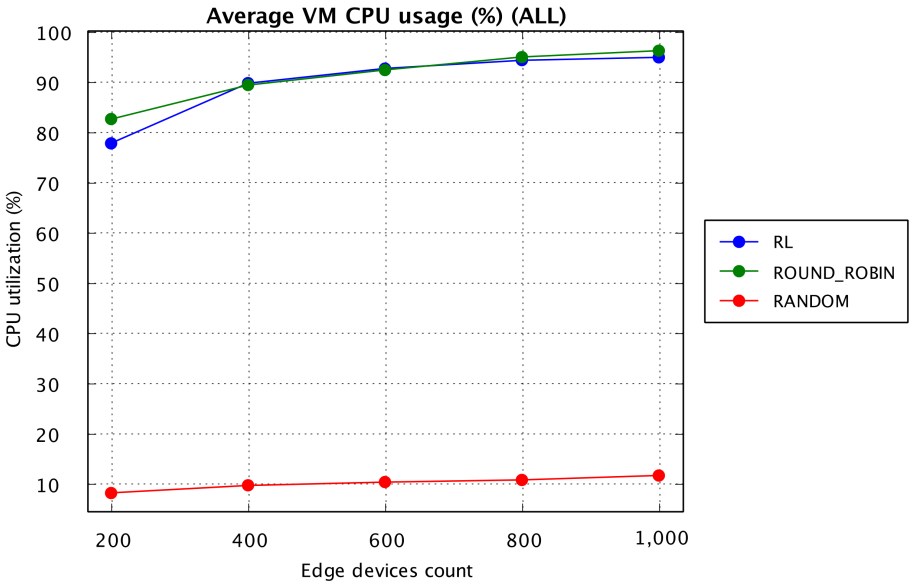


Figure 4.9. Avg CPU Usage Single Layer RL

The increase in task success rate is worth the trade off when considering that an increase of 80% is observed. This shows that the RL algorithm operates efficiently within the context of the requirements but leaves room for improvement with typical energy usage across the network hierarchy.

Further considerations include that lack of an ML optimised Edge device that would improve the results of computation efficiency and decrease energy usage; however, constraints are largely subject to the network HW and infrastructure simulated within the environment.

4.4.4 Average Execution Delay

One aspect where the RL implementation portrays efficiency is in the execution delay, hence our task latency requirements are met as observed in Figure 4.10. Execution delays continue on a downward trend as number of devices scale up and the RL algorithm builds its Qtable.

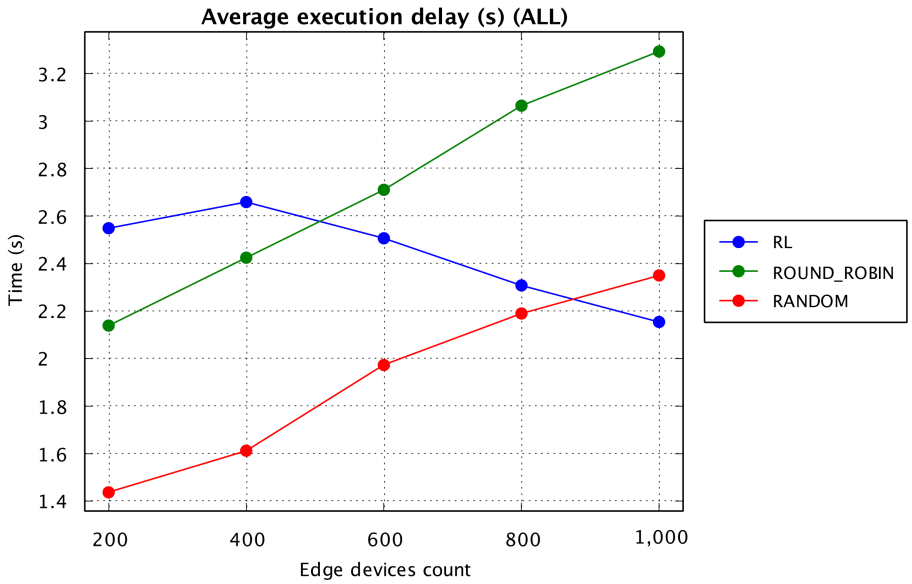


Figure 4.10. Avg Execution Delay Single Layer RL

We observe a decrease in initial latencies whereas the Random algorithm continuously grows as devices increase, each taking into consideration latency requirements per task. Bearing in mind that the execution delay is averaged

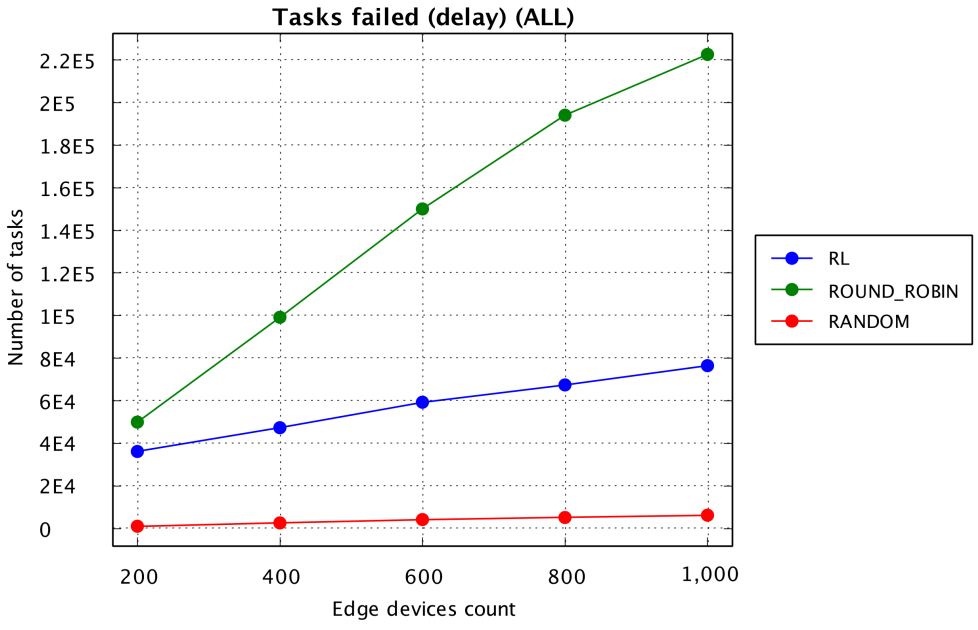


Figure 4.11. Task Failure Rate

across all tasks and initial tasks bear the brunt of the RL algorithm build-up as clarified in Figure 4.11.

4.5 Summary

Over the course of this chapter, we discover that our RL algorithm provides great increases in task efficiency and success rate, ensuring that more tasks meet latency requirements. The obvious negative aspect, however, lies in the power usage requirements that intelligence on the network edge requires.

Despite this, the simulation does not consider recent advances in neural network processing capabilities, where newer chipsets are being produced to handle CPU intensive tasks more efficiently, it also does not utilise GPUs effectively, as our measurement in the simulation is only conducted on average CPU usage.

Chapter 5 Multi-Layer RL in Resource Allocation

5.1 Introduction

This chapter consists of the design and implementation of our multi-layer algorithm in an attempt to improve learning and subsequently task success rate, at the end of which we will discuss the results. Using the parameters from Chapter 4, we can expand on our RL algorithm to use Multiple tiers to further optimise our Learning mechanism. Thus, theoretically improving the results of the algorithm.

Our QTables will then contribute to the next learning phases, which in this case must be done manually upon each iteration. Though out of the scope of this research, the QTables generated by the algorithm can then further be refined using a broader dataset of historical QTables, using granular environmental variables to ensure optimum performance for their respective environment.

5.2 Multi-Stage Implementation

Figure 5.1. Output analysis

Designing a multi-tiered reinforcement learning (RL) algorithm involves developing a system that can operate at multiple levels of abstraction and decision-making, allowing for more efficient and effective learning. Further implementing Fuzzy Logic for output analysis as seen in Figure 5.1 provides abstraction when observing task states, supplementing task state and allocation.

Thus, a tiered system can be introduced along multiple levels within the network to ensure maximum efficiency, effectively allocating tasks within multiple layers of the network stack.

Tier 1: Low-Level Control

At the lowest level, the agent interacts directly with the environment and takes actions based on the current state. The goal of this tier is to learn the basic skills and actions required to perform the task.

This tier can be trained using a simple RL algorithm such as Q-learning or SARSA. The agent receives a reward signal based on the outcome of its actions, and the algorithm updates its policy accordingly as seen in Figure 5.2.

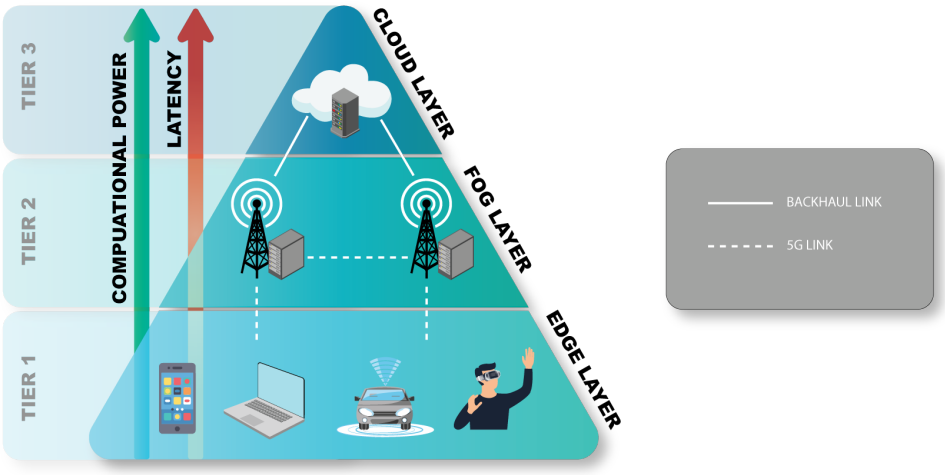


Figure 5.2. Hierarchy of the proposed network

Tier 2: Mid-Level Control

The second tier involves a higher-level controller that supervises the low-level agent and provides it with guidance and context. The mid-level controller is responsible for setting goals and sub-tasks, coordinating actions, and monitoring progress.

The mid-level controller can be trained using a hierarchical RL algorithm such as the MAXQ framework [103] or the options framework. These algorithms decompose the task into a hierarchy of sub-tasks, allowing the agent to learn to perform complex tasks by combining simpler skills.

Tier 3: High-Level Control

The top tier involves a meta-controller that supervises the mid-level controller and coordinates its behaviour. The meta-controller is responsible for selecting appropriate sub-tasks and adjusting the mid-level controller's parameters.

The meta-controller can be trained using a meta-RL algorithm such as the model-based RL or the population-based RL. As gathered from [55], we wish to ensure that the action space for any given subspace within our architecture is minimised to reduce-delay.

Further enhancements can be made by ensuring that the targeted action space's algorithm is catered to the delay sensitivity of its function. The last algorithm design is improved upon by including the continued. The value function of the state s under policy π is defined as follows:

$$V^{\pi} = \mathbb{E}_{\pi}[R_t | s_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t = s\right] \quad (5.1)$$

The quality of action a under policy π is then expressed as the expected value of the cumulative reward starting from the state s, acting on a, and then following the policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_t | {s_t = s \atop a_t = a}] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \left| {s_t = s \atop a_t = a}\right.\right] \quad (5.2)$$

The task of the MDP agent is to find the best possible policy by comparing value functions. π' is better than another policy π only if each state transition yields a higher value. The value and quality functions can be rewritten in accordance with the Bellman optimality equations as outlined above:

$$\begin{aligned} &\text{Value} && V^*(s) = \max_{\pi} V^{\pi}(s) = \max_{a} \sum_{s'} \mathcal{P}_a(s,s')[\mathcal{R}_a(s,s') + \gamma V^*(s')] \\ &\text{Quality} && Q(s,a) = \max_{\pi} Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}_a(s,s')\left[\mathcal{R}_a(s,s') + \gamma \max_{a'} Q^*(s',a')\right] \end{aligned} \quad (5.3)$$

To ensure the correct balance of exploitation and exploration, off-policy techniques are used to separate the policy searched from the one used to make

decisions, the learning process focuses on optimisation of the action-value function (q) using an iterative update based on previous values and temporal difference.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (5.4)$$

TABLE 5.1
NOTATION TABLE FOR Q POLICY EQUATION

Denotations	
s_t	Current state
a_t	Current action
γ	Discount factor
α	Learning rate = $\{0 \rightarrow 1\}$

5.3 Multi-Layer Results

5.3.1 Task Success Rate

As seen in Figure 5.3, the implementation of a multi-layered algorithm varied in its success initially, for reference, RL_MULTILAYER_EMPTY portrays an RL algorithm without QTables loaded, where QTables are initialised from the beginning, and built across the duration of the algorithm.

It is clear from the results gained that on initialisation, the algorithm performs poorly when compared to non-intelligent methods such as Round Robin, but as devices increase, which is the expected for an MEC node serving in an end-user environment, Multi-Layer RL provides an improved output of efficiency.

A similar trend is observed across all generated results, where output largely depends on the number of nodes. Figure 5.3 further portrays the relationship between the RL algorithm when QTables have been previously populated, clearly demonstrating the superiority over the ROUND_ROBIN algorithm, where task success rate immediately decreases once the edge device count begins to rise above 100.

Initial values show that with fewer devices, the single layered algorithm shows improved performance but once devices increase, so does the efficiency of the multi-layer algorithm.

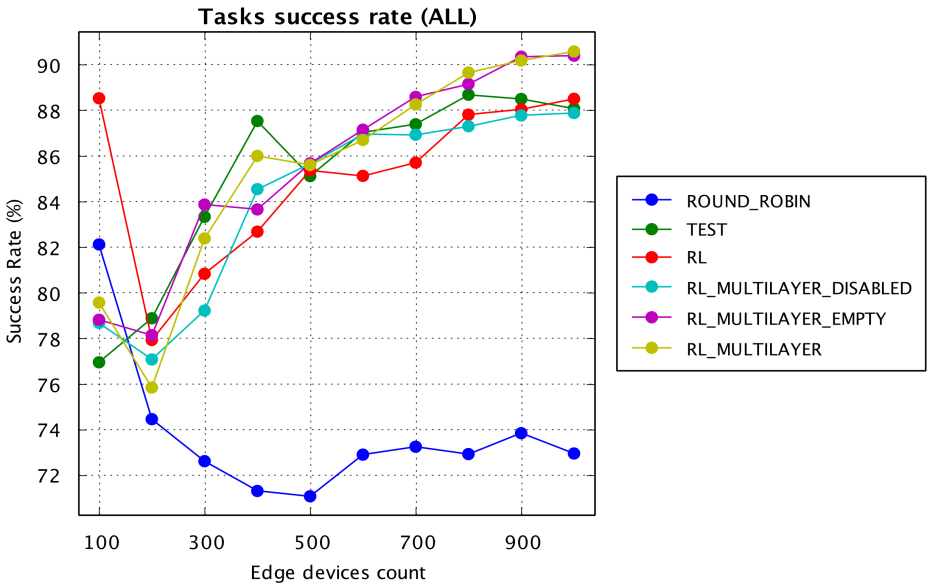


Figure 5.3. Task Success Rate Multi-Layer

5.3.2 Execution Delay

A similar observation is made for execution delay as seen above in Figure 5.4, as nodes increase, the 'intelligent' algorithms begin to outperform their non-intelligent counterparts by exponential increases, once again, when

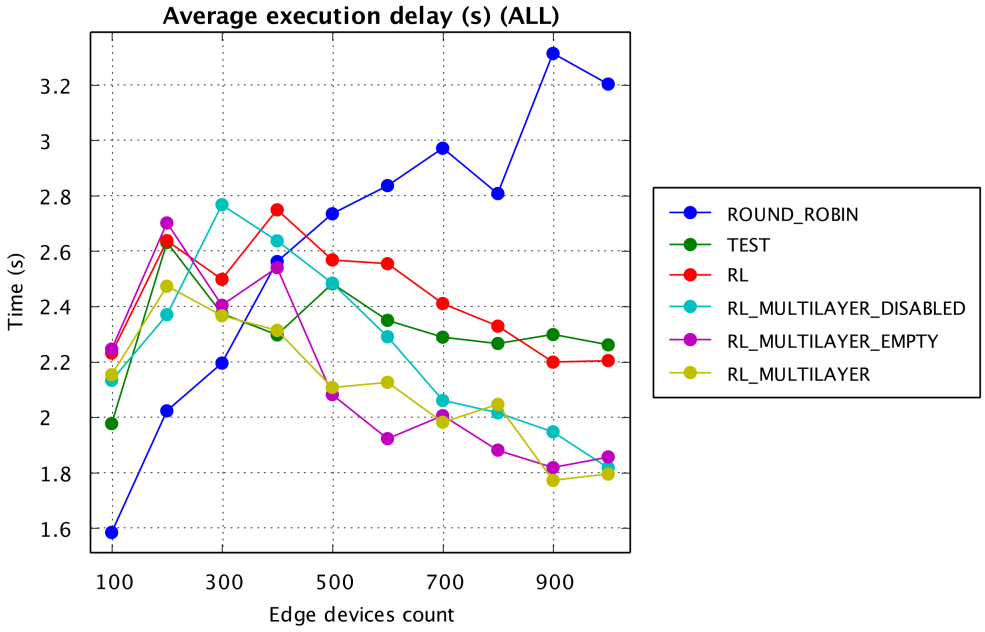


Figure 5.4. Multi-Layer RL Execution Delay

implemented in a multi-stage fashion, execution delays begin to decrease greatly.

This affects tasks with latency requirements, as more tasks can be successfully executed when start-up time of the tasks is lower, particularly helping in the fields of mobile health or robotic surgery.

5.3.3 Energy Usage

As with the single layer RL algorithm, we find that multi-layer RLs added complexity requires more computational power, therefore increased energy usage is incurred.

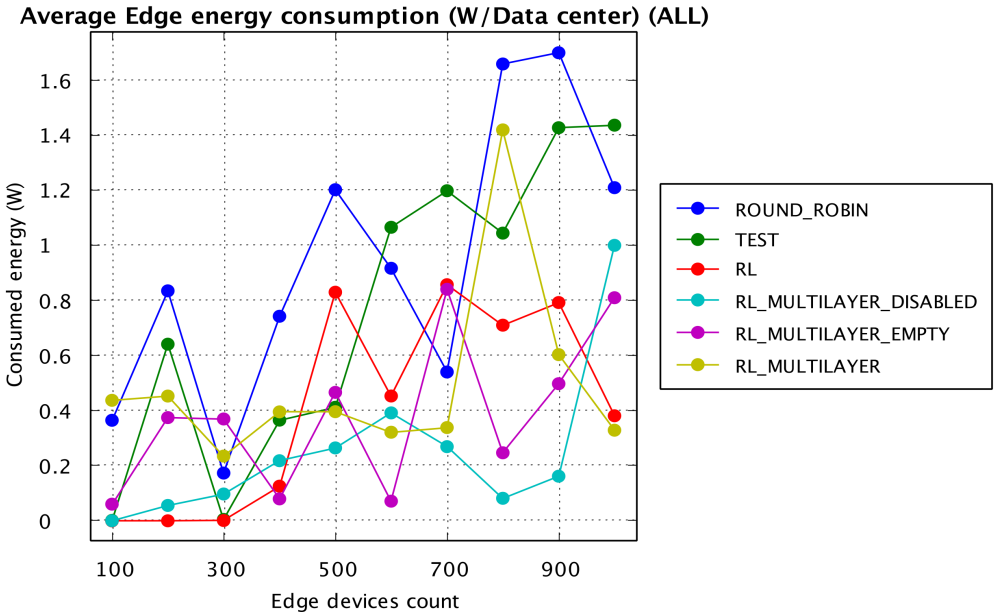


Figure 5.5. Avg Edge Energy Consumption Multi-Layer

Once again, computational decisions are increasingly handled by the cloud, however a pattern emerges where decision-making, which is now handled across the entire hierarchy using the tier system, is more evenly distributed, thus tasks can be effectively handled at the network edge once intelligence peaks (800 device mark, as seen in Figure 5.5).

Inconsistency is observed around the 800-device mark, where we notice a sharp increase in Edge consumed energy due to the offloading decision being allocated to edge. Despite the energy increase across both cloud and edge, average execution delay is reduced significantly once QTables have been loaded within the Multilayer algorithm.

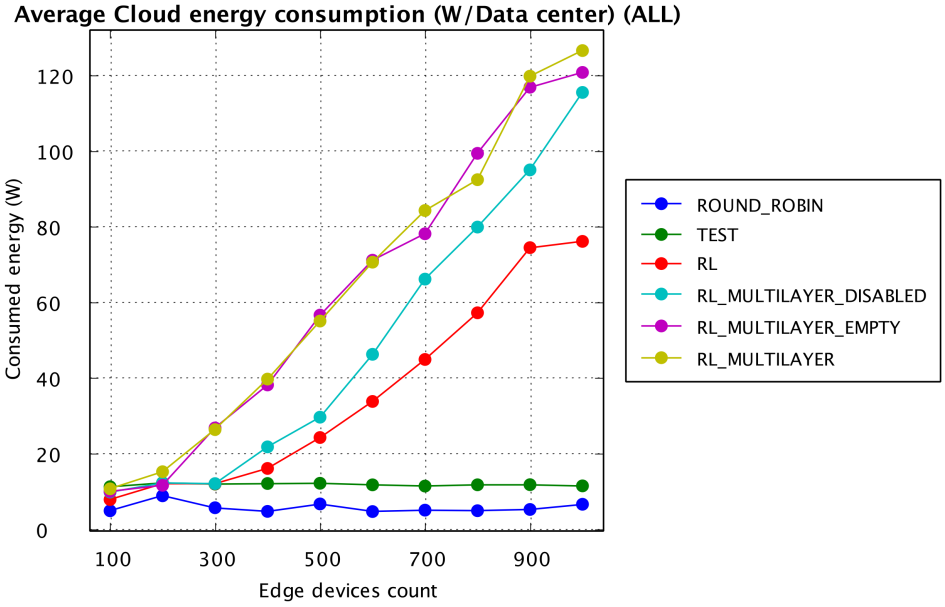


Figure 5.6. Avg Cloud Energy Consumption Multi-Layer

5.4 Further Enhancement

Using an approximation Q function to enhance success rate of tasks as seen in Eq. 5.5 ε-greedy algorithm uses exploration vs exploitation to maximise reward gaining advantages as implemented in Eq. 5.5, where using the parametrised function $Q(s,a;\theta)$ and θ are the parameters of the function approximator:

$$\theta \leftarrow \theta + \alpha \left[r_{t+1} + \gamma\, Q(s_{t+1}, a; \theta) - Q(s_t, a_t; \theta)\right] \nabla_\theta Q(s_t, a_t; \theta) \qquad (5.5)$$

TABLE 5.2
NOTATION TABLE FOR Q FUNCTION APPROXIMATION

Denotations	
$\boldsymbol{\theta}$	Model parameters (weights)
$\boldsymbol{\alpha}$	Learning rate (ranges from 0 to 1)

γ	Discount factor (ranges from 0 to 1)
$\boldsymbol{r_{t+1}}$	Reward at the next time step
$\boldsymbol{s_t}$	Current state
$\boldsymbol{s_{t+1}}$	Next state
$\boldsymbol{a_t}$	Current action
$\boldsymbol{Q(s_t, a_t;\ \theta)}$	Estimated Q-value for current state-action pair
$\boldsymbol{Q(s_{t+1}, a;\ \theta)}$	Estimated Q-value for next state and action

This equation is an update rule used in reinforcement learning to adjust the parameters of a function approximator, such as a neural network, in order to improve its performance. This type of update is commonly seen in algorithms like Q-learning with function approximation (e.g., Deep Q-Networks (DQN)).

$$\theta \leftarrow \theta + \alpha\,[\,\ldots\,]\nabla_\theta Q(s_t, a_t;\ \theta)$$

θ: The parameters (weights) of the function approximator, typically a neural network.

α: The learning rate, a scalar that controls how much the parameters are adjusted in each update step.

$\boldsymbol{\nabla_\theta Q(s_t, a_t;\ \theta)}$: The gradient of the Q-value function with respect to the parameters θ. This tells us how to change the parameters to maximize the Q-value.

$$r_{t+1} + \gamma\, Q(s_{t+1}, a;\ \theta) - Q(s_t, a_t;\ \theta)$$

$\boldsymbol{r_{t+1}}$: The immediate reward received after acting a_t in state s_t .

$\boldsymbol{\gamma}$ **(gamma)**: The discount factor, which determines the importance of future rewards.

$\boldsymbol{Q(s_{t+1}, a;\ \theta)}$: The estimated maximum future Q-value for the next state s_{t+1}. This represents the best possible future cumulative reward starting from s_{t+1}.

$\boldsymbol{Q(s_t, a_t;\ \theta)}$: The current Q-value estimate for taking action a_t in state s_t.

The term $r_{t+1} + \gamma Q(s_{t+1}, a; \theta) - Q(st, at; \theta)$ represents the temporal-difference (TD) error. It measures how far off the current Q-value estimate is from the "true" target value $r_{t+1} + \gamma Q(s_{t+1}, a; \theta)$.

The update rule $\theta \leftarrow \theta + \alpha[TD\ error]\nabla_\theta Q(s_t, a_t; \theta)$ uses this TD error to adjust the parameters θ. The gradient term $\nabla_\theta Q(s_t, a_t; \theta)$ guides the direction of the update to minimize this error.

Purpose of the Update

The goal is to iteratively adjust θ so that the Q-value function $Q(s, a; \theta)$ becomes a better approximation of the true action-value function. By applying this update repeatedly, the model learns to predict the expected cumulative reward for any given state-action pair more accurately.

ALGORITHM 5.1

MULTI-LAYER APPROXIMATION

Algorithm 2: ε-greedy Multilayer Q-Learning Algorithm

Parameters: discount factor γ, learning rate α, exploration rate ε, penalty factor δ, query reward factor ρ and query use penalty ω

```
begin
for each step t do
    Observe actual state s_t
    Determine feasible action set A' from A
    isQuery ← false
    e ← random number from [0, 1]

    if e < ε then
        a_t ← randomly select an action from A'
    else
        a_t ← arg min_{a ∈ A'} Q(s_t, a; θ)
    end

    if a_t is to ask a fog server then
        isQuery ← true
        Send the offloading request to a fog server
        a_t ← get the fog server decision
    end

    Execute or send the offloading action a_t
    Wait for the task to be completed
    Observe new state s_{t+1}
    Calculate reward C_t by (15)

    if isQuery then
        C_t ← ρ · C_t
        C_q_t ← ω · t · C_t
        δ = C_q_t + γ max_{a'} Q(s_{t+1}, a'; θ) - Q(a_t, a_t; θ)
        θ ← θ + α δ ∇_θ Q(s_t, a_t; θ)
    end

    δ = C_t + γ max_{a'} Q(s_{t+1}, a'; θ) - Q(s_t, a_t; θ)
    θ ← θ + α δ ∇_θ Q(s_t, a_t; θ)
end
```

5.5 Multi-Layer Approximation Results

We will now compare only the RL algorithms below, with round robin and random allocation once again as our control algorithms. The approximation algorithm often outperforms the multilayer algorithm in the previous section in regard to energy usage and also shows promising results over greater number of edge devices.

5.5.1 Task Success Rate

Using the function approximation algorithm, we notice a marginal increase in task success rate that once again, scales over time. We can rule out the RANDOM algorithm as being the least performant which is on par with expectations.

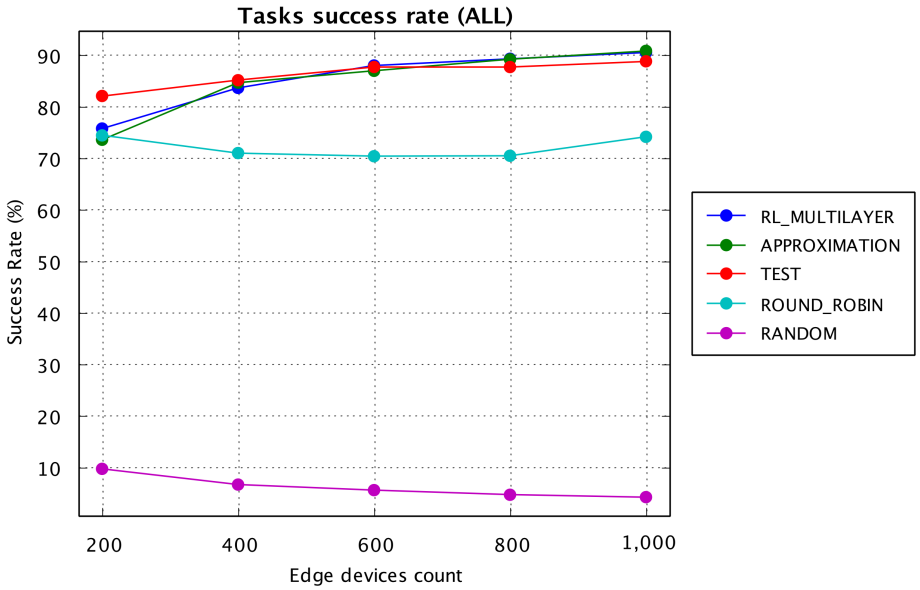


Figure 5.7. Task Success Rate Multi-Layer

It is noticeable however, that the ROUND ROBIN algorithm almost performs equivalently but as we will notice in the next section, there is a sharp increase in average execution delay which indicates that as devices increase, task failure rate will begin to decline sharply.

5.5.2 Execution Delay

Results in execution delay are varied, as there is a noticeable dip at the 600-device mark for our previous RL_MULTILAYER algorithm, although on conclusion of the simulation, we find that once again, a marginal increase is provided by our improved algorithm.

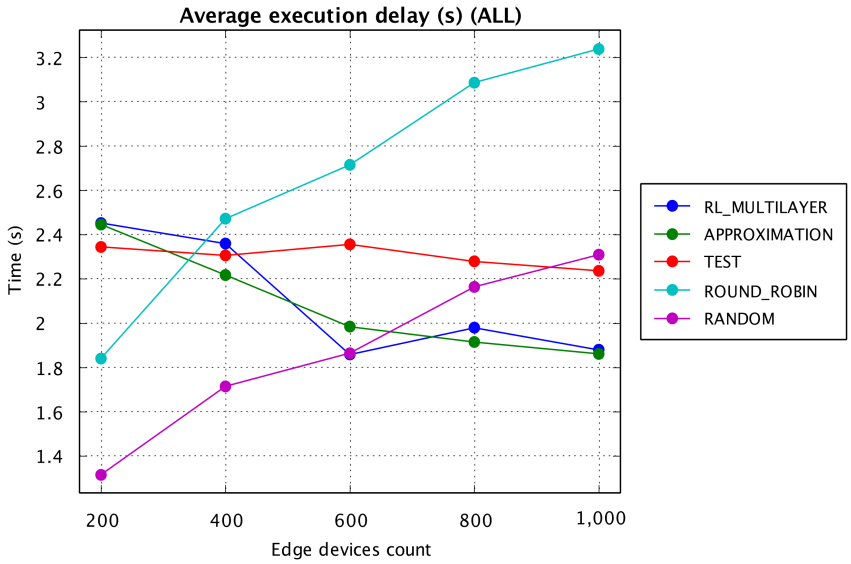


Figure 5.8. Avg Execution Delay Multi-Layer

This is largely due to the distribution of the task generator, as each task has its own specific requirements which are then allocated to the orchestrator. Despite the anomaly observer in the 600-device mark, the approximation algorithm outperforms the Multilayer algorithm across the board.

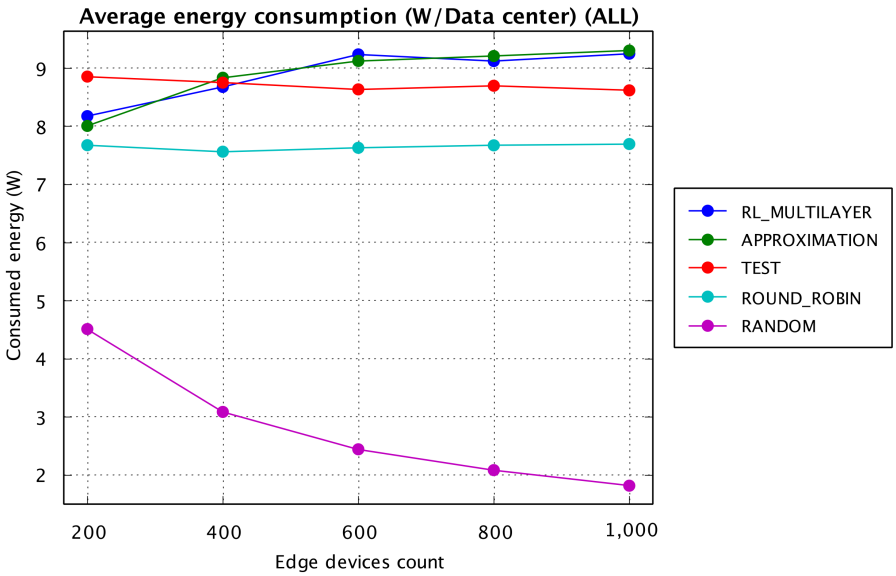


Figure 5.9. Avg Energy Consumption Multi-Layer

5.5.3 Power Usage

Test results stand, that with great intelligence, comes great power usage, and as the approximation algorithm is using more calculations, more CPU power is required to complete resource allocation.

Unfortunately, despite addressing the computational distribution for resource allocation, we continue with similar performance with energy usage without overhauling the entire edge architecture and changing simulation parameters. Another observation indicates that RANDOM assignment continuously declines in energy usage, however as the task failure rate is not sufficient and certainly does not meet QoS requirements for most applications, despite the

Due to the nature of simulation, at the time of writing results, the expected computational power available at the network edge has increased several fold, which in turn would also decrease execution delay as well as overall latency, ensuring more tasks are successful and meeting latency requirements.

Despite the anomalies observed where figures rise or dip sharply due to the nature of the task allocation and its random assignment when building task loads, we find that the latest improved algorithm consistently outperforms the multilayer algorithm without too great of an impact on performance.

5.6 Summary

This chapter portrayed that usage of a multi-layer approach within resource allocation can greatly increase efficiency of task success whilst reducing energy usage as the number of devices increases. There is a slight trade-off however, when it comes to task execution time, where RL can cause initial start-up delays. CPU usage also increases slightly, however as the algorithm matures, this also decreases with greater number of devices.

Chapter 6 Conclusion and Future Work

Over the course of this research, we dove into the resource allocation and MEC, compared simulation software and implemented our ε – greedy single layer and multi-layer algorithms that portrayed improvements at each stage.

6.1 Conclusion

This research covered many aspects of resource allocation within MEC environments. From the single layer implementation to the multi-layer, increases in efficiency were observed across all resource allocation as expected, however they did come with major disadvantages in energy usage, despite this, we must consider that our network configuration, particularly the configuration of the edge devices, was sourced from technology that began at the start of the thesis and therefore, does not take into account improvements in efficiency within silicone and neural network enabled processing.

Implementation of further enhancements, such as replay memory and approximate Q functions allowed more tasks to be successfully executed but once again, came at a cost in energy consumption. Simulating these environments gave us a chance to offset real-world implications and fine-tune our algorithm to respond to allocation tasks with greater efficiency, ensuring that once fully realised, gains in efficiency and QoS requirements for individual tasks could be met within industrial application.

As noted above, exponential increases in optimisation were observed in task success across all algorithmic implementations, and minor improvements could be made to return major gains in efficiency. Due to time implications, further improvements utilising neural network-based solutions could not be implemented as various research and software-based packages were introduced whilst this research was being conducted, such as ND4J, that could help us further integrate a neural network-based algorithm. Reward allocations could also be further refined to ensure the system is penalised for higher energy usage ensuring energy efficiency goals are met for resource allocation.

On the other hand, we successfully compared and discovered an optimum solution for the implementation and simulation for MEC environments that can apply to a cross-network architecture. We also developed and iterated upon intelligent resource allocation and subsequently multi-tiered resource allocation within the network edge, thereby increasing success rate of allocated tasks exponentially.

Some limitations were encountered within the simulation that constrained our simulation environment to fewer devices, issues such as this have since been addressed in PureEdgeSim but come at the cost of major architectural changes which require a rewrite of the custom edge orchestration modules and algorithms used over the course of the thesis.

To ensure validation, edge server specifications were maintained throughout the thesis and once decided, were not updated as new hardware was announced for the network environment. Though this may require addressing for the purpose of this research, luckily, we are just a parameter change away to being able to model and simulate any network environment with high accuracy.

Iteratively improving the resource allocation can lead to significant results over a larger number of devices, which would be the case when addressing individual cells within an Edge network or indeed a mobile network. We find that usage a RL can quickly and efficiently improve network operation but there is much to be gained in iterating QTables in a more efficient manner without overloading them with overheads and thus ensuring redundant information is omitted.

Thus, utilising methods like Experience Replay can help improve data efficiency and stability in training, particularly for off-policy algorithms such as the ones implemented in this research.

To conclude then, it is clear from the research we conducted that there is much to be gained with intelligence in resource allocation within the entire network stack, but as latency requirements become a major limitation in service provider offerings, including generative AI, we have just touched upon some of the many challenges that NPs face when implementing new architecture.

Thus, simulation and optimal resource allocation can help to provide the best route when it comes to intelligent service offerings at the network edge, readying us for 6G and beyond.

6.2 Future Work

As network connected devices increase alongside generative AI such as ChatGPT, ensuring that resources are allocated efficiently in both power usage and QoS becomes paramount to utilisation of technology. Despite the increases in bandwidth and speed in future technologies such as 6G, computational power must also be taken into consideration.

Technologies such as VR and AR have yet to see their full potential in application and will be further enhanced by MEC technology, but optimal resource allocation will prove to be a cornerstone in meeting QoS requirements for the vast variety of services expected to be handled by the network edge.

Despite our extensive coverage of optimal resource allocation, including the in-depth review of simulation software, implementation in real-world scenarios prove challenging and require consideration of numerous variables considering projected computation growth at the network edge with cost projections for MNOs to upgrade the technology.

Our multi-layered algorithm covered three layers however, there is potential for a ‘context aware’ MEC platform, that is able to dynamically identify its environment and select an appropriate allocation algorithm depending on projected usage from a historical context, as well as considering information on future network usage, adapting to use cases such as large events.

One major disadvantage was the lack of GPU implementation usage or more efficient, neural network enabled silicon that can more accurately gauge and drastically reduce the energy requirements for intelligence in the network edge. The above would have greatly impacted task execution delay, as RL tasks would have been handled more efficiently utilising GPU.

Unfortunately, as the testing was conducted using local computation with local hardware (and cost restraints), the number of edge devices was largely limited without the simulator simply giving up due to computational taxation. To further gauge the efficacy of the proposed algorithm in this research, it would be ideal to test the algorithm on a computational cluster via Elastic Compute (EC2) or Google Cloud Engine (GCE) using a computational cluster which would provide the added benefit of GPU based computation on NVIDIA architecture. This would remove the single device limitation and provide a more accurate representation of serving a greater number of devices of single edge nodes.

Considering that computational prowess continues to increase, newer technologies continue to emerge [104] [105], and energy efficiency has become a pivotal focus within both industry and academia, not only would we gauge a better idea of how the proposed algorithm would function in real-world application, but we could also understand the limitations and the trade-off between intelligent and non-intelligent resource allocation methods. In an ideal world, utilising AI based and simple allocation interchangeably until QTables are generated would be a greater solution, especially when considering initial deployment of edge nodes.

Additionally, with the advancement of AI, using Retrieval-Augmented Generation (RAG) to supplement and enhance contextual knowledge of the network during deployment and discovery of nodes would further bolster the success of the algorithm, ensuring that networks can pre-emptively and intelligently allocate resources efficiently whilst taking network conditions into consideration and using predictive pre-emptive measures to adjust accordingly when traffic is expected to fluctuate due to real-world conditions.

Intelligence in the network edge provides the kind of dynamicity required to keep up with growing demands and meeting with net-zero targets for organisations, specifically MNOs. As we enter industry 4.0, accompanied by the advancements and adoption of AI on a wider scale, an intelligent communication network is paramount to future-proof success.

Despite the focus and products of the research, it is important to consider how hierarchical continuous processes can enhance and optimise AI intelligence outside of LLMs. The natural progression of this research, with some lateral thinking, is determining how can infer contextual variables to adapt trained algorithms to new environments without incurring the same overhead, thus reducing TtE (Time to Efficiency). This will not only help to enhance optimisation in the communications space, but also ensure that continuous models can iteratively adapt to any use case.

Appendices

Appendix 1

Membership Functions

Variable	Fuzzy Term	Range Start	Range End
taskLength	Low	0	20000
	Medium	20000	100000
	High	100000	
taskMaxLatency	Low	0	6
	Medium	6	15
	High	15	
localCPU	Low	0	25
	Medium	25	50
	Busy	50	75
	High	75	
localMips	Low	0	30000
	Medium	30000	130000
	High	130000	
avgEdgeCPU	Low	0	25
	Medium	25	50
	Busy	50	75
	High	75	
avgCloudCPU	Low	0	25
	Medium	25	50
	Busy	50	75
	High	75	

References

[1] ‘Individuals using the Internet (% of population) | Data’. Accessed: Jul. 22, 2024. [Online]. Available: https://data.worldbank.org/indicator/IT.NET.USER.ZS?end=2022&skipRedirection=true&start=2022&view=map

[2] ‘5G today | Nokia’. Accessed: May 25, 2020. [Online]. Available: https://www.nokia.com/networks/5g/?did=d000000002hq&gclid=CjwKCAjw2a32BRBXEiwAUcugiBDdodmGfUorZjRLEvmEIBMIi1VYBkZ6tvJ6D1w5ojpuAbDSGYhOaRoCu8kQAvD_BwE

[3] ‘Press Releases : NTT DOCOMO to Launch 5G Service in Japan on March 25 | News & Notices | NTT DOCOMO’. Accessed: May 25, 2020. [Online]. Available: https://www.nttdocomo.co.jp/english/info/media_center/pr/2020/0318_00.html

[4] Ethem Alpaydın, ‘View on 5G Architecture’, *Version 3.0, June 2019*, no. June, pp. 21–470, 2019, [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf

[5] ‘How fast is 5G - 5G speeds and performance’. Accessed: Nov. 02, 2020. [Online]. Available: https://5g.co.uk/guides/how-fast-is-5g/

[6] Kris Beevers, ‘Why 5G is bringing edge computing and automation front and center | Network World’. Accessed: Jul. 14, 2019. [Online]. Available: https://www.networkworld.com/article/3255426/why-5g-is-bringing-edge-computing-and-automation-front-and-center.html

[7] ETSI White Paper No. 11, ‘Mobile edge computing—A key technology towards 5G. ETSI White Paper’, *ETSI White Paper*, vol. 11, no. 11, pp. 1--16., 2015.

[8] ‘Keeping the Internet up and running in times of crisis’. Accessed: Mar. 28, 2021. [Online]. Available: https://www.oecd.org/coronavirus/policy-responses/keeping-the-internet-up-and-running-in-times-of-crisis-4017c4c9/

[9] S. Yadav, ‘Six Important Questions to Ask Your Edge Computing Provider | GE Digital’. Accessed: Nov. 01, 2020. [Online]. Available: https://www.ge.com/digital/blog/six-important-questions-ask-your-edge-computing-provider

[10] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, ‘Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning’, *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2021, doi: 10.1109/TPDS.2020.3014896.

[11] S. Svorobej *et al.*, ‘Simulating fog and edge computing scenarios: An overview and research challenges’, *Future Internet*, vol. 11, no. 3, pp. 1–15, 2019, doi: 10.3390/fi11030055.

[12] H. Sun, Z. Zhang, R. Q. Hu, and Y. Qian, ‘Wearable communications in 5g: Challenges and enabling technologies’, *IEEE Vehicular Technology Magazine*, vol. 13, no. 3, pp. 100–109, 2018, doi: 10.1109/MVT.2018.2810317.

[13] Y. C. Hu, M. Patel, D. Sabella, and V. Young, ‘ETSI White Paper #11 Mobile Edge Computing - a key technology towards 5G’, 2015. Accessed: Jul. 15, 2019. [Online]. Available: www.etsi.org

[14] R. Vilalta *et al.*, ‘Control and Management of a Connected Car Using SDN/NFV, Fog Computing and YANG data models’, *2018 4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018*, pp. 344–346, 2018, doi: 10.1109/NETSOFT.2018.8460131.

[15] J. Marshall, ‘Poorly planned infrastructure could cost UK £23bn, Mace reveals | News | Building’. Accessed: Feb. 28, 2021. [Online]. Available: https://www.building.co.uk/news/poorly-planned-infrastructure-could-cost-uk-23bn-mace-reveals/5109591.article

[16] ‘Mobile network traffic update – Mobility Report - Ericsson’. Accessed: Feb. 28, 2021. [Online]. Available: https://www.ericsson.com/en/mobility-report/dataforecasts/mobile-traffic-update

[17] S. Antipolis, ‘ETSI Plugtests Report’, Jun. 2020.

[18] P. Dolan, R. Shaw, A. Tsuchiya, and A. Williams, ‘QALY maximisation and people’s preferences: a methodological review of the literature.’, *Health Econ*, vol. 14, no. 2, pp. 197–208, Feb. 2005, doi: 10.1002/hec.924.

[19] V. Socialcast and M. U. Guide, ‘No Title’, no. December, 2016.

[20] ‘Virtualization Essentials’. Accessed: Nov. 07, 2020. [Online]. Available: http://www.gartner.com/newsroom/id/1472714

[21] S. Vittal, M. K. Singh, and A. Antony Franklin, ‘Adaptive network slicing with multi-site deployment in 5G core networks’, *Proceedings of the 2020 IEEE Conference on Network Softwarization: Bridging the Gap Between AI and Network Softwarization, NetSoft 2020*, pp. 227–231, 2020, doi: 10.1109/NetSoft48620.2020.9165512.

[22] A. F. Aljulayfi and K. Djemame, ‘Simulation of an augmented reality application for driverless cars in an edge computing environment’, *5th International Symposium on Innovation in Information and Communication Technology, ISIICT 2018*, vol. 2019-Janua, 2019, doi: 10.1109/ISIICT.2018.8651268.

[23] F. Alvarez *et al.*, ‘An Edge-to-Cloud Virtualized Multimedia Service Platform for 5G Networks’, *IEEE Transactions on Broadcasting*, vol. PP, pp. 1–12, 2019, doi: 10.1109/TBC.2019.2901400.

[24] C. Sonmez, A. Ozgovde, and C. Ersoy, ‘Fuzzy Workload Orchestration for Edge Computing’, *IEEE Transactions on Network and Service Management*, vol. 4537, no. i, pp. 1–14, 2019, doi: 10.1109/TNSM.2019.2901346.

[25] N. Tziritas *et al.*, ‘Data Replication and Virtual Machine Migrations to Mitigate Network Overhead in Edge Computing Systems’, *IEEE Transactions on Sustainable Computing*, vol. 2, no. 4, pp. 320–332, 2017, doi: 10.1109/tsusc.2017.2715662.

[26] ‘Industrial Edge | Topic areas | Siemens Global’. Accessed: Apr. 02, 2021. [Online]. Available: https://new.siemens.com/global/en/products/automation/topic-areas/industrial-edge.html?gclid=CjwKCAjwgZuDBhBTEiwAXNofRKfDbS0hBvpFLoyz2MERwVxdeQQsQUWX_glm6vZVENlR6JegFsJ1mhoCwgEQAvD_BwE

[27] Nebbiolo Technologies Inc., ‘Fog vs Edge Computing’, p. 8, 2016, [Online]. Available: https://www.nebbiolo.tech/wp-content/uploads/whitepaper-fog-vs-edge.pdf

[28] J. Pan and J. McElhannon, ‘Future Edge Cloud and Edge Computing for Internet of Things Applications’, *IEEE Internet Things J*, vol. 5, no. 1, pp. 439–449, 2018, doi: 10.1109/JIOT.2017.2767608.

[29] M. Armbrust *et al.*, ‘A View of Cloud Computing Clearing the clouds away from the true potential and obstacles posed by this computing capability’, *Commun ACM*, vol. 53, no. 4, 2010, doi: 10.1145/1721654.1721672.

[30] newzoo, ‘Newzoo | Games & Esports Analytics and Market Research’. Accessed: Jun. 29, 2019. [Online]. Available: https://newzoo.com/

[31] A. Nadeem, ‘Performance Comparison of Data’, *Measurement*, pp. 84–89, 2005, doi: 10.1109/VETECF.2008.422.

[32] MTR, ‘ETSI GS MEC 002 V2.1.1’, 2018. Accessed: May 23, 2019. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

[33] sdxcentral, ‘What is the ETSI MEC ISG’s Role in Defining Edge Computing?’ Accessed: Apr. 10, 2019. [Online]. Available: https://www.sdxcentral.com/edge/definitions/what-is-etsis-role-in-defining-mec/

[34] ETSI, ‘MEC 003 - V2.1.1 - Multi-access Edge Computing (MEC); Framework and Reference Architecture’, vol. 1, pp. 1–21, 2019.

[35] R. Jain, ‘Introduction to Software Defined Networking (SDN)’, pp. 1–61, 2013, doi: https://dx.doi.org/10.1111/pace.13115.

[36] Y. Zhang, *Network Function Virtualization: Concepts and Applicability in 5G Networks*, First Edit. John Wiley & Sons, Inc., 2018.

[37] M. Li, F. R. Yu, P. Si, H. Yao, and Y. Zhang, ‘Software-Defined Vehicular Networks with Caching and Computing for Delay-Tolerant Data Traffic’, *IEEE International Conference on Communications*, vol. 2018-May, 2018, doi: 10.1109/ICC.2018.8422823.

[38] J. Wang, L. Zhao, J. Liu, and N. Kato, ‘Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach’, *IEEE Trans Emerg Top Comput*, vol. 9, no. 3, pp. 1529–1541, 2021, doi: 10.1109/TETC.2019.2902661.

[39] ‘A Beginner’s Guide to Deep Reinforcement Learning | Pathmind’. Accessed: Jan. 31, 2021. [Online]. Available: https://wiki.pathmind.com/deep-reinforcement-learning

[40] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second Edition. The MIT Press, 2018.

[41] ‘Types of Machine Learning - Supervised, Unsupervised, Reinforcement - TechVidvan’. Accessed: May 10, 2021. [Online]. Available: https://techvidvan.com/tutorials/types-of-machine-learning/

[42] D. Silver *et al.*, ‘Mastering the game of Go with deep neural networks and tree search’, *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[43] B. Andre, H. Shaobo, B. Diana, S. David, and P. Doina, ‘Fast reinforcement learning through the composition of behaviours | DeepMind’. Accessed: Feb. 07, 2021. [Online]. Available: https://deepmind.com/blog/article/fast-reinforcement-learning-through-the-composition-of-behaviours

[44] M. Naeem, S. T. H. Rizvi, and A. Coronato, ‘A Gentle Introduction to Reinforcement Learning and its Application in Different Fields’, *IEEE Access*, vol. 8, pp. 209320–209344, 2020, doi: 10.1109/ACCESS.2020.3038605.

[45] M. Abderrahim, A. Ben Letaifa, A. Haji, and S. Tabbane, ‘How to use MEC and ML to improve resources allocation in SDN networks?’, *Proceedings - 32nd IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2018*, vol. 2018-Janua, pp. 442–447, 2018, doi: 10.1109/WAINA.2018.00126.

[46] J. Li, H. Gao, T. Lv, and Y. Lu, ‘Deep reinforcement learning based computation offloading and resource allocation for MEC’, *IEEE Wireless Communications and Networking Conference, WCNC*, vol. 2018-April, pp. 1–6, 2018, doi: 10.1109/WCNC.2018.8377343.

[47] C. Mechalikh, H. Taktak, and F. Moussa, ‘PureEdgeSim: A Simulation Toolkit for Performance Evaluation of Cloud, Fog, and Pure Edge Computing Environments’, *2019 International Conference on High Performance Computing and Simulation, HPCS 2019*, pp. 700–707, 2019, doi: 10.1109/HPCS48598.2019.9188059.

[48] R. L. de Mantaras and L. Godo, ‘From fuzzy logic to fuzzy truth-valued logic for expert systems. A survey’, *1993 IEEE International Conference on Fuzzy Systems*, pp. 750–755, 1993, doi: 10.1109/fuzzy.1993.327536.

[49] ‘A Phenomenon in Modern Science or Who Are You Lotfi Zadeh? - Famous people - Visions of Azerbaijan Magazine’. Accessed: Nov. 05, 2020. [Online]. Available: http://www.visions.az/en/news/49/2bbd742d/

[50] ‘Fuzzy Logic (Stanford Encyclopedia of Philosophy)’. Accessed: Nov. 05, 2020. [Online]. Available: https://plato.stanford.edu/entries/logic-fuzzy/

[51] ‘What is “fuzzy logic”? Are there computers that are inherently fuzzy and do not apply the usual binary logic? - Scientific American’. Accessed: Nov. 05, 2020. [Online]. Available: https://www.scientificamerican.com/article/what-is-fuzzy-logic-are-t/

[52] ‘What is Fuzzy Logic in AI and What are its Applications? | Edureka’. Accessed: Nov. 07, 2020. [Online]. Available: https://www.edureka.co/blog/fuzzy-logic-ai/

[53] M. Jezewski, R. Czabanski, and J. Leski, ‘Introduction to Fuzzy Sets’, in *Theory and Applications of Ordered Fuzzy Numbers: A Tribute to Professor Witold Kosiński*, P. Prokopowicz, J. Czerniak, D. Mikołajewski, Ł. Apiecionek, and D. Ślęzak, Eds., Cham: Springer International Publishing, 2017, pp. 3–22. doi: 10.1007/978-3-319-59614-3_1.

[54] A. Robles-Enciso and A. F. Skarmeta, ‘A multi-layer guided reinforcement learning-based tasks offloading in edge computing’, *Computer Networks*, vol. 220, p. 109476, Jan. 2023, doi: 10.1016/J.COMNET.2022.109476.

[55] X. Xiong, K. Zheng, S. Member, L. Lei, S. Member, and L. Hou, ‘Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing’, vol. 8716, no. c, pp. 1–13, 2020, doi: 10.1109/JSAC.2020.2986615.

[56] X. Liu, J. Yu, J. Wang, and Y. Gao, ‘Resource Allocation With Edge Computing in IoT Networks via Machine Learning’, *IEEE Internet Things J*, vol. 7, no. 4, pp. 3415–3426, Apr. 2020, doi: 10.1109/JIOT.2020.2970110.

[57] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, ‘Resource management with deep reinforcement learning’, *HotNets 2016 - Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 50–56, Nov. 2016, doi: 10.1145/3005745.3005750.

[58] C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, ‘Machine Learning-Based Workload Orchestrator for Vehicular Edge Computing’, *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2239–2251, Apr. 2021, doi: 10.1109/TITS.2020.3024233.

[59] M. Abderrahim, A. Ben Letaifa, A. Haji, and S. Tabbane, ‘How to use MEC and ML to improve resources allocation in SDN networks?’, *Proceedings - 32nd IEEE International Conference on Advanced*

Information Networking and Applications Workshops, WAINA 2018, vol. 2018-Janua, pp. 442–447, 2018, doi: 10.1109/WAINA.2018.00126.

[60] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, 'A Double Deep Q-learning Model for Energy-efficient Edge Scheduling', *IEEE Trans Serv Comput*, vol. 1374, no. c, pp. 1–12, 2018, doi: 10.1109/TSC.2018.2867482.

[61] Huawei Technologies Ltd., '5G Network Architecture-A High Level View 5G Network Architecture 5G Network Architecture A High-Level Perspective Network Architecture-A High Level View 5G 3 5G Network Architecture-A High Level View Contents'. Accessed: Mar. 27, 2019. [Online]. Available: https://www.huawei.com/minisite/hwmbbf16/insights/5G-Nework-Architecture-Whitepaper-en.pdf

[62] 'Intel's new Core i9-11900K flagship processor will arrive in early 2021 - The Verge'. Accessed: Mar. 07, 2021. [Online]. Available: https://www.theverge.com/2021/1/11/22225541/intel-processor-11th-gen-ces-2021-chips-specs

[63] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, 'Collaborative cache allocation and computation offloading in mobile edge computing', *19th Asia-Pacific Network Operations and Management Symposium: Managing a World of Things, APNOMS 2017*, pp. 366–369, 2017, doi: 10.1109/APNOMS.2017.8094149.

[64] D. Gopi, S. Cheng, and R. Huck, 'Comparative analysis of SDN and conventional networks using routing protocols', *IEEE CITS 2017 - 2017 International Conference on Computer, Information and Telecommunication Systems*, pp. 108–112, 2017, doi: 10.1109/CITS.2017.8035305.

[65] X. Deng, J. Li, L. Shi, Z. Wei, X. Zhou, and J. Yuan, 'Wireless Powered Mobile Edge Computing : Dynamic Resource Allocation and Throughput', vol. 1233, no. c, 2020, doi: 10.1109/TMC.2020.3034479.

[66] S. Garg, N. Kumar, J. J. P. C. Rodrigues, and J. J. P. C. Rodrigues, 'Hybrid deep-learning-based anomaly detection scheme for suspicious flow detection in SDN: A social multimedia perspective', *IEEE Trans Multimedia*, vol. 21, no. 3, pp. 566–578, 2019, doi: 10.1109/TMM.2019.2893549.

[67] X. Xiong, K. Zheng, S. Member, L. Lei, S. Member, and L. Hou, 'Resource Allocation Based on Deep Reinforcement Learning in IoT

Edge Computing', vol. 8716, no. c, pp. 1–13, 2020, doi: 10.1109/JSAC.2020.2986615.

[68] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, 'Cache in the air: Exploiting content caching and delivery techniques for 5G systems', *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014, doi: 10.1109/MCOM.2014.6736753.

[69] Netflix, 'Rethinking Netflix's Edge Load Balancing - Netflix TechBlog - Medium'. Accessed: Jul. 14, 2019. [Online]. Available: https://medium.com/netflix-techblog/netflix-edge-load-balancing-695308b5548c

[70] V. Eudossiana, 'DYNAMIC RESOURCE ALLOCATION FOR WIRELESS EDGE MACHINE LEARNING WITH LATENCY AND ACCURACY GUARANTEES Mattia Merluzzi , Paolo Di Lorenzo , Sergio Barbarossa', pp. 9036–9040, 2020.

[71] R. I. Tinini, M. R. P. dos Santos, G. B. Figueiredo, and D. M. Batista, '5GPy: A SimPy-based simulator for performance evaluations in 5G hybrid Cloud-Fog RAN architectures', *Simul Model Pract Theory*, vol. 101, no. October 2019, p. 102030, 2020, doi: 10.1016/j.simpat.2019.102030.

[72] B. Linkleter, 'Open-Source Routing and Network Simulation | Open-Source Network Simulators'. Accessed: Jun. 17, 2019. [Online]. Available: http://www.brianlinkletter.com/open-source-network-simulators/

[73] C. Sonmez, A. Ozgovde, and C. Ersoy, 'EdgeCloudSim: An environment for performance evaluation of Edge Computing systems', *2017 2nd International Conference on Fog and Mobile Edge Computing, FMEC 2017*, pp. 39–44, 2017, doi: 10.1109/FMEC.2017.7946405.

[74] C. Mechalikh, H. Taktak, and F. Moussa, 'PureEdgeSim: A Simulation Toolkit for Performance Evaluation of Cloud, Fog, and Pure Edge Computing Environments', *2019 International Conference on High Performance Computing and Simulation, HPCS 2019*, pp. 700–707, 2019, doi: 10.1109/HPCS48598.2019.9188059.

[75] K. V. Katsaros and V. Glykantzis, 'Experimenting with cache peering in multi-tenant 5G networks', *21st Conference on Innovation in Clouds, Internet and Networks, ICIN 2018*, pp. 1–5, 2018, doi: 10.1109/ICIN.2018.8401623.

[76] T. Chowdhury, ‘How Netflix uses Big Data Analytics to ensure success’, 2017, [Online]. Available: https://upxacademy.com/netflix-data-analytics/

[77] ‘Eclipse ioFog: Evolving Toward Native Kubernetes Orchestration at the Edge | Eclipse Foundation’. Accessed: Nov. 20, 2019. [Online]. Available: https://blogs.eclipse.org/post/mike-milinkovich/eclipse-iofog-evolving-toward-native-kubernetes-orchestration-edge

[78] Y. C. Hsieh *et al.*, ‘Managed edge computing on Internet-of-Things devices for smart city applications’, *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, pp. 1–2, 2018, doi: 10.1109/NOMS.2018.8406133.

[79] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, ‘Network function virtualization: State-of-the-art and research challenges’, *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016, doi: 10.1109/COMST.2015.2477041.

[80] ‘The CLOUDS Lab: Flagship Projects - Gridbus and Cloudbus’. Accessed: Oct. 31, 2020. [Online]. Available: http://www.cloudbus.org/cloudsim/

[81] M. Whaiduzzaman *et al.*, ‘A Privacy-Preserving Mobile and Fog Computing Framework to Trace and Prevent COVID-19 Community Transmission’, *IEEE J Biomed Health Inform*, vol. 24, no. 12, pp. 3564–3575, 2020, doi: 10.1109/JBHI.2020.3026060.

[82] M. C. S. Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, ‘CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness’, *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, no. i, pp. 400–406, 2017, doi: 10.23919/INM.2017.7987304.

[83] T. Zaitoun, M. B. Issa, S. Banat, and W. Mardini, ‘Evaluation and Enhancement of the EdgeCloudSim using Poisson Interarrival time and Load capacity’, *2018 8th International Conference on Computer Science and Information Technology, CSIT 2018*, pp. 7–12, 2018, doi: 10.1109/CSIT.2018.8486288.

[84] R. Buyya, R. Ranjan, and R. N. Calheiros, ‘Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities’, *Proceedings of the 2009 International*

Conference on High Performance Computing and Simulation, HPCS 2009, pp. 1–11, 2009, doi: 10.1109/HPCSIM.2009.5192685.

[85] C. Sonmez, A. Ozgovde, and C. Ersoy, 'Fuzzy Workload Orchestration for Edge Computing', *IEEE Transactions on Network and Service Management*, vol. 4537, no. i, pp. 1–14, 2019, doi: 10.1109/TNSM.2019.2901346.

[86] 'CagataySonmez/EdgeCloudSim: EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems'. Accessed: Oct. 31, 2020. [Online]. Available: https://github.com/CagataySonmez/EdgeCloudSim

[87] J. Skirelis and D. Navakauskas, 'Classifier based gateway for edge computing', *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering, AIEEE 2018 - Proceedings*, 2018, doi: 10.1109/AIEEE.2018.8592162.

[88] S. Lee and H. Kim, 'ACO-based Optimal Node Selection Method for QoE Improvement in MEC Environment', pp. 5–8.

[89] G. S. S. Aujla, N. Kumar, S. Garg, K. Kaur, and R. Ranjan, 'EDCSuS: Sustainable Edge Data Centers as a Service in SDN-enabled Vehicular Environment', *IEEE Transactions on Sustainable Computing*, vol. 3782, no. c, pp. 1–1, 2019, doi: 10.1109/TSUSC.2019.2907110.

[90] W. Shi *et al.*, 'Edge Computing: Vision and Challenges', *IEEE Internet Things J*, vol. 3, no. 5, pp. 637–646, 2016, doi: 10.1109/JIOT.2016.2579198.

[91] Y. Ku and S. Dey, 'Sustainable Vehicular Edge Computing Using Local and Solar-Powered Roadside Unit Resources', 2019.

[92] H. Lim and T. Hwang, 'Energy-Efficient Computing for Wireless Powered Mobile Edge Computing Systems', pp. 1–5, 2019.

[93] R. Freymann, J. Shi, J.-J. Chen, and K.-H. Chen, 'Renovation of EdgeCloudSim: An Efficient Discrete-Event Approach', *2021 6th International Conference on Fog and Mobile Edge Computing, FMEC 2021*, Sep. 2021, Accessed: May 20, 2023. [Online]. Available: https://arxiv.org/abs/2109.03901v1

[94] C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, 'Machine Learning-Based Workload Orchestrator for Vehicular Edge Computing', *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2020, doi: 10.1109/tits.2020.3024233.

[95] R. F. Atallah, C. M. Assi, and M. J. Khabbaz, ‘Network Using Deep Reinforcement Learning’, *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 5, pp. 1669–1682, 2019.

[96] T. Zaitoun, M. B. Issa, S. Banat, and W. Mardini, ‘Evaluation and Enhancement of the EdgeCloudSim using Poisson Interarrival time and Load capacity’, *2018 8th International Conference on Computer Science and Information Technology, CSIT 2018*, pp. 7–12, 2018, doi: 10.1109/CSIT.2018.8486288.

[97] M. G. Voskoglou, ‘Comparison of the COG Defuzzification Technique and Its Variations to the GPA Index’, 2016.

[98] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, ‘On Enabling 5G Automotive Systems Using Follow Me Edge-Cloud Concept’, *IEEE Trans Veh Technol*, vol. 67, no. 6, pp. 5302–5316, 2018, doi: 10.1109/TVT.2018.2805369.

[99] ‘GSMArena.com - mobile phone reviews, news, specifications and more…’ Accessed: May 23, 2023. [Online]. Available: https://www.gsmarena.com/

[100] ‘TechRadar | The source for tech buying advice’. Accessed: May 23, 2023. [Online]. Available: https://www.techradar.com/uk

[101] Y. Yang, ‘A Deep Reinforcement Learning Architecture for Multi-Stage Optimal Control’, Johns Hopkins University, 2019. Accessed: Jun. 20, 2021. [Online]. Available: https://arxiv.org/abs/1911.10684v1

[102] ‘Bellman Equation and dynamic programming | by Sanchit Tanwar | Analytics Vidhya | Medium’. Accessed: Feb. 07, 2021. [Online]. Available: https://medium.com/analytics-vidhya/bellman-equation-and-dynamic-programming-773ce67fc6a7

[103] J. Shen, O. Gu, and H. Liu, ‘Multi-agent hierarchical reinforcement learning by integrating options into MAXQ’, *First International Multi-Symposiums on Computer and Computational Sciences, IMSCCS’06*, vol. 1, pp. 676–682, 2006, doi: 10.1109/IMSCCS.2006.90.

[104] D. Kim, D. Lim, K. Park, and Y. S. Ihn, ‘Quantum-correlation-based free-space optical link with an active reflector’, *Current Applied Physics*, vol. 41, pp. 156–162, Sep. 2022, doi: 10.1016/J.CAP.2022.06.018.

[105] Y. Al-Karawi, R. S. Alhumaima, and H. Al-Raweshidy, ‘Quality of Service of Quantum Entanglement in Mobile Networks’, *IEEE Access*, vol. 9, pp. 167242–167251, 2021, doi: 10.1109/ACCESS.2021.3136782.