

https://dspace.org/

Machine Learning-based Computation Offloading in Energy-Harvesting 5G Networks

Phiri, Francis Tendai

2025-06
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/49857
Phiri, Francis Tendai. (2025). Machine Learning-based Computation Offloading in Energy-Harvesting 5G Networks. [Master's dissertation, University of the Witwatersrand, Johannesburg]. WIReDSpace.

Downloaded from a DSpace repository. https://dspace.org/



MASTERS DISSERTATION

Machine Learning-based Computation Offloading in Energy-Harvesting 5G Networks

By **Francis Tendai Phiri** – Student No. 1637626

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Research Supervisors:

Prof. Fambirai Takawira

Dr. Chabalala Chabalala

June 2025

Declaration

I declare that this dissertation submitted for the Master of Science in Engineering at the University of the Witwatersrand is my original work, except where otherwise acknowledged. This dissertation has not been submitted to another university or higher institution for any degree.

Signature of Candidate: 

Date:

30 June 2025

Abstract

This dissertation explores the application of machine learning for enhancing computation offloading in 5G networks, driven by three key areas in Mobile Edge Computing (MEC) research. Firstly, the work in this dissertation addresses computation offloading for enhanced Mobile Broadband (eMBB) applications requiring high transmission data rates, in a multi-user edge network with Energy-Harvesting (EH) devices. Integrating the Simultaneous Wireless Information and Power Transfer (SWIPT) technology provides a sustainable energy source for resource-constrained User Equipment (UE). An objective function that improves energy efficiency by minimizing UE energy consumption and task queueing delays while maximizing offloading transmission data rates is proposed. Deep Reinforcement Learning (DRL) is utilized in this work to overcome the limitations of traditional optimization methods, which struggle with scalability. The proposed approach uses a state-of-the-art DRL model, Twin Delayed Deep Deterministic Policy Gradient (TD3), and is shown to achieve superior optimization compared to prior works that rely on earlier DRL variants.

Next, this work focuses on the coexistence of eMBB and Ultra-Reliable Low-Latency Communication (URLLC) users, addressing the same objectives. A coexistence mechanism that enables both user types to share channel resources effectively is designed. For URLLC users, who demand minimal latency and high reliability, the 5G New Radio (NR) is used for short packet transmissions in small Transmission Time Intervals (TTIs), and a strict transmission reliability constraint is introduced. Optimally slicing wireless channel resources using DRL and the 5G flexible frame structure for uplink transmissions ensures that both user types meet their Quality of Service (QoS) requirements.

Lastly, as future work, this dissertation shows how to tackle the user association problem in a system with multiple Next Generation Node Bs (gNBs). A Federated Learning (FL) approach is adopted, where several gNBs share knowledge of UE states through Deep Neural Network (DNN) model parameters, which are first trained at each gNB using information about respective associated users and are then transmitted from the gNBs to a centralized Mobility Management Entity (MME). The MME aggregates these models and sends updated parameters back to the gNBs. This method saves communication bandwidth as model parameters are significantly lighter than training data. A global objective function is introduced to guide model training, optimizing data rates between users and their

associated gNBs. Simulation results show that all hyperparameter tuning conducted only affects the convergence rate of the chosen DRL model. The TD3 model performs better in terms of convergence rate, and variance, than its earlier variant, Deep Deterministic Policy Gradient (DDPG), which is widely used in prior DRL-based computation offloading works. Several TD3 policies evaluated against each other have shown that reward shaping is a crucial element in DRL and, especially in this work, that it results in trade-offs among eMBB energy consumption, throughput, delay, and URLLC reliability.

Acknowledgments

I would like to pass my sincerest gratitude to my supervisors Prof. Fambirai Takawira and Dr. Chabalala Chabalala for their unwavering support and guidance throughout this research. I appreciate all the discussions and the meetings we had. This research would not have been carried to this standard of quality without Prof. Takawira's vast knowledge of telecommunications. His leadership and unmatched integrity never made me doubt his contributions and suggestions at any point. Dr. Chabala's consistent, honest, and ethical supervision has given me an invaluable attitude and spirit for researching as he does. I have learned the value of persistence, patience, and integrity in producing high-quality work.

I would also like to thank my family, my two aunts, Elizabeth Hove (PhD Candidate) and Theresa Madamombe, for enabling the pathway to start my MSc. I will never forget all their financial support, wisdom, and guidance in my first year of MSc. To my mom and dad, your moral support and understanding, especially at times when I had to disappear for long periods, gave me the time and energy to work on this research. To my friend, James Zungunde (Conducting similar research with both of my supervisors), I will not forget all the times we discussed topics, model training, GPUs, and several aspects of our research topics that are aligned. Your friendship, technical knowledge, and moral support carried me right to the end.

I want to thank SENTECH for their generous financial support in the first two years. The SENTECH bursary allowed me to carry out this research and provide for my upkeep. The Wits University Business Intelligence Services (BIS) department took me for an internship role that allowed me to pay for my school fees, gain valuable experience, and access some premium GPUs for training my deep learning models. Thank you for all your support and the opportunity to gain valuable experience in areas that also contributed to my research.

My heartfelt appreciation also goes to all my other friends, family, and loved ones who contributed in any way to my growth and sanity, helping me to conquer the Wits Edge.

Contents

Declaration i
Abstract ii
Acknowledgments iv
List of Figures viii
List of Tables xi
List of Symbols xii
List of Acronyms xv
Resulting Publications xvii
1. Introduction 1
1.1. Background 1
1.2. Problem Statement 3
1.3. Research Question 5
1.4. Research Contributions 5
1.5. Research Methodology 7
1.6. Dissertation Structure 7
2. Literature Survey 9
2.1. Background Concepts 9
2.1.1. Mobile Edge Computing 9
2.1.2. Energy-Harvesting 10
2.1.3. Coexistence of eMBB and URLLC users 13
2.1.4. User Association 14
2.2. System Architectures Considered in MEC Research 15
2.2.1. Single User Mobile Edge Computing with Energy Harvesting 15
2.2.2. Multi-User Mobile Edge Computing 16
2.2.3. Multi-User Mobile Edge Computing in Small Cell Networks 17
2.2.4. Vehicular Edge Computing (VEC) 17
2.2.5. Edge Computing with IoT Clustering 18
2.2.6. UAV-Assisted Mobile Edge Computing 18
2.2.7. Mobile Edge Computing with High Altitude Balloons 18
2.2.8. SWIPT-enabled Mobile Edge Computing 18

2.2.9. Mobile Edge Computing with Resource Slicing for eMBB and URLLC User Coexistence 19
2.3. System Model Parameters in MEC-related works 20
2.3.1. Computation Model 20
2.3.2. Energy Consumption Model 21
2.3.3. Transmission Model 21
2.3.4. Energy-Harvesting Model 22
2.4. Optimization Problems in MEC-related works 23
2.4.1. Energy Consumption Minimization 23
2.4.2. Task Computation Delay Minimization 23
2.4.3. Execution Cost Minimization 24
2.4.4. Throughput Maximization 25
2.5. Solution Architectures in MEC-related Works 25
2.5.1. Analytical Approaches 25
2.5.2. Lyapunov-based Algorithms 26
2.5.3. Game Theory 27
2.5.4. Lagrange Multiplier 27
2.5.5. Heuristic Approaches 28
2.5.6. Machine Learning Solutions 28
2.6. Summary 32
3. Computation Offloading for Energy-Harvesting eMBB Applications 33
3.1. Introduction 33
3.2. System Model 35
3.2.1. Transmission Model 35
3.2.2. Task Model 37
3.2.3. Offloading Policy 37
3.2.4. Queuing Model 37
3.2.5. Energy Consumption Model 43
3.2.6. Energy Harvesting Model 44
3.3. Problem Formulation 45
3.4. DRL Modelling 47
3.4.1. Reinforcement Learning 47
3.4.2. Q-learning 48
3.4.3. Deep-Q-Network 49

3.4.4. Deep Deterministic Policy Gradient (DDPG) 49
3.4.5. Twin Delayed Deep Deterministic Policy Gradient Model (TD3) 51
3.4.6. TD3 vs DDPG 55
3.4.7. TD3-based Computation Offloading Scheme 56
3.4.8. State and Action Refinement 58
3.5. Complexity Analysis 62
3.5.1. System Reward Complexity 62
3.5.2. TD3 Training Complexity 63
3.6. Simulation Results and Analysis 63
3.6.1. Evaluation Schemes 63
3.6.2. Technology Stack Used for Implementation 64
3.6.3. Simulation Setup 66
3.6.4. Analysis and Discussion 68
3.7. Conclusion 81
4. TD3-based Channel Resource Slicing for eMBB and URLLC Coexistence 83
4.1. Introduction 83
4.2. Multiplexing eMBB and URLLC traffic in Uplink Transmissions 83
4.2.1. Downlink Transmissions 83
4.2.2. 5G New Radio 84
4.2.3. Uplink Transmissions 84
4.2.4. Proposed Uplink Multiplexing Scheme 86
4.3. Results and Discussion 91
4.4. Conclusion 98
5. Conclusion and Future Work 100
5.1. Conclusion 100
5.2. Future Work 101
5.2.1. Future Work and Improvements 101
References 103

List of Figures

Figure 1. 1: Research methodology..... 7

Figure 2. 1: Conceptual model of Mobile Edge Computing..... 10

Figure 2. 2: Conceptual model of RF energy harvesting. 11

Figure 2. 3: Receiver architecture designs for RF-powered information receivers. (a) Separated receiver architecture; (b) Time switching architecture; (c) Power splitting architecture. Copied from [14]..... 12

Figure 2. 4: Network model with the coexistence of eMBB and URLLC users..... 13

Figure 2. 5: Illustration of user association in MEC 15

Figure 2. 6: Single user single server with energy harvesting. 16

Figure 2. 7: Multi-user multi-MEC server architecture. 16

Figure 2. 8: SWIPT-enabled MEC. 19

Figure 2. 9: DNN architecture. 29

Figure 2. 10: Reinforcement learning architecture. 30

Figure 3. 1: Single gNB with energy-harvesting eMBB users..... 35

Figure 3. 2: Communication channel frame structure..... 36

Figure 3. 3: Queuing model showing the offloading, local, battery energy, and MEC server queues. 38

Figure 3. 4: Structure of the DDPG model. 50

Figure 3. 5: TD3 training structure 54

Figure 3. 6: Structure of the TD3-based computation offloading scheme. 58

Figure 3. 7: Procedure for refining the RB allocation actions. 61

Figure 3. 8: State and action refinement between the environment and the TD3 actor model 62

Figure 3. 9: Reward, sum data rate, sum energy consumption, and sum delay under different actor learning rates. 69

Figure 3. 10: Reward, sum data rate, sum energy consumption, and sum delay under different numbers of hidden layers of the actor and critic DNNs. 70

Figure 3. 11: Reward, sum data rate, sum energy consumption, and sum delay under different numbers of neurons per hidden layer of the Actor and Critic DNNs. 71

Figure 3. 12: Reward, sum data rate, sum energy consumption, and sum delay under different experience replay memory sizes..... 72

Figure 3. 13: TD3 vs DDPG in reward, variance, and average reward. 73

Figure 3. 14: Effect of task arrival rate on the system reward, sum data rate, sum energy consumption, and sum delay on Policies 1 – 4. 75

Figure 3. 15: Average offloading ratios for Policies 1 – 4. 76

Figure 3. 16: Effect of task arrival rate on local and offload queue lengths (tasks and bits) of Policies 1 – 4..... 78

Figure 3. 17: Plots showing the probabilities of violating constraints (3.45) and (3.46), top left and top right, respectively, and probabilities of local and offload queueing delay violation, bottom left and bottom right, respectively. Probabilities averaged over 11 eMBB users. 79

Figure 3. 18: Plot showing the probability of violating the constraint (3.47) by Policies 1, 2, 3, and 4. Probability values averaged over 11 eMBB users..... 79

Figure 3. 19: Plot showing the probability of violating the constraint (3.47) by Policies 1, 2, 3, 4, and 5..... 80

Figure 4. 1: Multiplexing of eMBB and URLLC transmissions – copied from [11]..... 84

Figure 4. 2: Clustering eMBB users and URLLC users for puncturing..... 86

Figure 4. 3: eMBB sum data rate and URLLC sum data rate as the URLLC task arrival probability increases 92

Figure 4. 4: Average number of URLLC packets that arrive in a timeslot as the URLLC probability of task arrival increases**.** 93

Figure 4. 5: Percentage of successful and failed URLLC transmissions as the URLLC users' task arrival probability increases. 93

Figure 4. 6: Percentage of failed URLLC transmissions due to eMBB resource allocation and insufficient URLLC channel rate as the URLLC users' task arrival probability increases..... 94

Figure 4. 7: Average number of code blocks available for URLLC transmissions as the URLLC users' task arrival probability increases. 95

Figure 4. 8: Jairness fairness index for Policies 4, 5, and 6 across all task arrival probabilities..... 96

Figure 4. 9: Percentage per CB of failed URLLC transmissions due to insufficient URLLC channel rate. 97

Figure 4. 10: URLLC outage probabilities of Policies 4, 5, and 6 as the URLLC task arrival probability increases. 98

List of Tables

Table 1. 1: Comparison of our work with the most prominent existing works on MEC. .. 4

Table 3. 1: Google Colab GPU specifications. Table adapted from [74]. 66

Table 3. 2: Environment simulation parameters..... 67

Table 3. 3: Baseline hyperparameter settings for TD3 model. 67

Table 3. 4: DNN structure of TD3 model..... 68

Table 3. 5: Policies 1 to 4 and their respective reward functions used in training. 73

Table 3. 6: Percentage change in energy consumption as task arrival rate increases for Policies 1, 2, 3, and 4. 77

Table 4. 1: Multiplexing simulation parameters…….....91

Table 4. 2: Different policies and their respective reward functions..... 92

List of Symbols

$\mathcal{D}^{(m)}$, $\mathcal{U}^{(m)}$	Set of eMBB users, URLLC users associated with gNodeB m respectively
$\mathcal{M}$	Set of gNodeBs in the network
$D^{(m)}$, $U^{(m)}$	Total number of eMBB users and URLLC associated with gNodeB m respectively
d	Index for an eMBB user
u	Index for a URLLC user
M	Total number of gNodeBs
m	Index for a gNodeB
$\mathcal{B}^{(m)}$	Set of RBs at gNodeB m
b	Index for a RB
$r_{d,b}^{(m)}[t]$	Data rate of eMBB user *d* on assigned RB *b* in timeslot *t*
$f_{d,b}^{(m)}[t]$	Bandwidth of eMBB user *d* on assigned RB *b* in timeslot *t*
$\propto_{d,b}^{(m)}[t]$	Small-scale channel gain of eMBB user *d* on assigned RB *b* in timeslot *t*
$g_{d,b}^{(m)}[t]$	Large-scale channel gain of eMBB user *d* on assigned RB *b* in timeslot *t*
$P_d^{(m)}[t]$	Assigned transmit power of eMBB user *d* in timeslot *t*
N_o	Single sided White Gaussian Noise Power Spectral Density
$\sigma_{d,b}^{(m)}[t]$	Resource Block allocation variable for user *d* and RB *b*
$R_d^{(m)}[t]$	Achieved total data rate of eMBB user *d* over all allocated RBs
$z_d^{(m)}$	Size of a task in bits
$c_d^{(m)}$	Required CPU cycles for computing a task
T_d^{max}	Latency deadline on a task

$\lambda_d^{(m)}$	Task arrival rate
$x_d^{(m)}[t]$	Offloading ratio
$\mathcal{T}_d^{(m)}$	Task identifier
$T_{d,lc}^{(m),\mathcal{T}}$	End-to-end local task delay
$T_{d,off}^{(m),\mathcal{T}}$	End-to-end offloading task delay
$Q_{d,lc}^{(m)}[t]$	Number of tasks in the local queue at time slot t
$Q_{d,off}^{(m)}[t]$	Number of tasks in the offloading queue at time slot t
$\rho_{d,lc}^{(m)}[t]$	Local queue traffic intensity at time slot t
$\rho_{d,off}^{(m)}[t]$	Offloading queue traffic intensity at time slot t
τ_p	Computation time of a bit
T_s	Slot duration
$\mu_d^{(m)}$	Service rate in tasks per slot
$f_{d,lc}^{(m)}$	Local CPU clock cycle frequency of user *d*
$E_d^{(m)}[t]$	Total energy consumed in slot *t* by user *d*
$H_d^{(m)}[t]$	Battery energy level of user *d* in slot *t*
$\omega_d^{(m)}[t]$	Total energy harvested in slot *t* by user *d*
υ	Energy conversion efficiency
η	Transmission power of gNodeB
$G_d^{(m)}$	Distance between user *d* and gNodeB *m*
ϑ	Path loss factor exponent
$\Omega_d^{(m)}$	Combined gain of gNodeB transmission antennae and user *d* antennae
$\bar{g}_d^{(m)}[t]$	Average large-scale channel gain over RBs allocated to user *d* in slot *t*
$\varkappa[t]$	System reward in slot *t*

$\Theta_{max}^{(m)}$	Maximum allowed outage probability of URLLC transmissions
$a_{d,b}^{(m)}[t]$	Number of punctured mini-slots of user *d* on RB *b* in slot *t*
$\mathcal{K}$	Set of all eMBB and URLLC users in the network
$\beta_k^{(m)}$	User association variable of user *k* to gNodeB *m*
$\boldsymbol{\theta_g}$	Global model weights
$\boldsymbol{\theta_m}$	Local model weights of gNodeB *m*

List of Acronyms

3GPP	3rd Generation Partnership Project
B5G	Beyond 5G
BS	Base Station
CPU	Central Processing Unit
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
EE	Energy-Efficiency
EH	Energy-Harvesting
EI	Edge Intelligence
eMBB	Enhanced Mobile Broadband
FL	Federated Learning
gNB	Next Generation Node B
HD	High Definition
ID	Information Decoding
IoT	Information of Things
MCC	Mobile Cloud Computing
MDP	Markov Decision Process
MEC	Mobile Edge Computing
MINLP	Mixed Integer Non-Linear Problem
ML	Machine Learning
MME	Mobility Management Entity
mMTC	Massive Machine Type Communication
NR	New Radio
PA	Primary Actor
PC	Primary Critic
PN	Primary Network
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RF	Radio Frequency
SBS	Small-Cell Base Station
SDG	Sustainable Development Goals
sTTI	Small Transmission Time Interval
SVM	Support Vector Machine
SWIPT	Simultaneous Wireless Information and Power Transfer
TA	Target Actor

TC	Target Critic
TD3	Twin Delayed Deep Deterministic Policy Gradient
TN	Target Network
TTI	Transmission Time Interval
UE	User Equipment
URLLC	Ultra-Reliable Low-Latency Communication
VR	Virtual Reality

Resulting Publications

Conference Proceedings – Awarded second-best paper

F.T. Phiri, F. Takawira, and C. Chabalala, “Machine Learning-based Computation Offloading in Energy-Harvesting eMBB Network: A Twin Delayed DDPG Approach”, in the Proceedings of the Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2024, pp. 130-139, ISBN: 978-1-0370-1018-7.

CHAPTER ONE

1. Introduction

1.1. Background

Modern technological advancements in computing capacity have seen an increase in the use of mobile devices that have become an integral part of our day-to-day lives. As technology continues to evolve, new mobile applications that demand high computation capacities, such as facial recognition, natural language processing, augmented reality, and online gaming, are being developed to support the need for security, entertainment, Quality of Experience (QoE), and business operations. Mobile devices sometimes have limited battery energy, storage, and computation resources to support these applications. It is essential, therefore, to develop effective solutions that mitigate these mobile device constraints.

One technology in 5G wireless networks developed to tackle the mobile device limitations is Mobile Edge Computing (MEC). MEC brings large compute resources closer to mobile device users and allows them to run computationally intensive applications. Instead of running the applications on their local processing units, mobile device users can transmit computation tasks over wireless channels to high-performance edge servers that are situated within their Radio Access Network (RAN). Although this solution removes the bottlenecks of battery energy, storage, and processing capacity, it still requires efficient offloading schemes, especially when considering multiple users, limited wireless channel resources, and edge server compute resources.

The escalating global greenhouse and climate change have attracted a lot of attention from across all sectors. The International Commission on Climate Change (ICCC) has estimated that if the current trends in the use of fossil fuel and deforestation are not reversed, the average global temperature in the next century will rise by 6 degrees Celsius [3, 4], leading to extreme weather conditions. To slow down the greenhouse effect, the United Nations has come up with Sustainable Development Goals (SDGs) [3], which are to be pursued across all sectors [2]. In the telecommunications space, designing sustainable, energy-efficient cellular networks and systems has become an urgent goal to contribute to the SDGs. For this matter, cellular network operators face the costs and penalties of neglecting energy-efficient systems. A significant portion of the operational expenditure of cellular networks goes to pay the electricity bill, which is estimated to be more than 10

billion dollars a year [2]. Since Energy-Efficiency (EE) is directly tied to profitability, new energy-efficient solutions are required from the perspectives of both cellular operators and mobile device users to drive profitability and save the environment.

The several outcomes of the 5G project such as high data rates, low latency, and high device connectivity, have resulted in a classification of services offered by 5G wireless networks under the new 5G New Radio (NR). Two categories, which are the focus of this research, are: enhanced Mobile Broadband (eMBB), and Ultra-Reliable-Low-Latency-Communications (URLLC) [11]. The eMBB service is designed to support high data rate applications such as HD streaming and Virtual Reality (VR) applications [11]. URLLC service, on the other hand, targets mission-critical applications that require ultra-low latencies such as autonomous vehicles, remote surgery, and the tactile internet [11]. The 5G NR is a wireless transmission method designed to support URLLC services so that URLLC packet transmissions are localized in very short Transmission Time Intervals (TTI) [11]. The 5G NR also offers techniques that allow both types of service to efficiently share the same spectrum while meeting Quality of Service (QoS) requirements. The coexistence of URLLC and eMBB services on the same radio resource leads to a challenging scheduling problem that is not straightforward to tackle due to an underlying trade-off among latency, data rate, and reliability [11].

The advancements in wireless communications towards Beyond 5G Networks (B5G) will place very high demands on ultra-high data rates, spectrum efficiency, energy efficiency, extremely high reliability, and ubiquitous computing. The future 6G ecosystem will be envisioned to accommodate various stakeholders such as users, mobile devices, vehicles, sensors, machines, and things [5]. As such, B5G networks are anticipated to be highly complex and dynamic, requiring the need to move away from traditional algorithms that are difficult to implement, scale, and are inefficient for various network functions. Artificial intelligence (AI) and Machine Learning (ML) solutions can offer superior outcomes in complex networks [5]. AI and ML techniques can be used to learn sophisticated patterns from data collected from the dynamic network environment and to make predictions about problems that can then be addressed ahead of time. Edge Intelligence (EI) is a paradigm that introduces Artificial Intelligence on the edge of the network where edge servers can leverage powerful ML techniques to extract valuable knowledge and make adaptive decisions, thus offering distributed and low-latency services to end-users [6]. Finding appropriate machine learning algorithms and techniques for specific applications and tuning them for improved efficiency and performance is a task that needs to be addressed.

Motivated by several works in literature and the aforementioned concerns regarding the need for efficient offloading schemes, exploring energy-efficient solutions, addressing the issue of multi-service coexistence in 5G networks, and tapping into the power of machine learning for advancing telecommunication networks, the work in this dissertation seeks to develop an energy-efficient task offloading scheme that leverages the power of machine learning on the edge of networks for multi-service users.

1.2. Problem Rationale

Several works in literature have studied various MEC architectures and computation offloading schemes that leverage machine learning. The offloading schemes have been addressed as optimization problems that aim to do one or more of the following:

- Minimize energy consumption of mobile user devices and/or edge servers.
- Minimize the latency of task queueing and execution.
- Maximize the data rate and/or transmission reliability of eMBB users.
- Ensure the latency constraints of URLLC users are met.

The most prominent prior works in computation offloading research have several shortcomings, which are addressed in this dissertation. In [7], the authors studied a single-user offloading problem for an energy-harvesting mobile user. The mobile UE is equipped with energy-harvesting circuitry and powered purely by the harvested energy. The authors optimize the computation offloading decision by considering factors such as the length of the computation task queue, the executing state of the mobile device, and the state of the transmission unit. Though important to the work in this dissertation, the paper studied a very simple system architecture comprising a single EH user.

The papers in [8] and [9] studied a multi-user and multi-server computation offloading system. In this case, the computation offloading decision is more complicated due to the dynamic nature of channel resources, MEC computing resources, and the number of users. User association plays a vital role in this system to optimally connect users to the best servers based on conditions such as channel quality, distance to a server, and availability of compute resources on the edge servers. In [8], a machine learning solution was used for a digital twin-enabled network to manage user association for eMBB and URLLC services. The authors did not, however, consider the use of the 5G NR to support wireless transmissions, and this does not conform to the 5G standard. In [9], a game-theoretic approach was implemented to manage the user association. As mentioned before, traditional-based approaches may not offer the required optimization in very complex B5G

Table 1. 1: Comparison of this dissertation with the prominent existing works on MEC.

Papers	System Model Architectures				Optimization Problems and Solutions			
	Multi-Users	Multi-Servers	Energy-Harvesting	eMBB-URLLC Coexistence	Machine-Learning Solution	Energy Consumption Minimization	Delay Minimization	Throughput Maximization
Alsenwi *et al.* [11]	✓	×	×	✓	✓	×	×	✓
Wang *et al.* [30]	✓	✓	×	×	✓	✓	✓	×
Huang *et al.* [19]	✓	✓	×	×	✓	✓	×	×
Yun *et al.* [62]	✓	✓	×	✓	✓	✓	×	×
Min *et al.* [32]	×	✓	✓	×	✓	✓	×	×
Mao *et al.* [7]	×	×	✓	×	✓	×	×	×
Huang *et al.* [54]	✓	×	×	✓	✓	×	×	✓
Bairagi *et al.* [31]	✓	×	×	✓	×	×	×	✓
Tun *et al.* [33]	✓	×	×	✓	×	✓	×	✓
Chen *et al.* [55]	✓	✓	×	×	✓	×	×	×
Al-Ali *et al.* [63]	✓	×	×	✓	×	×	×	✓
Sui *et al.* [64]	✓	×	×	✓	×	✓	×	✓
Hsu *et al.* [65]	✓	×	×	✓	✓	×	×	✓
Li *et al.* [66]	✓	×	×	✓	✓	×	×	✓
This Dissertation	✓	✓	✓	✓	✓	✓	✓	✓

networks. The Simultaneous Wireless Information and Power Transfer (SWIPT) has been studied in [10] where RF energy-harvesting is done by computation offloading users. Since RFs carry both energy and information, energy-deficient UEs can simultaneously decode information and convert the downlink radio signal energy into battery-powered energy used to enhance computation offloading and device operations. This is an important concept in future green sustainable cellular networks. The study conducted did not, however, use a machine learning technique.

The study also did not consider the different types of 5G services. The papers in [8], [11], and [12] have studied the context of multiplexing URLLC and eMBB user services. The work in [11] considers a simple multiplexing problem over downlink traffic without any focus on the uplink computation offloading aspect, and the works in [8] and [12] introduced both services into computation offloading in the uplink scenario without using the 5G NR. Important to the work in this dissertation are the deep learning solutions implemented in all the multiplexing scenarios. The works in [11] and [12] in particular, made use of a Deep Reinforcement Learning (DRL) solution where in [11], the motivation

was to tackle the non-deterministic nature of the arrival of URLLC short-packets, and the problem involved throughput maximization for eMBB users while satisfying URLLC latency requirements.

In [12], the motivation for a DRL-based solution was to leverage a global experience replay memory in a multi-user and multi-server system, and the problem involved energy consumption minimization. The work in [8] made use of a DNN for user association in a multi-user and multi-server computation offloading system for energy consumption minimization. The implementation used a near-brute-force strategy using specified exploration and exploitation schemes for labelled data generation. MEC has been extensively studied in various contexts, which will be discussed in Chapter 2. Some works study system models that form one or more combinations of the architectures shown in Table 1.1. Table 1.1 shows the research contributions of some of the most prominent works on computation offloading. Papers that have studied energy harvesting have not included several issues in Table 1.1. Moreover, little has been done in energy harvesting and task delay minimization. This work presented in this dissertation seeks to compile a system model that addresses all the aspects in Table 1.1. To the best of the author's knowledge, no prior work has been able to perform this assimilation.

1.3. Research Question

In this dissertation, the shortcomings of the related prior works are addressed, and the following questions are answered:

How can Machine Learning be effectively applied to the joint optimization of energy consumption, transmission data rate, and task queueing delay in multi-service Mobile Edge Computing?

The main research question is broken into the following sub-questions:

1. How can a Mobile Edge Computing system be modelled that consists of Energy-Harvesting eMBB and URLLC users?
2. How can an objective function be formulated that optimizes energy consumption, transmission data rate, and task queueing delay?
3. How can Deep Reinforcement Learning be used to jointly optimize computation offloading and scheduling of eMBB and URLLC users?

1.4. Research Contributions

The main contributions of this dissertation are summarized as follows:

1. An MEC network system that incorporates energy harvesting in multi-user and multi-service computation offloading using the 5G New Radio is modeled. Multiple Next Generation Node Bs (gNBs) are considered, where each gNB has a server compute unit for processing user tasks. Users can, therefore, associate and perform computation offloading to any one of the servers. A Mobility Management Entity (MME) is used to allow communication between the several gNBs for user association management. To the best of the author's knowledge, this is the first work to investigate energy harvesting in task offloading multi-service networks using the 5G New Radio for uplink transmissions.

2. A formulation is done for an efficient joint task offloading and scheduling of eMBB and URLLC users' problem that minimizes energy consumption and task queuing delay, as well as maximizing the achievable uplink transmission data rates of the eMBB users subject to a resource allocation constraint, latency, and reliability constraints of URLLC users. This optimization problem is non-convex and NP-hard, which has a combinatorial complexity. Several works investigated multi-objective problems that consider only one or two problems of energy consumption minimization, task queueing delay minimization, and throughput maximization. To the best of the author's knowledge, this is the first work that considers a multi-objective function that addresses all 3 problems at the same time.

3. The main problem in contribution 2 is decomposed into two sub-problems that are solved on two timescales at the MME and the gNBs, respectively. At the gNBs, a state-of-the-art DRL algorithm, Twin Delayed Deep Deterministic Policy Gradient (TD3), is utilized to achieve better DRL performance compared to prior works that have mostly relied on the earlier variants of TD3. For future work, this dissertation shows how a DNN-based Federated Learning algorithm can be used for effective user association at the MME. To the best of the author's knowledge, this is the first work to utilize a TD3-based machine learning solution and to propose a federated learning-based solution to achieve the objectives in Contribution 2.

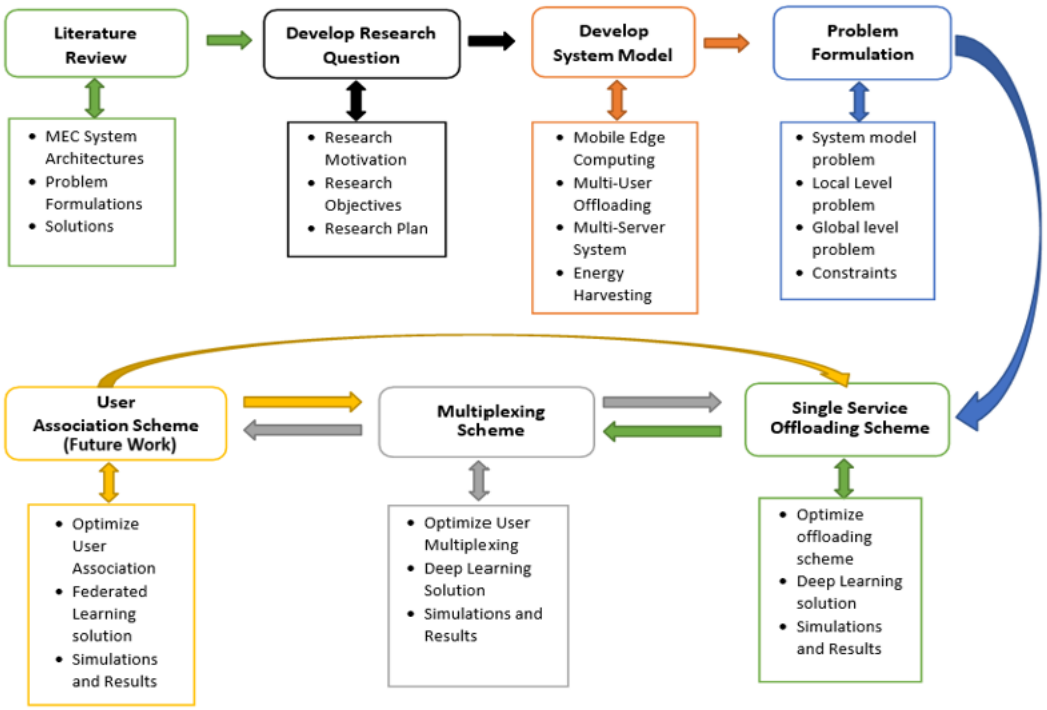

Figure 1. 1: Research methodology.

1.5. Research Methodology

The proposed research methodology aims to design a computation offloading scheme for a Mobile Edge Network that consists of eMBB and URLLC users that can associate with multiple MEC servers. The scheme's objective is to minimize the energy consumption of UE devices while guaranteeing QoS for both types of users. The target for eMBB users is to increase their transmission data rates, while that for URLLC users is to meet their latency and reliability requirements. The computation offloading scheme will use deep reinforcement learning. For future work, federated learning will be utilized to enhance the computation offloading scheme. The research methodology is a 7-stage process shown in Figure 1.1.

1.6. Dissertation Structure

The rest of this dissertation is organized as follows:

- Chapter 2 provides an extensive review of the literature that has motivated the work in this dissertation. Background concepts are outlined to provide an understanding of the basic system architectures, problems formulated, and solutions proposed in the literature.
- Chapter 3 introduces the work that investigates single-service computation offloading in energy-harvesting eMBB networks for a single gNB with multiple

users. The Chapter addresses the network system model, the problem formulation, and the DRL solution used to solve the optimization problem. Results are then discussed and analyzed.

- Chapter 4 extends the system model in Chapter 3 by introducing the multiplexing problem for eMBB and URLLC user coexistence in 5G NR. A constraint is formulated that ensures URLLC users' reliability requirements are satisfied. This constraint is assimilated into the original objective in Chapter 3, and results are discussed and analysed.
- Chapter 5 then provides a conclusion of the entire dissertation and discusses recommendations for future work, including a detailed description of the user association problem that aims to use a DNN-based Federated Learning system model, problem formulation, and solution framework for optimizing the association of users to several gNBs.

CHAPTER TWO

2. Literature Survey

This Chapter outlines the related works that have been done in MEC, which have motivated the work in this dissertation. Section 2.1 starts by introducing the background concepts used in the literature to understand the work done in the rest of this dissertation. Section 2.2 delves into detailed descriptions of the various system architectures considered in MEC literature. Section 2.3 describes, in general, the system model parameters and equations used to model the various architectures in MEC literature. Section 2.4 then looks at the optimization problems that have been considered in MEC literature. The Chapter then ends with Section 2.5, which outlines some of the solution architectures that have been used to solve the optimization problems in MEC literature.

2.1. Background Concepts

2.1.1. Mobile Edge Computing

Mobile Cloud Computing (MCC) is a networking technology that offers networking, computing resources, and internet services to user devices that are constrained in power, processing, and storage [13]. Users utilize wireless networks to access remote data centres or compute clusters that they can use to process or store large amounts of data [13]. The MCC paradigm has a centralized architecture with farms of cloud servers usually placed at one or a few locations far from users [13]. This approach imposes a high load on radio and backhaul networks and introduces high latencies, especially when processing must be done over large geographical distances. The concept of MEC, shown in Figure 2.1, was proposed by the European Telecommunications Standard Institute (ETSI) in 2014 as an evolution to MCC and was defined as a new platform that "provides IT and cloud-computing capabilities within the Radio Access Network (RAN) close to users [13].

UE devices have limited computation power, storage, and battery energy to sufficiently process heavy computational applications. Users, therefore, can offload some or all the computational tasks to a Base Station (BS) that is geographically located closer to them. This BS, which is typically within the RAN of the users, has an edge server that has a high computation capacity for handling user tasks. This brings several advantages to the users, such as extended battery life, enabling sophisticated applications, and providing higher storage capabilities [13]. The MEC server can either handle a user request and respond

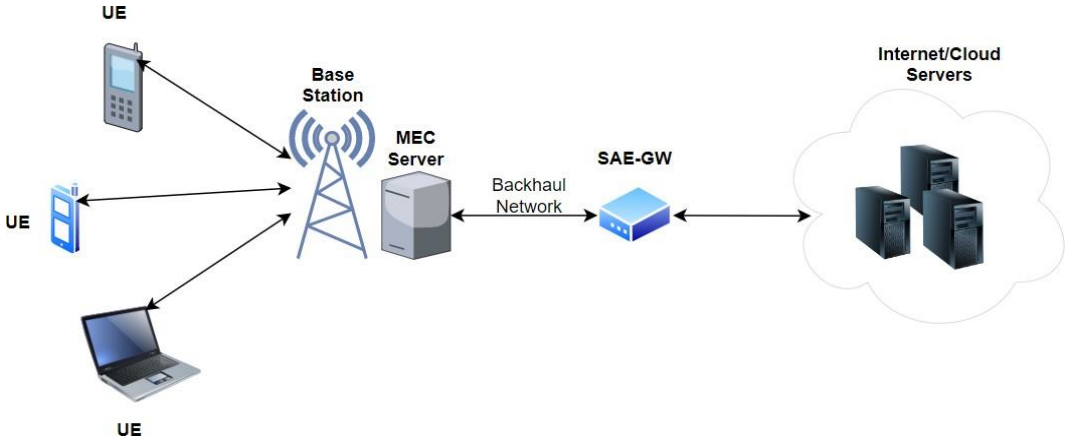

Figure 2. 1: Conceptual model of Mobile Edge Computing.

directly to the UE or forward the request to remote cloud servers. By deploying the server to the edge of the network, as in Figure 2.1, the backhaul network is decongested, thus reducing latency [13]. The MEC server typically, however, provides limited computational and storage resources compared to centralized servers in MCC. This limitation imposes a reduction in service performance, especially when there are too many service requests. Moreover, the offloading process is not always feasible or beneficial due to unpredictable channel conditions, the dynamic BS workload, latency constraints, computation complexity, energy consumption, or memory requirements [13]. Thus, the offloading process is preceded by a decision of whether to offload or not. This has motivated current research from both industry and academia to focus on improving the performance of MEC networks by studying different areas such as optimization of computation offloading decisions, wireless channel, and computation resource allocation, content caching, mobility management, etc [13]. Most of these research efforts are conducted to continuously promote latency reduction and energy efficiency in MEC networks [13]. By doing so, the next-generation MEC networks can meet data rate and latency QoS requirements and provide ubiquitous applications and services in the 5G and beyond networks.

2.1.2. Energy-Harvesting

Energy Efficiency (EE) is one of the most important aspects of the next generation 5G and beyond networks, which have a primary focus on green communications [14]. The ongoing advances in wireless network services and mobile devices call for the need to have energy solutions that can sustain ubiquitous services and applications for QoS provisions. The advent of the Internet of Things (IoT) has shown the urgent need for EE [14]. The various sensors and devices in IoT require sufficient energy to communicate reliably and

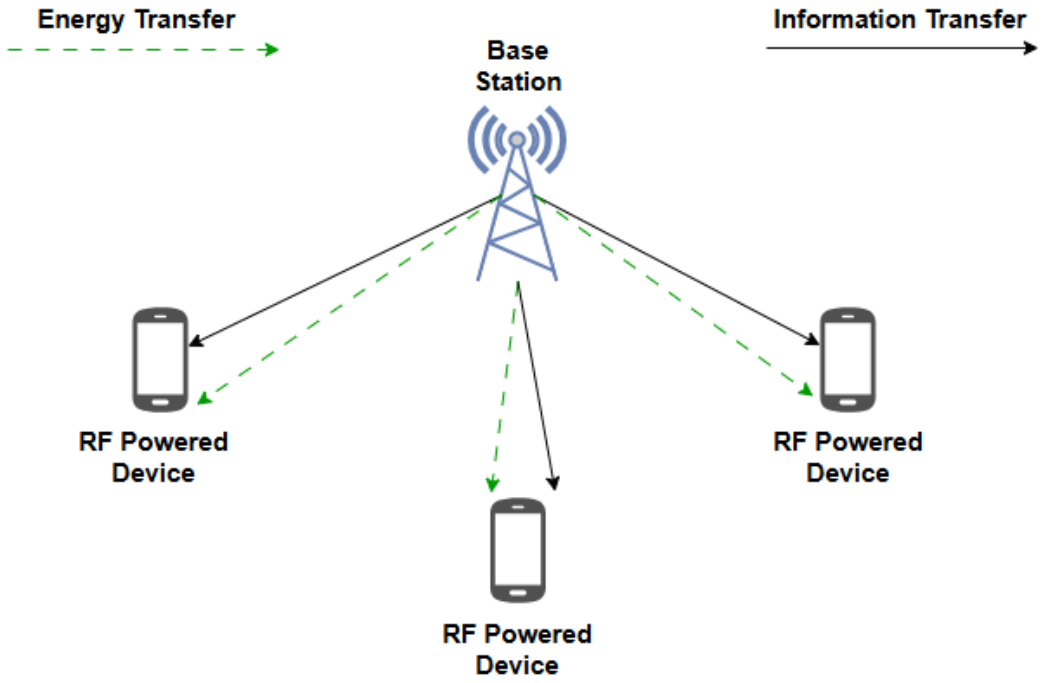

Figure 2. 2: Conceptual model of RF energy harvesting.

efficiently. The devices are usually powered by batteries as energy sources. Research has, however, shown that in some applications, relying on battery power is not a feasible option as they need constant replacements [14].

For instance, sensors embedded within building structures, the human body, or in very remote areas may increase the costs and risks associated with battery maintenance and replacement [14]. Harvesting energy from ambient sources is an excellent alternative to battery power for increasing device lifetime. However, the random and intermittent nature of energy sources such as solar, thermal, and wind makes it problematic and unreliable to be used where QoS is of utmost importance [14]. Radio Frequency (RF) energy harvesting offers significant benefits such as low cost, easy access to radio frequency energy, and perpetual availability [14]. The RF transmitter transmits information signals, which can be picked up by the UE device antennas. UE devices can be equipped with energy harvesting circuitry that converts the RF signal into electric energy that can be stored on the device battery or used immediately to power the device [14].

The paradigm for simultaneously receiving information and harvesting energy from RF signals is called Simultaneous Wireless Information and Power Transfer (SWIPT) [2]. In SWIPT, either dedicated or ambient RF sources are used for transmitting energy and information simultaneously [2]. This architecture promotes EE and Spectral Efficiency [2].

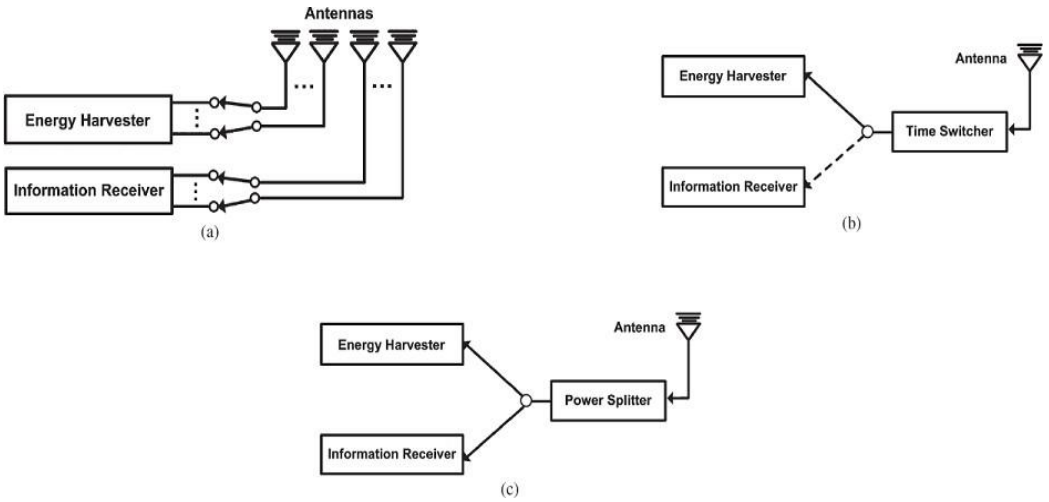

Figure 2. 3: Receiver architecture designs for RF-powered information receivers. (a) Separated receiver architecture; (b) Time switching architecture; (c) Power splitting architecture. Copied from [14].

In theory, it is possible to extract both energy and information from the same waveform in SWIPT [2]. However, practical implementation has been unattainable due to circuit design challenges, such as the differing power sensitivities of antennas on the receiver side [2]. To implement SWIPT, the signal needs to be divided into two parts: one for energy harvesting and the other for information decoding. There are two main techniques commonly used for the practical realization of SWIPT:

1. Separated Receiver – In this receiver architecture, the UE has separate antennas for both EH and Information Decoding (ID). EH and ID can, therefore, be done concurrently and independently [14].

2. Integrated Receiver – In this receiver architecture, there are no separate antennas for EH and ID. This architecture has two types of designs:
    a) Time Switching – This type of integrated receiver has a single antenna for doing both EH and ID. A switch is employed at the receiver to switch between EH and ID periodically [14]. The time-switching receiver needs precise information and energy scheduling, as well as synchronization of time, along with the advantage of simplicity in hardware.
    b) Power Splitting – this architecture involves splitting the signal using a power splitter, with one part of the signal used for EH and the other used for ID [14].

2.1.3. Coexistence of eMBB and URLLC users

The network services offered by the 5G NR fall under three categories, which are enhanced Mobile Broadband (eMBB), massive Machine Type Communications (mMTC), and Ultra-Reliable Low-Latency Communications (URLLC) [15]. The eMBB service is an advancement of the existing mobile broadband service in current LTE networks and is designed to accommodate high data rate applications such as high-definition video streaming [15]. The objective QoS of this type of service is, therefore, to achieve high data rates while satisfying moderate reliability. Due to the high data rate requirement, it is necessary to perform packet coding and transmission over large transmission blocks for long time intervals [15].

Figure 2.4 below shows the coexistence of eMBB and URLLC users in a downlink network. On the other hand, services like industrial automation, autonomous vehicles, and e-healthcare systems are classified as URLLC applications. These services target mission-critical communications. Applications such as remote surgery and drone operations need real-time operations that require ultra-high reliability and very low latency. Due to the low latency requirement, packet transmission must be localized in time with short TTIs [15]. URLLC traffic is naturally infrequent and needs to be addressed spontaneously [15]. The mMTC service aims to serve many IoT devices that send data intermittently with a low and fixed transmission rate [15]. Multiplexing eMBB and URLLC services in the same network over the same resource blocks while guaranteeing QoS provision for both types of services is a challenge that has stimulated a growing body of research in the 5G and Beyond networks [15]. In the 3rd Generation Partnership Project (3GPP) standard, two approaches have been proposed.

1. *Orthogonal Scheduling:* In this type of scheduling, a frequency band is reserved for URLLC transmissions [15]. This approach has the downside of wasting channel

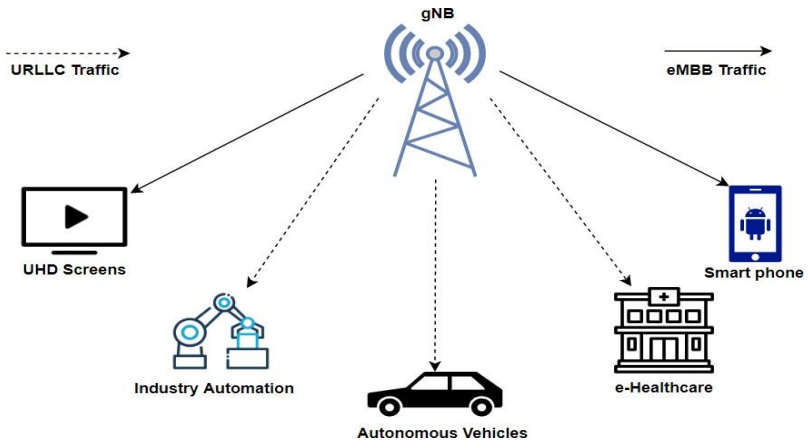

Figure 2. 4: Network model with the coexistence of eMBB and URLLC users.

resources, especially when there is no URLLC traffic. To this end, another approach is proposed that uses the spectrum more efficiently [15].

2. *Pre-emptive Scheduling (Puncturing):* In this mechanism, incoming URLLC packets are immediately scheduled on top of the ongoing eMBB transmissions in very short Transmission Time Intervals (sTTI) [15]. The immediate puncturing and transmission over sTTIs ensure that the URLLC service's strict latency requirements are satisfied. In downlink transmissions, the gNB, stops the eMBB transmissions during these TTI and schedules the URLLC packets [15]. This mechanism is effective in reducing URLLC latency, but the eMBB transmission data rate and reliability are negatively impacted [15]. Therefore, there is a need to create an effective coexistence mechanism that reduces or preserves eMBB transmission data rate and reliability while ensuring the latency demands of URLLC transmissions are satisfied [15].

2.1.4. User Association

Mobile edge networks consist of users who may not be stationary while offloading computation tasks. User association is a concept that defines which BS a user connects to at a particular point in time to perform computation offloading [4]. Figure 2.5 illustrates how a mobile offloading user may associate with multiple BSs. In the model, the mobile user can send offload requests for intensive computation to the MEC servers [4]. As the user moves around the network, there is a need to ensure the continuity of the computation of tasks. Since the user associates with a single base station at a particular point in time, coordination amongst the base stations is essential to allow for the choice of the best MEC server that gives the best computation offloading QoS and for a smooth and seamless handover procedure of task computations [4].

Energy consumption and latency in such a network have been associated with the level of service discontinuity and choice of BS by the user at a particular instance [16]. Resource allocation, therefore, becomes complex since the resources should be reallocated over time to adapt to the changes in the quality of communication channels (due to UE mobility) and the availability of computing resources at the MEC servers [16]. Several user association

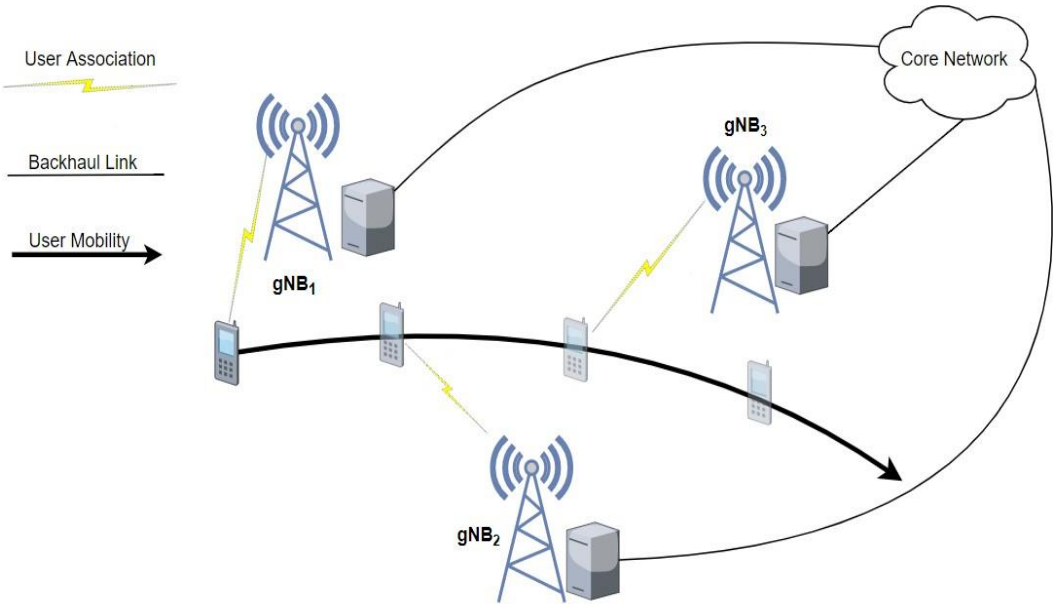

Figure 2. 5: Illustration of user association in MEC

schemes and handover procedures or algorithms have been studied in the literature, some of which adopt mobility predictions to proactively allocate channel and computation resources. In this dissertation, attention is given to the user association problem for users who wish to offload computation tasks to several MEC servers.

2.2. System Architectures Considered in MEC Research

This section gives an overview of the various system architectures proposed in the literature for MEC to bring computation closer to mobile users.

2.2.1. Single User Mobile Edge Computing with Energy Harvesting

Figure 2.6 illustrates an MEC system that consists of a single mobile device and a single MEC server. The mobile device is equipped with EH circuitry and powered purely by harvested renewable energy. The energy can be harvested from ambient renewable sources such as solar, thermal, and wind [17]. The MEC server could be a small data center managed by a telecom operator and can be accessed by a mobile device through a wireless channel. The mobile device is associated with a system-level clone at the MEC server, which runs a virtual machine and can execute the computation tasks on behalf of the mobile device [18].

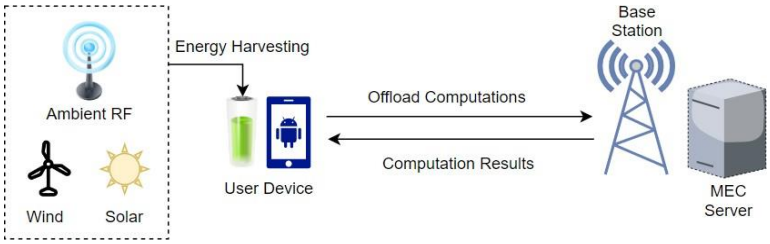

Figure 2. 6: Single user single server with energy harvesting.

By offloading the computation tasks for mobile-edge execution, the computation experience can be improved significantly. In a single-user scenario, the offloading decision of one user does not affect the performance of other users [18]. The main factors affecting the computation offloading decision in a single-user scenario are the length of the task queue, the executing state of the mobile device, and the state of the transmission unit [18]. This type of architecture was used in [18].

2.2.2. Multi-User Mobile Edge Computing

This model extends the single-user scenario where several users offload to either a single MEC server or multiple MEC servers. Figure 2.7 shows the typical system with users that can connect to any of the MEC servers. The MEC servers could be connected to a central point, like a Mobility Management Entity (MME) [8] or a cloud server [9], to manage and coordinate the resource allocation between the MEC servers. In this case, the computation offloading decision is more complicated because the network wireless channel resources, MEC computing resources, and the number of users are dynamically changing. For the case of multiple MEC servers, the concept of user association would play a vital role in managing which users connect to which MEC servers based on several conditions, such as

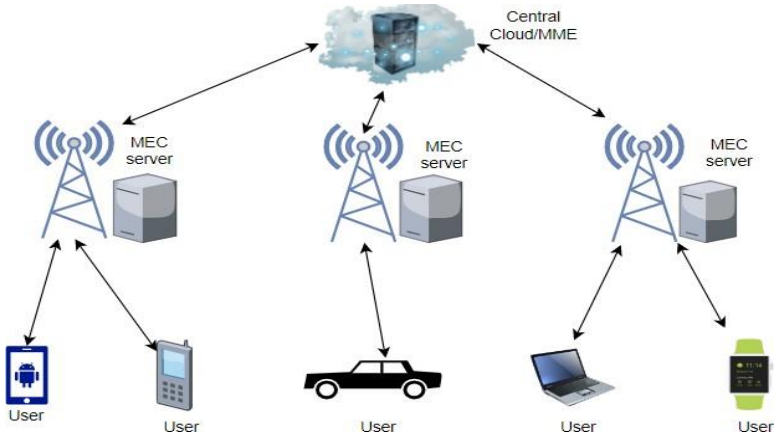

Figure 2. 7: Multi-user multi-MEC server architecture.

channel quality, availability of channel, and computation resources. This architecture has been used in several papers, including [8] and [9].

2.2.3. Multi-User Mobile Edge Computing in Small Cell Networks

This type of model takes the form of a multi-user and multi-MEC server architecture. The network consists of randomly distributed Small Cell Base Stations (SBSs) and randomly distributed users [19], [20]. The SBSs are equipped with MEC servers for computing offloaded tasks. The network is arranged into small cells where an SBS is connected to users within its vicinity. The goal of this type of architecture is to manage user association and the level of channel interference between the small cells (intercell interference) [19]. The SBSs would need to communicate with each other to manage an efficient allocation of channel and compute resources while minimizing intercell interference. The papers in [19] and [20] utilize this architecture to formulate their system models.

2.2.4. Vehicular Edge Computing (VEC)

This is a model that offers smart transportation systems (self-driving vehicles). It allows smart vehicles with limited onboard computing capacity and bandwidth connectivity to maximize computation capacity by offloading the overloaded tasks to neighboring nodes [21]. These smart vehicles use different Vehicle-to-Everything (V2X) communications to boost networking capabilities toward massive autonomous vehicle usage [21]. These include Dedicated Short-Range Communication (DSRC), 5G Millimetre Wave (5G-mmWave), and 6G Vehicle-to-Vehicle [21], [22]. The architecture comprises three levels as illustrated by the system model in [21].

At the highest level, there are multiple VEC servers, collectively referred to as the VEC pool, which manage coordination. The middle level comprises VEC servers, a key component of infrastructure edge computing, facilitating communication between vehicles and infrastructure [21]. The lower level consists of the heterogeneous V2X technologies. To minimize offloading delays, the VEC servers are positioned alongside their dedicated base stations at the roadside units within vehicular networks [21]. Here, V2X communications and computing are considered the services provided by the VEC servers to end devices called vehicles. The VEC architecture has been explored in [21], [22], and [23], which have all used a three-tier level architecture.

2.2.5. Edge Computing with IoT Clustering

This model follows the multi-user multi-MEC server three-layer hierarchical architecture. The MEC servers are attached to gateways. Each gateway is connected to a group of IoT devices. The gateways are then connected to a centralized cloud. The edge servers have limited computation capacity and spectrum resources such that a gateway can only allow a limited number of devices to be accessed at the same time [24]. This architecture is unique in that it explores a centralized user clustering to group the IoT users into different clusters according to users' priorities [24]. The cluster with the highest priority is designated to offload computation tasks to the edge server for execution, whereas the cluster with the lowest priority performs computation tasks locally [24]. This architecture has been utilized in [24], [25], [26], and [27], all with a similar type of clustering.

2.2.6. UAV-Assisted Mobile Edge Computing

Unmanned Aerial Vehicles (UAVs) are mobile and come equipped with cameras, sensors, computing resources, and communication devices [28]. For this reason, an MEC system can be managed by UAVs acting as mobile servers where users can offload tasks. However, UAVs are not capable of processing highly intensive computations due to limitations on computing resources and battery life [28]. To overcome this problem, the UAVs can offload some tasks to a nearby MEC server attached to a cellular BS with enough computing power. When offload requests occur in one area, one or more UAVs should move to that location to process the user tasks. If the data processing is too heavy to process locally, the UAV can cooperate with an MEC server attached to a BS on the ground [28]. This architecture is used in [28] and [29].

2.2.7. Mobile Edge Computing with High Altitude Balloons

High Altitude Balloons (HAB) have low deployment expenses and a large coverage range [20]. MEC servers can be deployed at each HAB to allow users to offload computation tasks to these HABs. Compared with UAVs, HABs do not have limited flight time and are bigger, therefore, they can be equipped with powerful computing resources and can continuously hover to serve ground users without the need to act as relays to ground base stations [20]. In [20], HABs are used for computation offloading.

2.2.8. SWIPT-enabled Mobile Edge Computing

RF energy harvesting is a paradigm within the EH literature that is proven to be suited for low-power IoT devices [10]. Since RFs carry information and energy, energy-deficient nodes can decode information and scavenge energy simultaneously. RF EH offers

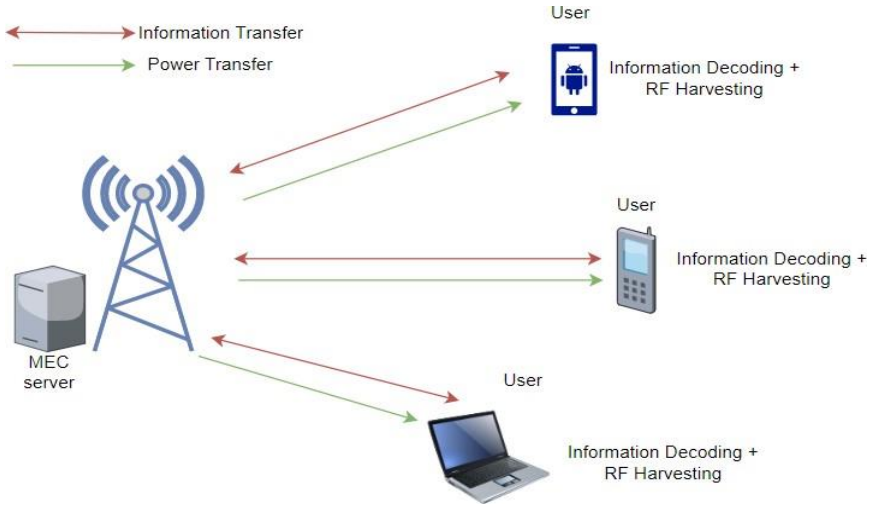

Figure 2. 8: SWIPT-enabled MEC.

significant advantages like low cost and easy access from BSs already transmitting information signals [10]. In the context of MEC, UE devices are modeled to have RF EH circuitry to harvest energy from downlink traffic frequency signals. This energy can be stored or used immediately to power the devices for computation offloading. Energy harvested provides good QoE for offloading users by providing energy for local computation, offloading, and minimizing task drop rate [10]. The system model given in Figure 2.8 shows an illustration of SWIPT-enabled MEC.

2.2.9. Mobile Edge Computing with Resource Slicing for eMBB and URLLC User Coexistence

As highlighted in section 2.1.3, 5G and emerging beyond-5G networks offer two influential types of services, namely eMBB and URLLC services [11]. Latency and reliability are major concerns for URLLC applications, while eMBB services require very high data rates [31]. The trade-off between latency, reliability, and spectral efficiency, sharing of radio resources between eMBB and URLLC services, leads to a challenging scheduling problem [31]. The coexistence of these services in the same spectrum has invited research in different contexts, particularly the MEC. This research area involves computation offloading from eMBB and URLLC users where radio, computation resources, and offloading decisions are all optimized and allocated efficiently to minimize energy consumption and latency while meeting both types of users' QoS requirements.

2.3. System Model Parameters in MEC-related works

Several parameters modeled in the literature of the different MEC architectures have laid the foundation for formulating the problems widely considered in the context of MEC. These parameters include the following:

2.3.1. Computation Model

The computation model consists of local device execution and, in some cases, the mobile edge execution model [30]. The local device computation depends on several factors. The CPU of the local device is the primary engine for local computation [32]. The performance of the CPU is controlled using the CPU-cycle frequency [32]. Computation tasks are initiated by mobile applications, and they arrive in packets, each with an input size in bits, execution deadline, and in some cases, the number of CPU cycles required for processing [8], [33]. Many studies characterize packet arrival processes using various statistical models [8], [12].

The local computation model also comes with local queuing models where packets that arrive at the MEC server are queued for processing by the server's CPU [8], [12]. The local computation model is thus usually modeled using a time delay that represents the amount of time it takes to process a particular task on the local UE device. This delay depends on the local queue delay, the local CPU frequency (f_L), the size of the task (z), and the number of CPU cycles per bit required for computing the task (w_L). A generalized equation for modeling the local computation delay (D_L) taken from [30] is shown below using (2.1). The parameter x denotes the offloading probability, whereas ($1-x$) denotes the local computation probability. The task size (z) is partitioned into offloaded data and local data using these probabilities [30]. The local computation delay is denoted by

$$D_L = \frac{w_L(1-x)z}{f_L}. \tag{2.1}$$

For partial offloading, task processing is done by the local device and the MEC server. For the multi-user case, the MEC server receives offloaded packets from several users requesting computations of their tasks. Packets arrive at the MEC server and are arranged in a queue, ready and waiting for processing by the MEC server's CPU [12]. Some works assume sufficient computation resources at the MEC server, in which case the computation delay is ignored [7]. Other works, such as [8],[30], and [12], however, consider finite MEC computation capacity which results in a delay at the MEC server.

Many works also assume that the output of the MEC server computation is of negligible size, thus ignoring the delay in transmitting the output packets back to the UE devices [7].

The total computation delay model at the MEC server is modeled as an equation that depends on the MEC server queuing delay, the CPU clock frequency of the MEC server (f_M), the size of the offloaded task, and the number of CPU cycles per bit required for computing the task (w_M). A generalized equation for modeling the MEC computation delay (D_M) taken from [30] is given as

$$D_M = \frac{w_M xz}{f_M}. \tag{2.2}$$

2.3.2. Energy Consumption Model

In the MEC architectures described in Section 2.2, the energy consumption is considered at the UE devices. UE devices expend energy during one of three operations: 1) normal device operation; 2) data transmission; and 3) local data computation [30]. The normal device operation tasks are those tasks that are not directly involved in the offload process. The local computation consumes energy, which is directly dependent on the size of a task, the local CPU-cycle frequency, and an energy consumption coefficient (ς) that represents the chip architecture of the user device [30]. The energy consumed in data transmission is used for transmitting the offloaded packets to the MEC server. This energy depends on the transmission power of the device (P_T), the offloaded packet size, and the channel rate (R) of the wireless transmission channel allocated to the UE device for communication with the MEC server. A generalized equation for modeling the energy consumption (E) taken from [30] is given by

$$E = \varsigma(f_L)^2(1-x)z + \left(P_T \frac{xz}{R}\right). \tag{2.3}$$

The equation is a sum of the local energy consumption and offloading energy, respectively.

2.3.3. Transmission Model

The UE devices send their computation tasks to the MEC server in a particular time slot with an uplink radio transmission data rate that depends on the wireless transmission channel gain, the transmit power, and the bandwidth allocated to the user for transmission [8]. Most MEC architectures model this transmission rate using Shannon's capacity equation. Ideally, this is considered in systems that only consider generic types of users whose packet transmissions are long enough to be approximated by Shannon's formula [30]. A generalized equation used for modeling Shannon's formula for data rate R is given by

$$R = f_B \log_2\left(1 + \frac{\alpha P_T}{N_o f_B}\right), \tag{2.4}$$

where f_B is the bandwidth allocated channel for transmission, $\frac{\alpha P_T}{N_o f_B}$ represents the Signal-to-Noise Ratio (SNR), which can be formulated differently depending on the nature of the wireless channel. In architectures that consider different types of users, such as in the coexistence of eMBB and URLLC users, the authors therein use a modified type of Shannon's capacity formula for the eMBB user with long packet transmissions [11], [33]. The equation is modified to account for the various fractional bandwidth allocations. For URLLC users, Shannon's capacity formula cannot accurately capture the short packet transmissions [11]. Instead, the achievable rate in URLLC transmissions falls in the finite block-length channel coding regime, which is derived in [34]. In the transmission model for MEC system models, the time delay (D_T) in transmitting packets from the UE device to the MEC server is a vital model in the latency reduction problems. Thus, it is modeled with (2.5), taken from [30], which shows dependency on the size of the offloaded packet and the radio transmission rate. Equation (2.5) is given as

$$D_T = \frac{xz}{R}. \tag{2.5}$$

2.3.4. Energy-Harvesting Model

For the EH system architectures, the UE devices are equipped with energy harvesting circuitry, such as RF energy harvester, photovoltaic modules, and wind turbines which can convert renewable resources (ambient RF signals, wind and solar) to electricity [32]. The harvested renewable energy can be stored in the battery to support local computing and computation offloading. Most works discretize the harvesting time horizon into time slots [32]. The amount of energy harvested in each time slot is modeled with a variable that depends on several factors such as the distance between the UE device and the energy transmitter (d), the energy conversion efficiency (υ), the power with which the energy is transmitted (η), a path loss factor exponent (τ) and the combined gain of the energy transmitter and the user device antenna (G) [32]. Equation (2.6), taken from [18], shows a generalized harvesting equation for energy harvested in slot t. Equation (2.6) is given as

$$\rho[t] = \upsilon\eta[t](d[t])^{-\tau}G. \tag{2.6}$$

The harvested energy in a time slot is then used in a model that shows the evolution of the battery energy level from the current time slot to the next time slot [32]. The harvested energy adds to the battery energy level (b). This evolution model also accounts for the energy consumed in local computation and the offloading of packets. Energy consumption

(E) takes away from the battery energy level. The generalized evolution equation taken from [18] is given as

$$b[t+1] = \max(0, b[t] - E[t] + \rho[t])\,. \tag{2.7}$$

2.4. Optimization Problems in MEC-related works

Based on the different parameter models described above, the different optimization problems in the context of MEC are formulated, and they are outlined below:

2.4.1. Energy Consumption Minimization

MEC comes from the need to improve the battery lifetime or energy efficiency of resource-constrained UE devices. By offloading tasks to MEC servers, energy can be saved at the local UE devices. In this way, users can run several applications seamlessly and with high-quality service and experience. However, energy is still consumed for offloading packets to an MEC server; in some cases, offloading might consume more energy than local computation. To minimize the energy consumption of the UE device, the offloading decision must be optimized [13]. In a multi-user system, there are several factors aside from the offloading decision that affect energy efficiency. These include user association, radio resource allocation, and computation resource allocation [8]. The works in the literature aim to minimize energy consumption by efficiently optimizing the allocation of channel and compute resources and managing offloading decisions [13]. Equation (2.8) gives a generalized energy minimization problem formulation. Equation (2.8) is given as

$$\min_{Optimization\,Variables} \sum_{k=1}^{K} E_k\,. \tag{2.8}$$

The minimization problem depends on optimization variables such as resource allocation, user association, and offloading probabilities. The total energy is summed up over a system that has a total of K users.

2.4.2. Task Computation Delay Minimization

The computation execution delay also plays a significant role in meeting QoS for mobile edge users. Real-time systems such as virtual reality and other mission-critical systems, such as autonomous vehicles, and remote drone operation, fall under the class of URLLC services that require minimal latency [11]. Computation offloading produces delays such as uplink transmission delay, MEC queuing delay, MEC computation delay, and sometimes the downlink transmission delay. To meet QoS requirements, related works on MEC aim to minimize the total delay by optimizing the same variables as in energy minimization. In

some cases, it might take longer to offload a task compared with processing it locally, therefore, an optimal offloading decision is required.

Channel resources must be optimally allocated in a multi-user system to reduce queueing delays. Computation resources at the MEC server must also be optimally allocated to avoid overloading the MEC server. In a multi-MEC server system, user association is important to minimize delay. A random association can lead to users connecting to base stations with which they have poor channel gains or which might be overloaded, while there is an option to connect to base stations with good channel conditions and with less workload. An optimized user association can reduce delays. Equation (2.9) shows a generalized time delay minimization problem formulation. Equation (2.9) is given as

$$\min_{Optimization\,Variables} \sum_{k=1}^{K} D_k \,. \tag{2.9}$$

The minimization problem depends on optimization variables such as resource allocation, user association, and offloading probabilities. The delay (D) is summed up over a system that has a total of K users.

2.4.3. Execution Cost Minimization

The UE devices might have insufficient energy. In EH system models, the harvested energy is intermittent and sporadic [10]. Some of the initiated computation tasks may not be executed. They can be dropped due to a lack of energy for local computation, while the wireless channel from the mobile device to the MEC server is in deep fading, i.e., the input task cannot be delivered wirelessly to the MEC server [7]. The execution cost quantifies the task-dropping cost; in some systems, it is used as a weighted sum of the execution delay and the task-dropping cost [7]. Thus, it is vital to manage radio and compute resources efficiently to avoid the task-dropping cost [7]. Compute resources like computing energy, power allocation, and CPU-cycle frequencies can be optimally adjusted on UE devices to minimize the task-dropping cost. Channel resources can be optimally allocated to minimize the deep fading scenarios and ensure tasks are always delivered for offload computations. Equation (2.10) shows a generalized execution minimization problem formulation,

$$\min_{Optimization\,Variables} \sum_{k=1}^{K} C_k \,. \tag{2.10}$$

The minimization problem depends on optimization variables such as CPU-cycle frequencies, power allocation, and channel gains. The execution cost (C) is summed up over a system that has a total of K users.

2.4.4. Throughput Maximization

In the architectures that involve the coexistence of eMBB and URLLC types of users, channel resources are allocated in a way that ensures the QoS requirements of both types of users are met. Most literature works use the pre-emptive puncturing type of scheduling for Resource Block (RB) allocation [11],[31]. Generally, the largest proportion of wireless traffic is produced by eMBB users [31]. URLLC traffic is naturally infrequent and needs to be addressed spontaneously and without delay [11]. In the pre-emptive scheduling scheme, URLLC traffic will be scheduled in short TTIs on top of the ongoing eMBB transmissions [11]. The base station stops eMBB transmission during short TTIs of URLLC transmission to meet URLLC latency requirements [11]. This method is efficient in meeting URLLC latency requirements; however, it may impact the eMBB rate and reliability. Optimization of RB allocation is done to meet URLLC latency requirements while maximizing the sum data rate of eMBB users. Equation (2.11) below shows a generalized throughput maximization problem formulation. Equation (2.11) is given by

$$\max_{Optimization\ Variables} \sum_{k=1}^{K} R_k \,. \tag{2.11}$$

The maximization problem depends on optimization variables such as RB allocation, power allocation, and puncturing decisions. The throughput (R) is summed up over a system that has a total of K users. Some literature works perform an integrated optimization of some of these optimization problems to simultaneously solve the different challenges in MEC.

2.5. Solution Architectures in MEC-related Works

This section gives descriptions of some of the solutions used in the literature to solve the optimization problems presented in Section 2.4.

2.5.1. Analytical Approaches

Analytical approaches aim to find an optimal solution out of all possible solutions [11]. The main objective function usually solves for multiple parameters, which makes it complex to solve (NP-hard or MINLP) and may be non-convex [11]. The problem is broken down into several convex sub-problems, which can be solved individually. The optimization

algorithm is then run iteratively using the simplified convex problems, which search through the entire space of possible solutions until the optimal solution is found [11]. In MEC, the optimal solution is the solution that gives the least amount of energy consumption, processing delay, weighted sum of energy consumption and delay, task execution cost, and maximum throughput. The papers in [11], [31], and [33] use analytical approaches to find the solution that maximizes the throughput of eMBB users in multi-user MEC systems.

The objective function relies on parameters like user association, power allocation, and RB allocation. The problem is broken down into sub-problems, each of which can relax some constraints and parameters to make them convex. In [35], an energy-efficient optimization problem is formulated to minimize the energy consumption by solving for the offload data size and the transmission delay per packet under delay constraints. The problem is broken down into two sub-problems, one of which is solved using the Lagrange function. In some cases, analytical approaches are considered slow and sub-optimal when the scale of the problem grows such that deep learning tools may be used to fully take over or used together with the optimization algorithms.

2.5.2. Lyapunov-based Algorithms

Lyapunov-based optimizations use a Lyapunov function to optimally control a dynamic system [7]. They are used extensively to ensure different forms of stability. In a multi-dimensional system, as in the MEC optimization problems, a Lyapunov function is a non-negative scalar measure of the multi-dimensional state of the system [7]. The function grows large if the system starts to move towards undesirable states. System stability is achieved by taking control actions that make the Lyapunov function drift in the negative direction towards zero [7]. Several MEC architectures have used Lyapunov optimization to address the stochastic nature of variables such as channel conditions, energy harvesting, and URLLC scheduling.

The papers in [7] and [36] use Lyapunov optimization in an EH-enabled MEC system. In [36], the authors formulate a stochastic optimization problem involving dynamic offloading and resource scheduling between local devices, BS, and the back-end cloud. The authors propose a Dynamic Offloading and Resource Scheduling algorithm (DORS) based on the Lyapunov optimization theory that minimizes energy consumption and computing resources. In [7], a Lyapunov Optimization-based Dynamic Computation Offloading (LODCO) algorithm is proposed to jointly decide the offloading decision, CPU-cycle frequencies for mobile execution, and transmit powers for computation offloading. The

objective is to minimize the execution cost. In both papers, the algorithm is used to turn the stochastic problems into deterministic problems that can be easily solved.

2.5.3. Game Theory

Game theory is a mathematical tool used to analyze the interactions between multiple players [37]. The combination of strategies that maximizes each player's outcome is called Nash equilibrium. A game reaches a Nash equilibrium when no player can improve their utility by altering their strategy without reducing the utility of the others [37]. Utility quantifies the satisfaction that a specific service provides for the decision-maker [37]. In the context of MEC, game theory is most suited in multi-user and/or multi-MEC server architectures to solve user association problems for spectrum efficiency optimization. The players can be the base stations as in [38], [39], and [40] or the users as in [41] or both as in [42] – [46], and the strategies are the corresponding user association decisions [37]. In non-cooperative game theory modeling [41] – [43], the players seek to maximize their utility and compete against each other by adopting different strategies, such as adjusting their transmit powers [37].

By contrast, cooperative game theory models a bargaining game as in [38], [39], and [40], where the players bargain with each other for the sake of attaining mutual benefits [37]. Game theory is ideal for designing distributed algorithms that offer flexible self-configuration capabilities, all while maintaining a low communication overhead [37]. The authors in [48] adopt a game-theoretic approach for achieving efficient computation offloading in a distributed manner for a multi-user case that is NP-hard to solve with a centralized optimal solution. In [47], the authors consider a multi-cell MEC scenario where the small cells serve multiple users. They propose an adaptive sequential offloading game approach to resolve the issues of intercell interference and finite computation resources. The players are the mobile users who make offloading decisions based on the current interference environment and available computation resources. The goal of the algorithm is to minimize energy consumption and latency.

2.5.4. Lagrange Multiplier

The Lagrange multiplier is mostly used in optimization-based algorithms to find the maximum or minimum of a multivariable function when there is some constraint on the input values [35]. Lagrange multipliers are widely used to solve challenging constrained optimization problems due to their advantage of allowing optimization to be solved without explicit parameterization in terms of the constraints [35]. The solution corresponding to the original constrained optimization is always the saddle point of the Lagrange function,

which can be identified at the stationary points [35]. In MEC, optimization problems often involve solving for multivariable objective functions. Some variables may include offloading data size, power allocation variables, user association variables, resource block allocation variables, etc.

The Lagrange function is, therefore, a powerful tool when finding the optimal stationary point at which the variable being solved (energy consumption, time delay, and throughput) is at its minimum or maximum. In [35], an energy-efficient resource allocation scheme is developed for offloading users. The Lagrange function is used in an optimization-based formulation to find optimal values of offload data size and slot time that minimize the energy in a multi-user system with a single BS. In [49], the optimization problem is transformed into solving a nonlinear equation with a linear constraint, leveraging the Lagrange multiplier to solve for minimizing energy, while in [50], the authors tackle the power allocation optimization problem using the Lagrange multiplier for energy consumption minimization in a multi-server system. In [9] and [51], the Lagrange multiplier is used in game-theoretic optimization.

2.5.5. Heuristic Approaches

Heuristic approaches are methods that aim to find practical solutions for optimization problems, especially in cases where exact solutions are computationally infeasible [61]. These methods are essential in computation offloading, where offloading decisions can significantly impact the system's overall efficiency and responsiveness [61]. Greedy algorithms are examples of heuristic algorithms that make optimal choices at each decision step with the hope of finding a global optimum or near optimum. They are straightforward and computationally efficient but may not always yield the best solution.

In [61], the authors used greedy algorithms for computation offloading for a network with multiple MEC servers. In this approach, tasks are offloaded to the nearest MEC server to minimize latency, without necessarily considering server load or system balancing. Genetic algorithms are another example of heuristic algorithms that are implemented using natural evolution theories, using a population of solutions that evolve over time. Solutions are crossed and mutated to produce new generations of solutions with improved fitness. In [61], a Genetic algorithm was used to optimize the placement of tasks across multiple edge servers by finding combinations that minimize latency and energy consumption.

2.5.6. Machine Learning Solutions

Machine learning is a subset of artificial intelligence that uses special algorithms and statistical models to allow systems to learn complex patterns from data without being

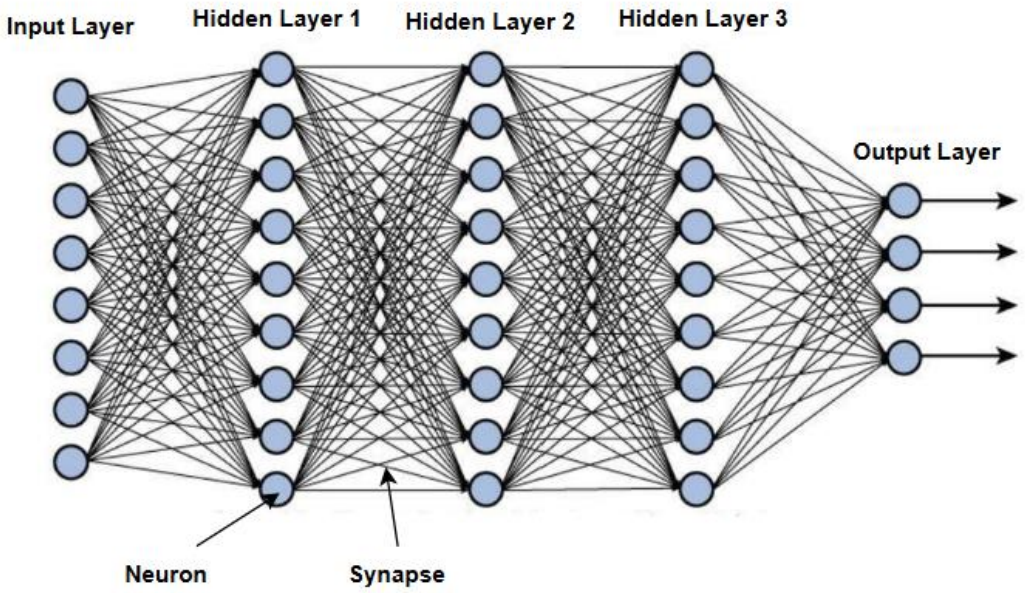

Figure 2. 9: DNN architecture.

explicitly programmed [6]. These models are trained on the data so that they can provide inferences and predictions on other sets of data that they have not been trained on. The field of machine learning is broad, and many ML algorithms can be used for various applications. In this study, the focus is given on Deep Neural Networks (DNNs) and Deep Reinforcement Learning (DRL), which are described below.

2.5.6.1. Deep Neural Networks

A DNN is an artificial neural network that consists of layers of neural units. Each neural unit represents weights and biases that are adjusted during training to model a dataset. The layers are connected using synapses. With these components, a DNN is designed to mimic the human brain. Figure 2.9 shows a DNN and its various components. During training, the DNN weights and biases are adjusted to fit a mathematical representation of a particular dataset [6]. A loss function is typically used to measure the difference between the DNN's outputs and true values. This measured difference is used to continually update the weights in a direction that minimizes it until the difference has become small enough to provide a high confidence level to perform predictions on unseen datasets [6]. DNNs are used in MEC literature for finding patterns in data such as user mobility, channel conditions, etc.

For a supervised mode of learning, some data about the network needs to be collected first, which consists of features and output variables that may need to be predicted. In a practical network environment, this may be difficult to accomplish as real-time data, such as channel gains, are highly dynamic and random, and in some cases, the data may be

difficult to collect. In [6], a digital twin is used to mimic an MEC network environment with several access points and offloading users. Network data about the users, such as channel gains, density of users, and task arrival rates, is monitored using sensors and sent to the digital twin. The digital twin is then used to train a DNN for user association prediction. Training data is acquired through some exploration and exploitation strategies. Each user association sample has a corresponding energy consumption value. An argmin function is used to continuously store samples that give low energy consumption values.

2.5.6.2. Deep Reinforcement Learning

Reinforcement Learning (RL) is a promising solution to tackle non-deterministic problems [11]. In a practical energy-harvesting MEC environment, the energy harvested is stochastic. Channel gains are stochastic, and this has an impact on the performance of optimization algorithms relying on channel gains. URLLC packet arrivals are considered sporadic and non-deterministic [11]. In multi-user, multi-server problems, user association is a non-deterministic problem. The number of users connecting to a BS could change over time. RL uses Markov Decision Processes (MDP) to train an artificial intelligence algorithm to predict future states using only the current state [11]. The MDP framework consists of an agent in a stochastic environment. This agent is exposed to a state space that is acted upon by an action space, and rewards are given for performing certain actions. The agent learns by associating actions with rewards. Actions that give negative rewards towards a particular state are undesirable, and the reverse is true.

In an MEC environment, a particular choice of offloading scheme, resource allocation, or user association can lead to either an increase or decrease in total energy consumption or time delay [33], [52], [53]. The works in [33], [52], and [53] have all considered energy and time delay optimization. State-action pairs that lead to a reduction in energy consumption and time delay are the favorable pairs. For coexistence mechanisms in

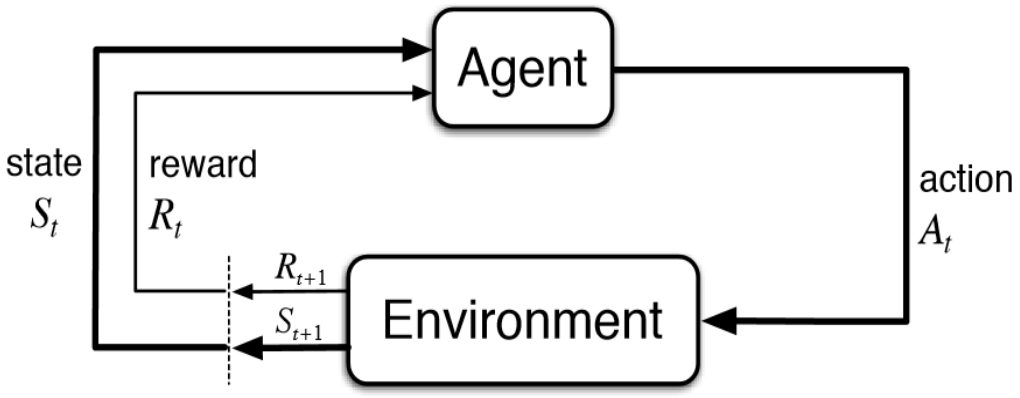

Figure 2. 10: Reinforcement learning architecture.

eMBB/URLLC networks, a particular URLLC scheduling that maximizes eMBB throughput while meeting URLLC latency requirements gets positive rewards; the reverse is true [11], [54]. In some works, such as energy harvesting MEC [32], the offloading scheme relies on a RL algorithm that learns using a state space of the current battery level, radio transmission rate from the previous time slot, and the predicted amount of harvested energy. The actions are: choice of a BS to offload to and the corresponding offloading ratio. DRL algorithms have, however, been regarded as slow at converging to an optimal policy. Recent approaches tackle this problem by using the DRL algorithm together with another optimization algorithm, as done in [11].

2.5.6.3. Federated Learning

Federated learning is a concept that removes a centralized approach to training machine learning models [30]. The conventional centralized approach involves gathering data from different sources into one server, where a machine learning model is stored. The model is then trained using the data from all the sources. In an MEC system, a machine learning model stored at an MEC server might be used to learn certain aspects of the different users associated with it. It is implausible to send all user data over the limited channel resources [30]. Moreover, the MEC server might not have enough storage capacity to store all the user data [19]. User data privacy is another issue when the data is being moved away from the users' mobile devices. The data could be intercepted during transmission or at the centralized server [19]. The federated approach is widely used in literature to address all these problems. The model parameters stored at the central server are transmitted to all the UE devices. The UE devices use their localized data and computation resources to train and update the model parameters from the central server. After several rounds of training, the updated parameters are sent back to the central server, which then aggregates all the parameters from the different users into a global model.

The global model is transmitted back to the users. These training rounds are done to keep all users' local models up to date with the global model. In the context of MEC federated learning, which has been used in [19], [21], [55], and [56], is particularly useful for multi-server systems where multiple base stations that serve users need to communicate with each other to coordinate channel resources, manage intercell interference [19], optimize user association to minimize latency and energy consumption [30], [55]. In [19], a DRL model is used in a small cell network consisting of several base stations, each associated with users in its vicinity. System optimization is achieved through coordination among different small cells for computation offloading, channel allocation, and power control to alleviate inter-cell interference, hence improving the offloading performance

[19]. The coordination is done via global observations and actions of each SBS by sharing the DRL model parameters and converging to a global optimal DRL model stored at each SBS. In [30], the system consists of multiple HABs where users offload their computation tasks. The system optimizes a weighted sum of energy consumption and offload delay by using a federated global model that is shared among the HABs to predict optimal user association.

2.6. Summary

MEC is a broad field in which several architectures have been used to provide seamless experiences and high-quality service to resource-constrained users. Different architectures have been studied, such as single-user energy harvesting MEC, multi-user multi-server MEC, Vehicular Edge Computing, UAV-assisted MEC, MEC with High Altitude Balloons, and SWIPT-enabled MEC. Due to a similar MEC offloading concept, these system architectures share generalized parameter models that include a computation model, energy consumption model, transmission model, and energy harvesting model. Several works have formulated similar objective functions, most of which aim to solve energy consumption and time delay problems. Energy and latency minimization have been associated with enabling the delivery of high-quality experiences and services.

The concept of the coexistence of eMBB and URLLC users has also been studied in MEC. The coexistence problems are mostly solved with schemes that aim to maximize eMBB throughput while meeting URLLC latency requirements. Literature work in the field of MEC has come up with different solutions to tackle the optimization problems. These solutions aim at optimizing computation offloading by optimizing parameters that reduce energy consumption and time delay. Some of these parameters include radio and computation resource allocation, offloading decisions, transmit power allocations, and user association. Some of the notable approaches that solve optimization problems using these parameters include analytical approaches, Lyapunov approaches, Game theory, DRL, Lagrange multipliers, and federated learning.

CHAPTER THREE

3. Computation Offloading for Energy-Harvesting eMBB Applications

This Chapter presents the first part of the system model, which considers a computation offloading network with a single gNB and multiple energy-harvesting eMBB users. A full description of the system model is presented. A problem formulation is introduced that comprises energy consumption minimization, task queueing delay minimization, and throughput maximization. The solution framework that introduces Deep Reinforcement Learning, specifically the Twin Delayed DDPG, is then presented. The Chapter then concludes with results and discussions that present the findings of this part of the dissertation.

3.1. Introduction

The 5th Generation (5G) wireless New Radio (NR) offers support for a range of services that fall under three categories, namely, massive Machine-Type Communications, enhanced Mobile Broadband (eMBB), and Ultra-Reliable Low-Latency (URLLC) [11]. The eMBB service is designed to support high data rate applications while satisfying moderate reliability [11]. A key challenge for this service is the limited processing power and battery energy at the User Equipment (UE). Mobile Edge Computing (MEC) has addressed this problem where UEs can offload computation-intensive tasks to powerful edge servers thus reducing computation delay and conserving battery energy. UEs, however, still incur energy costs in the offloading process, and in some cases, depending on channel conditions, incur more energy costs in offloading than local computation. Therefore, optimizing offloading probabilities and the UE's transmission powers is necessary for an energy-efficient and latency-reduction offloading scheme.

Energy Harvesting (EH) is another promising paradigm that prolongs battery lifetime and enhances the quality of UEs' experiences [32]. The EH module enables offloading devices to capture ambient energy such as solar radiation, wind energy, and Radio Frequency (RF) energy to supplement the battery energy required for processing [32]. For computation offloading to support high data rates over wireless channel resources while preserving UEs' battery energies, there is a need to optimize the allocation of scarce channel resources in multi-user networks.

URLLC services are designed to support applications requiring ultra-low latencies with high transmission reliability [11]. URLLC transmissions are generally random [11] and have very short packet sizes that require low data rates. The 5G New Radio has been designed with a physical layer that enables the Quality of Service (QoS) requirements of both URLLC and eMBB services to be achieved. This physical layer is a standard for 5G wireless transmission and leaves a crucial challenge of scheduling both types of services on the same spectrum for effective computation offloading in uplink scenarios. There is a need to explore an effective solution for scheduling both types of service on the same spectrum, that considers the random nature, latency, and reliability requirements of URLLC transmissions as well as the data rate requirements of the eMBB service.

In an MEC network with multiple MEC servers, solving an optimization problem that encompasses EE, latency reduction, throughput maximization, and effective scheduling of different types of services using the 5G NR for EH eMBB and URLLC users by considering optimization variables such as user association, offloading ratios, transmission power allocation, and channel resource allocation, is a very complex problem. When the scale of the problem grows, such as when the system acquires more users or edge servers, it becomes very challenging to solve using traditional optimization algorithms. On the other hand, Deep Learning (DL) solutions can potentially solve such a complex problem in real time [32]. Deep Neural Networks (DNNs) have been widely used to approximate optimal policies in various computation offloading works, as done in [6],[8],[12],[19], and [53]. Simplistic DNN implementations, however, require labeled data for training, which may not be available in practical systems. To overcome this issue, the Deep Reinforcement Learning (DRL) approach has been proposed, which does not require labeled training data but learns the reward-maximizing behavior of Markov Decision Processes (MDPs) over time using feedback from the real environment. The classic DRL algorithm, which has been widely used in computation offloading problems, Deep Deterministic Policy Gradient (DDPG), has been overtaken by a state-of-the-art Twin Delayed DDPG (TD3) algorithm. The TD3 has outperformed the DDPG by a wide margin, as shown in [57].

In this Chapter, a computation offloading network for EH eMBB users is modeled, and an optimization problem that jointly minimizes energy consumption, task computation delay, and maximizes the channel data rate of the users is formulated. A TD3-based algorithm is then implemented to bring a new DRL solution into the research area of DRL-based computation offloading.

3.2. System Model

A multi-user MEC system is considered and shown in Figure 3.1, where a single gNB, with index m, serves a set of $D^{(m)}$ eMBB users, with each user indexed by $d^{(m)} \in \mathcal{D}^{(m)} = \{1, \dots, d^{(m)}, \dots, D^{(m)}\}$. Each eMBB UE generates computation-intensive tasks that must be executed within a deadline. The UEs are equipped with battery storage and EH circuitry. They can harvest RF energy and use the EH circuitry to convert that into electrical energy used for processing or storage. The energy-powered UEs can either process the tasks locally or perform a full or partial offloading to an MEC server that is attached to the gNB. The gNB provides uplink wireless channel resources to the eMBB users for computation offloading. The MEC server has compute resources to process the offloaded tasks. The downlink channel is used to transmit the results of computation tasks.

3.2.1. Transmission Model

The wireless communication channel used for computation offloading is implemented using the 5G NR standard, which leverages multiple waveform configurations [58]. Wireless channel resources are allocated through variable-length Transmission Time Intervals (TTIs) [57]. The total uplink channel bandwidth is divided into Resource Blocks (RBs). Each RB $b^{(m)} \in \mathcal{B}^{(m)} = \{1,2, \dots, B^{(m)}\}$ is made up of 12 sub-carriers. Figure 3.2 shows the configuration of RBs. In every TTI, the uplink scheduler of the gNB allocates RBs to users with pending offload requests. In particular, the gNB allocates to the users,

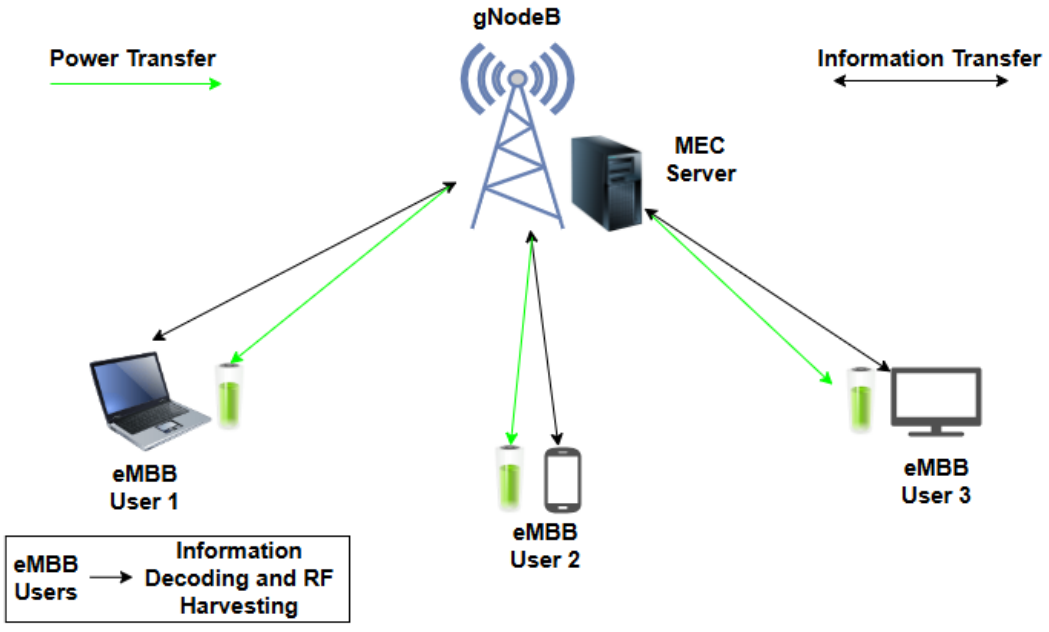

Figure 3. 1: Single gNB with energy-harvesting eMBB users.

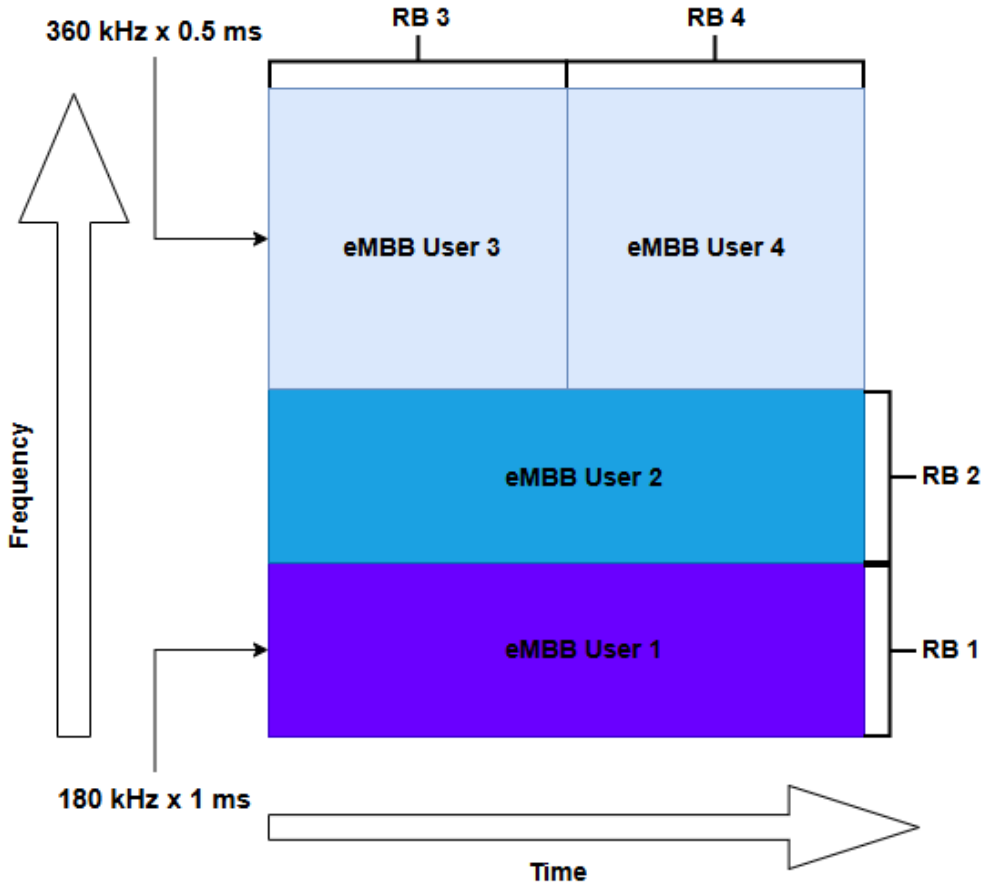

Figure 3. 2: Communication channel frame structure.

transmit powers, RBs, and offloading ratios while considering their QoS demands. The data rate achieved by the eMBB users in offloading computation tasks varies depending on the transmission power, noise interference, and channel bandwidth. The channel rate of eMBB user $d^{(m)}$ who is connected to the gNB m, over the allocated RB $b^{(m)}$ in the TTI t is formulated using Shannon's capacity [57] in (3.1),

$$r_{d,b}^{(m)}[t] = f_b^{(m)} \log_2 \left(\frac{\propto_{d,b}^{(m)}[t] g_{d,b}^{(m)}[t] P_d^{(m)}[t]}{N_o f_b^{(m)}} \right), \tag{3.1}$$

where $f_b^{(m)}$ is the bandwidth of the RB b, $\propto_{d,b}^{(m)}$ is the small-scale channel gain of user $d^{(m)}$ on RB b, $g_{d,b}^{(m)}$ is the large-scale channel gain of user $d^{(m)}$ on RB b, $P_d^{(m)}$ is the uplink transmission power of the eMBB user $d^{(m)}$, and N_o is the single-sided additive white gaussian noise power spectral density. If the user is allocated multiple RBs for transmission, the data rate of the user over all the allocated RBs in the TTI t is given as

$$R_d^{(m)}[t] = \sum_{b=1}^{B^{(m)}} \sigma_{d,b}^{(m)}[t] r_{d,b}^{(m)}[t], \tag{3.2}$$

where $\sigma_{d,b}^{(m)}[t]$ is the RB scheduling indicator in the time slot t, and is defined as

$$\sigma_{d,b}^{(m)}[t] = \begin{cases} 1, & if\ RB\ b\ is\ allocated\ to\ user\ d\ at\ time\ t \\ 0, & otherwise \end{cases}. \tag{3.3}$$

3.2.2. Task Model

This work considers local computing, the task processing done at the UEs. The edge server is assumed to be powerful enough to process offloaded tasks and send back results within one TTI. The task, $\mathcal{T}_d^{(m)}$, of eMBB user $d^{(m)}$, generated in slot t has a profile characterized by ($z_d^{(m)}$, $c_d^{(m)}$,T_d^{max}) where $z_d^{(m)}$ (bits/task) is the task size in bits, $c_d^{(m)}$ (cycles/task) is the number of CPU cycles required to process the task, and T_d^{max} is a QoS constraint that puts a deadline on the task for queueing and computation. The variable $c_d^{(m)}$ is computed as

$$c_d^{(m)} = k_1' z_d^{(m)}, \tag{3.4}$$

where k_1' (cycles/bit) is the number of CPU cycles required to process one bit. The parameter $z_d^{(m)}$ follows a geometric distribution. The parameter $\lambda_d^{(m)}[t]$ (tasks/slot) is used to denote the task arrival rate in the slot t, which follows a Poisson distribution. One TTI is reserved for edge processing and downlink transmissions.

3.2.3. Offloading Policy

A UE transmits tasks over a wireless channel. UEs can perform partial offloading where a task is partitioned into bits that are offloaded and bits that are computed locally. A UE d generates tasks in timeslot t according to the task arrival rate $\lambda_d^{(m)}[t]$. All the tasks generated in t are individually split according to the ratio $x_d^{(m)}[t]$ where $0 \leq x_d^{(m)}[t] \leq 1$.The parameter $z_{d,off}^{(m)} = x_d^{(m)}[t] z_d^{(m)}$ is the ratio of task $\mathcal{T}_d^{(m)}$ that goes into the offloading queue, whereas $z_{d,lc}^{(m)} = (1 - x_d^{(m)}[t]) z_d^{(m)}$ is the ratio of task $\mathcal{T}_d^{(m)}$ that goes into the local queue. The size $z_d^{(m)}$of task $\mathcal{T}_d^{(m)}$remains constant throughout the timeslot t.

3.2.4. Queuing Model

The queue model of the local UE and the MEC server is illustrated in Figure 3.3. The peak task arrival rate can be higher than the offloading rate, resulting in an offloading queue.

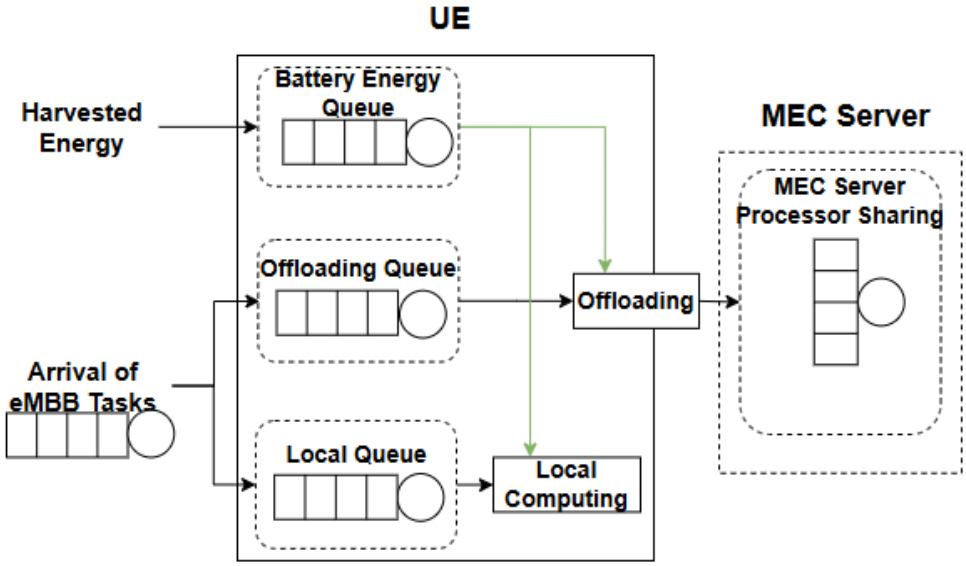

Figure 3. 3: Queuing model showing the offloading, local, battery energy, and MEC server queues.

Likewise, the peak task arrival rate can be higher than the local UE service rate, resulting in a local queue. The offloading rate is determined by the allocated channel resources, while the local UE service rate is determined by the clock frequency of the UE's processor. In both queues, packets are served in a first-come-first-serve order. The MEC server is a powerful processing unit that adopts a processor-sharing scheme to avoid queueing delays. Depending on the available processing units, the total service rate is equally allocated to all packets on the server.

3.2.4.1. Local Queueing Model

Let $T_{d,lc}^{(m),\mathcal{T}}$ denote the end-to-end delay of an arriving task $\mathcal{T}_d^{(m)}$ on the local queue of the UE $d^{(m)}$. $T_{d,lc}^{(m),\mathcal{T}}$ is computed as

$$T_{d,lc}^{(m),\mathcal{T}} = T_{d,lq}^{(m),\mathcal{T}} + T_{d,lcomp}^{(m),\mathcal{T}}, \tag{3.5}$$

where $T_{d,lq}^{(m),\mathcal{T}}$ is the amount of time the task $\mathcal{T}_d^{(m)}$ spends in the local queue before getting processed and $T_{d,lcomp}^{(m),\mathcal{T}}$ is the amount of processing time of task $\mathcal{T}_d^{(m)}$. The computation of $T_{d,lcomp}^{(m),\mathcal{T}}$, adapted from [12], is given by

$$T_{d,lcomp}^{(m),\mathcal{T}} = \frac{(1 - x_d^{(m)}[t])c_d^{(m),\mathcal{T}}[t]}{f_{d,lc}^{(m)}}, \tag{3.6}$$

where $f_{d,lc}^{(m)}$ is the local UE CPU clock frequency measured in cycles per slot and $c_d^{(m),\mathcal{T}}[t]$ denotes the number of CPU cycles required to process the task $\mathcal{T}_d^{(m)}$. The parameter $T_{d,lcomp}^{(m),\mathcal{T}}$ has the units of slots, where each slot has a fixed duration of 1 ms. The local queueing time of task $\mathcal{T}_d^{(m)}$ is computed as

$$T_{d,lq}^{(m),\mathcal{T}} = \sum_{q=1}^{Q_{d,lc}^{(m)}[t]} z_{q,lc}^{(m)} \tau_p \,, \tag{3.7}$$

where $z_{q,lc}^{(m)}$ is the size in bits of the local queue task q, $Q_{d,lc}^{(m)}[t]$ is the number of tasks in the local queue, and τ_p is the local computation time per bit, and is calculated as

$$\tau_p = \frac{k_1'}{f_{d,lc}^{(m)}}, \tag{3.8}$$

where τ_p has the unit of slots. Since both $Q_{d,lc}^{(m)}[t]$ and $z_{q,lc}^{(m)}$ are random variables, $T_{d,lq}^{(m),\mathcal{T}}$ is expressed as a value that is dependent on both variables. Equation (3.7) then changes to

$$\frac{T_{d,lq}^{(m),\mathcal{T}} | Q_{d,lc}^{(m)}[t]}{\tau_p} = \sum_{q=1}^{Q_{d,lc}^{(m)}[t]} z_{q,lc}^{(m)} \,, \tag{3.9}$$

where (3.9) measures the length of the local queue in bits. Let $\frac{T_{d,lq}^{(m),\mathcal{T}} | Q_{d,lc}^{(m)}[t]}{\tau_p}$ be denoted by $L_{d,lc}^{(m)}[t] | Q_{d,lc}^{(m)}[t]$ which represents the local queue length in bits. The variable $L_{d,lc}^{(m)}[t] | Q_{d,lc}^{(m)}[t]$ is a sum of random geometric variables. Equation (3.9) can, therefore, be expressed as

$$L_{d,lc}^{(m)}[t] | Q_{d,lc}^{(m)}[t] = \sum_{q=1}^{Q_{d,lc}^{(m)}[t]} z_{q,lc}^{(m)} \,. \tag{3.10}$$

A local queueing delay violation probability is introduced which measures the probability that an arriving task, $\mathcal{T}_d^{(m)}$, will have a queueing delay, $T_{d,lq}^{(m),\mathcal{T}}$, that exceeds the maximum allowable queueing delay, $T_{d,lq}^{max}$. The local queuing delay violation probability is expressed as follows: when a task arrives in the task queue the number of bits already in the queue awaiting processing is given by $L_{d,lc}^{(m)}[t]$. Let $T_{d,lq}^{max}|_{L_{d,lc}^{(m)}=L_{d,lc}^{max}}$ denote the maximum local queueing delay above which the delay violation probability is violated. The parameter $L_{d,lc}^{max}$ is the maximum queue length that gives $T_{d,lq}^{max}$. The probability of the queueing delay

violation is reduced to the probability of the length of the queue as done in [12] and expressed as

$$\begin{aligned}\epsilon^{max} &= \Pr\left(T_{d,lq}^{(m),\mathcal{T}} > T_{d,lq}^{max}|_{L_{d,lc}=L_{d,lc}^{max}}\right), \\ &\approx \Pr\left(L_{d,lc}^{(m)}[t] > L_{d,lc}^{max}\right),\end{aligned} \tag{3.11}$$

where $L_{d,lc}^{max}$ is computed as

$$L_{d,lc}^{max} = \frac{T_{d,lq}^{max}}{\tau_p}. \tag{3.12}$$

A calculation is done for $\Pr\left(L_{d,lc}^{(m)}[t] > L_{d,lc}^{max}\right)$ and this is determined as

$$\Pr\left(L_{d,lc}^{(m)}[t] > L_{d,lc}^{max}\right) = \sum_{L_{d,lc}^{(m)}=L_{d,lc}^{max}+1}^{\infty} \Pr\left(L_{d,lc}^{(m)}[t]\right). \tag{3.13}$$

The probability that $L_{d,lc}^{(m)}[t]$ is greater than $L_{d,lc}^{max}$ is computed by summing all the probabilities of the values of $L_{d,lc}^{(m)}$ which are greater than $L_{d,lc}^{max}$ as shown in (3.13). The complement rule of probability is then applied; therefore, (3.13) becomes

$$\Pr\left(L_{d,lc}^{(m)}[t] > L_{d,lc}^{max}\right) = 1 - \sum_{L_{d,lc}^{(m)}=0}^{L_{d,lc}^{max}} \Pr\left(L_{d,lc}^{(m)}[t]\right). \tag{3.14}$$

As explained in (3.9) and (3.10), the value of $L_{d,lc}^{(m)}[t]$ is to be calculated given a particular value of $Q_{d,lc}^{(m)}[t]$. When $L_{d,lc}^{(m)} = 0$, $\Pr\left(L_{d,lc}^{(m)}[t]\right)$ is the probability that there are zero bits in the local queue. $\Pr(L_{d,lc}^{(m)}[t])$ is, therefore, calculated as

$$\Pr\left(L_{d,lc}^{(m)}[t]\right) = \sum_{Q_{d,lc}^{(m)}=0}^{\infty} \Pr\left(L_{d,lc}^{(m)}[t]\middle|Q_{d,lc}^{(m)}[t]\right) \Pr\left(Q_{d,lc}^{(m)}[t]\right). \tag{3.15}$$

When $Q_{d,lc}^{(m)}[t] = 0$, $\Pr\left(Q_{d,lc}^{(m)}[t]\right)$ is the probability that there are zero tasks in the local queue. To find $\Pr\left(L_{d,lc}^{(m)}[t]\middle|Q_{d,lc}^{(m)}[t]\right)$ the negative binomial distribution is used, which models either a number of trials or a number of failures in a sequence of independent and identically distributed Bernoulli trials before a fixed number of successes occur [59]. As explained in [59], $\Pr\left(L_{d,lc}^{(m)}[t]\middle|Q_{d,lc}^{(m)}[t]\right)$ is, therefore, defined by

$$\Pr\left(L_{d,lc}^{(m)}[t]\middle|Q_{d,lc}^{(m)}[t]\right)=\binom{L_{d,lc}^{(m)}-1}{Q_{d,lc}^{(m)}-1}p^{Q_{d,lc}^{(m)}}v^{L_{d,lc}^{(m)}-Q_{d,lc}^{(m)}}, \tag{3.16}$$

where $v=1-p$. To compute the $\Pr\left(Q_{d,lc}^{(m)}[t]\right)$ in (3.15), the M/G/1 queue system is considered, which has been used for the computation offloading model in [12]. The authors in [12] defined the probability that there are y tasks in the queue using (3.17). The work in this dissertation approximates the queueing system for this part of the work of finding $\Pr\left(Q_{d,lc}^{(m)}[t]\right)$, on account of the queueing system not having been studied before. Equation (3.17) is, therefore, adopted for the queue model to find $\Pr\left(Q_{d,lc}^{(m)}[t]\right)$ as

$$\Pr\left(Q_{d,lc}^{(m)}[t]=y\right)=\left(\rho_{d,lc}^{(m)}[t]\right)^{y}\left(1-\rho_{d,lc}^{(m)}[t]\right), \tag{3.17}$$

where $\rho_{d,lc}^{(m)}[t]$ is a local queueing traffic intensity given by

$$\rho_{d,lc}^{(m)}[t]=\frac{\bar{z}_{d}^{(m)}[t]\left(1-x_{d}^{(m)}[t]\right)\lambda_{d}^{(m)}[t]}{\mu_{d}^{(m)}[t]}. \tag{3.18}$$

The parameter $\bar{z}_{d}^{(m)}[t]$ is the average task size in bits of the tasks that arrive in the slot t. $\mu_{d}^{(m)}$ represents a service rate in terms of the number of bits serviced per slot and is calculated as

$$\mu_{d}^{(m)}[t]=\frac{f_{d,lc}^{(m)}}{k_{1}'}. \tag{3.19}$$

To ensure the stability of the local queue at any point in time, the constraint that $\rho_{d,lc}^{(m)}[t]\leq 1$ is applied. Since the expressions for $\Pr\left(L_{d,lc}^{(m)}[t]\middle|Q_{d,lc}^{(m)}[t]\right)$ and $\Pr(Q_{d,lc}^{(m)}[t])$ are now given, $\Pr\left(L_{d,lc}^{(m)}[t]>L_{d,lc}^{max}\right)$ can be computed. For the local queueing violation probability to be satisfied, the inequality in (3.20) must hold as follows,

$$1-\sum_{L_{d,lc}^{(m)}=0}^{L_{d,lc}^{max}}\Pr\left(L_{d,lc}^{(m)}[t]\right)<\epsilon^{max}. \tag{3.20}$$

3.2.4.2. Offload Queueing Model

Let $T_{d,off}^{(m),\mathcal{T}}$ denote the end-to-end delay of the task $\mathcal{T}_{d}^{(m)}$ on the offloading queue of UE $d^{(m)}$. $T_{d,off}^{(m),\mathcal{T}}$ is computed as

$$T_{d,off}^{(m),\mathcal{T}} = T_{d,oq}^{(m),\mathcal{T}} + T_{d,trans}^{(m),\mathcal{T}}, \tag{3.21}$$

where $T_{d,oq}^{(m),\mathcal{T}}$ is the time task $\mathcal{T}_d^{(m)}$ spends in the offloading queue before being transmitted to the edge server and $T_{d,trans}^{(m),\mathcal{T}}$ is the transmission time of task $\mathcal{T}_d^{(m)}$ given as a single TTI. The parameter $T_{d,oq}^{(m),\mathcal{T}}$ is computed as

$$T_{d,oq}^{(m),\mathcal{T}} = \left(Q_{d,off}^{(m)}[t] + 1\right) \frac{\mathbb{E}\left(z_{d,off}^{(m)}\right)}{T_s \mathbb{E}\left(R_d^{(m)}[t]\right)}. \tag{3.22}$$

Since $z_{d,off}^{(m)}$ of a particular task is a random variable, an average value is used that is taken over the offloading queue at time t. The variable $Q_{d,off}^{(m)}[t]$ is the number of tasks in the offload queue. T_s is the slot duration. The value of $\mathbb{E}\left(R_d^{(m)}[t]\right)$ is calculated using an average as

$$\mathbb{E}\left(R_d^{(m)}[t]\right) = \mathbb{E}\left[\sum_{b=0}^{B} f_{d,b}^{(m)} \log_2\left(1 + \frac{\propto_{d,b}^{(m)}[t] g_{d,b}^{(m)}[t] P_d^{(m)}[t]}{N_o f_{d,b}^{(m)}}\right)\right]. \tag{3.23}$$

To ensure the stability of the offload queue at any point in time, the constraint $\rho_{d,off}^{(m)}[t] \leq 1$ is applied where $\rho_{d,off}^{(m)}$ is the offloading queue traffic intensity given as

$$\rho_{d,off}^{(m)}[t] = \frac{x_d^{(m)}[t] \lambda_d^{(m)}[t] \bar{z}_d^{(m)}[t]}{T_s \mathbb{E}\left(R_d^{(m)}[t]\right)}. \tag{3.24}$$

The offload queuing delay violation probability is calculated using the same derivation as the local queuing delay violation probability. The $\Pr\left(L_{d,off}^{(m)}[t] > L_{d,off}^{max}\right)$ is, therefore, calculated as

$$\Pr\left(L_{d,off}^{(m)}[t] > L_{d,off}^{max}\right) = 1 - \sum_{L_{d,off}^{(m)}=0}^{L_{d,off}^{max}} \Pr\left(L_{d,off}^{(m)}[t]\right), \tag{3.25}$$

where $\Pr\left(L_{d,off}^{(m)}[t]\right)$ is defined using the following equations:

$$\Pr\left(L_{d,off}^{(m)}[t]\right) = \sum_{Q_{d,off}^{(m)}=0}^{\infty} \Pr\left(L_{d,off}^{(m)}[t] \middle| Q_{d,off}^{(m)}[t]\right) \Pr\left(Q_{d,off}^{(m)}[t]\right), \tag{3.26}$$

$$\Pr\left(L_{d,off}^{(m)}[t]\middle|Q_{d,off}^{(m)}[t]\right)=\binom{L_{d,off}^{(m)}-1}{Q_{d,off}^{(m)}-1}p^{Q_{d,off}^{(m)}}v^{L_{d,off}^{(m)}-Q_{d,off}^{(m)}}, \tag{3.27}$$

$$\Pr\left(Q_{d,off}^{(m)}[t]=y\right)=\left(\rho_{d,off}^{(m)}[t]\right)^{y}\left(1-\rho_{d,off}^{(m)}[t]\right). \tag{3.28}$$

The parameter $\rho_{d,off}^{(m)}[t]$ is defined in (3.24). For the offload queueing violation probability to be satisfied, the following inequality must hold,

$$1-\sum_{Q_{d,off}^{(m)}=0}^{L_{d,off}^{max}}\Pr\left(L_{d,off}^{(m)}[t]\right)<\epsilon^{max}. \tag{3.29}$$

The total delay for task $\mathcal{T}_d^{(m)}$ is given as the maximum between the local delay and the offloading delay as

$$T_d^{(m)}=\max\left(T_{d,lc}^{(m),\mathcal{T}},T_{d,off}^{(m),\mathcal{T}}\right). \tag{3.30}$$

3.2.5. Energy Consumption Model

The device energy is consumed during local computations and transmitting tasks from the offloading queue. The local energy consumption equation in [8] is used to derive the one utilized in this work. The clock cycle frequency, $f_{d,lc}^{(m)}$(cycles/slot), measures the number of computation cycles that the local CPU uses in a slot for task computation. The constant k_1' (cycles/bit) represents the number of computation cycles required for processing one bit. Equation (3.31) is, therefore, used to compute the maximum number of bits (ψ^{max}) that can be processed by the local CPU in a slot. The variable ψ^{max} is given by

$$\psi^{max}=\frac{f_{d,lc}^{(m)}}{k_1'}. \tag{3.31}$$

The local energy consumption equation in [8] is for a task arrival process that follows a Bernoulli distribution, where all the CPU cycles are used on a single task, and thus, the energy consumption equation derives the energy per task. Equation (3.32), taken from [8], shows the local energy consumption and is defined as

$$E_{d,lc}^{(m)}=\varsigma\left(f_{d,lc}^{(m)}\right)^{2}c_d^{(m)}. \tag{3.32}$$

where ς is the effective switched capacitance depending on the chip architecture [8] and $\varsigma\left(f_{d,lc}^{(m)}\right)^2$ is a calculation for the energy consumption per CPU cycle [8]. The units of (3.32) are J/task, which represents the amount of energy consumed for computing a task that requires $c_d^{(m)}$ cycles of computation. The energy consumed in this work for local computations is calculated as follows: if the sum of task sizes in bits on the local queue exceeds ψ^{max}, the CPU uses all $f_{d,lc}^{(m)}$ cycles/slot to process only ψ^{max} bits. The remaining tasks and their bits are carried over into the next timeslot. Equation (3.32) thus changes to (3.33). The energy used to process ψ^{max} bits in slot t is given by

$$E_{d,lc}^{(m)}[t] = \varsigma\left(f_{d,lc}^{(m)}\right)^3. \tag{3.33}$$

The typical value of ς is 10^{-15} [8]. If the sum of task sizes in the queue at slot t is less than ψ^{max} bits, the proportion rule is used, given by

$$E_{d,lc}^{(m)}[t] = \frac{\sum_q^{Q_{d,lc}^{(m)}[t]} z_{d,q}^{(m)}}{\psi^{max}} \varsigma\left(f_{d,lc}^{(m)}\right)^3. \tag{3.34}$$

The energy consumed in transmitting offloaded tasks by the user $d^{(m)}$ is given by,

$$E_{d,off}^{(m)}[t] = \frac{T_s}{l} P_d^{(m)}[t] \sum_{b=1}^{B} \sigma_{d,b}^{(m)}[t], \tag{3.35}$$

where l is the number of RB elements in a timeslot, illustrated in Figure 3.2, and T_s is the slot time. The total energy consumed by the user b in timeslot t is given by

$$E_d^{(m)}[t] = E_{d,lc}^{(m)}[t] + E_{d,off}^{(m)}[t]. \tag{3.36}$$

3.2.6. Energy Harvesting Model

As described in [32], the EH circuitry of the UE converts RF wave energy into usable electrical energy. The UEs are also equipped with batteries to store the harvested energy. The harvested energy is used to support both local computing and computation offloading. The amount of energy harvested in a time slot by the user $d^{(m)}$ is denoted by $\omega_d^{(m)}[t]$. The battery level at the beginning of time slot t is denoted by $H_d^{(m)}[t]$ and, as adapted from [32], evolves as

$$H_d^{(m)}[t] = \max\left\{0, H_d^{(m)}[t-1] - E_d^{(m)}[t] + \omega_d^{(m)}[t]\right\}. \tag{3.37}$$

The amount of energy harvested in timeslot t is modeled in [32] as

$$\omega_d^{(m)}[t] = \upsilon\eta \left(G_d^{(m)}\right)^{\vartheta} \Omega_d \bar{g}_d^{(m)}[t], \tag{3.38}$$

where υ is the energy conversion efficiency, which depends on the architecture of the EH circuitry, η is the transmission power of the gNB, $G_d^{(m)}$ is the distance between the gNB m and the user device, ϑ is the path loss factor exponent and $\Omega_d^{(m)}$ is the combined gain of the RF transmitter antenna and the user device receiver antenna. The variable $\bar{g}_d^{(m)}[t]$ is the average of the large-scale channel gains of the RBs allocated to the user $d^{(m)}$ in slot t.

3.3. Problem Formulation

The goal of the optimization problem is to jointly minimize the sum energy consumption and sum task delay while maximizing the sum throughput of eMBB UEs. This optimization is achieved by tuning 3 key variables, namely, the offloading ratio ($x_d^{(m)}$), RB allocation ($\sigma_{d,b}^{(m)}$) and transmission power ($P_d^{(m)}$) which all have a bearing on the amount of energy consumed, computation delay, and achieved throughput. The optimization problem is, therefore, formulated using **P0** as

$$\mathbf{P0}: \max_{\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t]} \frac{\sum_{d\in\mathcal{D}} R_d^{(m)}[t]}{\sum_{d\in\mathcal{D}} \left(T_d^{(m)} \times E_d^{(m)}[t]\right)}$$

$$\textbf{s.t} \quad 0 \leq x_d^{(m)}[t] \leq 1, \quad \forall d \in \mathcal{D}^{(m)} \tag{3.39}$$

$$P_d^{(m)}[t] \leq P_d^{max}, \quad \forall d \in \mathcal{D}^{(m)} \tag{3.40}$$

$$\sum_{d=1}^{D} \sigma_{d,b}^{(m)}[t] \leq 1, \qquad \forall b \in \mathcal{B}^{(m)} \tag{3.41}$$

$$H_d^{(m)}[t-1] + \omega_d^{(m)}[t] > E_d^{(m)}[t], \quad \forall d \in \mathcal{D}^{(m)} \tag{3.42}$$

$$\rho_{d,lc}^{(m)}[t] \leq 1, \quad \forall d \in \mathcal{D}^{(m)} \tag{3.43}$$

$$\rho_{d,off}^{(m)}[t] \leq 1, \quad \forall d \in \mathcal{D}^{(m)} \tag{3.44}$$

$$\Pr\left(L_{d,lc}^{(m)}[t] > L_{d,lc}^{max}\right) < \epsilon^{max}, \quad \forall d \in \mathcal{D}^{(m)} \tag{3.45}$$

$$\Pr\left(L_{d,off}^{(m)}[t] > L_{d,off}^{max}\right) < \epsilon^{max}, \quad \forall d \in \mathcal{D}^{(m)} \tag{3.46}$$

$$R_d^{(m)}[t] \geq R^{min}, \quad \forall d \in \mathcal{D}^{(m)} \tag{3.47}$$

$\mathbf{P_m}[t]$, $\boldsymbol{\sigma}_\mathbf{m}[t]$, and $\mathbf{x_m}[t]$ represent the vectors for the transmit powers, RB allocations, and offloading ratios of all eMBB users, respectively. Constraint (3.39) ensures that the offloading ratio is a value between 0 and 1. Constraint (3.40) ensures that each user's transmit power does not exceed a threshold. Constraint (3.41) ensures that each RB is allocated to a single user. Constraint (3.42) ensures that computation tasks are not dropped due to inadequate energy. Constraints (3.43) and (3.44) ensure stability on the local and offloading queues, respectively. Constraints (3.45) and (3.46) ensure that the local and offload queueing delay violation probabilities, respectively, are satisfied. Constraint (3.47) ensures that each UE achieves at least a minimum data rate to ensure that packets in the offloading queue can be transmitted.

At this point, an elaboration is done on one of the contributions that is listed in Chapter 1. To the best of the author's knowledge, this is the first work that formulates a problem aimed at jointly maximizing the sum of users' data rate, minimizing the sum of users' energy consumption, and minimizing the sum of users' task queueing delays, thus making this a new objective function in the computation offloading body of literature. Furthermore, the combination of the constraints introduced, specifically, constraints (3.43) to (3.47), has not been studied as constraints before. This work, therefore, offers a significant stride by contributing the study of a new multi-objective function and constraints that are relevant for efficient queueing and offloading models.

Problem **P0** is a nonlinear and non-convex problem because the objective function is in fractional form. To transform the problem into a tractable form, problem **P0** is translated into subtractive form by adapting a nonlinear fractional programming approach as done in [71]. The maximum energy efficiency $q_*^{(m)}$ of the considered system can be defined by

$$\begin{aligned} q_*^{(m)} &= \frac{\sum_{d\in\mathcal{D}} R_d^{(m)}[t]}{\sum_{d\in\mathcal{D}}\left(T_d^{(m)} E_d^{(m)}[t]\right)}, \\ &= \max_{\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t]} \frac{\sum_{d\in\mathcal{D}} R_d^{(m)}[t]}{\sum_{d\in\mathcal{D}}\left(T_d^{(m)} E_d^{(m)}[t]\right)}. \end{aligned} \tag{3.48}$$

Let $\sum_{d\in\mathcal{D}}\left(T_d^{(m)}E_d^{(m)}[t]\right)$ be represented by the function $\mathcal{E}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])$ and $\sum_{d\in\mathcal{D}^{(m)}}R_d[t]$ be represented by the function $\mathcal{R}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])$. The fractional objective function above is equivalent to a subtractive form expressed as $f(x)=\mathcal{R}_d^{(m)}(x)-q\mathcal{E}_d^{(m)}(x)$. Therefore, the maximum energy efficiency q^* is maximized if and only if

$$\max_{\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t],q_E,q_T}\mathcal{R}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])-q_{E,*}^{(m)}E_d^{(m)}[t]-q_{T,*}^{(m)}T_d^{(m)}$$

$$=\max_{\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t],q}\mathcal{R}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])-q^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])$$

$$=\mathcal{R}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])-q_*^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])$$

$$=0. \tag{3.49}$$

Therefore, the original problem **P0** can be rewritten as **P1**, given by

$$\mathbf{P1}:\max_{\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t],q}\mathcal{R}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])-q^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])$$

Subject to (3.39) – (3.47).

3.4. DRL Modelling

3.4.1. Reinforcement Learning

Reinforcement Learning (RL) forms the building blocks of the Twin Delayed DDPG used in this work, thus, it is reviewed. RL is composed of an environment within which an agent can act and learn. Each action performed by the agent has an associated reward value or penalty. The agent must learn actions, through trial and error, exploration, and exploitation, that maximize future rewards. This removes the need for labeled data to train the agent. The environment has a Markov property, and the RL framework can be integrated to model a Markov Decision Process (MDP). RL, therefore, offers a solution for handling complex decision-making processes. The following components are used in the RL training procedure:

- *Episode:* An episode consists of a sequence of timesteps in which the agent performs an action in a given state, giving rise to a reward associated with that action and a transition into a future state. An episode is therefore made up of a

sequence of states, actions, and rewards $(s[t], a[t], r[t], s[t+1], \dots, s[T], a[T], r[T], s[T+1])$ where t is a timestep within the episode and T is the terminal timestep that ends the episode.

- *Policy* (π)*:* The policy is a function that maps states to a probability distribution over the actions. The policy represents the agent's behavior and guides it in choosing actions.
- *Immediate Reward* $(r[t])$*:* the immediate reward is used to provide feedback to the agent's action at timestep t. Good actions will be given a high reward whereas bad actions are either given a low reward or penalty.
- *Cumulative Reward* (J)*:* The cumulative reward is given by $J = \sum_{\tau=1}^{T} \varepsilon^{\tau} r[t-\tau]$ where τ is a discount rate that represents the impact of future rewards on the current cumulative reward.
- *Action value function (Q-value):* A Q-value is used in Q-learning-based Reinforcement Learning, where an action value $Q(s[t], a[t])$ represents the expectation of the cumulative reward after taking the action $a[t]$ while in state $s[t]$ using the policy π.

3.4.2. Q-learning

Q-learning is one of the widely used RL algorithms, which makes use of a state-action value $Q(s[t], a[t])$, stored in a Q-table, to choose an appropriate action, $a[t]$, according to the current state, $s[t]$ and policy π. The target state-action value is calculated as done in [67] by

$$Q(s[t], a[t]) = r[t] + \varepsilon \max_{a[t+1]} Q(s[t+1], a[t+1]). \tag{3.50}$$

At the start of each episode, the environment state is initialized. For each subsequent timestep, the action $a[t]$ is chosen according to the current state $s[t]$ and policy π. The reward $r[t]$ is then calculated using a reward function that depends on the environment, and the environment then transitions into state $s[t+1]$. Q-learning is then used to update the state-action value $Q(s[t], a[t])$ as done in [67] given by

$$Q(s[t], a[t]) = Q(s[t-1], a[t-1]) + \phi\left(r[t] + \varepsilon \max_{a[t]} Q(s[t+1], a[t+1]) - Q(s[t-1], a[t-1])\right), \tag{3.51}$$

where $\phi \in (0,1)$ is a learning rate. This procedure is repeated until the terminal step is reached.

3.4.3. Deep-Q-Network

In the traditional Q-learning algorithm, a Q-table is used to store the Q-values of all state-action pairs [67]. In applications that have a large state and action space, such as the one considered in this work, it becomes impractical and computationally heavy to accurately compute, store, and retrieve all the Q-values from one large Q-table [67]. Advances have been made in line with this work to combine deep learning with reinforcement learning to come up with a less computationally heavy framework that can approximate the Q-values of different state-action pairs using prediction [67]. This leads to a field of Deep Reinforcement Learning. Deep-Q-Network (DQN) makes use of a Deep Neural Network (DNN) with parameters $\boldsymbol{\theta_\pi}$ to approximate the Q function and generate action values.

The input to the DNN is a state s and the output is an action, $Q(s[t], a[t]|\boldsymbol{\theta_\pi})$. An action selection strategy, such as epsilon-greedy, is then used to choose one action, $a[t] = \underset{a[t]}{\operatorname{argmax}}\, Q(s[t], a[t]|\boldsymbol{\theta_\pi})$ [67]. DNN training, in general, requires a loss function optimization that aims to minimize the deviation between labeled training samples and the prediction outputs of the DNN. The weights of the DNN are updated using backpropagation and stochastic gradient descent in the direction that minimizes the loss [67]. The same training approach applies to DQN training, where the goal of the DQN is to make the Q-value prediction as close as possible to the target Q-value. The Q-learning algorithm is used to provide the target Q-value (Q^T) as done in [67] by using

$$Q^T = r[t] + \varepsilon \max_{a[t+1]} Q(s[t+1], a[t+1]|\boldsymbol{\theta_T}), \tag{3.52}$$

where $\boldsymbol{\theta_T}$ are the DNN weights of the Target DNN. A loss function, such as the Mean Squared Error (MSE), is then used to compute the difference between Q^T and the predicted $Q(s[t], a[t]|\boldsymbol{\theta_\pi})$ as done in [67] by using

$$L(\boldsymbol{\theta_\pi}) = \mathbb{E}[(Q(s[t], a[t]|\boldsymbol{\theta_\pi}) - Q^T)^2]. \tag{3.53}$$

The target network is used to provide labels and to improve the stability and convergence of training [67].

3.4.4. Deep Deterministic Policy Gradient (DDPG)

DDPG is another DRL algorithm that is built on top of the DQN. DDPG is an actor-critic framework-based algorithm where the Actor is used to generate actions, and the critic is used to guide the actor to produce better actions [67]. The DQN architecture has several shortcomings that DDPG has addressed. Below, is a discussion the benefits of DDPG over DQN:

- *Suitability for Continuous Action Spaces:* DQN is designed for discrete action spaces. It cannot handle continuous action spaces directly because it relies on a finite set of Q-values for discrete actions [67]. DDPG is specifically designed for continuous action spaces, making it more suitable for real-world tasks such as robotics [67]. DDPG has been widely used in different MEC architectures to provide a complex decision-making framework for dynamic wireless networks.
- *Exploration Efficiency:* DQN uses the epsilon-greedy action selection strategy for randomly selecting an action from a set of predicted actions with probability $\varepsilon \in (0,1)$ inorder to promote exploration of the environment [67]. This works well for a finite, discrete action space but is inefficient in large and complex action spaces. DDPG uses a noise-based exploration strategy such as the Ornstein-Uhlenbeck process, which is better suited for exploring large continuous action spaces effectively [67].
- *Scalability:* Scaling DQN to high-dimensional action spaces can be impractical as the Q-value network must output Q-values for all actions. DDPG scales easily to high-dimensional action spaces since it outputs a single continuous action vector. Figure 3.4 shows the DDPG framework, which is made of Primary and Target Networks, and an Experience Replay Memory [67]. The equations and process for training the DDPG DNNs are explained in the sections for the Twin Delayed architecture, where the differences between the two are highlighted.

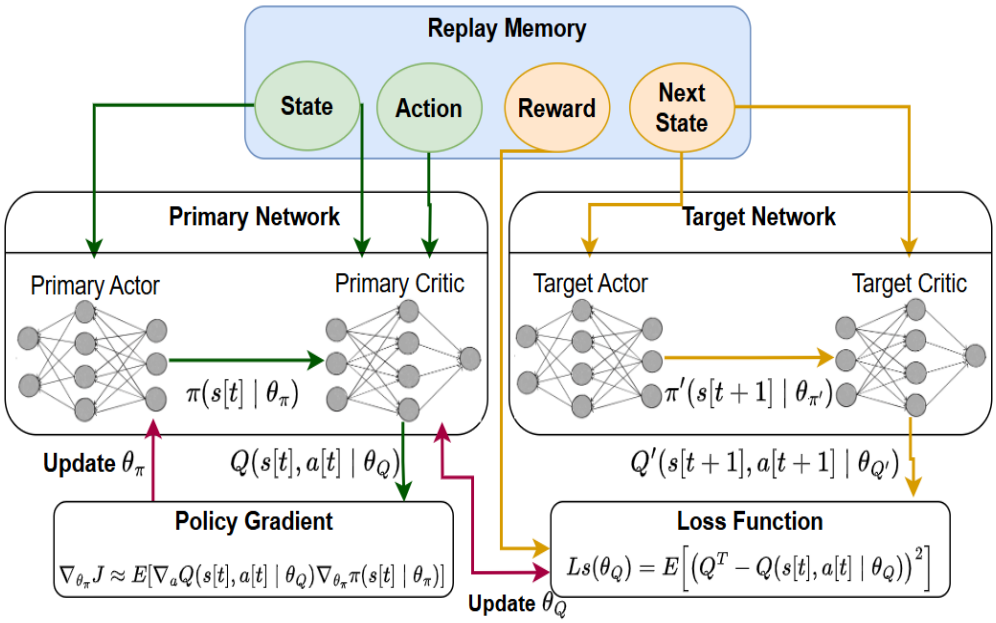

Figure 3. 4: Structure of the DDPG model.

3.4.5. Twin Delayed Deep Deterministic Policy Gradient Model (TD3)

The DDPG algorithm is a widely used DRL-based algorithm for solving optimization problems in computation offloading works, as done in [8], [11], [12], [19], [53], [54], and [58] due to its ability to handle continuous action spaces. This dissertation proposes a state-of-the-art variant of the DDPG called the Twin Delayed DDPG, which takes into account the interaction between the error in function approximation in both policy and Q-value updates [57]. The TD3 has been shown to outperform the DDPG by a wide margin, as demonstrated in [57], producing higher rewards than the latter. In value-based reinforcement learning methods such as deep-Q-learning, function approximation errors are known to lead to overestimations and suboptimal policies. This problem persists in an actor-critic setting, and the TD3 is built to minimize this problem. The algorithm builds on double-Q-learning by taking the minimum value between a pair of critics to limit overestimation.

3.4.5.1. Overestimation Bias

Overestimation bias is an attribute of Q-learning where the maximization of a noisy value estimate induces a consistent overestimation [57]. In function approximation problems, this noise is inevitable because of the imprecision of the estimator [57]. This error is further induced by the nature of temporal difference learning, in which an estimate of the value function is updated using the estimate of a subsequent state [57]. This phenomenon leads to an accumulation of errors as erroneous estimates are used in each update [57]. The accumulated errors can cause bad states to be estimated as high values, resulting in divergent behavior and ultimately suboptimal policies [57].

In Q-learning with discrete actions, the value estimate is updated with a greedy target $Q^T = r + \varepsilon \max_{a[t+1]} Q(s[t+1], a[t+1])$ [57], however, if the target is prone to error ϵ, then the maximum of the value combined with this error will typically exceed the actual maximum, $\mathbb{E}_\epsilon[\max_{a[t+1]} Q(s[t+1], a[t+1]) + \epsilon)] \geq \max_{a[t+1]} Q(s[t+1], a[t+1])$ [57]. Consequently, even an error that initially has a zero mean can lead to value updates that consistently overestimate, and this bias is then carried through the Bellman equation in (3.50) [57]. While in the discrete setting, overestimation bias is an obvious artifact from the analytical maximization, the presence and effects of overestimation bias are less clear in an actor-critic setting where the policy is updated via gradient descent [57]. In [57], the authors have proved that the value estimate will be an overestimation under some basic

assumptions, and they propose a clipped variant of Double Q-learning in an actor-critic setting to reduce overestimation bias.

3.4.5.2. Clipped Double Q-Learning for Actor-Critic

The clipped variant of double-Q-learning introduces two separate value estimates that are independent of each other. They upper bound the less-biased value estimate by the biased estimate. This results in taking the minimum between the two estimates, to give the target update of the clipped double-Q-learning algorithm, as shown in [57] by

$$Q^T = \mathcal{r} + \varepsilon \min_{i=1,2} Q_i'\left(s', a' \middle| \boldsymbol{\theta}_{\boldsymbol{Q_i}}^{\boldsymbol{T}}\right). \tag{3.54}$$

Clipped double-Q-learning ensures that the value target does not cause any extra overestimation compared to the standard Q-learning target [57]. Although this update rule might lead to an underestimation bias, it is much more desirable than an overestimation bias [57]. This is because, unlike overestimated actions, the values of underestimated actions are not directly propagated through the policy update [57].

3.4.5.3. TD3 Training

The TD3 follows the Actor-Critic framework. The Actor is used to generate actions, and the Critic is used to evaluate the action values, thereby guiding the Actor to produce better actions over time. The TD3 consists of three modules: Primary Network (PN), Target Network (TN), and Experience Replay Memory (ERM). The PN consists of three DNNs, which are the Primary Actor (PA) DNN, denoted by $\pi(s[t]|\boldsymbol{\theta_\pi})$ where $s[t]$ is an input and $\boldsymbol{\theta_\pi}$ represents the DNN weights, the first Primary Critic (PC) DNN, denoted by $Q_1\left(s[t], a[t]\middle|\boldsymbol{\theta_{Q_1}}\right)$, and the second primary critic DNN, denoted by $Q_2(s[t], a[t]|\boldsymbol{\theta_{Q_2}})$.

The TN is used to produce target values for training the primary critic DNNs, and it follows the same structure as the PN but with different parameters denoted by $\pi'(s[t+1]|\boldsymbol{\theta_\pi^T})$, $Q_1'(s[t+1], a[t+1]|\boldsymbol{\theta_{Q_1}^T})$ and $Q_2'(s[t+1], a[t+1]|\boldsymbol{\theta_{Q_2}^T})$. The ERM's function is to store a finite number of the most recent experience tuples. A single tuple consists of a state $s[t]$, the corresponding selected action $a[t]$, the next state $s[t+1]$, and the immediate reward $\mathcal{r}[t]$. For notational simplicity, the experience tuple $< s[t], a[t], s[t+1], \mathcal{r}[t] >$ shall be represented by $< s, a, s', \mathcal{R} >$. Random sampling of experience tuples is used to break the correlation among experiences in a training batch. The TD3 training equations formulated in this paper are derived from the original TD3 paper in [57].

Primary Actor DNN Training: The computation offloading and resource allocation policy is represented by the primary actor DNN with weights, $\boldsymbol{\theta_\pi}$, that map the state s to an action

a. At each policy update, the noise $\epsilon \sim \mathcal{N}(0, \sigma)$, is added to the state variable to enable the policy to explore random actions. The generated action is therefore described as [57]

$$a = \pi(s|\boldsymbol{\theta_\pi}) + \epsilon. \tag{3.55}$$

The primary actor weights $\boldsymbol{\theta_\pi}$, are updated using the deterministic policy gradient as [57]

$$\nabla_{\theta_\pi} J = \mathbb{E}\left[\nabla_a Q_1(s, a|\boldsymbol{\theta_{Q_1}}) \nabla_{\theta_\pi} \pi(s|\boldsymbol{\theta_\pi})\right], \tag{3.56}$$

where $Q_1(s, a|\boldsymbol{\theta_{Q_1}})$ is a state-action Q-value produced by the first primary critic. At each training step, a mini-batch of W experience tuples is sampled randomly from the ERM, $< s_w, a_w, s'_w, R_w >,\ w \in \{1,2,\dots,W\}$ and is used by the critic networks to produce a corresponding mini-batch of q_1 and q_2 values $<q_{1,w}, q_{2,w}>$. The primary actor weights are then updated by first performing (3.56) and then updating the weights $\boldsymbol{\theta_\pi}$ as done in [57] by

$$\boldsymbol{\theta_\pi} = \boldsymbol{\theta_\pi} - \frac{lr_\pi}{W} \sum_{w=1}^{W} \left[\nabla_a Q_1(s_w, a_w|\boldsymbol{\theta_{Q_1}}) \nabla_{\theta_\pi} \pi(s_w|\boldsymbol{\theta_\pi})\right], \tag{3.57}$$

where lr_π is the learning rate of the primary actor DNN.

Primary Critic DNN Training: The pair of primary critic DNNs $Q_1(s, a|\boldsymbol{\theta_{Q_1}})$ and $Q_2(s, a|\boldsymbol{\theta_{Q_2}})$ are responsible for evaluating the performance of the selected action based on the state-action Q-value function. The state-action Q-values are calculated using the Bellman equation given in [57] as

$$Q(s, a|\boldsymbol{\theta_Q}) = \mathbb{E}\left[r + \varepsilon Q(s', a'|\boldsymbol{\theta_Q})\right]. \tag{3.58}$$

In this case, both the primary critics take the current state s and the current chosen action a as input to calculate the Q-value, $Q(s, a|\boldsymbol{\theta_Q})$, of each state-action pair. The primary critic DNNs update their weights by minimizing the loss function $L(\boldsymbol{\theta_{Q_1}}, \boldsymbol{\theta_{Q_2}})$ defined in [57] as

$$L(\boldsymbol{\theta_{Q_1}}, \boldsymbol{\theta_{Q_2}}) = \mathbb{E}\left[\sum_{i=1}^{2} \left(Q_T - Q_i(s, a|\boldsymbol{\theta_{Q_i}})\right)^2\right], \tag{3.59}$$

where Q_T is the target value obtained from the two target critics, $Q'_1(s', a'|\boldsymbol{\theta^T_{Q_1}})$ and $Q'_2(s', a'|\boldsymbol{\theta^T_{Q_2}})$ as done in [57] and given by

$$Q_T = \mathcal{R} + \varepsilon \min\left[Q'_1(s', a'|\boldsymbol{\theta^T_{Q_1}}), Q'_2(s', a'|\boldsymbol{\theta^T_{Q_2}})\right]. \tag{3.60}$$

The gradient of the loss $L(\boldsymbol{\theta_{Q_1}}, \boldsymbol{\theta_{Q_2}})$ is computed on each of the primary critics' weights as done [57] and given as

$$\nabla_{\boldsymbol{\theta_{Q1}}} L = \mathbb{E}\left[\left\{\sum_{i=1}^{2}\left(Q_T - Q_i(s,a|\boldsymbol{\theta_{Q_i}})\right)^2\right\}\nabla_{\boldsymbol{\theta_{Q1}}} Q_1(s,a)\right], \tag{3.61}$$

$$\nabla_{\boldsymbol{\theta_{Q2}}} L = \mathbb{E}\left[\left\{\sum_{i=1}^{2}\left(Q_T - Q_i(s,a|\boldsymbol{\theta_{Q_i}})\right)^2\right\}\nabla_{\boldsymbol{\theta_{Q2}}} Q_2(s,a)\right]. \tag{3.62}$$

At each training step, the primary critics are updated by first performing (3.61) and (3.62) and then updating the weights as done in [57] and given by

$$\boldsymbol{\theta_{Q_i}} = \boldsymbol{\theta_{Q_i}} - \frac{lr_{Q_i}}{W}\sum_{w=1}^{W}\left[\nabla_{\boldsymbol{\theta_{Q_i}}} L_w\right], \tag{3.63}$$

where the index i is used to denote the 2 primary critics and lr_{Q_i} is the learning rate of each of the critics.

Target Network Training: A delayed update of the target DNNs' weights is performed as done in [57] and given by

$$\boldsymbol{\theta_{\pi}^{T}} = \mu\boldsymbol{\theta_{\pi}} + (1-\mu)\boldsymbol{\theta_{\pi}^{T}}, \tag{3.64}$$

$$\boldsymbol{\theta_{Q_i}^{T}} = \mu\boldsymbol{\theta_{Q_i}} + (1-\mu)\boldsymbol{\theta_{Q_i}^{T}}, \tag{3.65}$$

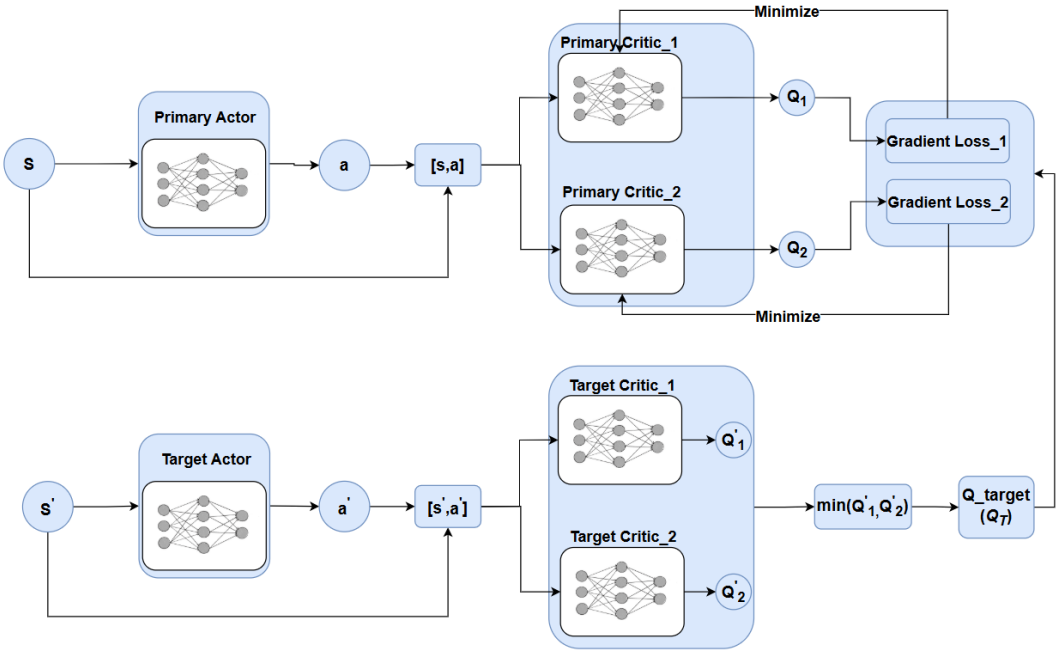

Figure 3. 5: TD3 training structure

where $0 \leq \mu \leq 1$ controls how much update is done. Figure 3.5 shows the network structure of the TD3 training.

3.4.6. TD3 vs DDPG

DDPG and TD3 are designed for applications that exhibit continuous action spaces, making them suitable for this work [57]. The improvements made to the DDPG framework leading to the TD3 architecture have been discussed in Sections 3.4.5.1 and 3.4.5.2. The differences between TD3 and DDPG and the advantages of TD3 over DDPG are discussed below.

3.4.6.1. Q-value Estimation

DDPG: Uses a single Q-network to estimate the value of actions, which can lead to overestimation of the Q-values due to approximation errors in the critic network [57].

TD3: Uses a twin-critic architecture, where two separate Q-networks are trained, and the minimum of the two Q-values is used as the target [57]. This reduces the overestimation bias and leads to more stable training.

3.4.6.2. Policy Updates

DDPG: Updates the primary actor network (policy) and the critic network at every step. This can lead to instability, especially when the critic's Q-function is not yet accurate [57].

TD3: Delays updates to the actor network. The actor is only updated every few steps, for instance, once every two critic updates. This ensures the critic has had enough updates to provide accurate Q-values before the actor tries to improve [57].

3.4.6.3. Net Computational Cost

Increased Cost (TD3): The Twin-Critic Architecture roughly doubles the computational cost for critic updates compared to DDPG [57].

Reduced Cost (TD3): The delayed policy updates mean that the Actor updates are less frequent, reducing the computational burden for actor-network training [57].

The increased cost of updating the twin critics is significantly higher than the reduction offered due to delayed policy updates. TD3, therefore, is more computationally costly than DDPG. However, despite its higher computational cost, TD3's benefits in stability and performance typically outweigh the added expense [57]. With the twin-critic and policy smoothing offering a more stable training process, this often leads to faster convergence and fewer training iterations required to achieve optimal performance.

3.4.7. TD3-based Computation Offloading Scheme

This work now leverages DRL to solve P1. P1 is transformed into an MDP, and an appropriate DRL-based algorithm is proposed. An MDP is characterized by a finite set of states, actions, and transition probabilities of moving from state $s[t]$ to $s[t+1]$, after executing an action, and an immediate reward function for performing an action while in state $s[t]$. The gNB is modeled as an agent that observes the network environment states and performs appropriate actions that maximize long-term rewards. At each time slot, the agent receives the observation $s[t]$, takes an action $a[t]$, and receives an immediate reward $r[t]$ and environment then evolves to the next state $s[t+1]$. The state space, action space, and reward function for this system are defined as follows:

System State Space: The state observed by the gNB in time slot t is defined as $s[t] = \{\boldsymbol{\alpha}_\mathbf{m}[t], \mathbf{g}_\mathbf{m}[t], \mathbf{H}_\mathbf{m}[t], \mathbf{Q}_\mathbf{m}^\mathbf{lc}[t], \mathbf{Q}_\mathbf{m}^\mathbf{off}[t]\}$

- $\boldsymbol{\alpha}_\mathbf{m}[t] = \{\alpha_1^{(m)}[t], \dots, \alpha_D^{(m)}[t]\}$ are the instant small-scale channel gains between the UEs and the gNB.
- $\mathbf{g}_\mathbf{m}[t] = \{g_1^{(m)}[t], \dots, g_D^{(m)}[t]\}$ are the instant large-scale channel gains between the UEs and the gNB.
- $\mathbf{H}_\mathbf{m}[t] = \{H_1^{(m)}[t], \dots, H_D^{(m)}[t]\}$ are the battery states of the UEs.
- $\mathbf{Q}_\mathbf{m}^\mathbf{lc}[t] = \{Q_{1,lc}^{(m)}[t], \dots, Q_{D,lc}^{(m)}[t]\}$ are the local task queue lengths of the UEs.
- $\mathbf{Q}_\mathbf{m}^\mathbf{off}[t] = \{Q_{1,off}^{(m)}[t], \dots, Q_{D,off}^{(m)}[t]\}$ are the offloading task queue lengths of the UEs.

Action Space: The gNB performs actions in timeslot t defined as $a[t] = \{\mathbf{x}_\mathbf{m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{P}_\mathbf{m}[t]\}$

- $\mathbf{x}_\mathbf{m}[t] = \{x_1^{(m)}[t], \dots, x_D^{(m)}[t]\}$ denotes the offloading ratios of all UEs.
- $\boldsymbol{\sigma}_\mathbf{m}[t] = \{\sigma_1^{(m)}[t], \dots, \sigma_D^{(m)}[t]\}$ denotes the RB allocations done by the gNodeB to all UEs.
- $\mathbf{P}_\mathbf{m}[t] = \{P_1^{(m)}[t], \dots, P_D^{(m)}[t]\}$ denotes the transmit powers allocated by the gNodeB to all UEs.

System Reward: A reward function is formulated that correlates with the desired objective, P1. Constraints (3.42) to (3.47) are assimilated into the reward to ensure that the model learns behavior or actions that take these constraints into account. Constraints (3.39) to (3.41) are hardcoded into the gNB's functionality by configuring 3 individual functions,

each function for each action type. The function modifies the model outputs in a way that confines them to the respective constraint. For instance, the power values from the model output are values between 0 and 1. The power constraint function then performs normalization to bring the range of 0 to 1 to the range of 0 to P^{max} values. The reward is formulated as

$$\begin{aligned} r[t] \quad &= \mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)} \mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) \\ &+ q_1^{(m)} \sum\nolimits_{d\in\mathcal{D}} V_{1,d}^{(m)} + q_2^{(m)} \sum\nolimits_{d\in\mathcal{D}} V_{2,d}^{(m)} + q_3^{(m)} \sum\nolimits_{d\in\mathcal{D}} V_{3,d}^{(m)} \\ &+ q_4^{(m)} \sum\nolimits_{d\in\mathcal{D}} V_{4,d}^{(m)} + q_5^{(m)} \sum\nolimits_{d\in\mathcal{D}} V_{5,d}^{(m)} + q_6^{(m)} V_6^{(m)} \end{aligned} \tag{3.66}$$

where all the V terms are defined as follows:

$$V_{1,d}^{(m)} = F(H_d[t-1] + \omega_d[t] - E_d[t]), \tag{3.67}$$

$$V_{2,d}^{(m)} = F\left(1 - \rho_{d,lc}^{(m)}[t]\right), \tag{3.68}$$

$$V_{3,d}^{(m)} = F\left(1 - \rho_{d,off}^{(m)}[t]\right), \tag{3.69}$$

$$V_{4,d}^{(m)} = F\left(\epsilon^{max} - \left\{1 - \sum_{Q_{d,lc}^{(m)}=0}^{L_{d,lc}^{max}} \Pr\left(L_{d,lc}^{(m)} \middle| Q_{d,lc}^{(m)}\right) \Pr\left(Q_{d,lc}^{(m)}\right)\right\}\right), \tag{3.70}$$

$$V_{5,d}^{(m)} = F\left(\epsilon^{max} - \left\{1 - \sum_{Q_{d,off}^{(m)}=0}^{L_{d,off}^{max}} \Pr\left(L_{d,off} \middle| Q_{d,off}^{(m)}\right) \Pr\left(Q_{d,off}^{(m)}\right)\right\}\right), \tag{3.71}$$

$$V_6^{(m)} = F\left(\sum\nolimits_{d\in\mathcal{D}} \log_2\left\{\frac{R_d^{(m)}}{R^{min}}\right\}\right), \tag{3.72}$$

where $F(x)$ is a piecewise function i.e. $F(x) = \phi$ if $x \geq 0$, and $F(x) = x$ if $x < 0$. The variable ϕ is a hyperparameter that is tuned empirically. Equation (3.66) is meant to incentivize the agent to make actions that perform the intended optimization. Equations

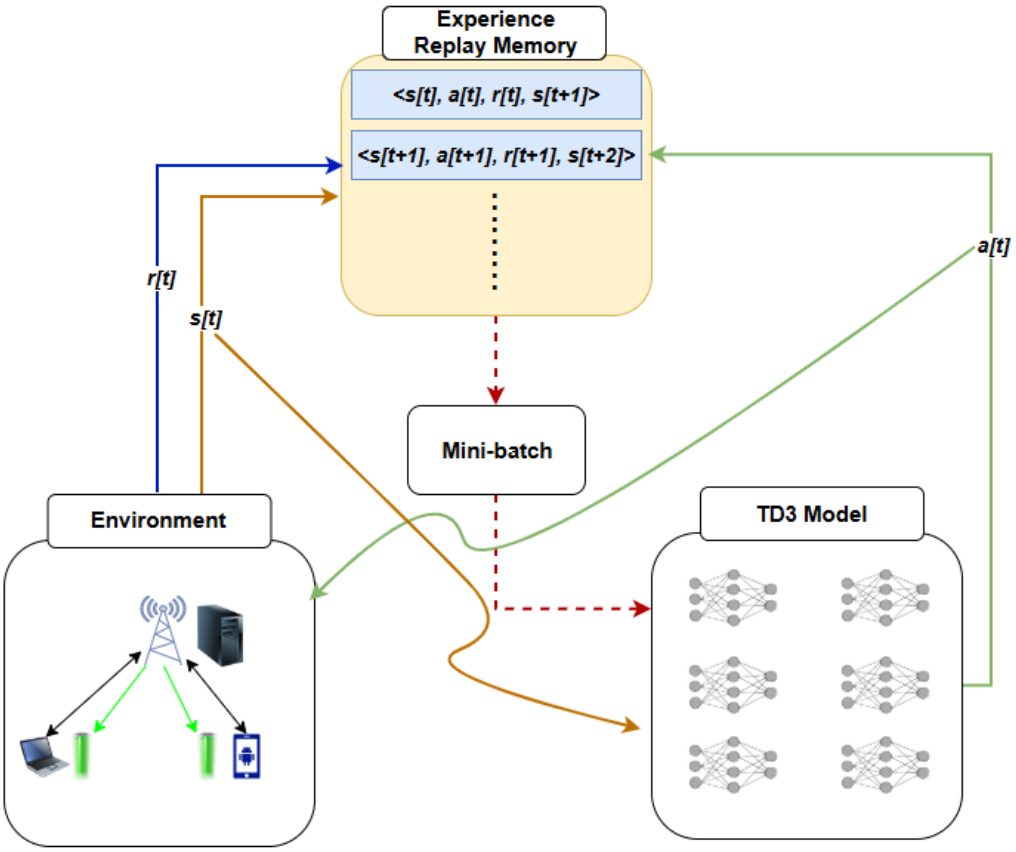

Figure 3. 6: Structure of the TD3-based computation offloading scheme.

(3.67) – (3.72) are introduced into the reward to ensure constraints (3.42) – (3.47) are satisfied. The parameters $q_0^{(m)}$ – $q_6^{(m)}$ are multipliers introduced to increase the order of the respective V term values so that the ensuing values are of the same order as the data rate values. These are also tuned empirically. The goal of the gNB agent is to learn a computation offloading policy that maximizes the expected cumulative discounted rewards, i.e.

$$J = \sum_{\tau=1}^{T} \varepsilon^{\tau}\, r[t-\tau], \tag{3.73}$$

where $\varepsilon \in [0,1)$ is a constant that represents the discount factor and T is a terminal timestep of an episode of model training. Figure 3.6 shows a consolidation of the system model network environment and the TD3 Model.

3.4.8. State and Action Refinement

Preprocessing of state space values and post-processing of action space values are done to ensure that values moving between the environment and the TD3 DNNs conform with what is required on both sides, thus ensuring TD3 learning that achieves convergence.

Algorithm 3.1: TD3-based Computation Offloading Algorithm

Initialize primary actor DNN $\pi(s|\boldsymbol{\theta_\pi})$ with parameters $\boldsymbol{\theta_\pi}$;
Initialize primary critic DNNs $Q_1(s,a|\boldsymbol{\theta_{Q_1}})$, $Q_2(s,a|\boldsymbol{\theta_{Q_2}})$; with parameters $\boldsymbol{\theta_{Q_1}}$ and $\boldsymbol{\theta_{Q_2}}$ respectively;
Initialize target actor DNN $\pi'(s|\boldsymbol{\theta_\pi^T})$ with parameters $\boldsymbol{\theta_\pi^T}$;
Initialize target critic DNNs $Q_1'(s,a|\boldsymbol{\theta_{Q_1}^T})$, $Q_2'(s,a|\boldsymbol{\theta_{Q_2}^T})$ with parameters $\boldsymbol{\theta_{Q_1}^T}$ and $\boldsymbol{\theta_{Q_2}^T}$ respectively;
Initialize experience replay memory;
for *each episode* **do**
    Initialize environment setup {users, communication channel, gNB};
    for *each time slot t, t ∈ {1,…,T}* **do**
        gNB acquires environment state $s[t] = \{\boldsymbol{\alpha_m}[t], \mathbf{g_m}[t], \mathbf{H[t]}, \mathbf{Q_m^{lc}}[t], \mathbf{Q_m^{off}}[t]\}$;
        gNB performs the action $a[t] = \{\mathbf{x[t]}, \boldsymbol{\sigma}\mathbf{[t]}, \mathbf{P[t]}\}$;
        Calculate energy consumed, task delay, and throughput for each user $d^{(m)}$, using (3.36), (3.30) and (3.2) respectively;
        Calculate the total reward achieved based on (3.66) and update the environment to the next state $s[t+1]$;
        Sample a mini-batch of W tuples from the experience replay memory;
        Perform $a'_w \leftarrow \pi'(s'_w|\boldsymbol{\theta_\pi^T}) + \epsilon$ for each sample, w, of the mini-batch;
        Compute 2 target Q-values $<q'_{1,w}, q'_{2,w}>$ for each sample by performing $q'_{i,w} = Q'_i(s', a'|\boldsymbol{\theta_{Q_i}^T})$;
        Calculate 2 corresponding primary critic Q-values;
        Compute both Primary Critics' losses using (3.59) and minimize these losses using (3.61) and (3.62);
        if *t mode d* **then** (Update every 2 iterations)
            Update Primary Actor using the policy gradient in (3.56) and (3.57);
            Update all target DNNs using Poliak averaging in (3.64) and (3.65);
        end
    end
end

3.4.8.1. State Space Feature Scaling

State inputs are normalized at each step of training to a range of values between 0 and 1. Normalization is a process of changing the scale of values in a given range. The maximum

and minimum values of the original range of values would be required to perform such a scaling, and the normalization is calculated as

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{3.74}$$

where X_{norm} is the normalized state value that goes into the DNN. X is the original value, which could be a very small value, such as a large-scale channel gain with an order of magnitude in the 10^{-10} to 10^{-15} range or a battery energy capacity value that could be as large as 2660 mAh. The parameters X_{max} and X_{min} are the maximum and minimum values of the range of the X feature value, and they are computed by performing an initial simulation run for a large number of timesteps without model training and extracting and storing all random feature values. When the initial simulation run is terminated, the X_{max} and X_{min} for each state feature are found from a set of all stored random features generated in the initial simulation run. Feature scaling is done for the following reasons:

- *Improved Training Convergence:* DNNs learn by adjusting weights based on the gradient of the loss function. If input features are on varying scales, the optimization algorithm will take inefficient steps because gradients will vary significantly for different input dimensions. Normalization thus ensures that all state features contribute similarly to the gradient, resulting in faster and more efficient learning and convergence of training.
- *Consistency Across Features:* Features with different scales may cause some to dominate the learning process. DNNs are particularly sensitive to input magnitude. Normalization ensures that all features are on the same scale, preventing bias toward any particular feature.

3.4.8.2. Action Space Scaling

The Actor DNN outputs values between 0 and 1 due to the sigmoid activation function that is used on the output layer. The maximum transmit power of each user is set to 20 dBm, thus, the scale of 0 to 1 from the DNN is normalized to a scale of 0 to 20 dBm of power values. The offloading actions are set to values between 0 and 1 even in the environment, hence, the DNN output values for offloading actions are used as they are.

3.4.8.3. RB Allocation Actions

The RB allocation action values require 2 processes before being applied to the environment. First, the DNN's continuous output values between 0 and 1 must be turned

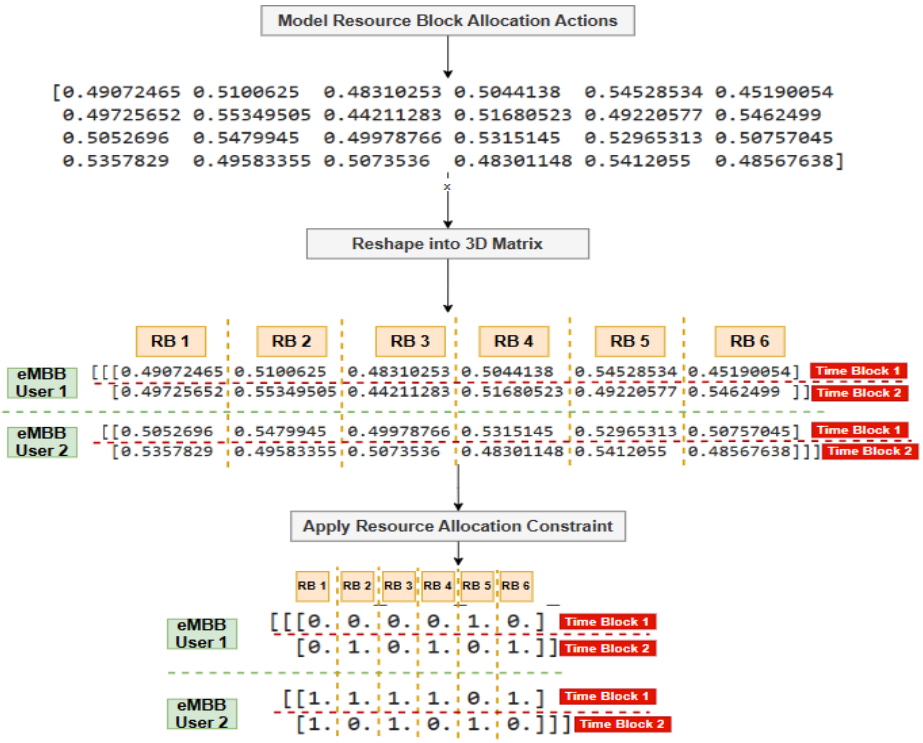

Figure 3. 7: Procedure for refining the RB allocation actions.

into binary allocators, 0 indicating that a RB is not assigned and 1 indicating that the RB is assigned. Second, the RB allocation constraint is applied, ensuring that no single RB is assigned to multiple users. A function is used that couples both processes. The function starts by extracting the continuous values corresponding to the RB allocation actions from the vector of the DNN outputs into a one-dimensional RB allocation action matrix. It then reshapes the RB allocation matrix into a 3D shape that conforms to dimensions corresponding to the number of eMBB users in the environment, the maximum number of RBs that can be allocated, and the maximum number of RB units that a single timeslot can be split into. The 5G reconfigurable frame structure is modeled using this 3D matrix as shown in Figure 3.7 below. Each RB column can only have a maximum of 2 binary ones, indicating either the allocation of both time blocks of a particular RB to a single user or both time blocks allocated to different users. To determine which user or users will be allocated the binary ones of each RB column, the function extracts the RB column and finds which user or users have the first 2 maximum continuous values. It then sets these continuous values to binary ones and the rest to binary zeros. The process is repeated for all the other RB columns. The process is shown in Figure 3.7.

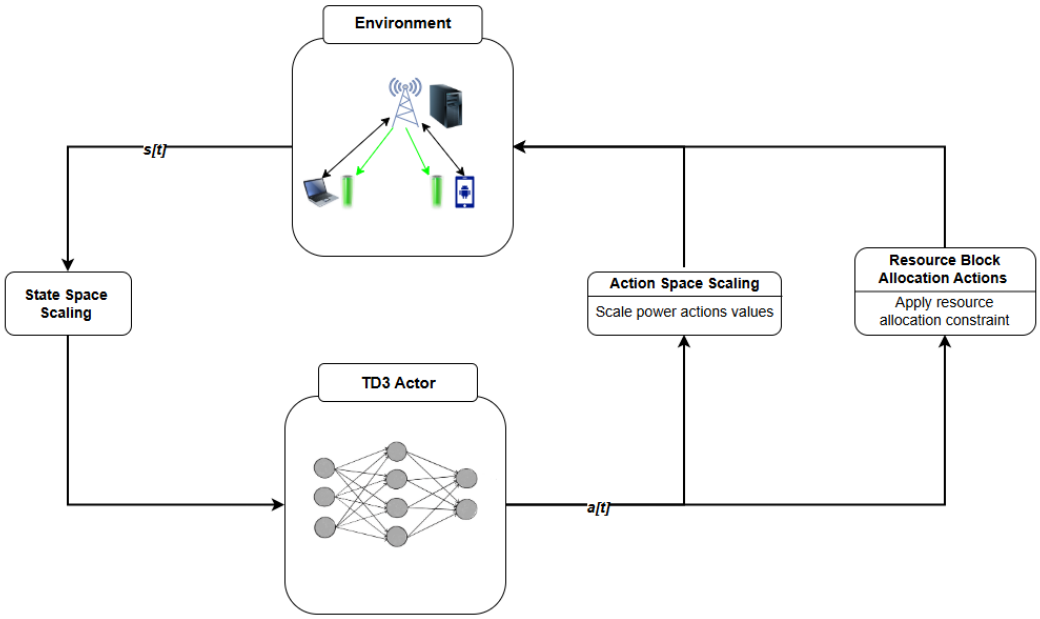

Figure 3. 8: State and action refinement between the environment and the TD3 actor model

3.5. Complexity Analysis

The TD3-based Computation Offloading scheme consists of two main stages outlined in Algorithm 1. The first stage is a calculation of the system reward using (3.66), and this follows the calculations of the three variables in the objective function, which are the achieved throughput, energy consumption, and task delay. The second stage is the TD3 training. The overall complexity of the computation offloading scheme depends on the individual complexities of these two stages.

3.5.1. System Reward Complexity

The complexity of the calculation in (3.2) for the achieved throughput depends on the number of RBs and users. The total complexity of (3.2) over the number of users and RBs is then $D^{(m)}O\left(B^{(m)}\right)$. The complexity of calculating the task delay largely depends on the number of tasks in each queue and the number of users. The single-user local queue complexity is $O\left(Q_{d,lc}^{(m)}\right)$ and that for the offloading queue is $O(Q_{d,off}^{(m)})$. Taking over all the users, the task delay complexity is then $D^{(m)}\left\{O\left(Q_{d,lc}^{(m)}\right) + O\left(Q_{d,off}^{(m)}\right)\right\}$. The single-user energy consumption complexity largely depends on the number of RBs as defined in (3.35). The overall energy consumption complexity is thus defined by $D^{(m)}\{O\left(B^{(m)}\right)\}$. The

overall complexity of calculating the achieved throughput (R), task delay (T), and energy consumption (E) is defined as:

$$C(R,T,E) = O\left(D^{(m)} max\left\{B^{(m)}, Q_{d,lc}^{(m)}, Q_{d,off}^{(m)}\right\}\right). \quad (3.75)$$

3.5.2. TD3 Training Complexity

As defined earlier, the TD3 consists of 6 neural networks, which are, Primary Actor (PA), 2 Primary Critics (PC), Target Actor (TA), and 2 Target Critics (TC). The subscript $\xi = \{PA, PC, TA, TC\}$ is used to denote a specific neural network. The TD3 training complexity largely depends on the number of neural networks, the number of states, and/or actions in each neural network (d_ξ), number of hidden layers (L_ξ) and neurons (n_ξ) in each neural network and the number of operations done in each neural network. The operations done in each neural network consist of the forward pass, where the states and/or actions are passed into the network, and the backward pass, where a gradient update is performed to update the network's weights.

The forward pass of the PA has a complexity of $O(L_{PA} n_{PA} d_{PA})$ where $d_{PA} = 3D^{(m)}$ is the dimensionality of the action space, which consists of 3 action variables per user. The backward pass of the PA has the same complexity as the forward pass. The forward pass of each PC has a complexity of $O(L_{PC} n_{PC} d_{PC})$ where $d_{PC} = 7D^{(m)}$ is the dimensionality of the PC, which takes as input both the state (4 states per user) and action (3 actions per user) spaces. The backward pass of each PC has the same complexity as the forward pass. The forward and backward passes of the TCs follow the same dimensionalities and complexities as the PCs, $O(L_{TC} n_{TC} d_{TC}) = O(L_{PC} n_{PC} d_{PC})$. The overall complexity of TD3 training is defined as $O(\max\{2L_{PA} n_{PA} d_{PA}, 4L_{PC} n_{PC} d_{PC}\})$.

To determine the overall complexity of the TD3-based Computation Offloading scheme, $C(R,T,E)$, $C(TD3)$, the number of training steps, $t \in \{1, \dots, T\}$, and episodes $e \in \{1, \dots, E\}$ are considered. Since each episode e has T timesteps, the overall complexity is calculated as: $O(eT\max\{C(R,T,E), C(TD3)\})$.

3.6. Simulation Results and Analysis

3.6.1. Evaluation Schemes

This section presents the numerical results used to validate the proposed TD3-based computation offloading algorithm. To verify the performance of the proposed algorithm, the following benchmark algorithms are used:

- *DDPG:* The TD3 learning is evaluated against its earlier DRL variant. The DDPG is trained using the same environment, states, actions, and reward function as the TD3. The DDPG is the benchmark that is widely used in DRL-based computation offloading literature.
- *Different TD3 Policies:* The TD3 performance is evaluated against different TD3 Policy variants that differ in the reward shaping.

3.6.2. Technology Stack Used for Implementation

For reproducibility, this section provides details of the tools and technologies that have been used in the implementation of the system model and deep learning models.

3.6.2.1. Python

Python is an open-source, high-level, and general-purpose programming language that is well known for its simplicity, readability, and versatility [72]. Python is very famous and widely used among the Artificial Intelligence community due to its several features that make it the most suitable programming language for machine learning projects [72].

- Python's syntax is very intuitive, resembling natural language, thus making it easy to learn [72].
- Python comes with rich standard libraries that simplify machine learning application development, tasks, and operations. Libraries such as Numpy for numerical computations, Pandas for data manipulation and analysis, Matplotlib/Seaborn for visualizations, scikit-learn for machine learning algorithms, Tensorflow, Keras, and PyTorch for deep learning [72]. These libraries are well-documented and continuously updated by the community
- Python has a big community maintaining, updating libraries, and providing extensive tutorials and learning resources [72].

Given these advantages and others, Python is a highly suitable choice for developing machine learning applications when compared to other programming languages. It has been widely accepted in academia and industry as the de facto language for teaching, research, and innovation in machine learning applications [72].

3.6.2.2. PyTorch

PyTorch is an open-source machine-learning Python library developed by Facebook's AI research lab [73]. It is widely used for developing deep learning models and provides and very flexible framework for experimentation and production. PyTorch is built on top of Python and provides strong support for GPU acceleration, making it efficient for both

research and production environments [73]. PyTorch has several key features that make it the most suitable deep learning library such as:

- *Dynamic Computation Graphs:* PyTorch constructs computation graphs in real-time as operations are executed allowing easy debugging and flexible models [73].
- *Pythonic Syntax:* Since PyTorch is built on top of Python, this makes it easy for Python programmers to learn [73].
- *Strong GPU support:* With a single line command, PyTorch seamlessly offloads computations to a GPU, enabling fast training times and support for training large models [73].

With Tensorflow being an alternative library to PyTorch for deep learning applications, PyTorch has unique advantages that are application-dependent:

PyTorch has gained more popularity in industry and academia than Tensorflow due to its ease of use and Pythonic syntax [73]. In this regard, PyTorch has an extensive community that continuously updates the library to make it more user-friendly and efficient. PyTorch integrates easily with other Python libraries like Numpy, Pandas, and Matplotlib [73]. Tensorflow, on the other hand, often requires converting between Tensorflow Tensors and Numpy, making it cumbersome to use [73].

3.6.2.3. Google Colab

Google Colab is a cloud-based platform that allows Google users to write, run, and share Python code in a Jupyter Notebook environment [74]. It is well suited for training deep learning models, offering both free and paid access to GPUs and TPUs. Google Colab is hosted on Google's cloud infrastructure, and this removes the need to acquire local hardware [74]. Colab integrates seamlessly with Google Drive and files can be cloned from a public GitHub repository making it easy to share files with external sources and applications [74]. Colab comes with many pre-installed libraries including PyTorch and all the other machine learning libraries, which essentially reduces setup time. Users are allocated runtimes on a CPU, GPU, or TPU with limited connection time depending on the subscription [74]. The Free-tier version offers support for many small-to-medium projects. The Colab Pro and Pro+ tiers offer longer runtimes, faster GPUs, and more memory. Table 3.1 shows the specifications of GPUs that Colab offers, and these are offered based on subscription.

Table 3. 1: Google Colab GPU specifications. Table adapted from [74].

GPU	Specifications			
	Compute Capability	**CUDA Cores**	**Memory**	**Peak FP32 Performance**
NVIDA Tesla K80	3.7	2496	12 GB GDDR5	~4.1 TFLOP
NVIDIA Tesla T4	7.5	2560	16 GB GDDR6	~8.1 TFLOPs
NVIDIA Tesla P100	6.0	3584	16 GB HBM2	~10.6 TFLOPs
NVIDIA Tesla V100	7.0	5120	16 GB HBM2	~15.7 TFLOPs

3.6.3. Simulation Setup

A network topology that consists of a single MEC server attached to a gNB is simultaed. EH eMBB users are randomly distributed within a radius of the cell of 100 m. The large-scale channel gain of each user follows a path loss model and shadowing. The path loss model is $35.3 + 37.6 \log_{10}(\mathrm{G_d})$ [8], where $\mathrm{G_d}$ is the distance between the gNB and a UE. The shadowing is lognormal distributed with an 8 dB standard deviation. The small-scale channel fading follows Rayleigh Fading. The system consists of 12 RBs where each RB has 12 subcarriers. Each subcarrier has a bandwidth of 15 kHz. Table 3.2 shows the full list of the environment simulation parameters. The environment is created using the OpenAI Gym python library for producing the state and action spaces. The architectures of the primary and critic DNNs are defined in Table 3.4. A sigmoid function is used in the output layer of each DNN so that the output values of the DNNs are in the range of 0 to 1. The exploration noise, $\epsilon \sim (0, \sigma)$ used in (3.44), is set as a Gaussian random variable and directly applied to the output of the sigmoid function to encourage exploration of actions. When the Experience Replay Memory capacity is full, new transitions are set to overwrite the oldest transitions. Table 3.3 lists all the baseline hyperparameter values used for the TD3 model. Hyperparameter tuning will be performed to find the best hyperparameters for simulating the environment parameters.

Table 3. 2: Environment simulation parameters.

Parameter Symbol	**Description**	**Value**
T_s	Slot Duration	1 ms
T_d^{max}	Maximum allowable queueing delay	5 ms
$\lambda_d^{(m)}$	Task arrival rate	Poisson (5) tasks/slot
P_d^{max}	Max UE transmit power	20 dBm
$f_{d,lc}^{(m)}$	UE Computation Capability	40 000 cycles/slot
$z_d^{(m)}$	eMBB UE task size	Geometric (600) bits
k_1	CPU cycles required to process one byte	330 cycles/byte
ς	Switched Capacitance	10^{-18}
υ	Energy Conversion Efficiency	1
η	Transmission power of gNB	37 dBm
$G_d^{(m)}$	Distance between a UE and gNB	Uniform (30, 100) m
N_o	Thermal Noise Density	-174 dBm/Hz
ϑ	Path loss factor component	1.5
$\Omega_d^{(m)}$	Combined gain of gNB and a UE	10 dB
H_d^{max}	Max battery capacity	26640 J (7.4 Wh)
ϵ^{max}	Maximum queue delay violation probability	0.2
$D^{(m)}$	Number of eMBB users	11
R^{min}	Minimum data rate requirement	500 kilobits/s

Table 3. 3: Baseline hyperparameter settings for TD3 model.

Hyperparameter Definition	**Hyperparameter Value**
Actor learning rate	10^{-8}
Critic learning rate	10^{-5}
Discount factor	0.99
Experience Replay Memory size	10 000 Samples
Training Batch size	100
Soft Update Coefficient	0.005
Action Noise	Gaussian (0,0.3)

Episode Timesteps	100

Table 3. 4: DNN structure of TD3 model.

DNN	**Structure**
Primary Actor	Layer 1 (state_dim, 400), relu Layer 2 (400, 300) relu Layer 3 (300, action_dim), sigmoid
Primary Critic 1	Layer 1 (state_dim + action_dim, 400), relu Layer 2 (400, 300) relu Layer 3 (300, 1), sigmoid
Primary Critic 2	Layer 1 (state_dim + action_dim, 400), relu Layer 2 (400, 300) relu Layer 3 (300, 1), sigmoid
Target Actor	Layer 1 (state_dim, 400), relu Layer 2 (400, 300) relu Layer 3 (300, action_dim), sigmoid
Target Critic 1	Layer 1 (state_dim + action_dim, 400), relu Layer 2 (400, 300) relu Layer 3 (300, 1), sigmoid
Target Critic 2	Layer 1 (state_dim + action_dim, 400), relu Layer 2 (400, 300) relu Layer 3 (300, 1), sigmoid

3.6.4. Analysis and Discussion

3.6.4.1. Hyperparameter Tuning – Actor Learning Rate

Figure 3.9 shows the TD3 training results under different actor learning rates for a computation offloading system with 11 users. The Figure shows the system reward that only consists of the sum of energy consumption, the sum of data rates, and the sum of task delays. The constraints have not been included in the reward equation. Only the actor learning rate is varied. Typically, the critic network's learning rate is set higher than that of the actor network to enhance sensitivity to value function approximation [68]. The first observation is that the TD3 performs the required optimization of energy consumption minimization, throughput maximization, and task delay minimization. This can be explained by the choice of environment parameters as well as the model's actions over time. To maximize the throughput, the model produces higher power values in the action

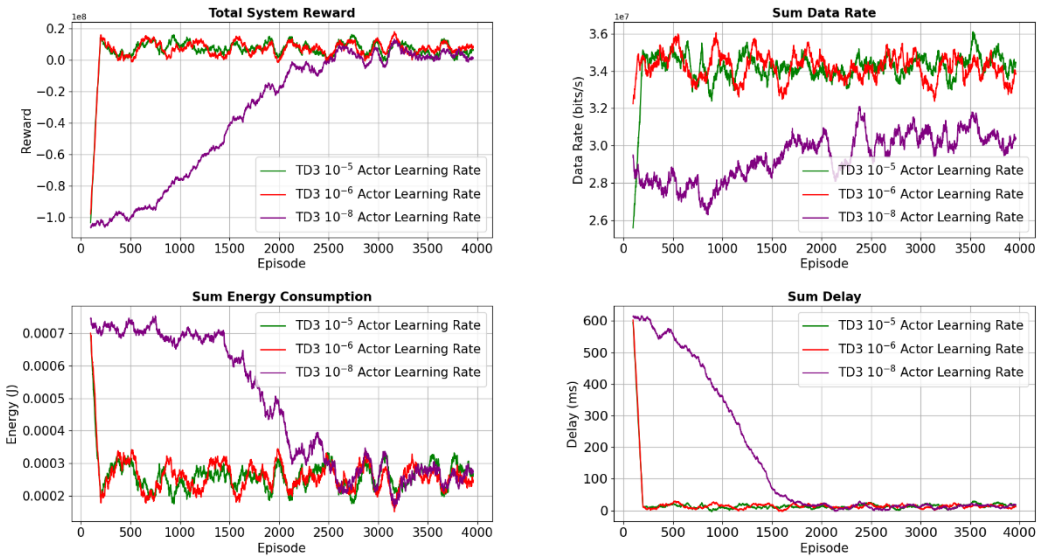

Figure 3. 9: Reward, sum data rate, sum energy consumption, and sum delay under different actor learning rates.

space. The RB allocation also plays a role in increasing the data rate, where the model sometimes may allocate more RBs to users that have high channel gains, however, this leads to an unfair distribution of channel resources, resulting in longer offloading queueing delays for users that have bad channel gains. The task delay component of the reward effectively prevents a highly skewed distribution of RBs since longer queue delays would reduce the reward. The increase in throughput, therefore, mostly depends on the power values in the action space.

Increasing the transmission power values leads to a corresponding increase in the energy consumed for offloading tasks, as defined in (3.35). The plots in Figure 3.9, however, show a reduction in energy consumption, and this is explained by the offloading action done by the model. The energy consumed for offloading is much less in comparison to the energy consumed during local computation. The energy increase that comes from high transmission powers is less than the energy increase from performing local computation. The model, therefore, chooses high offloading ratios for most of the users to reduce the sum of energy consumption. The reduction in the task delay is explained by the high offloading ratios and the increasing data rates. Since the task delay on the offloading queues depends on the data rate, as shown by (3.22), the model's actions of performing more offloading than local computation to reduce energy consumption and increasing power values to increase the data rates are further induced by the need to reduce the task delay and putting tasks on the offloading queue where the model has control over the data rate actions and can effectively reduce the delay by increasing the data rate. If the tasks were to

be sent to the local queues, which have a fixed computation capacity and thus a fixed rate of finishing the tasks, the sum delay would be high since the model cannot have control over the devices' computation rate.

Figure 3.9 also shows that increasing the actor learning rate increases the rate of convergence, which is expected. The learning rate controls the amount by which the model updates its weights in the direction that increases the reward during training. Increasing the learning rate increases the amount of updating of the model's weights. All models converge to the same levels of reward since they all perform similar actions to maximize the reward. The data rates plot, however, shows that the model with a 10^{-8} learning rate converges to lower values of data rates compared to the other learning rates. This indicates that the learning rate has an impact on the model's power actions. The slower learning rate may get the model stuck in a sub-optimal action space of one of the actions, even though the aggregated reward has converged to the optimal values.

The energy and delay drops experienced during the early episodes for the 10^{-6} learning rate are due to the fact that the 10^{-6} learning rate has a faster rate of DNN weight updates compared to the 10^{-8} learning rate. TD3 10^{-5} learning rate results in the same rate of weight updates as TD 10^{-6} learning rate, indicating that any learning rate higher than 10^{-6} does not provide any faster convergence.

3.6.4.2. Hyperparameter Tuning – Number of DNN Hidden Layers

Figure 3.10 shows the effect of changing the number of hidden layers on the total reward, the sum of the data rate, the sum of energy consumption, and the sum of the task delay. It

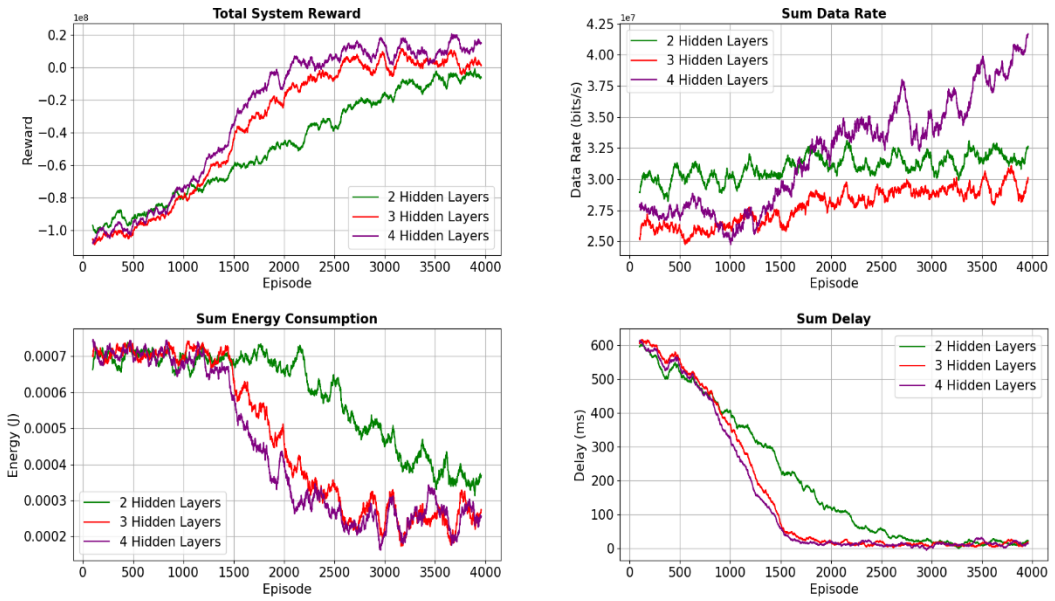

Figure 3. 10: Reward, sum data rate, sum energy consumption, and sum delay under different numbers of hidden layers of the actor and critic DNNs.

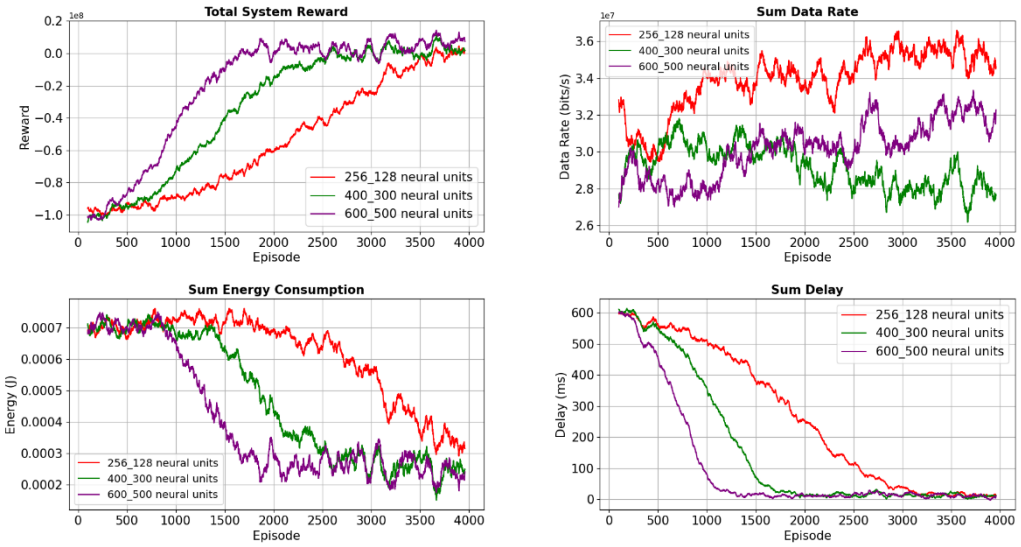

Figure 3. 11: Reward, sum data rate, sum energy consumption, and sum delay under different numbers of neurons per hidden layer of the Actor and Critic DNNs.

is noted that 4000 episodes may not be enough for the convergence of the reward under two hidden layers and the sum data rate under four hidden layers. Google Colab imposes a limit on the runtimes depending on the subscription. The TD3 model could only run up to 4000 episodes before the runtime would disconnect. The plots show a significant improvement in the convergence rate as the number of hidden layers increases. This is because adding hidden layers improves TD3's ability to model increasingly complex relationships and patterns in the training data, thus allowing it to find the optimal actions much quicker. This, however, comes at a computational cost associated with additional network layers. Adding layers beyond 4 resulted in longer runtimes, where the model could not reach convergence before the runtime would disconnect.

3.6.4.3. Hyperparameter Tuning – Number of Neurons per Hidden Layer

Figure 3.11 shows the effect of varying the number of neurons per hidden layer on the performance metrics. The number of hidden layers is fixed at 2 hidden layers. The first model is trained with 256 neurons in the first layer and 128 neurons in the second layer. The second model has 400 and 300 neurons in the respective layers, and the third model has 600 and 500 neurons. Once again, increasing the number of neurons increases the convergence rate of training due to improving the model's ability to learn complex patterns and relationships in the training data and enabling it to find the optimal actions much quicker. The same runtime limitations as with varying the number of hidden layers also

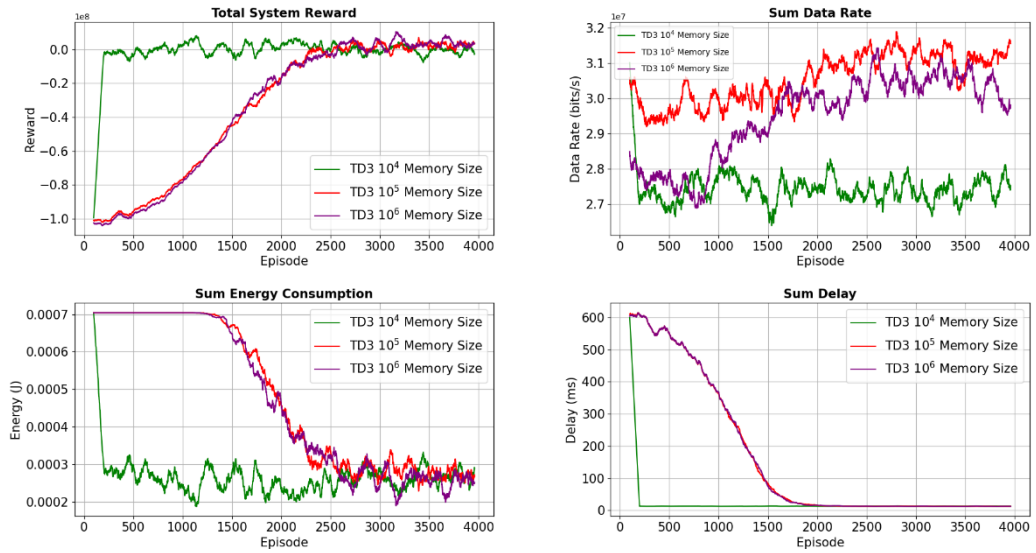

Figure 3. 12: Reward, sum data rate, sum energy consumption, and sum delay under different experience replay memory sizes.

apply to increasing the number of neurons due to the increase in the computational cost associated with increasing the neural units.

3.6.4.4. Hyperparameter Tuning – Experience Replay Memory Size

Figure 3.12 shows the effect of changing the size of the Experience Replay Memory on the performance metrics. As shown by the plots, a smaller memory size of 10^4 leads to faster convergence than memory sizes of 10^5 and 10^6. Since training samples are drawn in batches using a random uniform sampling, older experiences that lead to smaller rewards can be drawn at any point in time for batch learning. These older experiences may slow down learning if the model continuously samples them for training. A larger memory size stores a lot of older experience samples than a smaller memory size.

A smaller memory size has very recent training samples that have high rewards and that reflect the current state of the environment and policy. The model, therefore, can learn very quickly from recent samples with high rewards. A smaller memory size reduces the probability of sampling batches of older experiences with smaller rewards for training. This, however, increases the exploitation of actions and reduces exploration. This type of exploitation can work for the type of environment modeled in this work, which doesn't require extensive exploration to find the optimal policy. The actions of performing more offloading and increasing transmission power levels are not too complicated to require

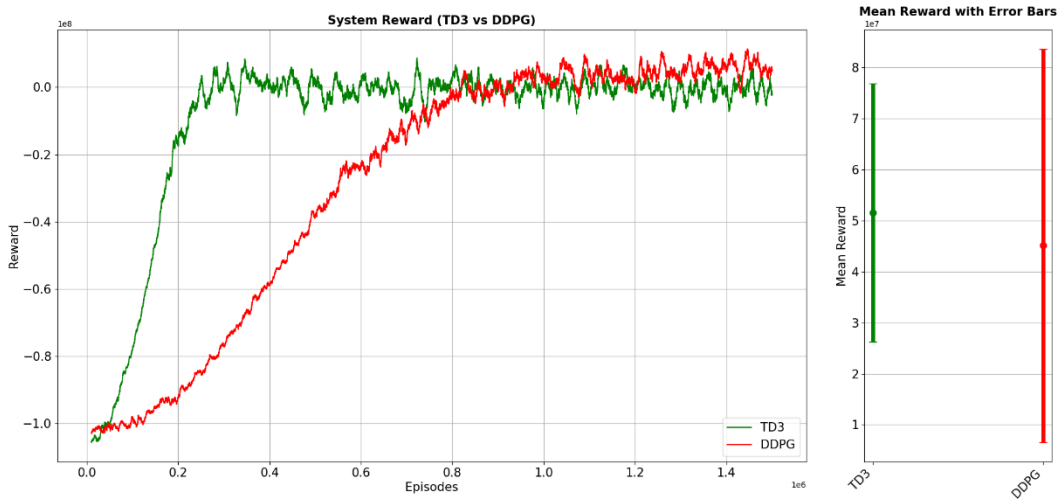

Figure 3. 13: TD3 vs DDPG in reward, variance, and average reward.

extensive exploration. For much more complex environments with varying degrees of actions, reducing the memory size could lead to the exploitation of sub-optimal actions.

3.6.4.5. TD3 vs DDPG

Figure 3.13 shows the differences in policy training in the same environment between TD3 and DDPG. Both models are trained using the same hyperparameters shown in Table 3.3. The figure shows that TD3 converges much quicker than DDPG. The reason for this can be explained by the plot, which shows the average reward and the error bars in training. DDPG shows too much variation in its learning, and this instability in Q-value estimation leads to suboptimal learning. TD3 has clipped double-Q-learning, with two separate critic networks estimating Q-values and taking the smaller of the two for learning. This improves the stability of the learning process, allowing for faster convergence.

3.6.4.6. Effect of Task Arrival Rate on Performance Metrics

In this section, the task arrival rate is varied against 4 different Policies and compared in terms of the reward, the sum of energy consumption, the sum of task queueing delays, and the sum of data rates. The 4 Policies differ in the reward construction used during training. Table 3.5 shows the different Policies and their respective reward functions.

Table 3. 5: Policies 1 to 4 and their respective reward functions used in training.

Policy	**Reward Function**
Policy 1 (π_1)	$\mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t])$
	$\mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t])$

Policy 2 (π_2)	$+ q_2^{(m)} \sum_{d \in \mathcal{D}} V_{2,d}^{(m)}$
Policy 3 (π_3)	$\mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)} \mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t])$ $+ q_3^{(m)} \sum_{d \in \mathcal{D}} V_{3,d}^{(m)}$
Policy 4 (π_4)	$\mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)} \mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t])$ $+ q_2^{(m)} \sum_{d \in \mathcal{D}} V_{2,d}^{(m)} + q_3^{(m)} \sum_{d \in \mathcal{D}} V_{3,d}^{(m)}$

Table 3.5 shows that only two constraints are experimented with, constraints (3.43) and (3.44). The contribution of constraint (3.43) is studied using Policy 2, and constraint (3.44) using Policy 3, and the resultant performance of both constraints using Policy 4. Policy 1 is the baseline model that does not have any constraints built into the reward. The weights of $q_0^{(m)}, q_2^{(m)}$ and $q_3^{(m)}$ are determined experimentally. The value of each weight is a multiplier that brings the respective summation of a V term to the range of values of the sum data rate $\mathcal{R}_d^{(m)}$. Only constraints (3.43) and (3.44) are experimented with while leaving out all the other constraints for the following reasons:

1. Constraint (3.42) is always satisfied even when excluded from the reward function during training. The magnitude of the battery energy level, $H_d[t-1]$, is too high compared to the energy consumed, $E_d[t]$, in each time slot, so at no point does the energy consumption become higher than the battery energy level.
2. Constraints (3.45) and (3.46) are computationally expensive. Calculations of the probabilities of queueing delay violation using (3.13) – (3.17), (3.25) – (3.28) slow down model training time significantly. However, as will be explained, the model's actions for Policy 2 also favor constraint (3.45). The same is true for Policy 3 and constraint (3.46).
3. Constraint (3.47) is already captured in Policy 1. If users are not allocated channel resources to satisfy the constraint (3.47), their offload queue lengths will increase, thus increasing the task queueing delay. Policy 1, therefore, ensures constraint (3.47) is satisfied to minimize the sum of the delay term in the reward.

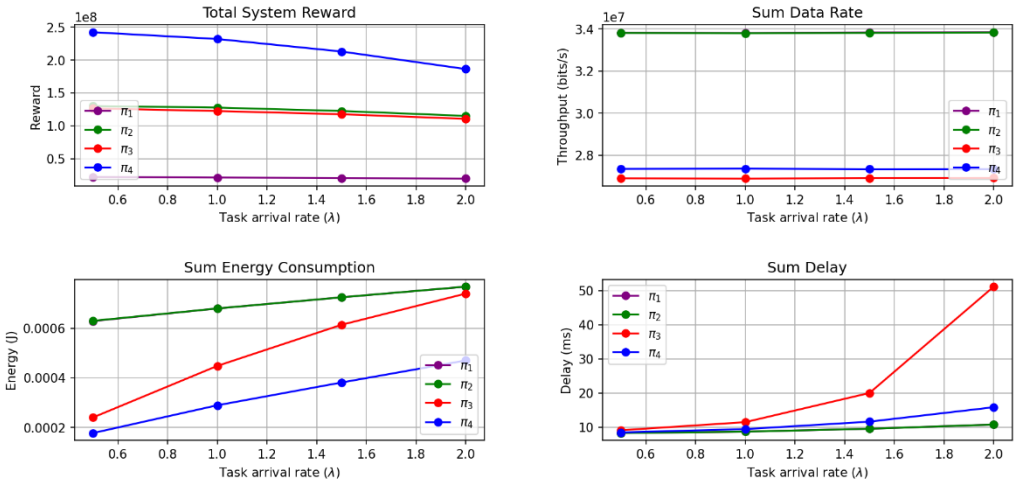

Figure 3. 14: Effect of task arrival rate on the system reward, sum data rate, sum energy consumption, and sum delay on Policies 1 – 4.

Figure 3.14 shows the effect of changing the task arrival rate on the system reward, sum of data rate, sum of energy consumption, and sum of delay against 4 different Policies. The task arrival rate is changed during inference, where the models are already trained. The inference is done over a simulation run of 50000 timesteps. The performance metrics (reward, sum energy, sum data rate, and sum delay) shown in the plots are values averaged over 50000 timesteps. The system reward plot shows that Policy 4 has higher rewards than all the other Policies, followed by Policies 2 and 3, and lastly Policy 1. This is attributed to the reward functions of the individual policies, which are shown in Table 3.5. There is a positive correlation between reward values and the number of terms in the reward function. The variation in the sum data rates across the different Policies can be explained by Figure 3.15, which shows the average optimal offloading ratios by each model. The values are averaged over 50000 timesteps, and over the 11 users used in the inference simulation run. There is a positive correlation between the sum data rates and the offloading ratios. Policy 3 has the constraint (3.46) that seeks to ensure minimal violation of the probability of offload queueing delay violation. To achieve this, the model assigns small offloading ratios to prevent long offloading queues. However, sending tasks to the local queues increases energy consumption due to local processing. To compensate for the increase in energy consumption, the model starts assigning small power values for the transmission of the tasks on the offloading queue. The reduction of power values counters the increase in local energy consumption but also leads to a reduction in the transmission data rates.

Policy 2 is the direct opposite of Policy 3. It has the constraint (3.45) that seeks to ensure minimal violation of the probability of local queueing delay violation. To achieve this, the

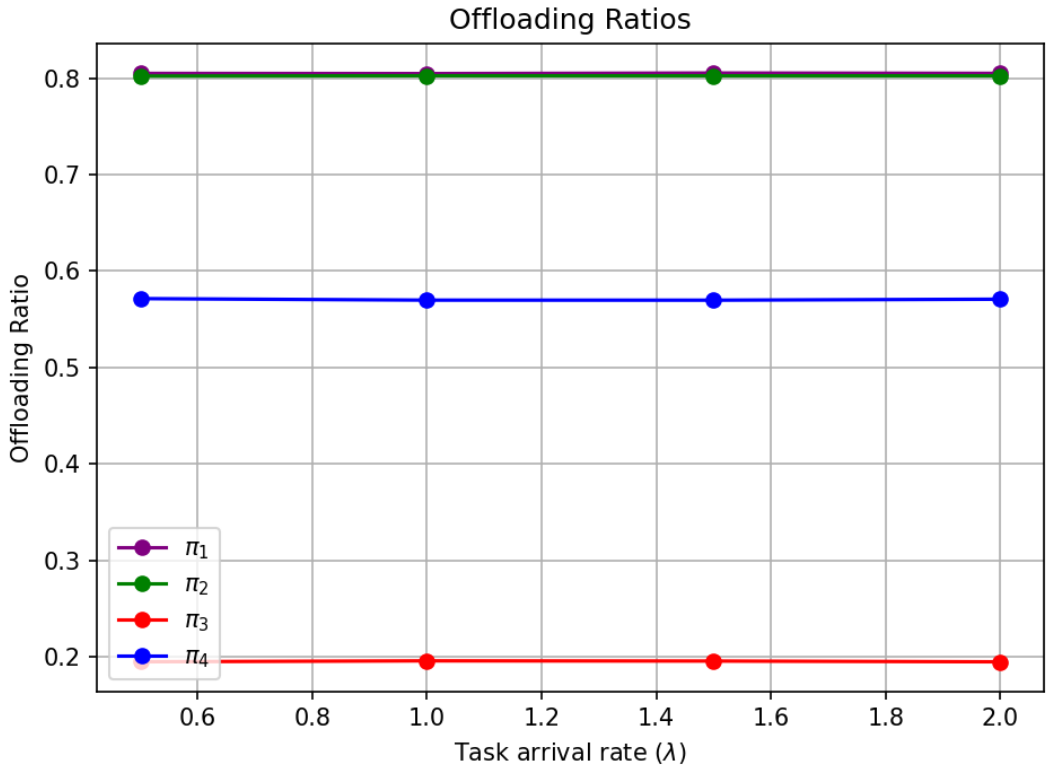

Figure 3. 15: Average offloading ratios for Policies 1 – 4.

model assigns high offloading ratios to prevent long local queues. Sending more tasks to the offloading queues results in the model increasing the transmission power values to increase the data rates, so that the demand for offloading is met by the high transmission data rates. Policy 1 acts in a similar manner to Policy 2, where in the absence of any constraint term in the reward, the model performs more offloading than local computing because offloading results in high data rates, small task queueing delays, and small energy consumption values, compared to local computing.

Policy 4, on the other hand, involves both constraints (3.45) and (3.46) in the reward function. As a result, the model seeks to balance between offloading and local computing to achieve a high reward. The sum data rates of Policy 4 compared to other Policies also reflect this balance. The plot for the sum energy consumption shows an increase in energy consumption as the task arrival rate increases for all Policies. Increasing the task arrival rate leads to an increase in the bits available for processing in each time slot. The calculation of the local energy consumption in (3.34) shows and explains the direct relation between the bit sizes and the energy consumption. Policy 1 and Policy 2 show identical values across all task arrival rates, indicating that these Policies follow the same offloading strategies, and this is proven in Figure 3.15, where the average offloading ratios of the two Policies overlap. Policy 3 starts with a much lower energy consumption than Policies 1 and 2 at

Table 3. 6: Percentage change in energy consumption as task arrival rate increases for Policies 1, 2, 3, and 4.

Task Arrival Rate Increase	**Percentage Change in Energy Consumption**			
	Policy 1	Policy 2	Policy 3	Policy 4
0.5 to 1	≈ 8.1%	≈ 8%	≈ 86%	≈ 63%
1 to 1.5	≈ 6.5%	≈ 6.5%	≈ 37%	≈ 31%
1.5 to 2	≈ 5.9%	≈ 6%	≈ 20%	≈ 23%

lower task arrival rates but then increases more rapidly than all the other Policies to catch up with Policy 1 and 2 energy consumption values. Policy 4 consistently has the lowest energy consumption across all task arrival rates, making it the most energy efficient. Table 3.6 below shows the percentage changes in the energy consumption of each model as the task arrival rate increases.

Table 3.6 shows that Policies 1 and 2 have the smallest percentage increases in energy consumption, making them more stable across different task arrival rates. Their increases range from 5.9% to 8.1%, indicating a linear trend. Policy 3 has the highest percentage increase at lower arrival rates; however, as explained before, the smaller energy consumption compared to Policies 1 and 2 at lower task arrival rates suggests that it starts very energy efficient, but loses its advantage as load increases. Policy 4 shows a high percentage increase at lower task arrival rates. It, however, remains more energy efficient than all Policies across all task arrival rates. The percentage increase for Policies 3 and 4 declines as the task arrival rate increases, suggesting a possible convergence beyond a task arrival rate of 2.

The plot for the sum delays shows that the general trend is an increase in delay as the task arrival rate increases for all the Policies. This aligns with the plots in Figure 3.16, which indicate that increasing the task arrival rate increases the load or the number of tasks in the queues at any time. An increase in the queue length leads to a corresponding increase in queueing delay for any task arriving in any of the queues. Once again, Policies 1 and 2 show very similar behavior across all task arrival rates. Delay values for both Policies slightly increase as the task arrival rate increases, showing stability across all rates, and this indicates an ability to handle loads without significant performance degradation. Policies 1 and 2 send about 80% of arriving tasks to the offload queue, where the model can optimize the allocation of channel resources and transmission power values to minimize the queueing delay.

Policy 3 starts with reasonable delays like the other Policies but starts to deteriorate significantly beyond a task arrival rate of 1. This suggests that Policy 3 scales poorly under

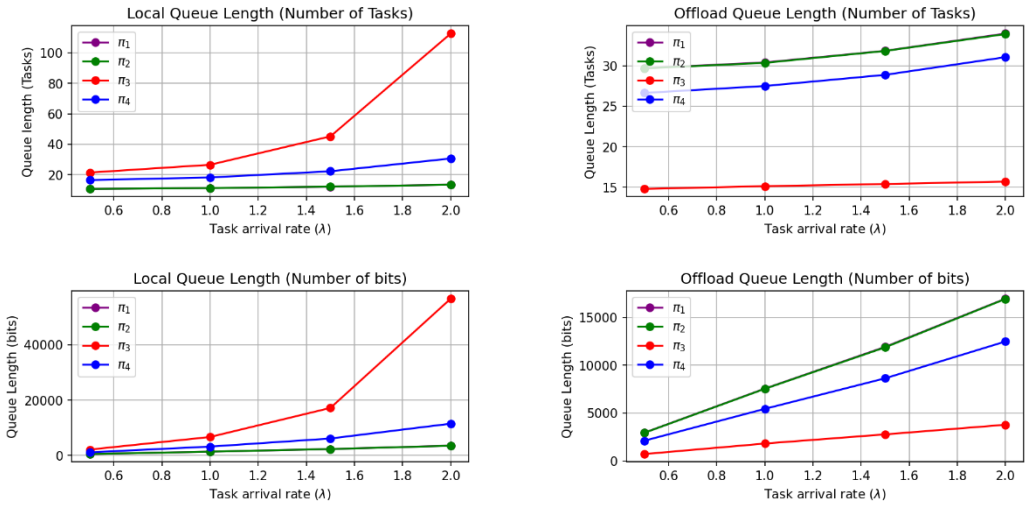

Figure 3. 16: Effect of task arrival rate on local and offload queue lengths (tasks and bits) of Policies 1 – 4.

high loads. Since Policy 3 has lower offloading ratios, most tasks are sent to the local queue for local processing, where the CPU clock frequency is fixed. Once the local queue length reaches ψ^{max}, shown in (3.31), the same amount of time is allocated for processing ψ^{max} in every time slot. The plot suggests that the value of ψ^{max} is reached at a task arrival rate of somewhere between 1 and 1.5, where the delay starts to rise exponentially. As the local queue length grows due to increasing task arrival rates, so does the queueing delay. Policy 4 performs better than Policy 3 but worse than Policies 1 and 2, maintaining reasonable delays up to an arrival rate of 1.5 before accelerating upwards at 2. Policy 4 sends about 60% of arriving tasks to the offloading queue, resulting in the decongestion of the local queue. Moreover, the model can optimize the allocation of channel resources to minimize the offloading delay.

Figure 3.16 shows the evolution of the queue lengths as the task arrival rate increases for all the Policies. The queue lengths for the different Policies are also explained by the offloading ratios in Figure 3.15. These plots support the conclusion derived from the sum delay plot. The task delay has a direct correlation to the queue length. The local queue length for Policy 3 grows quicker than the other policies, resulting in high delay values as shown by the sum delay plot in Figure 3.14.

Figure 3.17 shows plots of

a) Probability of violating the local traffic load constraint, i.e. Pr $\left(\rho_{d,lc}^{(m)} > 1\right)$.

b) Probability of violating the offload traffic load constraint i.e. Pr $\left(\rho_{d,off}^{(m)} > 1\right)$.

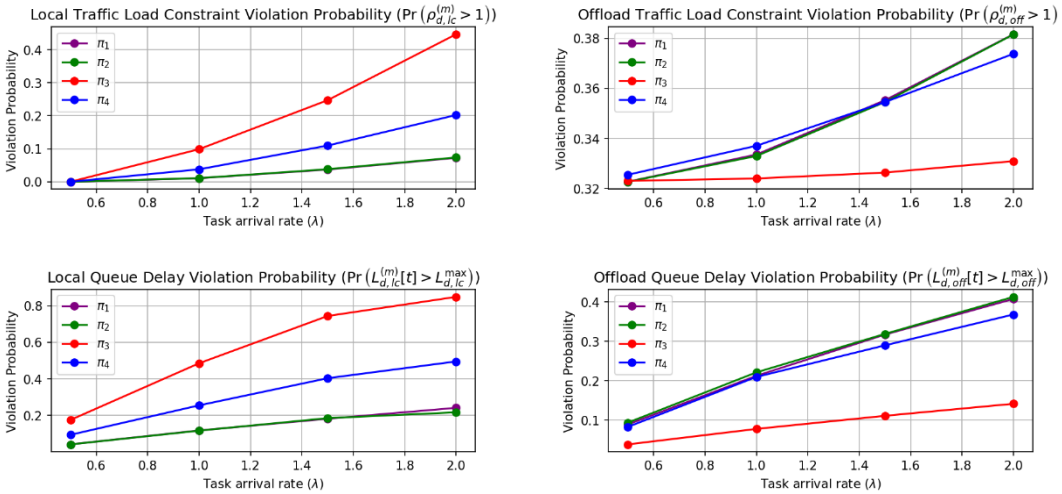

Figure 3. 17: Plots showing the probabilities of violating constraints (3.45) and (3.46), top left and top right, respectively, and probabilities of local and offload queueing delay violation, bottom left and bottom right, respectively. Probabilities averaged over 11 eMBB users.

c) Local delay violation probability i.e. Pr $\left(L_{d,lc}^{(m)}[t] > L_{d,lc}^{max}\right)$.

d) Offload delay violation probability i.e. Pr $\left(L_{d,off}^{(m)}[t] > L_{d,off}^{max}\right)$.

The first observation is that the local queue violation probabilities a) and c) show a positive correlation across all the models with the task arrival rate. The same observation is true for

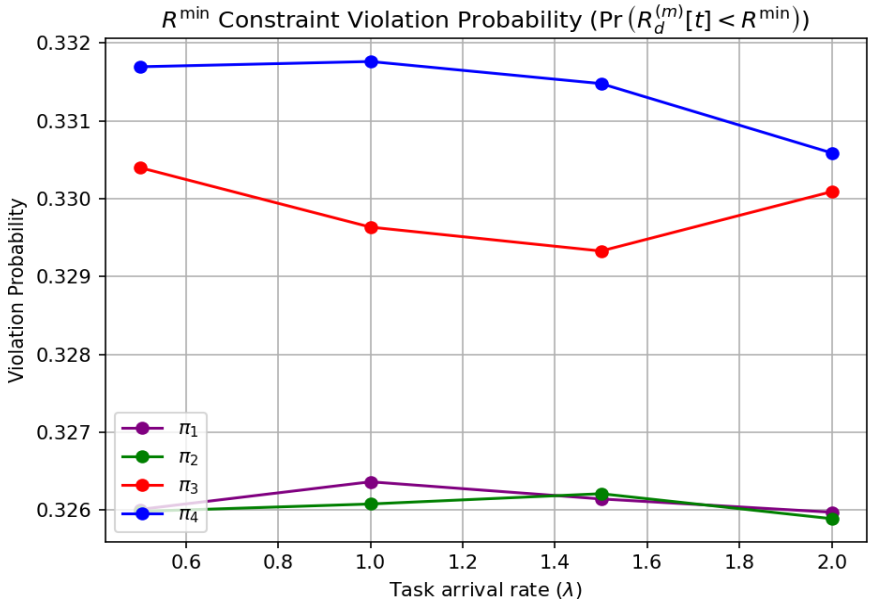

Figure 3. 18: Plot showing the probability of violating the constraint (3.47) by Policies 1, 2, 3, and 4. Probability values averaged over 11 eMBB users.

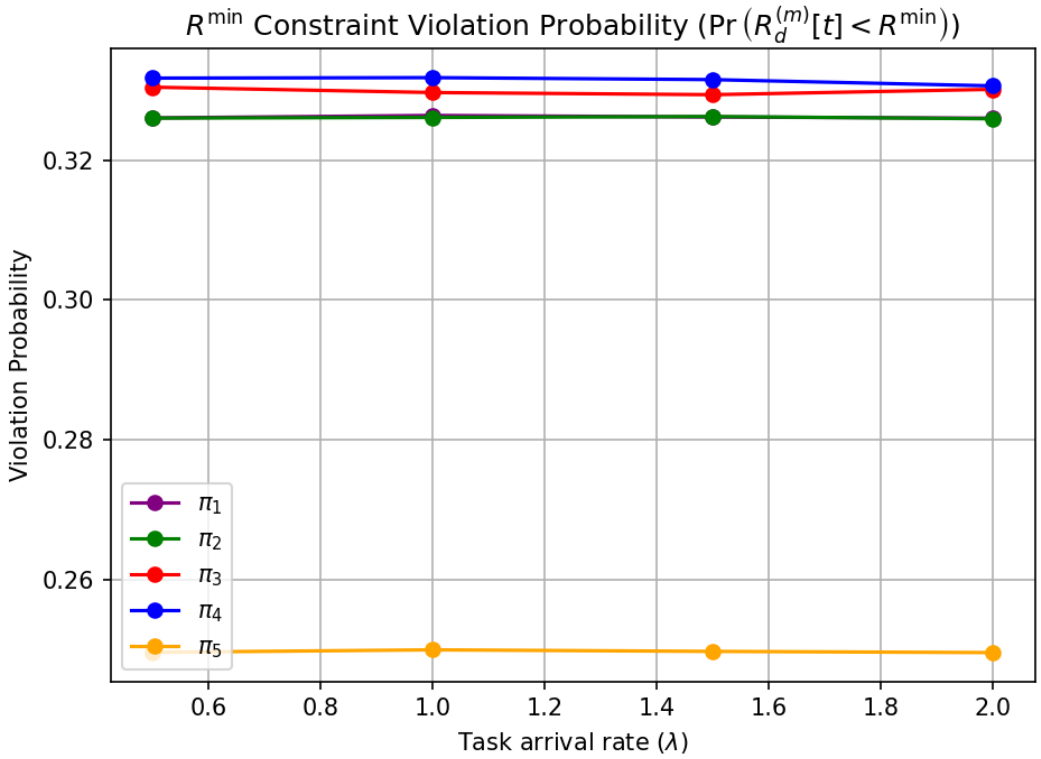

Figure 3. 19: Plot showing the probability of violating the constraint (3.47) by Policies 1, 2, 3, 4, and 5.

the offload queue violation probabilities b) and d). Increasing the task arrival rate increases the load on both queues and thus the queue lengths and this leads to an increase in the violation probabilities as calculated in (3.14), (3.18), (3.24), and (3.25). The performance of the Policies in ascending order of the local queue violation probabilities a) and d) are Policy 3, Policy 4, and then Policies 1 and 2. The reverse order of the Policies is true for the ascending order of performance of the offload queue violation probabilities. This is once again explained by the plots in Figure 3.16, which show the local and offload queue lengths under the different policies. Figure 3.18 shows the probability of violating constraint (3.47) calculated as an average of 11 eMBB users over 50000 timesteps. There seems to be no correlation of this probability with the offloading ratio, as shown in all the other plots. Moreover, there are no significant differences in the probabilities across the various Policies. This is explained by the fact that the constraint (3.47) has not been included in any of the reward terms of the individual Policies during training. The models do not have the motive to behave in any way that correlates with the constraint. The violation probabilities, however, are small, indicating that the resource allocation done by the model and the RB distribution among the eMBB users do consider the delay term in

the reward functions of all the Policies. RBs should be distributed to the eMBB users in a way that prevents long offloading queue delays.

The correlation between model behavior and inclusion of the constraint (3.47) is validated by the plot in Figure 3.19, which shows the addition of Policy 5, whose reward term includes constraint (3.46) as

$$r_{\pi_5} = r_{\pi_4} + q_6^{(m)} V_6^{(m)}, \tag{3.76}$$

where r_{π_5} is an extension of r_{π_4}. The reward term for Policy 4 (r_{π_4}) has been chosen to experiment with constraint (3.47) because it has been deemed the best performing model that balances the performance of all metrics. In Figure 3.19, Policy 5 performs better than the other Policies due to adding the constraint term in the reward that incentivizes the model for distributing RBs fairly amongst the users.

3.7. Conclusion

This Chapter introduces a computation offloading network system model comprising multiple energy-harvesting eMBB user devices. A problem is formulated that aims to reduce the sum of the devices' energy consumption and the sum of the task delays while maximizing the sum of the data rates. Several constraints are introduced to ensure limits on the offloading ratio of the users and the transmission powers, transmission collision prevention, energy consumption versus battery energy level, probability of queueing delay violations, and resource allocation fairness. The problem is converted into a Markov Decision Process for which a state-of-the-art DRL algorithm based on a TD3 actor-critic solution is proposed. Results show that under specific environment parameters, the TD3 model performs the required optimization of maximizing the sum data rates, minimizing both the sum of energy consumption and the sum of the task delays. Hyperparameter tuning shows that the various models always converge to the same level of reward, indicating that the environment involved is simple enough for the model to figure out the optimal actions. However, tuning the hyperparameters leads to faster convergence, and the optimal hyperparameters are found to be an actor learning rate of 10^{-5} or 10^{-6}, 4 hidden layers of both the actor and critic DNNs, 600 and 500 neurons in layer 1 and layer 2 of the actor and critic DNNs, and an Experience Replay Memory size of 10^4 training samples. The TD3 model is also evaluated against its earlier variant, DDPG, which is widely used in prior works on computation offloading. Although both models converge to the same level of reward, the TD3 is found to converge much quicker with less training variation. Ablation studies are conducted using 4 different policies that are trained using different reward

functions. Two constraints are experimented with, and the task arrival rate is varied to explore its impact on the 4 policies against several performance metrics. Policy 4, which balances between offloading and local computing, is found to be the best-performing model across most of the performance metrics.

CHAPTER FOUR

4. TD3-based Channel Resource Slicing for eMBB and URLLC Coexistence

4.1. Introduction

This Chapter investigates a coexistence mechanism for slicing wireless channel resources to accommodate both eMBB and URLLC users on the same spectrum while ensuring their QoS requirements are satisfied. eMBB user service is designed for high data rate applications, and this service requirement is addressed in Chapter 3, where the objective function involves maximizing the sum data rate, $\sum_{d\in\mathcal{D}} R_d^{(m)}$, of eMBB users. Chapter 3 also introduces the 5G New Radio with a flexible frame structure. In this Chapter, the URLLC service is introduced, whose QoS requirements are very low transmission latencies and high reliability, by extending the system architecture, transmission model, and the problem formulation in Chapter 3 to include URLLC users, short packet transmissions over eMBB channel RBs, and a transmission reliability constraint. To satisfy the low latency requirement, URLLC transmissions are localized in time with short Transmission Time Intervals (sTTI) [11].

In practice, URLLC transmissions are random and unpredictable. Thus, it is necessary to dynamically and intelligently allocate resources to URLLC users by interacting with the environment [11]. Since it is established that DRL is best suited for making real-time decisions in non-deterministic problems, it is appropriate for addressing dynamic resource allocation in coexistence networks. The power of TD3 is leveraged for RB allocation while ensuring URLLC QoS constraints are satisfied. In this regard, the state space, action space, and reward function in Chapter 3 are modified to accommodate URLLC user states, actions, and transmission reliability reward.

4.2. Multiplexing eMBB and URLLC traffic in Uplink Transmissions

4.2.1. Downlink Transmissions

The problem of multiplexing eMBB and URLLC traffic in the downlink transmission is solved in [11]. The authors in [11] consider the downlink channel used to transmit data from the gNB to UEs. The gNB schedules all downlink transmissions. Suppose that the

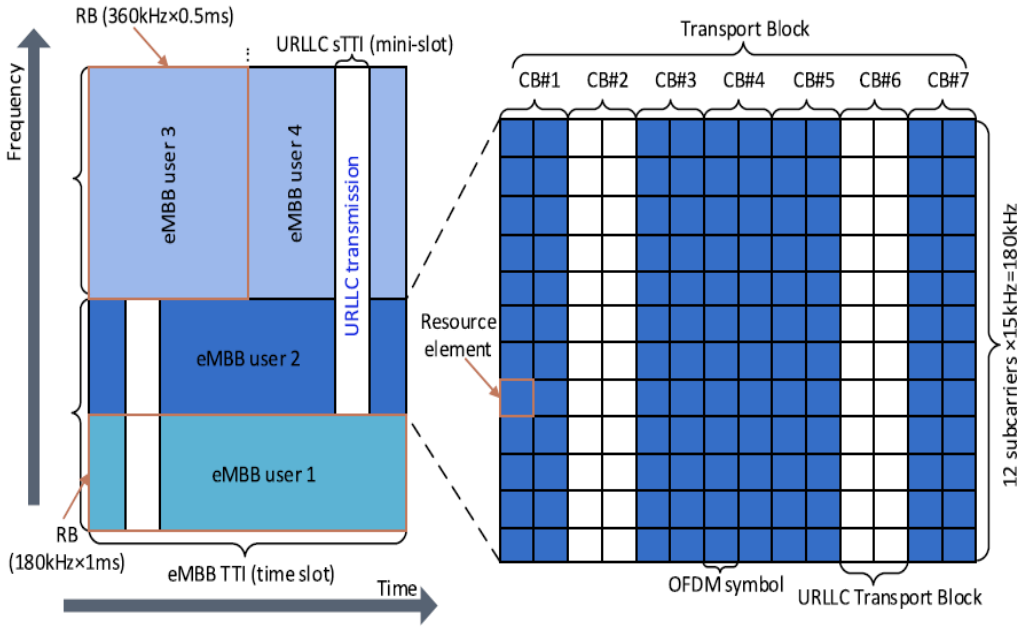

Figure 4. 1: Multiplexing of eMBB and URLLC transmissions – copied from [11].

gNB is allocated the available frequency resources for transmitting data to several eMBB UEs. If an urgent URLLC packet arrives in the middle of the long eMBB slot, the gNB can use the pre-emptive scheduling mechanism (puncturing). With this mechanism, the gNB can reallocate some resources for the transmission of the URLLC packet. At the end of the eMBB slot, the gNB will notify eMBB UEs about the corrupted OFDM symbols and schedule the retransmission of lost eMBB data.

4.2.2. 5G New Radio

The 5G New Radio can support both types of service as outlined in [11]. The frame structure used in 5G NR can assume multiple waveform configurations that depend on the number of frequency subcarriers and the TTI for transmission. A subcarrier spacing of 15 kHz is adopted, which is suitable for outdoor small-cell networks [11]. A RB consists of 12 subcarriers, making a total of 180 kHz bandwidth and a TTI of 1 ms. A RB can also be 360 kHz and 0.5 ms TTI. The RB is divided into 7 smaller Code Blocks (CB), and each CB makes up a small TTI that is utilized for transmitting a single URLLC packet. Figure 4.1 shows an illustration of a single TTI.

4.2.3. Uplink Transmissions

In the Uplink (UL) channel for computation offloading, the multiplexing problem becomes more complicated because eMBB and URLLC UEs use different channel access schemes.

To transmit an eMBB packet in UL, a UE uses a Grant-Based (GB) access scheme which consists of the following steps [60]:

1. The eMBB user sends a scheduling request to the gNB [60].
2. After receiving a grant containing information about the allocated channel resource and transmission parameters, the UE can start data transmission [60].

Because of the long process for obtaining an access grant, the GB scheme cannot satisfy very strict latency requirements for URLLC [60]. To solve this issue, the NR specification introduces a Grant-Free (GF) channel access scheme. The gNB allocates periodic time-frequency resource blocks for the transmission of URLLC packets in advance. For that, using the control channel, the gNB notifies the URLLC UEs about the parameters of the GF transmissions [60]:

1. Location of the allocated RB in the time-frequency resource grid and its periodicity [60],
2. Transmission parameters, such as the modulation and coding scheme, transmission power, and the number of repetitions [60].

When a new URLLC packet arrives at the UE, it can use the first available time-frequency RB allocated for GF transmissions. The GF resources can be shared by multiple UEs, which results in collisions between URLLC transmissions. To reduce collision probability and to provide the high reliability needed for URLLC transmissions, the gNB should allocate enough GF resources such that URLLC transmissions occur very rarely [60]. There are several ways to divide common UL channel resources between GB eMBB transmissions and GF URLLC transmissions. The two schemes described in [60] are the Orthogonal Multiple Access (OMA) and the Non-Orthogonal Multiple Access (NOMA)

- *Orthogonal Multiple Access (OMA):* In this scheme, the whole frequency band is split into two sub-bands: the first one (called the GB sub-band) is used for dynamic scheduling of GB eMBB transmissions, the second one (called the GF sub-band) is allocated only for GF URLLC transmissions [60]. As mentioned before, to provide high reliability, URLLC transmissions shall occur very rarely, which leads to underutilization of the GF sub-band [60]. To overcome this issue, several studies propose to use the Non-Orthogonal Multiple Access (NOMA).
- *Non-Orthogonal Multiple Access (NOMA):* In this scheme, the whole frequency band can be used for both GB eMBB transmissions and GF URLLC transmissions.

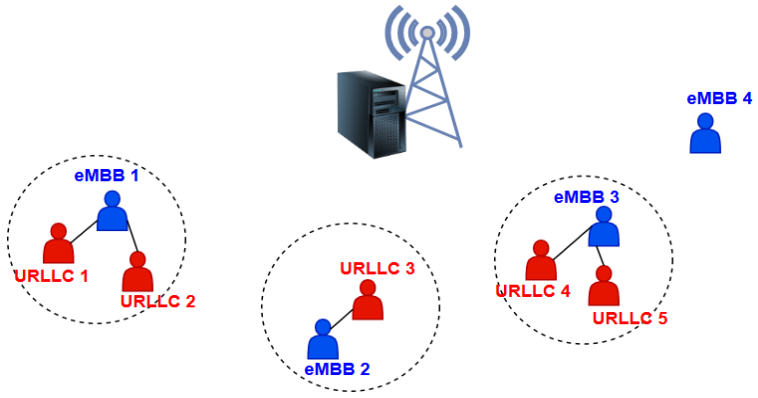

Figure 4. 2: Clustering eMBB users and URLLC users for puncturing.

Since URLLC transmissions and eMBB transmissions overlap in time and frequency, they interfere [60]. To achieve high reliability for URLLC, the power allocated for URLLC transmission is much higher than the power allocated for eMBB transmissions [60].

4.2.4. Proposed Uplink Multiplexing Scheme

4.2.4.1. Implementation

The proposed uplink multiplexing scheme draws from the NOMA scheme, where the whole frequency band is used for both types of transmissions. The URLLC transmissions are superimposed onto the eMBB transmissions. To achieve high reliability, the URLLC users are allocated transmit power that is much higher than that of the eMBB users in their frequency-time RBs. The reliability of the URLLC users is thus controlled by adjusting the power allocation of eMBB users.

The eMBB optimization is done at the gNB. UEs convey all state parameters to the gNB. URLLC users are linked to specific eMBBs as shown in Figure 4.2. URLLC 1 and 2 are linked to eMBB 1 and will only puncture RBs assigned to eMBB 1. The resource allocation actions are broadcast from the gNB to eMBB users, and hence to URLLC users. The gNB will reserve mini-slots for the individual URLLC users linked to a particular eMBB user. This means that each URLLC will be assigned a particular CB to use for the transmission of its packets. If a URLLC packet arrives, it will have to wait until the URLLC user's assigned interval for their CB has arrived. The gNB will know which URLLC has transmitted from the header.

The gNB allocates RBs and transmission power to eMBB users at the beginning of each time slot. Any incoming URLLC traffic is superimposed onto the ongoing eMBB transmissions. This superposition of the URLLC users' transmissions onto eMBB users'

transmissions impacts the data rate of eMBB users. The URLLC reliability is achieved by ensuring that its outage probability is less than a certain threshold, $\Theta_{max}^{(m)}$, where $\Theta_{max}^{(m)}$ is a small positive value ($\Theta_{max}^{(m)} < 1$). Let $L_s^{(m)}[t]$ be the random variable denoting the number of arrived URLLC packets at a mini slot $s \in S = \{1,2,\dots,S\}$ of the time slot t, and $L^{(m)}[t] = \sum_{s \in S} L_s^{(m)}[t]$. Then the URLLC reliability constraint is defined as

$$\Pr\left[\sum_{u \in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t] \leq l^{(m)} L^{(m)}[t]\right] \leq \Theta_{max}^{(m)}. \tag{4.1}$$

The parameter $R_u^{(m)}[t]$ is a slot channel rate for URLLC user u. $\mathcal{U}^{(m)'}$ represents the set of URLLC users that transmit packets in the timeslot t. $l^{(m)}$ is the packet size of a URLLC user, which is a fixed constant. There is a direct relation between the loss data rate of eMBB users within each time slot and the amount of their superimposed resources. Let $w_d^{(m)}$ define the weight of puncturing eMBB user $d \in \mathcal{D}^{(m)}$ and $0 \leq w_d^{(m)} \leq 1 \in$ and $\mathbf{w}^{(m)} = \{w_1^{(m)}, w_2^{(m)}, \dots, w_D^{(m)}\}$. The higher the $w_d^{(m)}$ value, the higher the probability of the eMBB user being punctured by the traffic of URLLC users. Let S be the total number of mini-slots in a time slot and $a_{d,b}^{(m)}[t] \in \{0,1,\dots,S\}$ represents the number of punctured mini-slots from the resource block b of a particular eMBB user at a particular time slot t. The parameter $a_{d,b}^{(m)}[t]$ is related to $w_{d,b}^{(m)}$ by

$$w_{d,b}^{(m)}[t] = \frac{a_{d,b}^{(m)}[t]}{S}. \tag{4.2}$$

Therefore, the eMBB data rate is approximated as

$$r_{d,b}^{(m)}[t] = f_b^{(m)}\left(1 - w_{d,b}^{(m)}[t]\right)\log_2\left(1 + \frac{\alpha_{d,b}^{(m)}[t] g_{d,b}^{(m)}[t] P_d^{(m)}[t]}{N_o f_b^{(m)}}\right). \tag{4.3}$$

The achieved URLLC data rate taken from [11] is approximated as

$$R_{u,b}^{(m)}[t] = f_b^{(m)}\left(\frac{1}{S}\right) \log_2\left(1+\frac{\propto_{u,b}^{(m)}[t] g_{u,b}^{(m)}[t] P_u^{(m)}[t]}{\sum_{d=1}^{D}\sigma_{d,b}^{(m)}[t] \propto_{d,b}^{(m)}[t] g_{d,b}^{(m)}[t] P_{d,b}^{(m)}[t] + N_o f_b^{(m)}}\right) - \sqrt{\frac{O_{u,b}^{(m)}[t]}{e_{u,b}^{(m)}[t]}}\ Q^{-1}(v), \tag{4.4}$$

where $e_{u,b}^{(m)}$ is the length of the CB in symbols (i.e., number of symbols in a mini-slot), $Q^{-1}(\cdot)$ is the inverse of the Gaussian Q-function, $v > 0$ is the transmission error probability, and $O_{u,b}^{(m)}[t]$ represents the characteristic of the channel called the channel dispersion, i.e. $O_{u,b}^{(m)}[t]$ determines the stochastic variability of the channel of the user u at timeslot t relative to a deterministic channel with the same capacity, and is given by

$$O_{u,b}^{(m)}[t] = 1 - \frac{1}{\left(1+\frac{\propto_{u,b}^{(m)}[t] g_{u,b}^{(m)}[t] P_u^{(m)}[t]}{N_o f_b^{(m)}}\right)^2}. \tag{4.5}$$

4.2.4.2. URLLC Users Outage Probability

A closed-form expression of the outage probability in (4.1) is derived and used in the objective function. Each URLLC user's packet arrival is modeled using a Bernoulli distribution. In any timeslot, a URLLC user either has a packet to transmit or not. In this regard, a Binomial distribution for all URLLC users' packet arrivals is considered. The parameter $L^{(m)}[t]$, which represents the total number of URLLC users' packets arriving in the slot t, is Binomially distributed with U and p, where U (number of trials) is the total number of URLLC users and p is the probability of packet transmission of a URLLC user. $L^{(m)}[t] \sim Binom(U, p)$, hence, the CDF of the Binomial distribution is used to turn (4.1) into a deterministic form as follows:

$$\Pr\left[\sum_{u\in \mathcal{U}^{(m)}{}'} T_s R_u^{(m)}[t] \le l^{(m)} L^{(m)}[t]\right] \tag{4.6a}$$

$$= \Pr\left[\frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}} \leq L^{(m)}[t]\right] \tag{4.6b}$$

$$= 1 - \Pr\left[\frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}} \geq L^{(m)}[t]\right] \tag{4.6c}$$

$$= 1 - \Pr\left[L^{(m)}[t] \leq \frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}}\right] \tag{4.6d}$$

$$= 1 - F_L^{(m)}\left(\frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}}\right) \tag{4.6e}$$

Here, $F_L^{(m)}$ is the cumulative distribution function (CDF) of the random variable $L^{(m)}$ and is calculated as done in [69] and given by

$$F_L^{(m)} = \sum_{i=0}^{\lfloor k \rfloor} \binom{U}{i} p^i (1-p)^{U-i}, \tag{4.7}$$

where $k = \frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}}$ and $\lfloor k \rfloor$ is the floor under k i.e, the greatest integer less than or equal to k [69]. Thus, (4.1) is rewritten as follows:

$$\Pr\left[\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t] \leq l^{(m)} L^{(m)}[t]\right] \leq \Theta_{max}^{(m)}, \tag{4.6f}$$

$$1 - F_L^{(m)}\left(\frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}}\right) \leq \Theta_{max}^{(m)}, \tag{4.6g}$$

$$F_L^{(m)}\left(\frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}}\right) \geq 1 - \Theta_{max}^{(m)}, \tag{4.6h}$$

$$\frac{\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t]}{l^{(m)}} \geq F_L^{-1^{(m)}}\left(1-\Theta_{max}^{(m)}\right), \tag{4.6i}$$

$$\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t] \geq l^{(m)} F_L^{-1^{(m)}}\left(1-\Theta_{max}^{(m)}\right), \tag{4.6j}$$

$$\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t] - l^{(m)} F_L^{-1^{(m)}}\left(1-\Theta_{max}^{(m)}\right) \geq 0. \tag{4.6k}$$

The URLLC users' QoS requirement is assimilated into the objective function formulated in Chapter 3 by incorporating (4.6k) as a constraint into the objective function as follows:

- The eMBB RB data rate equation changes from (3.1) to (4.3).
- The new constraint (4.6k) is added. Thus, P1 now has constraint (3.48) in the form of (4.6k):

$$\mathbf{P1}: \max_{\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t],q} \mathcal{R}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t]) - q^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t],\boldsymbol{\sigma}_\mathbf{m}[t],\mathbf{x_m}[t])$$

Subject to (3.39) – (3.47), (4.6k)

- The URLLC channel gains are added as states in the original state space. Thus, the small-scale ($\boldsymbol{\alpha}_\mathbf{m}[t]$) and large-scale ($\mathbf{g_m}[t]$) channel gain vectors in the state space $s[t]$ defined in Chapter 3 are changed as follows:
  - $\boldsymbol{\alpha}_\mathbf{m}[t] = \{\alpha_1^{(m)}[t], \dots, \alpha_D^{(m)}[t], \alpha_1^{(m)}[t], \dots, \alpha_U^{(m)}[t]\}$ are the instant small-scale channel gains between the UEs and the gNB.
  - $\mathbf{g_m}[t] = \{g_1^{(m)}[t], \dots, g_D^{(m)}[t], g_1^{(m)}[t], \dots, g_U^{(m)}[t]\}$ are the instant large-scale channel gains between the UEs and the gNB.
- Another reward term is added to the reward equation of (3.66)

$$r[t] = r[t] + q_7^{(m)} V_7^{(m)}, \tag{4.8}$$

where $V_7^{(m)}$ is given by

$$V_7^{(m)} = F\left(\sum_{u\in \mathcal{U}^{(m)'}} T_s R_u^{(m)}[t] - l^{(m)} F_L^{-1^{(m)}}\left(1-\Theta_{max}^{(m)}\right)\right). \tag{4.9}$$

4.3. Results and Discussion

In this Section, the performance of the TD3 model is assessed against several performance metrics. In attempting to transmit short packets, URLLC users may fail to transmit due to the following two reasons:

1. The achievable URLLC user data rate is less than the size of the packet that needs to be transmitted.
2. The linked eMBB user of the URLLC user that has a packet to transmit may not be allocated any RB for transmission.

If either of these scenarios occurs, the URLLC packet is dropped. Like the testing results of Chapter 3, the performance of the TD3 model is tested against three different Policies, which are trained using different reward functions. The reward functions of the three Policies are shown in Table 4.2. The environment setup is the same as the one in Chapter 3, except URLLC users are now involved, and multiplexing occurs in each timeslot. Table 4.1 shows the additional URLLC environment simulation parameters. The URLLC users' small-scale and large-scale channel gains are modeled in the same way as those of eMBB users. URLLC users are linked to eMBB users with whom they have the shortest distance.

Table 4. 1: Multiplexing simulation parameters.

Parameter	**Value**
Transmission Error Probability (v)	10^{-5}
Length of the CB in symbols (Number of symbols per mini slot) ($e_{u,b}^{(m)}[t]$)	2
URLLC Task Size ($l^{(m)}$)	32 bytes
Maximum Outage Probability ($\theta_{max}^{(m)}$)	0.2
URLLC User Transmit Power ($P_u^{(m)}$)	40 dBm
Probability of URLLC users task arrival (p)	0.5
Number of URLLC users (U)	16

Table 4. 2: Different policies and their respective reward functions.

Policy	Reward Function
Policy 4 (π_4^{mul})	$\mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t])$ $+ q_2^{(m)}\sum_{d\in\mathcal{D}} V_{2,d}^{(m)} + q_3^{(m)}\sum_{d\in\mathcal{D}} V_{3,d}^{(m)}$
Policy 5 (π_5^{mul})	$\mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t])$ $+ q_2^{(m)}\sum_{d\in\mathcal{D}} V_{2,d}^{(m)} + q_3^{(m)}\sum_{d\in\mathcal{D}} V_{3,d}^{(m)} + q_6^{(m)}V_6^{(m)}$
Policy 5 (π_6^{mul})	$\mathcal{R}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t]) - q_0^{(m)}\mathcal{E}_d^{(m)}(\mathbf{P_m}[t], \boldsymbol{\sigma}_\mathbf{m}[t], \mathbf{x_m}[t])$ $+ q_2^{(m)}\sum_{d\in\mathcal{D}} V_{2,d}^{(m)} + q_3^{(m)}\sum_{d\in\mathcal{D}} V_{3,d}^{(m)} + q_7^{(m)}V_7^{(m)}$

Figure 4.3 shows the plots for eMBB sum data rate and URLLC sum data rate as the URLLC task arrival probability is changed from 0.1 to 0.5. The eMBB plot shows that the eMBB sum data rate decreases as the URLLC task arrival probability increases. This trend is observed across all Policies. Policy 4 achieves the highest eMBB sum data rate, followed by Policy 5, and Policy 6 has the lowest sum data rate. Increasing the task arrival probability increases the number of URLLC packets that arrive in each time slot, i.e. $L^{(m)}[t]$ increases as p increases. As $L^{(m)}[t]$ increases, this results in more channel resources, specifically Code Blocks, being punctured by the URLLC traffic, reducing the available bandwidth for the eMBB users. The decline in eMBB sum data rate from a p of 0.1 to 0.5 is gradual, indicating that the increase in URLLC traffic is not big enough to impact the eMBB

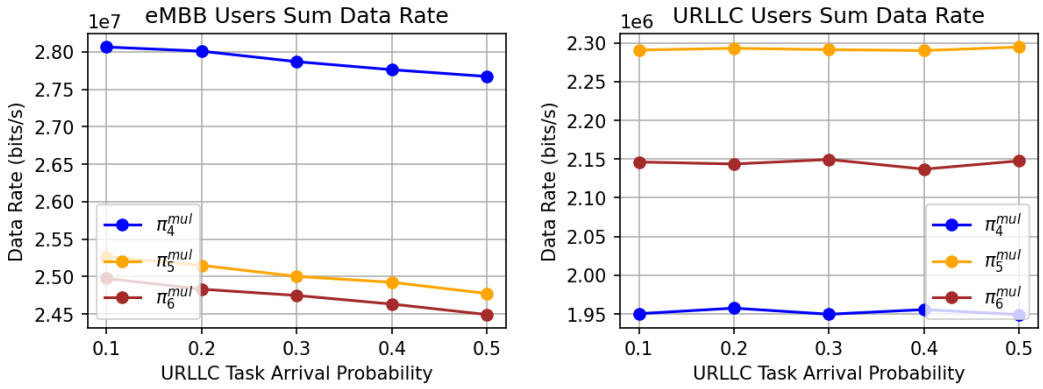

Figure 4. 3: eMBB sum data rate and URLLC sum data rate as the URLLC task arrival probability increases

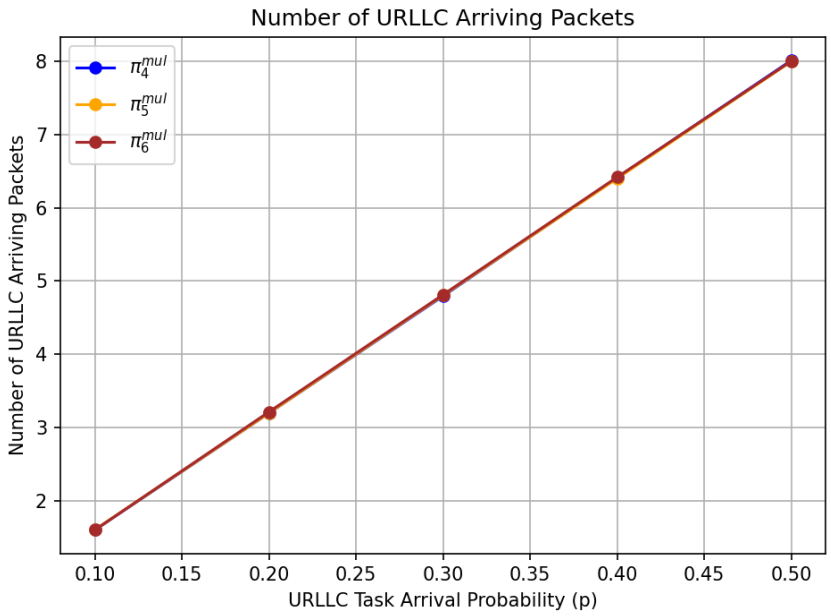

Figure 4. 4: Average number of URLLC packets that arrive in a timeslot as the URLLC probability of task arrival increases**.**

resource allocation significantly. The URLLC sum data rate remains relatively stable across all the task arrival probabilities with minor variations. Policy 5 achieves the highest URLLC sum data rate, followed by Policy 6 and then Policy 4. Policy 4's low URLLC sum data rate indicates less puncturing of eMBB channel resources. Indeed, the eMBB sum data rate plot supports this suggestion since the sum eMBB data for Policy 4 is the highest. Figure 4.4 and Figure 4.5 give plots that can explain these differences.

Figure 4.4 shows the average number of URLLC packets that arrive in a timeslot to be transmitted. The task arrival probability is set to be the same for all users. The number of packets arriving is an average of over 50000 timesteps of a single inference simulation run.

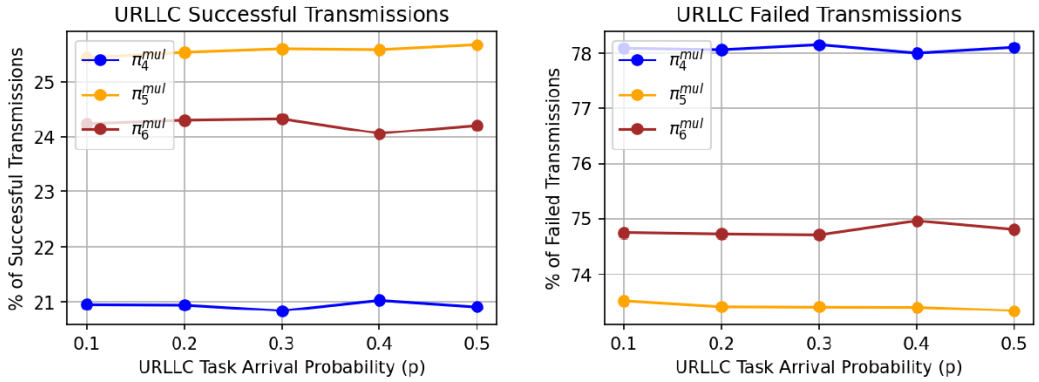

Figure 4. 5: Percentage of successful and failed URLLC transmissions as the URLLC users' task arrival probability increases.

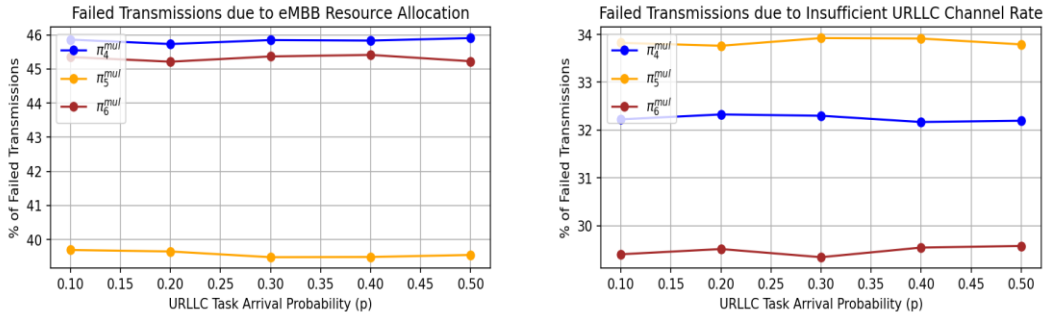

Figure 4. 6: Percentage of failed URLLC transmissions due to eMBB resource allocation and insufficient URLLC channel rate as the URLLC users' task arrival probability increases.

The graphs for all Policies overlap and show a linear trend. At this point, URLLC users only produce packets that need to be transmitted, and this is not affected by the Policies. The task arrival follows a Bernoulli distribution. At p of 0.5, an average of 8 packets are produced, meaning that 8 users out of a total of 16 URLLC users (50%) produce packets that need to be transmitted, which correlates with a p of 0.5. After URLLC users produce packets to be transmitted, the URLLC users then puncture the Code Blocks of the eMBB user that they are linked with to transmit their short packets. If the eMBB user has been allocated RBs in the same time slot that a URLLC user wants to transmit their packet, and when the channel rate between the URLLC user and the gNB is sufficient to carry the packet, i.e $R_{u,b}^{(m)}[t] \geq l^{(m)}$, then the URLLC user can puncture and transmit. If, however, the linked eMBB user is not allocated any RB or if the URLLC user's channel rate is less than the packet size, then the packet is dropped, and transmission fails. Successful transmission, therefore, depends on eMBB resource allocation policies and the signal link between the URLLc users and the gNB.

Figure 4.5 shows that the percentage of successful and failed transmissions remains consistent with minor changes for Policies 4, 5, and 6 across all task arrival probabilities, indicating that the resource allocation and the signal links between the URLLC users and the gNB also remain consistent throughout the simulation runs for all the Policies. The minor deviations are due to the randomness introduced by small-scale and large-scale channel gains. Policy 4 registers the highest number of failed transmissions (about 78% of packets arriving), and this tallies with the plot in Figure 4.3, which shows that the lowest URLLC sum data rate of the 3 Policies comes from Policy 4. The URLLC sum data rate is therefore positively correlated with the percentage of successful transmissions because successful transmissions mean that either the eMBB resource allocation is effective enough

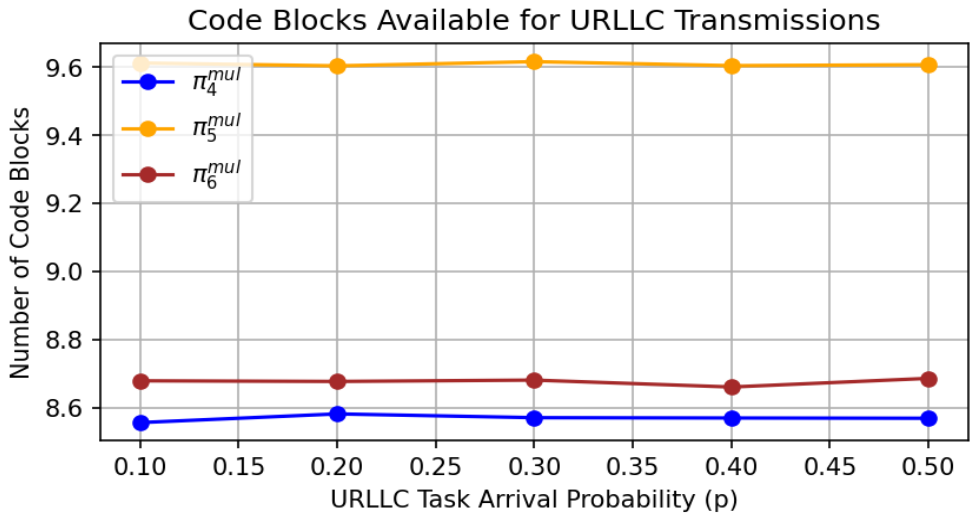

Figure 4. 7: Average number of code blocks available for URLLC transmissions as the URLLC users' task arrival probability increases.

to meet the demand for URLLC transmissions or the signal links between the URLLC users and the gNB are good enough to carry the packets.

An investigation is carried out to find out the contributions to the failure of URLLC transmissions of both the eMBB resource allocation and transmission signal links of URLLC users. Figure 4.6 shows the percentage contributions of eMBB resource allocation and the URLLC user channel rate to failed transmissions. The figure shows that eMBB resource allocation contributes more than the URLLC channel rate for all Policies. Furthermore, when it comes to failed transmissions due to URLLC channel rate, the difference between the Policy with the highest contribution and the Policy with the smallest contribution (Policy 5 and Policy 6 respectively) is about 5% compared to the 7% difference of failed transmissions due to eMBB resource allocation between Policy 4 and Policy 5. This suggests that the eMBB resource allocation is more critical to the success of URLLC transmissions. Further investigations also cement this conclusion by showing that the eMBB resource allocation contributes to the amount of channel rate available for URLLC transmissions.

Figure 4.7 shows the average number of eMBB Code Blocks that are exposed to URLLC users to enable transmissions of their small packets. Figure 4.7 shows an inverse correlation with the plot of failed transmissions due to eMBB resource allocation in Figure 4.6, thus providing an explanation that the higher the number of Code Blocks available for URLLC transmissions, the lower the URLLC transmission failure rate, and the inverse is true. To explain why Policy 4 has the least number of CBs available for URLLC

transmission, followed by Policy 6 and then Policy 5, the Jairness Fairness Index (JFI) is introduced, which measures the fairness of the allocation of channel resources amongst the users [70]. This index is a number between 0 and 1, with 0 indicating the lowest fairness and 1 indicating the highest fairness of allocation of resources [70]. A value of 1 would mean that every user in the system has the same share of resources. The JFI is measured over the 50000-timestep simulation run for all the Policies, and Figure 4.8 shows the results.

Figure 4.8 shows a horizontal linear trend across all task arrival probabilities for all Policies, and this shows consistency in the fairness of the allocation of resources among eMBB users as the URLLC task arrival probability changes. Policy 5 has the highest JFI, followed by Policy 6, then Policy 4. The order of fairness between the Policies directly contrasts with Figure 4.7, showing a positive correlation between the JFI and the number of CBs available for URLLC short packet transmissions. The clustering that happens to link URLLC users to eMBB users ultimately requires that each eMBB user be allocated channel resources for URLLC users to be able to transmit their packets. If one or more eMBB users are not allocated RBs for transmission, the corresponding URLLC users linked to those eMBB users will consequently not be exposed to CBs for transmission, and this increases the transmission failure rate. It follows then that increasing the equality of distribution of channel resources among eMBB users increases the likelihood of exposure of CBs available for URLLC transmissions.

The reason for the differences in the JFIs of the Policies is explained by the reward construction of the individual Policies shown in Table 4.1. Policy 5 has the highest JFI due

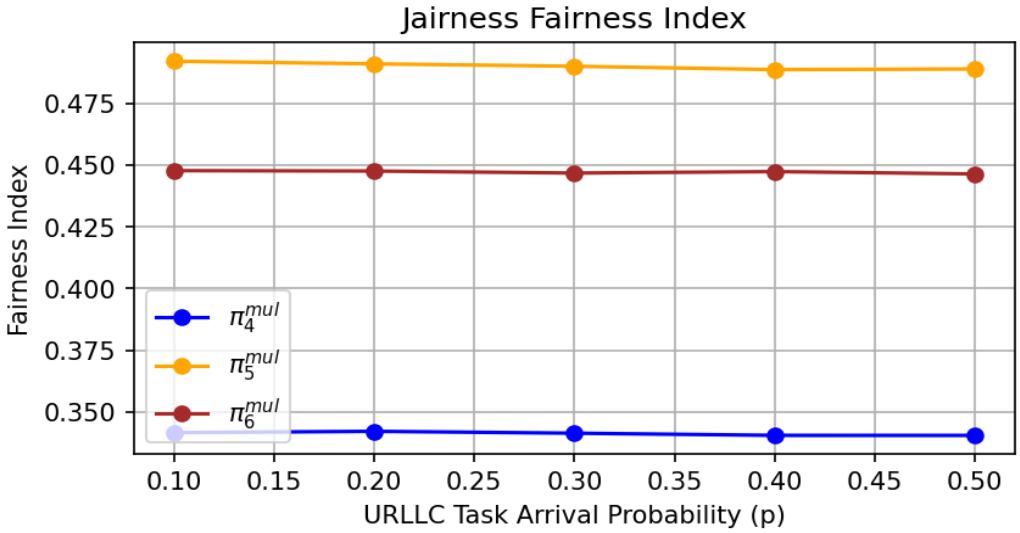

Figure 4. 8: Jairness fairness index for Policies 4, 5, and 6 across all task arrival probabilities.

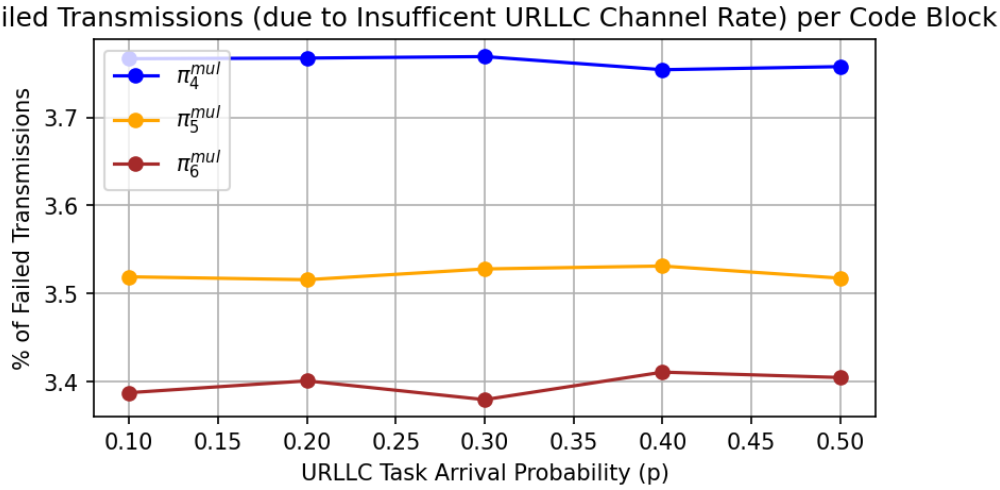

Figure 4. 9: Percentage per CB of failed URLLC transmissions due to insufficient URLLC channel rate.

to the presence of the $q_6^{(m)}V_6^{(m)}$ term in the reward that is defined in Chapter 3. This term incentivizes the model during training to allocate a portion of the available spectrum (at least R^{min}) to each eMBB user in the system. The model is penalized if this requirement is not met. Policy 4 has the lowest JFI due to the absence of this term in its reward function. Policy 6 performs better than Policy 4 and worse than Policy 5 in terms of JFI. Although the reward function for Policy 6 doesn't have the $q_6^{(m)}V_6^{(m)}$ term, it has the $q_7^{(m)}V_7^{(m)}$ as shown in Table 4.2, which is related to the constraint (4.6k) that aims to reduce the outage probability. The parameter $V_7^{(m)}$, as shown in its definition in (4.9), provides an incentive for the model to reduce the outage probability by increasing the URLLC users' sum data rate. During training, the model, therefore, attempts to distribute channel resources fairly among eMBB users to increase the exposure of CBs to URLLC users, thus increasing their sum data rate. The incentive to distribute channel resources fairly is brought about by $q_7^{(m)}V_7^{(m)}$ in Policy 6's reward is less than the incentive given by $q_6^{(m)}V_6^{(m)}$ in Policy 5's reward, and this explains why Policy 6 performs slightly worse than Policy 5 in terms of JFI.

Figure 4.9 shows the percentage per CB of failed URLLC transmissions due to inadequate URLLC Channel Rate. The percentage difference between the Policies is small, registering about a 0.35% difference between the Policy with the highest percentage (Policy 4) and the Policy with the smallest percentage (Policy 6). This once again reflects the small deviation amongst the Policies in the percentage of failed URLLC transmissions due to insufficient channel rate in Figure 4.6. Despite the small deviation in percentage, it is still

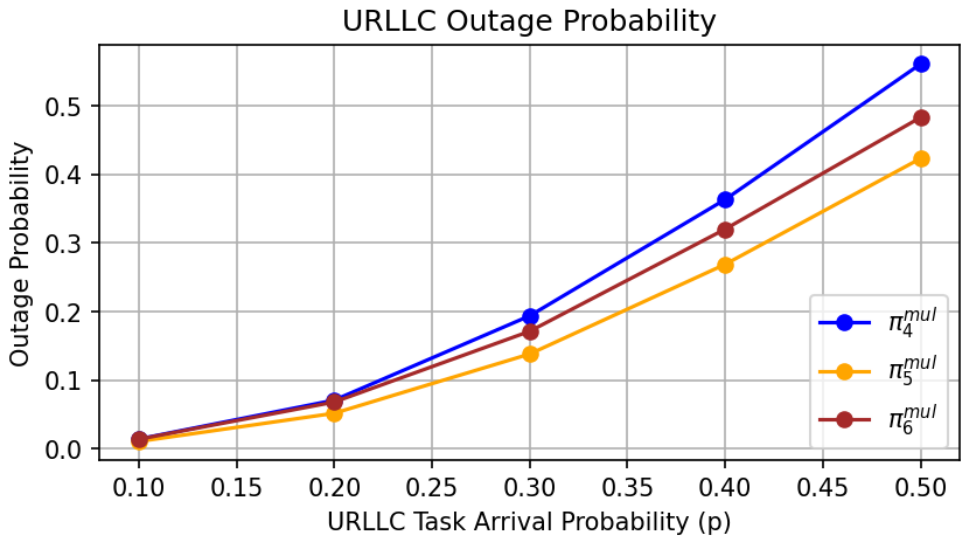

Figure 4. 10: URLLC outage probabilities of Policies 4, 5, and 6 as the URLLC task arrival probability increases.

noteworthy to distinguish the efficacy of successful transmissions for each CB available for URLLC transmission. In terms of efficacy, the policies are ranked as follows: Policy 6 shows the highest efficiency, followed by Policy 5, and then Policy 4.

Figure 4.10 shows the outage probabilities of the 3 Policies as the URLLC task arrival probability increases. The outage probability is calculated using the CDF of the binomial distribution as worked out in (4.6e). The general trend shows that as the URLLC task arrival rate increases, the outage probability increases for all Policies. This trend is a result of an increased likelihood of transmission failure due to eMBB resource allocation and insufficient URLLC channel rate as the URLLC traffic increases. Policy 3 has the highest outage probability across all arrival rates. It reaches over 0.5 outage probability at a p of 0.5. Policy 5 performs the best, maintaining the lowest outage probabilities at all levels. Policy 6 performs slightly worse than Policy 5 but better than Policy 3. The outage probability plot reflects the plot in Figure 4.5, which shows the percentage of failed transmissions of the different Policies as the URLLC task arrival probability increases, and so the reasons for the differences in the levels of outage probabilities between the Policies are the same.

4.4. Conclusion

This Chapter investigates a coexistence mechanism for slicing wireless channel resources to accommodate both eMBB and URLLC users on the same spectrum. The system model and problem formulation in Chapter 3 are extended to include URLLC users and the

URLLC reliability constraint. Different from prior works on multi-service computation offloading that only consider downlink transmissions, a resource-slicing mechanism for uplink transmissions using the 5G flexible frame structure is developed. URLLC users are clustered into groups around an eMBB user they are in close proximity with. The TD3 model is trained and used for RB allocation for eMBB users, and URLLC users can perform transmissions by puncturing Code Blocks from the RBs that are allocated to the eMBB user they are linked to. The performance of this slicing scheme is investigated through a performance metric that measures the average rate of successful URLLC transmissions, and this is tested against three Policies that have different reward functions used in training. Results show that on average, about 26% of URLLC tasks that are produced in a timeslot through a Bernoulli process from all the URLLC users are successfully transmitted by the best-performing Policy, Policy 5. Further investigations show that there are two causes of the failure of URLLC packet transmissions, namely, failure due to eMBB resource allocation and failure due to insufficient URLLC channel rate. From the best-performing model, Policy 5, about 40% of URLLC packets fail due to eMBB resource allocation, and about 34% fail due to insufficient channel rate. The failure due to eMBB resource allocation is attributed to the fairness of the allocation of channel resources among the eMBB users. Policy 5 is found to have the smallest failure rate because it has the highest fairness that has been calculated using the Jairness Fairness Index. This conclusion suggests that improving the fairness of the allocation of resources can reduce the URLLC users' transmission failure rate for this multiplexing scheme. The Policy's fairness index is controlled by how the reward is shaped to cater for the fairness in RB allocation.

CHAPTER FIVE

5. Conclusion and Future Work

This Chapter provides a detailed overview of the work that has been documented in this dissertation and the key findings resulting from the simulations that have been carried out. The Chapter also outlines potential improvements to the work and/or additions and suggestions for future research.

5.1. Conclusion

Chapter 1 of the thesis introduces the foundation of the work in this dissertation, which lays out the need to develop an energy-efficient task offloading scheme that utilizes machine learning on the edge of networks for multi-service users. The motivation for this work is outlined by addressing several gaps in the literature associated with computation offloading. Based on these gaps, the dissertation's research contributions include modelling an MEC network that incorporates energy harvesting in multi-user and multi-service computation offloading using the 5G New Radio, formulating a joint task offloading and scheduling of eMBB and URLLC users' problem minimizes energy consumption and task queueing delay as well as maximizing the achievable uplink data rate of eMBB users subject to latency and reliability constraints of URLLC users, and proposing a TD3-based deep reinforcement learning solution. Chapter 2 then cements the foundation of this research by providing a comprehensive literature review that aims to give a detailed understanding of core concepts, system architectures, system model parameters, optimization problems, and solution frameworks that are used in the existing body of literature on computation offloading. Chapter 3 provides the work that is done for the proposed system model that comprises energy-harvesting eMBB users in a computation offloading setting that involves a single gNB. Chapter 3 also differentiates this dissertation's work from the literature by making use of the 5G flexible frame structure for uplink transmissions and using a state-of-the-art deep reinforcement learning solution for an optimization problem that jointly considers energy consumption, task queueing delay, and throughput. Results for Chapter 3 involve hyperparameter tuning, where optimal hyperparameters for TD3 training are found to improve the convergence rate of the model. Some of the hyperparameters are, an actor learning rate of 10^{-5} or 10^{-6}, 4 DNN hidden layers of both the actor and critics, and an experience replay memory size of 10^{4} training samples.

Results also show that the TD3 performs better than the widely used deep reinforcement learning model in literature in terms of the convergence rate. Chapter 3 results also validate TD3 model performance by testing it using 4 different Policies against several key metrics. The models are trained using different reward functions, and Policy 4 is found to be the best performing due to its ability to balance between offloading and local computing, thus satisfying most constraints that exhibit contradicting patterns caused by either offloading or local computing. Chapter 4 extends the system model and problem formulated in Chapter 3 to include URLLC users and a URLLC transmission reliability constraint. A multiplexing scheme is developed that differentiates this dissertation's work from the literature by using the 5G flexible frame structure for both eMBB and URLLC uplink transmissions. The multiplexing scheme is tested against several key metrics, and once again, using 3 different Policies that are trained using different reward functions. The best-performing model, Policy 5, is found to have an average URLLC transmission success rate of about 26%. Through several plots and derivations, it is found that the key factor to the transmission success rate is the fairness of allocation of channel resources among eMBB users, which Policy 5 does better than the other Policies.

5.2. Future Work

Based on the insights gained from this work, several improvements could be made, and more research work could be explored. The following are potential improvements and work that can be conducted in the future.

5.2.1. Future Work and Improvements

- **Fairness Improvement:** Chapter 4 concludes that the URLLC users' transmission success rate could be increased by improving the fairness of the allocation of resources. Work can be carried out to explore how fairness can be improved, and investigations done to find out how much the transmission success rate can be increased by.
- **Energy Harvesting Results**: The work in Chapters 3 or 4 does not investigate any contributions made by energy harvesting that was included as one of the key technologies for improving energy efficiency. Simulations should be carried out to find the impact of energy harvesting on model performance, especially against energy efficiency metrics.
- **Multi-agent Deep Reinforcement Learning:** During simulations for Chapters 3 and 4, it was noted that the TD3 model tends to perform similar offloading and transmission power actions across all users. This is why a single offloading ratio

was persistent across all task arrival rates in Chapter 3. Having a single actor DNN performing actions for all users in the system may not produce an optimal solution because it performs the same gradient ascent across all actor output DNN neurons. The actions are, therefore, aggregated due to a similarly aggregated system reward function. A multi-agent approach could help solve this problem, where different actors for each user are used, although this approach could be computationally intensive.

- **Investigate Different Sampling Mechanisms:** The sampling used in this work is a random-based sampling where training samples were extracted randomly from the experience replay memory. A suggestion is to test different sampling methods, such as importance sampling, stratified sampling, bootstrap sampling, and cluster sampling, which could improve the TD3 learning.
- **User Mobility:** Real-life telecommunication systems involve mobile users. A suggestion is to improve the system model by adding a user mobility model.
- **User Association:** User association is important in network systems, especially when energy efficiency is concerned. For a system that has multiple gNBs, users need to be optimally connected to the gNB that may offer low latency, low offloading energy consumption, and high data rate for offloading. To build upon the goal of using machine learning, for future work, Federated Learning is leveraged to achieve high energy efficiency. Future work will involve formulating an appropriate system model, problem, solution, and conducting simulation experiments to establish the effectiveness of this approach.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017, doi: 10.1109/COMST.2017.2745201.

[2] K. Han, D. Liu, Y. Chen and K. K. Chai, "Energy-Efficient User Association in HetNets: An Evolutionary Game Approach," 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, Sydney, NSW, Australia, 2014, pp. 648-653, doi: 10.1109/BDCloud.2014.18.

[3] A. Bandi, "A Review Towards AI Empowered 6G Communication Requirements, Applications, and Technologies in Mobile Edge Computing," 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2022, pp. 12-17, doi: 10.1109/ICCMC53470.2022.9754049.

[4] X. Huang, S. Leng, S. Maharjan and Y. Zhang, "Multi-Agent Deep Reinforcement Learning for Computation Offloading and Interference Coordination in Small Cell Networks," in IEEE Transactions on Vehicular Technology, vol. 70, no. 9, pp. 9282-9293, Sept. 2021, doi: 10.1109/TVT.2021.3096928.

[5] Y. Mao, J. Zhang and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices," in IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590-3605, Dec. 2016, doi: 10.1109/JSAC.2016.2611964.

[6] R. Dong, C. She, W. Hardjawana, Y. Li and B. Vucetic, "Deep Learning for Hybrid 5G Services in Mobile Edge Computing Systems: Learn From a Digital Twin," in IEEE Transactions on Wireless Communications, vol. 18, no. 10, pp. 4692-4707, Oct. 2019, doi: 10.1109/TWC.2019.2927312.

[7] T. Mahn, M. Wirth and A. Klein, "Game Theoretic Algorithm for Energy Efficient Mobile Edge Computing with Multiple Access Points," 2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2020, pp. 31-38, doi: 10.1109/MobileCloud48802.2020.00013.

[8] R. Dong, C. She, W. Hardjawana, Y. Li and B. Vucetic, "Deep Learning for Hybrid 5G Services in Mobile Edge Computing Systems: Learn From a Digital Twin," in IEEE Transactions on Wireless Communications, vol. 18, no. 10, pp. 4692-4707, Oct. 2019, doi: 10.1109/TWC.2019.2927312.

[9] T. Mahn, M. Wirth and A. Klein, "Game Theoretic Algorithm for Energy Efficient Mobile Edge Computing with Multiple Access Points," 2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2020, pp. 31-38, doi: 10.1109/MobileCloud48802.2020.00013.

[10] N. Ashraf, S. A. Sheikh, S. A. Khan, I. Shayea and M. Jalal, "Simultaneous Wireless Information and Power Transfer With Cooperative Relaying for Next-Generation Wireless Networks: A Review," in IEEE Access, vol. 9, pp. 71482-71504, 2021, doi: 10.1109/ACCESS.2021.3078703.

[11] M. Alsenwi, N. H. Tran, M. Bennis, S. R. Pandey, A. K. Bairagi and C. S. Hong, "Intelligent Resource Slicing for eMBB and URLLC Coexistence in 5G and Beyond: A Deep Reinforcement Learning Based Approach," in IEEE Transactions on Wireless Communications, vol. 20, no. 7, pp. 4585-4600, July 2021, doi: 10.1109/TWC.2021.3060514.

[12] Yun, Y. Goh, W. Yoo and J. -M. Chung, "5G Multi-RAT URLLC and eMBB Dynamic Task Offloading With MEC Resource Allocation Using Distributed Deep Reinforcement Learning," in IEEE Internet of Things Journal, vol. 9, no. 20, pp. 20733-20749, 15 Oct.15, 2022, doi: 10.1109/JIOT.2022.3177425

[13] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," in IEEE Communications Surveys & Tutorials, vol.

19, no. 3, pp. 1628-1656, thirdquarter 2017, doi: 10.1109/COMST.2017.2682318.

[14] A. Padhy, S. Joshi, S. Bitragunta, V. Chamola and B. Sikdar, "A Survey of Energy and Spectrum Harvesting Technologies and Protocols for Next Generation Wireless Networks," in IEEE Access, vol. 9, pp. 1737-1769, 2021, doi: 10.1109/ACCESS.2020.3046770.

[15] M. Alsenwi, N. H. Tran, M. Bennis, A. Kumar Bairagi and C. S. Hong, "eMBB-URLLC Resource Slicing: A Risk-Sensitive Approach," in IEEE Communications Letters, vol. 23, no. 4, pp. 740-743, April 2019, doi: 10.1109/LCOMM.2019.2900044.

[16] J. Plachy, Z. Becvar, E. C. Strinati and N. d. Pietro, "Dynamic Allocation of Computing and Communication Resources in Multi-Access Edge Computing for Mobile Users," in IEEE Transactions on Network and Service Management, vol. 18, no. 2, pp. 2089-2106, June 2021, doi: 10.1109/TNSM.2021.3072433.

[17] N. Ashraf, S. A. Sheikh, S. A. Khan, I. Shayea and M. Jalal, "Simultaneous Wireless Information and Power Transfer With Cooperative Relaying for Next-Generation Wireless Networks: A Review," in IEEE Access, vol. 9, pp. 71482-71504, 2021, doi: 10.1109/ACCESS.2021.3078703.

[18] Y. Mao, J. Zhang and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices," in IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590-3605, Dec. 2016, doi: 10.1109/JSAC.2016.2611964.

[19] X. Huang, S. Leng, S. Maharjan and Y. Zhang, "Multi-Agent Deep Reinforcement Learning for Computation Offloading and Interference Coordination in Small Cell Networks," in IEEE Transactions on Vehicular Technology, vol. 70, no. 9, pp. 9282-9293, Sept. 2021, doi: 10.1109/TVT.2021.3096928.

[20] Y. Du, J. Li, L. Shi, T. Liu, F. Shu and Z. Han, "Two-Tier Matching Game in Small Cell Networks for Mobile Edge Computing," in IEEE Transactions on Services Computing, vol. 15, no. 1, pp. 254-265, 1 Jan.-Feb. 2022, doi: 10.1109/TSC.2019.2937777.

[21] S. B. Prathiba, G. Raja, S. Anbalagan, K. Dev, S. Gurumoorthy and A. P. Sankaran, "Federated Learning Empowered Computation Offloading and Resource Management in 6G-V2X," in IEEE Transactions on Network Science and Engineering, doi: 10.1109/TNSE.2021.3103124.

[22] C. Pan et al., "Asynchronous Federated Deep Reinforcement Learning-Based URLLC-Aware Computation Offloading in Space-Assisted Vehicular Networks," in IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2022.3150756.

[23] N. Kammoun, R. Abassi and S. Guemara, "Towards a New Clustering Algorithm based on Trust Management and Edge Computing for IoT," 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), 2019, pp. 1570-1575, doi: 10.1109/IWCMC.2019.8766492.

[24] X. Liu, J. Yu, J. Wang and Y. Gao, "Resource Allocation With Edge Computing in IoT Networks via Machine Learning," in IEEE Internet of Things Journal, vol. 7, no. 4, pp. 3415-3426, April 2020, doi: 10.1109/JIOT.2020.2970110.

[25] Q. Liu, L. Cheng, T. Ozcelebi, J. Murphy and J. Lukkien, "Deep Reinforcement Learning for IoT Network Dynamic Clustering in Edge Computing," 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2019, pp. 600-603, doi: 10.1109/CCGRID.2019.00077.

[26] C. S. M. Babou et al., "Hierarchical Load Balancing and Clustering Technique for Home Edge Computing," in IEEE Access, vol. 8, pp. 127593-127607, 2020, doi: 10.1109/ACCESS.2020.3007944.

[27] N. Kammoun, R. Abassi and S. Guemara, "Towards a New Clustering Algorithm based on Trust Management and Edge Computing for IoT," 2019 15th

International Wireless Communications & Mobile Computing Conference (IWCMC), 2019, pp. 1570-1575, doi: 10.1109/IWCMC.2019.8766492.

[28] K. Kim and C. S. Hong, "Optimal Task-UAV-Edge Matching for Computation Offloading in UAV Assisted Mobile Edge Computing," 2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS), 2019, pp. 1-4, doi: 10.23919/APNOMS.2019.8892864.

[29] Y. Wang, H. Wang and X. Wei, "Energy-Efficient UAV Deployment and Task Scheduling in Multi-UAV Edge Computing," 2020 International Conference on Wireless Communications and Signal Processing (WCSP), 2020, pp. 1147-1152, doi: 10.1109/WCSP49889.2020.9299765.

[30] S. Wang et al., "Federated Learning for Task and Resource Allocation in Wireless High-Altitude Balloon Networks," in IEEE Internet of Things Journal, vol. 8, no. 24, pp. 17460-17475, 15 Dec.15, 2021, doi: 10.1109/JIOT.2021.3080078.

[31] A. K. Bairagi et al., "Coexistence Mechanism Between eMBB and uRLLC in 5G Wireless Networks," in IEEE Transactions on Communications, vol. 69, no. 3, pp. 1736-1749, March 2021, doi: 10.1109/TCOMM.2020.3040307.

[32] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu and W. Zhuang, "Learning-Based Computation Offloading for IoT Devices With Energy Harvesting," in IEEE Transactions on Vehicular Technology, vol. 68, no. 2, pp. 1930-1941, Feb. 2019, doi: 10.1109/TVT.2018.2890685.

[33] Y. K. Tun, D. H. Kim, M. Alsenwi, N. H. Tran, Z. Han and C. S. Hong, "Energy Efficient Communication and Computation Resource Slicing for eMBB and URLLC Coexistence in 5G and Beyond," in IEEE Access, vol. 8, pp. 136024-136035, 2020, doi: 10.1109/ACCESS.2020.3011167.

[34] Y. Polyanskiy, H. V. Poor, and S. Verdu, “Channel coding rate in the finite blocklength regime,” *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307–2359, May 2010.

[35] C. You, K. Huang, H. Chae and B. -H. Kim, "Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading," in IEEE Transactions on Wireless Communications, vol. 16, no. 3, pp. 1397-1411, March 2017, doi: 10.1109/TWC.2016.2633522.

[36] F. Zhao, Y. Chen, Y. Zhang, Z. Liu and X. Chen, "Dynamic Offloading and Resource Scheduling for Mobile-Edge Computing With Energy Harvesting Devices," in IEEE Transactions on Network and Service Management, vol. 18, no. 2, pp. 2154-2165, June 2021, doi: 10.1109/TNSM.2021.3069993.

[37] D. Liu et al., "User Association in 5G Networks: A Survey and an Outlook," in IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1018-1044, Secondquarter 2016, doi: 10.1109/COMST.2016.2516538.

[38] D. Liu, Y. Chen, K. K. Chai, and T. Zhang, “Joint uplink and downlink user association for energyefficient HetNets using Nash bargaining solution,” in Proc. IEEE 79th Veh. Technol. Conf. (VTC Spring), May 2014, pp. 1–5.

[39] D. Liu, Y. Chen, K. K. Chai, and T. Zhang, “Nash bargaining solution-based user association optimization in HetNets,” in Proc. IEEE 11th Consum. Commun. Netw. Conf. (CCNC), Jan. 2014, pp. 587–592.

[40] D. Liu, Y. Chen, K. K. Chai, T. Zhang, andM. Elkashlan, “Opportunistic user association for multiservice HetNets using Nash bargaining solution,” IEEE Commun. Lett., vol. 18, no. 3, pp. 463–466, Mar. 2014

[41] L. Chen, B. Wang, X. Chen, X. Zhang, and D. Yang, “Utility-based resource allocation for mixed traffic in wireless networks,” in Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), Apr. 2011, pp. 91–96

[42] O. Semiari,W. Saad, S. Valentin, M. Bennis, and B. Maham, “Matching theory for priority-based cell association in the downlink of wireless small cell networks,” in Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP), May 2014, pp. 444–448.

[43] N. Namvar, W. Saad, B. Maham, and S. Valentin, “A context-aware matching game for user association in wireless small cell networks,” in Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP), May 2014, pp. 439–443.

[44] M. Haddad, P. Wiecek, E. Altman, and H. Sidi, “A game theoretic approach for the association problem in two-tier HetNets,” in Proc. 25th Int. Teletraffic Congr. (ITC), Sep. 2013, pp. 1–9.

[45] W. Saad, Z. Han, R. Zheng, M. Debbah, and H. Poor, “A college admissions game for uplink user association in wireless small cell networks,” in Proc. IEEE INFOCOM, Apr. 2014, pp. 1096–1104.

[46] M. Hong and Z.-Q. Luo, “Distributed linear precoder optimization and base station selection for an uplink heterogeneous network,” IEEE Trans. Signal Process., vol. 61, no. 12, pp. 3214–3228, Jun. 2013.

[47] M. Deng, H. Tian and X. Lyu, "Adaptive sequential offloading game for multi-cell Mobile Edge Computing," 2016 23rd International Conference on Telecommunications (ICT), 2016, pp. 1-5, doi: 10.1109/ICT.2016.7500395

[48] X. Chen, L. Jiao, W. Li and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," in IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 2795-2808, October 2016, doi: 10.1109/TNET.2015.2487344.

[49] Yang, Li & Xu, Gaochao & Ge, Jiaqi & Liu, Peng & Fu, Xiaodong. (2020). Energy-Efficient Resource Allocation for Application Including Dependent Tasks in Mobile Edge Computing. KSII Transactions on Internet and Information Systems. 14. 2422-2443. 10.3837/tiis.2020.06.006.

[50] J. Xue and Y. An, "Joint Task Offloading and Resource Allocation for Multi-Task Multi-Server NOMAMEC Networks," in IEEE Access, vol. 9, pp. 16152-16163, 2021, doi: 10.1109/ACCESS.2021.304988

[51] Tong, M.; Wang, X.; Li, S.;Peng, L. Joint Offloading Decision and Resource Allocation in Mobile Edge Computing-Enabled Satellite-Terrestrial Network.

Symmetry 2022,14, 564. (14) (PDF) Joint Offloading Decision and Resource Allocation in Mobile Edge Computing-Enabled Satellite-Terrestrial Network. Available from: https://www.researchgate.net/publication/359217649_Joint_Offloading_Decision_and_Resource _Allocation_in_Mobile_Edge_Computing-Enabled_Satellite-Terrestrial_Network [accessed Aug 12 2022]

[52] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan and Y. Zhang, "Deep Learning Empowered Task Offloading for Mobile Edge Computing in Urban Informatics," in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 7635-7647, Oct. 2019, doi: 10.1109/JIOT.2019.2903191.

[53] Y. Dai, K. Zhang, S. Maharjan and Y. Zhang, "Edge Intelligence for Energy-Efficient Computation Offloading and Resource Allocation in 5G Beyond," in IEEE Transactions on Vehicular Technology, vol. 69, no. 10, pp. 12175-12186, Oct. 2020, doi: 10.1109/TVT.2020.3013990.

[54] Y. Huang, S. Li, C. Li, Y. T. Hou and W. Lou, "A Deep-Reinforcement-Learning-Based Approach to Dynamic eMBB/URLLC Multiplexing in 5G NR," in IEEE Internet of Things Journal, vol. 7, no. 7, pp. 6439-6456, July 2020, doi: 10.1109/JIOT.2020.2978692.

[55] M. Chen, O. Semiari, W. Saad, X. Liu and C. Yin, "Federated Echo State Learning for Minimizing Breaks in Presence in Wireless Virtual Reality Networks," in IEEE Transactions on Wireless Communications, vol. 19, no. 1, pp. 177-191, Jan. 2020, doi: 10.1109/TWC.2019.2942929.

[56] H. Chen, S. Huang, D. Zhang, M. Xiao, M. Skoglund and H. V. Poor, "Federated Learning over Wireless IoT Networks with Optimized Communication and Resources," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2022.3151193.

[57] Fujimoto, S., Hoof, H. & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. Proceedings of Machine Learning

Research 80:1587-1596 Available from https://proceedings.mlr.press/v80/fujimoto18a.html.

[58] M. Elsayed and M. Erol-Kantarci, "AI-Enabled Radio Resource Allocation in 5G for URLLC and eMBB Users," 2019 IEEE 2nd 5G World Forum (5GWF), Dresden, Germany, 2019, pp. 590-595, doi: 10.1109/5GWF.2019.8911618.

[59] Ross, G. J. S., and D. A. Preece. “The Negative Binomial Distribution.” *Journal of the Royal Statistical Society. Series D (The Statistician)*, vol. 34, no. 3, 1985, pp. 323–35. *JSTOR*, https://doi.org/10.2307/2987659. Accessed 8 Nov. 2024.

[60] I. Gerasin, A. Krasilov and E. Khorov, "Flexible Multiplexing of Grant-Free URLLC and eMBB in Uplink," 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications, London, UK, 2020, pp. 1-6, doi: 10.1109/PIMRC48278.2020.9217168.

[61] S. Singh and D. H. Kim, "Joint Optimization of Computation Offloading and Resource Allocation in C-RAN With Mobile Edge Computing Using Evolutionary Algorithms," in IEEE Access, vol. 11, pp. 112693-112705, 2023, doi: 10.1109/ACCESS.2023.3322650.

[62] J. Yun, Y. Goh, W. Yoo and J. -M. Chung, "5G Multi-RAT URLLC and eMBB Dynamic Task Offloadingwith MEC Resource Allocation using Distributed Deep Reinforcement Learning," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2022.3177425.

[63] M. Al-Ali, E. Yaacoub and A. Mohamed, "Dynamic Resource Allocation of eMBB-uRLLC Traffic in 5GNew Radio," 2020 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 2020, pp. 1-6, doi: 10.1109/ANTS50601.2020.9342844.

[64] W. Sui, X. Chen, S. Zhang, Z. Jiang and S. Xu, "Energy-Efficient Resource Allocation With Flexible Frame Structure for Hybrid eMBB and URLLC Services," in IEEE Transactions on Green Communications and Networking, vol. 5, no. 1, pp. 72-83, March 2021, doi: 10.1109/TGCN.2020.3028202.

[65] Y. -H. Hsu and W. Liao, "eMBB and URLLC Service Multiplexing Based on Deep Reinforcement Learning in 5G and Beyond," 2022 IEEE Wireless Communications and Networking Conference (WCNC), 2022, pp. 1467-1472, doi: 10.1109/WCNC51071.2022.9771707.

[66] J. Li and X. Zhang, "Deep Reinforcement Learning-Based Joint Scheduling of eMBB and URLLC in 5GNetworks," in IEEE Wireless Communications Letters, vol. 9, no. 9, pp. 1543-1546, Sept. 2020, doi: 10.1109/LWC.2020.2997036

[67] Liang Huang, Xu Feng, Cheng Zhang, Liping Qian, Yuan Wu, Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing, Digital Communications and Networks, Volume 5, Issue 1, 2019, Pages 10-17, ISSN 2352-8648, https://doi.org/10.1016/j.dcan.2018.10.003.

[68] Jiadong Dong, Kai Pan, Chunxiang Zheng, Lin Chen, Shunfeng Wu, Xiaolin Zhang, A Dual-Agent Approach for Coordinated Task Offloading and Resource Allocation in MEC, Journal of Electrical and Computer Engineering Volume 2023, Article ID 6134837, 16 pages, https://doi.org/10.1155/2023/6134837

[69] 1.3.6.6.18. binomial distribution. Available at: https://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm (Accessed: 20 February 2025).

[70] Ubaid Ullah Khan, Naqqash Dilshad, Mubashir Husain Rehmani, Tariq Umer, Fairness in Cognitive Radio Networks: Models, measurement methods, applications, and future research directions, Journal of Network and Computer Applications,Volume 73, 2016, Pages 12-26, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2016.07.008.

[71] K. Lee, C. Yoon, O. Jo and W. Lee, "Joint Optimization of Spectrum Sensing and Transmit Power in Energy Harvesting-Based Cognitive Radio Networks," in *IEEE Access*, vol. 6, pp. 30653-30662, 2018, doi: 10.1109/ACCESS.2018.2843395

[72] *Welcome to Python.org* (no date) *Python.org*. Available at: https://www.python.org/ (Accessed: 28 February 2025).

[73] *Get started* (no date) *PyTorch*. Available at: https://pytorch.org/ (Accessed: 28 February 2025).

[74] *Colab.google* (no date) *colab.google*. Available at: https://colab.google/ (Accessed: 28 February 2025).