Energy-aware Computational Offloading using Reinforcement Learning Techniques

by

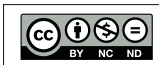
Meriem MECHENNEF

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN INFORMATION TECHNOLOGY ENGINEERING
M.A.Sc.

MONTREAL, DECEMBER 08, 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

 Meriem Mechennef, 2023



This Creative Commons license allows readers to download this work and share it with others as long as the author is credited. The content of this work cannot be modified in any way or used commercially.

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Aris Leivadeas, Thesis supervisor
Department of Software Engineering and Information Technology, École de Technologie Supérieure

Mr. Zbigniew Dziong, Chair, Board of Examiners
Department of Electrical Engineering, École de Technologie Supérieure

Mr. Julien Gascon-Samson, Member of the Jury
Department of Software Engineering and Information Technology, École de Technologie Supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON NOVEMBER 21, 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

This work would have not been possible without the support of many people.

I am deeply grateful to my supervisor, Professor Aris Leivadeas, not only for his invaluable academic guidance and patience but also for the trust he placed in me and his strong and continuous support throughout my studies. My heartfelt gratitude also goes to Marios Avgeris, his guidance and the time he invested in our collaborative work are deeply appreciated. I am fortunate to have been mentored by them and provided with the necessary psychological support.

From the bottom of my heart, I would like to deeply thank my parents, Abbes and Lila, for their unconditional love, support, and sacrifices that have been the foundation of my journey. I extend my appreciation to my cherished younger brothers, Mohamed and Kamel, I'm grateful for their presence in my life.

A profound gratitude to FMB for the constant support and encouragement. I deeply appreciate the warmth and positivity you have brought to my life; it has been a driving force.

A special mention goes to my dear friend Rafael Amado. I sincerely want to express my deep gratitude for the significant contribution you have made to this journey.

To my dear close friends and family members who will easily identify themselves, I want to say a big thank you.

Déchargement informatique sensible à l'énergie en utilisant des techniques d'apprentissage par renforcement

Meriem MECHENNEF

RÉSUMÉ

Face à l'évolution rapide de la 5G et à l'avènement de l'ère 6G, le déchargement informatique devient un élément clé pour l'exécution de tâches mobiles au sein des infrastructures de calcul en périphérie. Ce changement de paradigme implique le transfert de tâches de calcul gourmandes en ressources vers des serveurs externes à proximité du réseau, offrant ainsi la possibilité d'optimiser l'efficacité. Cependant, afin d'assurer une qualité de service constante pour les nombreux utilisateurs impliqués, une planification méticuleuse des décisions de déchargement doit être prise, ce qui implique potentiellement un transfert de tâches entre sites pour répondre aux diverses exigences d'application des utilisateurs mobiles.

Dans ce mémoire, notre attention se porte sur une infrastructure informatique de périphérie multi-accès avec multi-utilisateurs et collaboration multi-sites, où les appareils mobiles ont la capacité de décharger leurs tâches informatiques vers les sites en périphérie disponibles. Notre objectif est de minimiser le délai de bout en bout subi par ces tâches et la consommation d'énergie du système. Ces deux mesures cruciales constituent collectivement le coût global de l'ensemble du système et sont fondamentales pour l'expérience de l'utilisateur. Le défi central consiste à coordonner ces objectifs, garantissant une convergence harmonieuse de l'optimisation des performances, de la satisfaction de l'utilisateur et de l'efficacité énergétique.

Pour relever ce défi, nous introduisons un mécanisme sophistiqué en deux étapes basé sur l'apprentissage par renforcement, une approche de pointe qui nous permet d'affiner de manière itérative les décisions des appareils mobiles concernant le déchargement des tâches vers les serveurs en périphérie, ainsi que les décisions de ces derniers concernant le transfert des tâches entre eux. Ce processus itératif d'optimisation est au cœur de notre approche, guidant la coordination fluide des tâches informatiques pour atteindre un équilibre délicat entre de faibles latence et l'efficacité énergétique. La première étape est celle où les appareils mobiles déterminent de manière autonome s'ils doivent décharger leurs tâches vers les serveurs au périphérie auxquels ils sont connectés ou les exécuter localement. Cette décision de déchargement des tâches distribuée est prise en utilisant un mécanisme itératif basé sur l'apprentissage par renforcement appelé Stochastic Learning Automata (SLA). La deuxième étape, qui assure l'équilibrage de la charge sur l'infrastructure en périphérie, est réalisée à l'aide d'un Deep Q-Network (DQN) formé hors ligne et utilisé à la fin de chaque itération de la première étape. Ces deux étapes sont intégrées dans un mécanisme de déchargement informatique coopératif multi-tours qui optimise de manière itérative les décisions prises à la fois par les appareils mobiles et les sites en périphérie, conduisant finalement à la convergence stable du problème d'optimisation.

Nos résultats expérimentaux avec différents nombres d'appareils mobiles et sites en périphérie montrent que notre solution réduit la latence et la consommation d'énergie des appareils mobiles. Comparé à la solution proposée par des travaux antérieurs qui ne prend pas en charge l'équilibrage

de la charge au niveau de l'infrastructure en périphérie, notre solution obtient de meilleurs résultats.

Mots-clés: déchargement de tâches, répartition de charge, apprentissage par renforcement, automates d'apprentissage stochastique, apprentissage par renforcement profond, minimisation de la latence, minimisation de la consommation d'énergie

Energy-aware Computational Offloading using Reinforcement Learning Techniques

Meriem MECHENNEF

ABSTRACT

In the rapidly evolving landscape of 5G and the emerging 6G era, computational offloading is becoming a game-changer for mobile task execution within edge computing infrastructures. This paradigm shift involves the transfer of resource-intensive computational tasks to external servers nearby in the network, offering the potential for optimized efficiency. Yet, to ensure consistent Quality of Service (QoS) for the numerous users involved, meticulous planning of the offloading decisions should be made, which potentially involves inter-site task transferring to meet the diverse application requirements of mobile users.

In this thesis, our focus extends to a multi-user Multi-Access Edge Computing (MEC) infrastructure with multi-site collaboration, where Mobile Devices (MDs) have the capability to offload their computational tasks to the available Edge Sites (ESs). Our goal is to minimize end-to-end delay experienced by these tasks and the energy consumption of the system. These two important measures collectively constitute the overall cost of the entire system and are fundamental for the user experience. The central challenge is to coordinate these goals, ensuring a seamless convergence of performance optimization, user satisfaction, and energy efficiency.

To tackle this challenge, we introduce a sophisticated two-stage mechanism based on Reinforcement Learning (RL), a cutting-edge approach that enables us to iteratively refine the MDs' decisions regarding task offloading to ESs, as well as the ESs' decisions regarding the transfer of tasks between themselves. This iterative optimization process lies at the core of our approach, guiding the seamless coordination of computational tasks to achieve a careful balance between low delays and energy efficiency. The first stage is where individual MDs autonomously determine whether to offload their tasks to the attached ES or execute the tasks locally. This distributed task-offloading decision is done using an iterative RL-based mechanism called Stochastic Learning Automata (SLA). The next stage which ensures load balancing across the edge infrastructure is achieved using an offline-trained Deep Q-Network (DQN) employed at the end of each iteration in the first stage. These two stages are integrated into a multi-round cooperative computational offloading mechanism which iteratively optimizes the decisions made by both the MDs and the ESs, ultimately leading to the stable convergence of the optimization problem.

Our experimental results using different numbers of MDs and ESs show that our framework decreases the delay of the tasks and the energy consumption of MDs. Compared to the solution proposed by prior work which doesn't support load balancing at edge infrastructure our solution gives better results.

Keywords: task offloading, load balancing, edge computing, reinforcement learning, stochastic learning automata, deep reinforcement learning, latency minimization, energy consumption minimization.

TABLE OF CONTENTS

Page

INTRODUCTION 1
0.1 Context 1
0.2 Problematic 2
0.3 Objectives 3
0.4 Structure of the thesis 4

CHAPTER 1 BACKGROUND AND RELATED WORK 5
1.1 Multi-Access Edge Computing 5
1.2 Task Offloading 6
1.3 Reinforcement Learning 8
1.3.1 Markov Decision Process 9
1.3.2 Stochastic Learning Automata 13
1.3.3 Q-Learning 14
1.4 Deep Reinforcement Learning 15
1.4.1 Deep Q-Network 15
1.4.2 DQN key components 16
1.5 Related Work 18
1.5.1 Heuristic-based solutions 18
1.5.2 Game theory-based solutions 20
1.5.3 ML-based solutions 20
1.5.4 Load balancing after task offloading solutions 21

CHAPTER 2 SYSTEM MODEL & ALGORITHM DESIGN 25
2.1 System Model 25
2.1.1 Computation Models 26
2.1.1.1 Local Execution 26
2.1.1.2 Remote Execution 27
2.1.2 Problem Formulation and Analysis 31
2.2 Energy-Aware Computational Offloading Mechanism 33
2.2.1 First Stage: MD-to-ES Task Offloading 33
2.2.2 Second Stage: ES-to-ES Transferring 35
2.2.2.1 QL-based Task Transferring Training 36
2.2.2.2 DQN-based Task Transferring Training 36
2.2.3 Two Stage Cooperative MEC Computational Offloading 39

CHAPTER 3 PERFORMANCE EVALUATION 43
3.1 Benchmark solution 43
3.2 Evaluation parameters 44
3.3 Numerical Results 45
3.3.1 Training time comparison 45

3.3.2 Online convergence behavior 46
3.3.3 Comparative evaluation 47
3.4 Result discussion 49

CONCLUSION AND RECOMMENDATIONS 51

LIST OF REFERENCES 55

LIST OF FIGURES

Page

Figure 1.1 Architecture of Multi-access Edge Computing 6

Figure 1.2 Offloading procedure 7

Figure 1.3 Basic Interaction in RL 10

Figure 1.4 Basic QL steps 15

Figure 1.5 Basic DQN steps 16

Figure 2.1 Two-stage Cooperative MEC Computational Offloading 39

Figure 3.1 Training time 45

Figure 3.2 Convergence Behavior 46

Figure 3.3 Benchmarking: Delay 47

Figure 3.4 Benchmarking: Energy 48

LIST OF ALGORITHMS

Page

Algorithm 2.1 Learning Automata Offloading Algorithm 34

Algorithm 2.2 QL-based Task Transferring Training 36

Algorithm 2.3 DQN-based Task Transferring Training 38

Algorithm 2.4 Two-stage Cooperative MEC Computational Offloading 40

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AP	Access Point
AR	Augmented Reality
BS	Base Station
CC	Cloud Computing
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
EC	Edge Computing
ER	Experience Replay
ES	Edge Site
IoT	Internet of Things
ISG	Industry Specification Group
IT	Information Technology
MD	Mobile Device
MDP	Markov Decision Process
ML	Machine Learning
MSE	Mean Squared Error
MEC	Mobile Edge Computing/Multi-access Edge Computing

NN	Neural Network
QL	Q-Learning
QoS	Quality of Service
RAN	Radio Access Network
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
SLA	Stochastic Learning Automata
TD	Temporal Difference
UE	User equipment
VR	Virtual Reality

INTRODUCTION

0.1 Context

The expeditious development of the Internet of Things (IoT) alongside the advent of the 5G/6G wireless networks has given rise to a plethora of diverse mobile applications, such as Augmented Reality (AR) and Virtual Reality (VR). These applications are typically computationally intensive and time-sensitive, for instance, in order to provide an authentic VR experience, VR systems often require latency of less than 10ms (Chu, Jia, Yu, Lui & Lin, 2023). Despite advancements in MDs technology, their limited computational capabilities and battery lifetime still fall short of the proper execution of such applications. This drives the necessity to offload their workloads to resource-rich devices.

Leveraging Cloud Computing (CC), with its abundance of computational resources, can alleviate the issue to some extent. Yet, it's crucial to acknowledge that the QoS could be negatively affected by high transmission delays caused by the geographical distance between MDs and remote cloud servers (Avgeris *et al.*, 2022; Akhlaqi & Hanapi, 2023). To mitigate this, MEC arises as a viable solution, its fundamental principle involves pushing the computing capabilities at the edge of the wireless network. MEC architecture employs wireless Access Points (APs) to establish a connection between end users ESs, thus enabling MDs to offload computationally intensive tasks to their corresponding local ESs for processing without incorporating excessive communication pathways toward a remote cloud infrastructure. This mechanism, known as task offloading, is key in addressing the challenge of limited computational resources, minimizing the latency of computing tasks, and enhancing energy efficiency (Saeik *et al.*, 2021).

For the purposes of this thesis, the acronym MEC will be used exclusively to denote Multi-access Edge Computing, as per the expanded definition established by the MEC Industry Specification Group (ISG) during the 2016 MEC World Congress. This shift reflects the

expanded support of MEC for multiple access technologies, including fixed and wireless networks (Singh, Sukapuram & Chakraborty, 2023).

0.2 Problematic

Despite the several advantages yielded by employing Edge Computing (EC) for task offloading such as low latency and energy efficiency, it also raises new challenges that require meticulous attention.

One significant research obstacle in MEC systems is the decision-making process related to offloading - whether to offload a task to the ES or to execute it locally on the MD. Improperly planned computational offloading may result in various disturbing phenomena during the transmission and processing of the tasks, including unexpected increases in propagation and processing delay, as well as downtime (Bouhoula, Avgeris, Leivadeas & Lambadaris, 2022). In that case, the trade-off between on-device execution and offloading at the local ES should be investigated.

Moreover, unlike the cloud, the computational resources available at the edge infrastructure are confined to micro data centers, consisting of only a few servers. There might be scenarios where the offloading of tasks from MDs to their associated local ESs becomes impractical due to varying loads at the edges. For instance, a situation where multiple MDs within the coverage of an ES offload their tasks simultaneously could cause a substantial increase in response latency, as the ES becomes suddenly overwhelmed with offloaded tasks. Therefore, redirecting these offloaded tasks for processing from overloaded local ES to underloaded remote ones can potentially alleviate the pressure on overloaded sites and enable load balancing across the edge infrastructure. Ideally, the offloading decision and resource allocation should be jointly optimized to enhance the overall QoS.

Overall, carefully determining the offloading strategy in a multi-site edge infrastructure, for achieving load balancing while minimizing processing delay and energy consumption is an imminent challenge. It results in a mixed integer programming problem that is non-convex and difficult to solve.

0.3 Objectives

In this thesis, our main objective is the joint optimization of task offloading decisions and computation resource allocation within cooperative MEC environment. We consider a multi-site MEC infrastructure wherein each MD has a single computationally intensive task to be processed. Our aim is to allocate computational resources among the competing MDs such that we minimize the weighted sum cost in terms of the completion time and the energy consumption for all the MDs, while simultaneously ensuring a load balancing across the edge infrastructure.

By leveraging the power of RL, we propose a novel two-stage computational offloading mechanism to achieve the said objectives. In the first stage, the offloading decision problem is solved using a decentralized approach based on SLA which enables distributed and autonomous decision-making. MDs act as SLA and in every iteration of this process, each one independently selects whether to offload to the attached ES or execute the task locally on the device (MD-to-ES offloading), aiming at achieving its objectives. For the second stage, we integrate an ES-to-ES cooperative offloading mechanism which aims to balance the workload in the infrastructure, by proactively transferring the MD's offloaded tasks from the overloaded to underloaded sites. An offline-trained DQN is utilized to make these decisions at the end of each iteration of the first stage and the updated processed delays are fed back to the MDs for them to reconsider their offloading decisions. Finally, these two stages are combined into a multi-round cooperative computational offloading mechanism, which iteratively optimizes the offloading and transferring decisions until they converge to a stable solution.

0.4 Structure of the thesis

Our thesis is organized into four distinct chapters, each with a specific focus, detailed as follows. In Chapter 1, the main general topics in this thesis are described, and then we overview the relevant related work within MEC. In Chapter 2, we outline our system, formulate our task offloading optimization problem, and detail our proposed solution. In Chapter 3, the performance evaluation of our proposed framework is presented and examined.

CHAPTER 1

BACKGROUND AND RELATED WORK

In this chapter, we provide a background view of MEC, task offloading techniques, and RL. Furthermore, we overview the relevant related work within MEC.

1.1 Multi-Access Edge Computing

MEC has recently been at the forefront of both academic and industrial focus, viewed as an influential technology to boost computational performance and prolong the lifespan of MDs. It provides a new ecosystem in which operators can open their Radio Access Network (RAN) edge to authorized third parties, such as application developers or content providers, expanding the scope of services and applications that can be offered to end-users. (Singh *et al.*, 2023; Zhou, Jadoon & Khan, 2023)

This groundbreaking approach restructures the conventional network architecture, typically comprising just a client and a server, by incorporating an extra component in the mix. Specifically, MEC servers represent this new intermediary element placed at the periphery of the network (e.g., base stations and access points), forming a novel development framework comprising the Client, Edge server, and Remote Server. The standard two-layer structure of the MEC system is illustrated in Figure 1.1. At the devices' layer, a variety of MDs exist, capable of exchanging information with local APs within the edge environment. The decision of whether to offload specific tasks to the MEC servers is first considered at this layer. In the edge layer, ESs are situated, each consisting of a small data center connected to a wireless AP to provide offloading services for the resource-constrained MDs of its coverage area.

By extending cloud computing capabilities and IT services nearer to the users, MEC allows the execution of heavy-duty and latency-sensitive applications on resource-limited MDs. Essentially, when local resources are not adequate for prompt processing, devices can offload their tasks to MEC servers and due to their proximity, devices do not need to endure high latency to utilize their services (Avgeris, Mechennef, Leivadeas & Lambadaris, 2023). Consequently, MEC

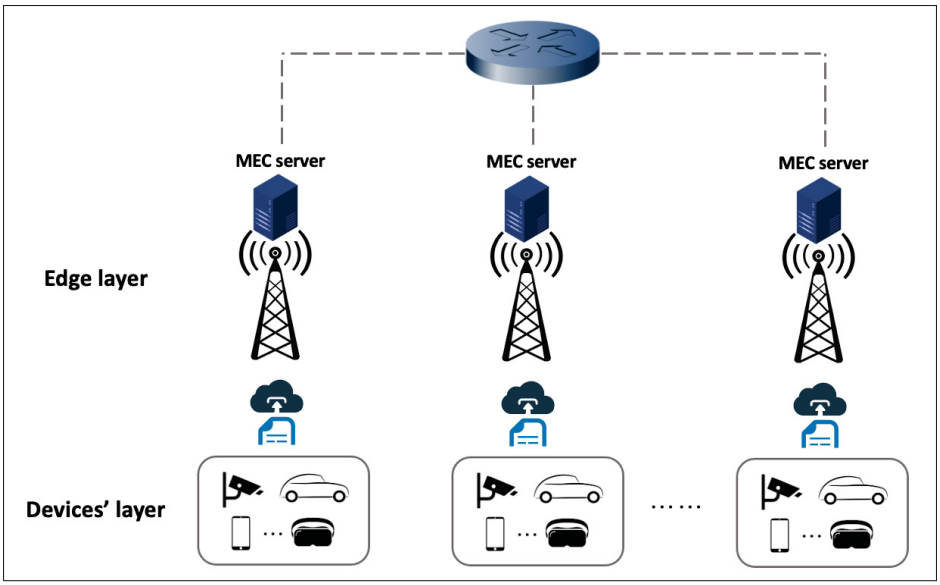


Figure 1.1 Architecture of Multi-access Edge Computing

provides substantial cuts in both latency and energy consumption for devices, while concurrently relieving network congestion. MEC opens the door to a multitude of novel key applications, the primary use cases currently recognized are AR and VR, connected vehicles, industrial IoT and Healthcare, among others (Avgeris, 2021).

1.2 Task Offloading

MEC addresses the deficiencies of MD execution resources by performing task offloading. It is a key point in the research of MEC and stands out as a significant challenge to fully leverage the benefits that MEC presents. As mentioned earlier, task offloading is a process of transferring computational tasks from one platform to another, and with proper design and planning, it can improve the computational efficiency of mobile entities, leading to a reduction in both latency and the overall energy consumption of the ecosystem (Shakarami, Ghobaei-Arani & Shahidinejad, 2020).

A task is a unit of work or requests to be completed remotely as a part of application components, it can be either application code, process, data, or service requests. The MD's decision to offload

or locally execute a task depends on different factors, such as QoS requirements and the device's battery life. Tasks meant for remote processing are sent over the wireless network to a distant server. Meanwhile, tasks kept on the device are processed locally using its available computing power. The final output of the application is then generated by combining the results of both local and remote tasks (Saeik *et al.*, 2021). Figure 1.2 shows these two types of task offloading and are described as follows:

- *Binary/Full Offloading*: In this type of offloading, the whole application data is transferred and processed at the edge server. The key to a successful (or beneficial) full offloading is to allocate sufficient radio and computation resources such that both the transmission time and energy can be minimized (Zhang, 2023).
- *Partial Offloading*: In this case, MDs offload part of the data to the edge server, while the rest of the data are processed locally. This type of offloading can benefit from both local and remote resources and it consists of a transmission procedure, a remote execution procedure, and a result send-back procedure. However, it introduces an additional layer of complexity, requiring the determination and scheduling of tasks to be offloaded. This process must consider the potential energy and resource constraints of the end device (Saeik *et al.*, 2021).

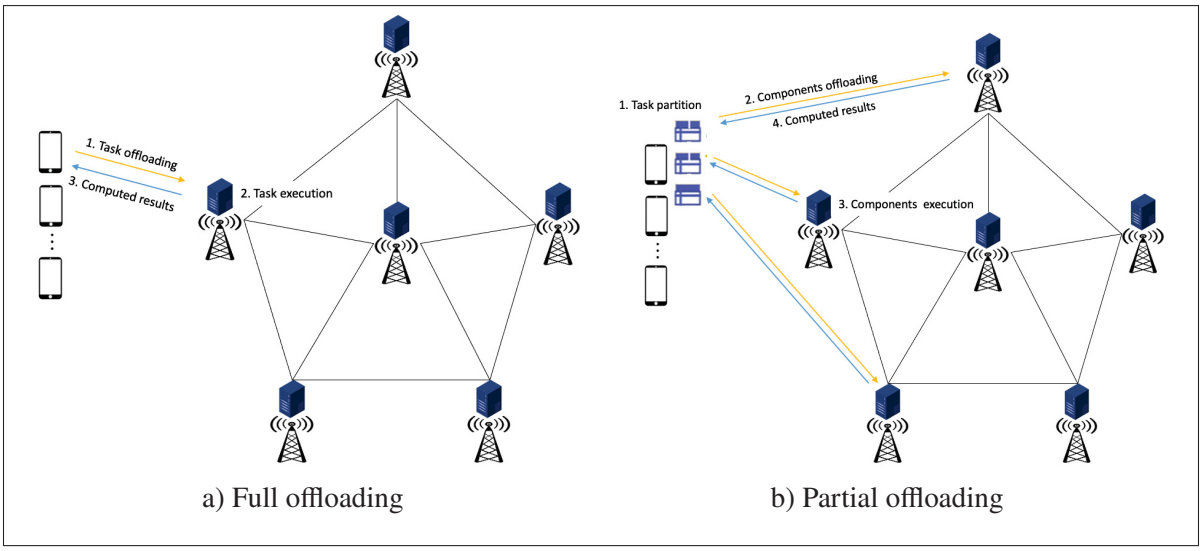


Figure 1.2 Offloading procedure

The offloading scheme includes the following key components (Lin, Zeadally, Chen, Labiod & Wang, 2020):

- Task partition: If a task is partitionable as in the case of partial offloading, we need to optimally partition the task before offloading. Otherwise, the whole task should be offloaded to an edge server.
- Offloading decision: It is a critical decision to decide whether to offload the task to an edge server or execute it locally, that needs to be planned properly. This offloading decision engine estimates the required energy and completion time in both cases and then decides to whether or not the MDs will save energy and time by offloading a given task. For example, offloading is not beneficial if the MDs consume energy on offloading the task more than on executing it on the device. Therefore, an offloading decision is vital to make the offloading beneficial for MDs.
- Resource allocation: The available bandwidth and computational resources are limited, thus good system performance would require a judicious distribution of these resources between the users.

In order to fully exploit the benefits of task offloading an intelligent solution that harnesses the potential of Artificial Intelligence (AI) is required. In the subsequent section, we introduce RL which is a pivotal element of our approach.

1.3 Reinforcement Learning

RL is a sub-category of Machine Learning (ML). Unlike supervised and unsupervised learning, RL relies on a trial-and-error approach. RL can start learning from scratch and attain human-level decision-making gradually through a set of actions and rewards from these actions. Hence, it can solve sequential decision-making problems, where the selected action not only affects the immediate reward achieved but impacts the upcoming situations and, through that, all subsequent rewards. The agent learns the best behavior in an uncertain, potentially complex environment while having no knowledge about which actions to take, but instead must discover which ones yield the most reward by trying them. These two characteristics —delayed reward and trial-and-

error search— are the two most important distinguishing features of RL. (Rezagholizadeh, 2022; Sutton & Barto, 2018)

Typically, an RL setup is composed of four main components: *environment, agent, action* and *reward*. The environment provides the agent, which is the learning entity that interacts with it, with different possible states that could exist in the problem it has to react to and feedback which is the reward signal to tell it what actions led to success (or failure). These four components represent a *Markov decision process* (MDP).

1.3.1 Markov Decision Process

MDP is a generalized framework used to model decision-making problems in RL. It provides a mathematical formalism for modeling an agent's interactions with an environment defining the structure and components of the RL problem (Sutton & Barto, 2018; Arulkumaran, Deisenroth, Brundage & Bharath, 2017; Chen, Qu, Tang, Low & Li, 2021a). MDP is defined by a 5-tuple as (S, A, P, R, γ), where:

- S and A denote a finite state and action set respectively.
- $P(s_{(t+1)}|s_t, a_t)$ is the state transition probabilities or transition dynamics, it gives the probability distribution of the next state given the current state and action. In most real-life applications, the transition dynamics are not known due to the fact that it is hard to come up with a model of the environment and this is where model-free RL algorithms can be used (Sadiki, Bentahar, Dssouli, En-Nouaary & Otrok, 2023)
- R is an immediate reward after performing the action a_t under state s_t.
- $\gamma \in [0, 1]$ is a discount factor to reflect the diminishing importance of current rewards on future ones, with lower values placing more emphasis on immediate rewards.

The basic concept behind RL is illustrated in Figure 1.3, where the agent and environment interact in a sequence of discrete time steps, t. At each time step $t = 1, 2, 3, ...$, the agent is in a certain state s_t and takes one of the possible actions a_t. The agent's action causes the environment to change from s_t to $s_{(t+1)}$, generating a reward $r_{(t+1)}$. The agent then performs

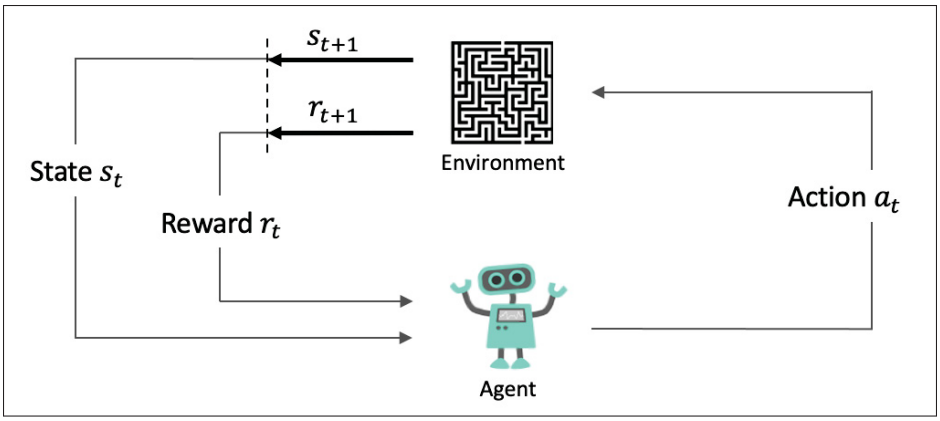


Figure 1.3 Basic Interaction in RL

another action for this new state $s_{(t+1)}$ triggering a reward, and so on. The rewards provided by the environment determine the best sequence of actions.

This process of selecting an action from a given state transitioning to a new state and receiving a reward happens sequentially over and over again which creates a *trajectory* or *episode* that shows a sequence of state, actions, and rewards (Sutton & Barto, 2018). In other words, an episode or trajectory represents one full run through all steps that end at a terminal point. In episodic tasks, this process ends when a terminal state is reached, which is followed by resetting the environment to a standard starting state or to a random sample from possible states. The next episode then starts independently from the previous one.

Policies and Value Functions

Any strategy that the agent follows to decide which action to pick in a given state, is called a policy. Formally, a policy π is a mapping from the state space to potential actions. It can be deterministic, meaning it prescribes a single action for each state, or stochastic, where it provides a probability distribution over actions. The choice of policy significantly influences the agent's performance and its ability to learn and adapt to the environment. Notably, there are two main categories of RL methods based on their treatment of policies: on-policy and off-policy methods. On-policy methods focus on improving the policy currently in use, directly seeking

the optimal policy. In contrast, off-policy methods, concentrate on learning a value function and then derive a policy based on the learned values. (Li, 2022)

Each episode ends in a terminal state in T time steps and yields a cumulative reward called a return defined as

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{1.1}$$

This reward will be discounted by γ (*discount rate* or *factor* $(0 < \gamma < 1)$) to the exponent of the time step. This controls how soon future rewards are ignored. In other words, if $\gamma = 0$, then only the present reward is taken into consideration. With this we define the discounted return as

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + ... = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{1.2}$$

The goal of the RL agent is to learn an optimal policy denoted by π_*, that maximizes this *expected discounted return*. The agent should know how good it is to be in a specific state or, how good it is to perform a given action in a specific state. This notion of how good a state or a state-action pair is is given in terms of expected return. Almost all RL algorithms consider two value functions:

- **State-Value Function**: The state-value function represents the expected return an agent can obtain by starting from state s at time t and following a particular policy π. In other words, it tells us the total return we can expect in the future if we start from that state

$$V_\pi(s) = \mathbb{E}_\pi(G_t | s_t = s) \tag{1.3}$$

- **Action-Value Function**: The action-value function or Q-function for brevity, is closely related to the state-value function but adds another dimension by considering not only the state but also a specific action. It represents the expected return an agent can obtain by from state s at time t, taking action a and following policy π.

The output from this function is called a Q-value, the letter Q refers to the quality of taking a given action in a given state

$$Q_\pi(s, a) = \mathbb{E}_\pi(G_t | s_t = s, a_t = a) \tag{1.4}$$

Once we have the respective Value Functions, we use them to compare the policies learned by the agent. The one with the highest Value Function yields higher expected returns and is called optimal policy denoted π_*. The equivalent optimal state-value and action-value functions are $V_*(s)$ and $Q_*(s, a)$, respectively (Winder, 2020). In contrast, the action-value Function enables the derivation of a policy independently of any model, this makes it necessary for model-free methods such as Q-Learning (QL).

Bellman Optimality Equation

A fundamental property that is frequently used in MDP called *Bellman optimality equation* represents the recursive relationship of the value function (Zhang, 2020). Q_* must satisfy this equation defined as

$$Q_*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q_*(s', a'] \tag{1.5}$$

The *Bellman optimality equation* states that, for any state-action pair (s, a) at time t, the expected return from starting in state s, choosing action a and following the optimal policy, is going to be the expected reward we get from taking action a in state s, which is r_{t+1} plus the maximum expected discounted return that can be achieved from any possible next state-action pair (s', a'). Since the agent is following an optimal policy, the following state s' will be the state from which the best possible next action a' can be taken at time $t + 1$.

Solving an RL problem basically means finding the Optimal Policy. There exist different ways of searching through an environment's states to estimate the expected return. When the model is known ahead, obtaining the optimal value function becomes a planning problem that can be

solved by **Dynamic Programming (DP)**. This technique performs bootstrapping, updating the value functions based on other currently estimated values, after a single time step. In many problems, prior knowledge of the environment's model is not available. In this case, sampling methods can be instead used, in which one estimates the state values from the experience generated by the agent-environment interaction. A frequently employed approach is **Monte Carlo (MC)**, wherein learning involves updating the value function by utilizing the average sample returns obtained upon concluding an episode. Another common method is **Temporal Difference (TD)**, which combines the advantages of both MC and DP. It learns directly from real experience without requiring the model of the environment like MC and DP. It bootstraps without waiting to reach the end of an episode to get the final outcome. (Li, 2022; Winder, 2020; Sutton & Barto, 2018; Rezagholizadeh, 2022).

This thesis focuses on model-free off-policy RL methods. Subsequently, we will introduce two algorithms employed in this study, namely SLA and QL.

1.3.2 Stochastic Learning Automata

An automaton represents a machine or control system engineered to autonomously execute a pre-established sequence of operations or react to encoded instructions. Learning is characterized as any enduring modification in behavior stemming from previous experiences. Therefore, a learning system should inherently possess the capability to enhance its behavior over time, gradually progressing toward a predetermined objective. The term stochastic emphasizes the flexible nature of the automaton, this latter does not adhere to predefined rules; instead, it adjusts according to environmental shifts. This adaptive mechanism is a product of its learning process. The stochastic automaton embarks on problem-solving without any prior knowledge of the optimal action. Initially, it assigns equal probabilities to all the actions, it randomly selects one action, observes the corresponding environmental response, updates action probabilities based on this feedback, and repeats this process. When a stochastic automaton operates in this way to enhance its performance, it can be termed a learning automaton. (Unsal, Kachroo & Bay, 1999; Stoica, Popa & Pah, 2008)

In summary, SLA is one of the RL tools. It is an adaptive, online decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment (Rasouli, Razavi & Faragardi, 2020)

1.3.3 Q-Learning

QL is a fundamental algorithm in the realm of RL. It is a classic model-free off-policy TD RL method. In this case, the learned action-value function Q directly approximates the optimal action-value function Q_*, independent of the policy being followed (Sutton & Barto, 2018). As shown in Figure 1.4, QL is a Lookup-Table-based approach, it uses a Q-table of State-Action Values (Q-values). This Q-table has a row for each state and a column for each action and it maps state and action pairs to a Q-value. The algorithm begins by initializing all the values to zero and, as the agent interacts with the environment and receives feedback in the form of rewards, the algorithm progressively refines these Q-values through iterative updates. QL performs TD updates to state-action pairs as follows

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{1.6}$$

where, α is the learning rate, $(0 < \alpha < 1)$. It balances the weight of what has already been learned (existing information) with the weight of the new observation (new information) during the update process. The core idea behind this update is to find the TD between the predicted Q-value ($r_t + \gamma \max_a Q(s_{(t+1)}, a)$) and its current value ($Q(s_t, a_t)$). The algorithm terminates when either a specified number of iterations have been reached or all Q-values have converged. At this stage, the algorithm outputs the optimal policy, which suggests the best action to take in each state.

Although QL has played a pivotal role in addressing diverse RL problems it faces significant limitations. In a situation characterized by extensive state spaces the algorithm may converge

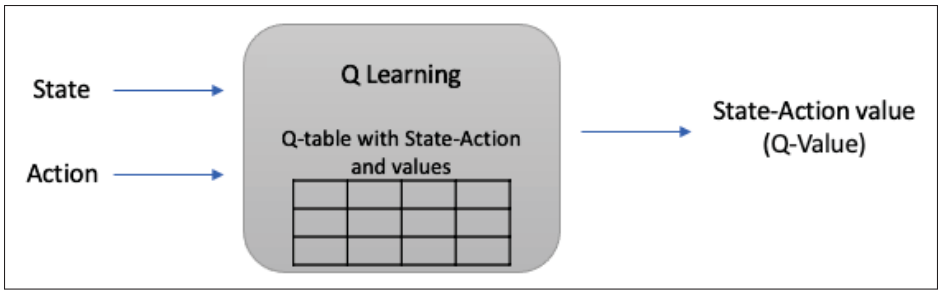


Figure 1.4 Basic QL steps

slowly or fail to converge altogether in complex environments. This calls for more sophisticated approaches that can handle such challenges effectively.

1.4 Deep Reinforcement Learning

As discussed above, simple RL lacked scalability and was limited to fairly low-dimensional problems. The rise of deep learning in recent years has provided new tools to overcome these problems, leading to the creation of the exciting field of DRL. The usage of Deep Neural Network (DNN) as function approximator to learn Q-values gives the agent the ability to learn from complex environments and allows to scale to previously intractable decision-making situations where the state space and action space are high-dimensional (Sadiki *et al.*, 2023).

1.4.1 Deep Q-Network

DQN is a combination of the model-free, off-policy QL with DNN. As shown in Figure 1.5, the architecture of a DQN comprises DNN that replaces the Q-table and serves as the foundation for approximating the Q-function. This DNN is designed to take in the state space as input and output Q-values for each possible action.

The DNN architecture used by DQN comprises an *input layer*, *hidden layers*, and an *output layer*. The input layer is responsible for receiving external data (state space), while the output layer generates the final outcome. Each hidden layer only connects to its adjacent two layers.

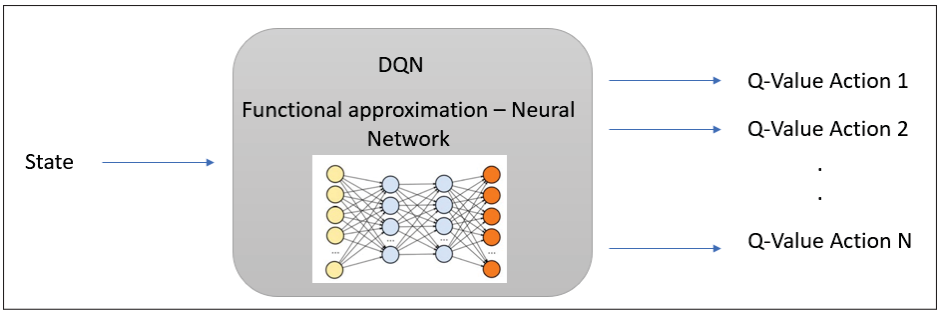


Figure 1.5 Basic DQN steps

Increasing the number of hidden layers enhances the network's capacity for approximating complex functions (Li, 2022).

1.4.2 DQN key components

In this section, we will delve into the key components that make DQN such a successful algorithm. These components include Experience Replay (ER), Target Networks, and the Epsilon-Greedy exploration strategy.

1. *Experience Replay*: The agent's experiences are stored in a database $\mathcal{D}$ called replay buffer. In each training step, the algorithm selects samples from the buffer $\mathcal{D}$ and feeds them to the network for the training. The Q-values obtained will be used to obtain new experiences, and these experiences will be then stored in the memory pool $\mathcal{D}$. This allows to use old data to train the model, making training more sample-efficient. Also, the correlation between consecutive observations is broken by taking a random subset when training.
2. *Fixed Target Q-network*: The second mechanism is to create a target network which is a clone of the Q-network but has a different weight. This target network is updated frequently but slowly compared to the primary Q-network. This will reduce the correlations between the target and estimated Q-values, thereby stabilizing the algorithm.

   These two mechanisms help to improve sample efficiency and stability of training (Winder, 2020).

3. *Epsilon-Greedy exploration strategy (ϵ-greedy)*: The ϵ-greedy algorithm is both the most popular and the simplest method for balancing the trade-off between "exploration" and "exploitation" phases. ϵ is a hyperparameter between 0 and 1, a small ϵ value means the agent mostly exploits its current knowledge, while a larger value implies a greater emphasis on exploration. t. The value of ϵ once chosen remains constant for the behavior policy. (Sewak, 2019)

    - **Greedy action** (probability of $1 - \epsilon$): the agent selects the action that has the highest estimated Q-value according to the Q-network. This is the exploitation part of the strategy, where the agent chooses the best-known action.
    - **Random action** (probability of ϵ): the agent selects a random action. This is the exploration part of the strategy, where the agent takes a random action to learn more about the environment.

4. *Loss function*: To approximate the Q-values, DQN uses DNN with weights θ. For every iteration, these weights get updated to converge towards the optimal Q-values. The Q-network can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$, where the loss function at time step t is defined as

$$L_t(\theta_t) = \mathbb{E}[r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{t-1}) - Q(s_t, a_t; \theta_t))^2] \tag{1.7}$$

    This means, given a transition (s_t, a_t, r_t, s_{t+1}), the weights θ of the Q-network are updated to minimize the squared error loss between the current predicted Q-value $Q(s_t, a_t)$ and the target Q-value of $(r_t + \gamma \max_{a'} Q(s_{(t+1)}, a')$

In our work, first the SLA is used to solve the offloading decision problem and then we propose a QL approach to balance the offloaded workload at the edge infrastructure. Later, to overcome the curse of dimensionality, we propose a DQN method to solve the same problem.

1.5 Related Work

The computational offloading problem falls into the knapsack resource allocation category, which is known to be NP-hard. This difficulty arises because computation offloading, involving binary offloading or partitioning an application for offloading, often leads to an integer programming formulation. Additionally, the allocation of resources, such as bandwidth and computing resources, among mobile users adds further complexity to the problem. Consequently, it is notably challenging to solve it. (Saeik *et al.*, 2021; Zhao, Geng & Jin, 2023)

To mitigate this complexity and attain optimal or nearly optimal results in a feasible time frame, approximate solutions based on ML, Game Theory, and heuristics have been proposed in the literature.

1.5.1 Heuristic-based solutions

In (Yang, Zhang, Li, Guo & Ji, 2018), the authors studied the problem of computation offloading for MEC in 5G systems. Specifically, this study aimed to enhance the energy efficiency of entities offloading tasks in the system. The problem was formulated as an optimization problem, seeking to minimize energy consumption while meeting delay requirements. In the formulation model, both task transmission and task computation at the MEC server were considered and to solve the problem, the authors proposed using an artificial fish swarm algorithm (AFSA). This heuristic algorithm provides global convergence, obtaining the global optimization solution for the problem under consideration (Maray & Shuja, 2022).

(Tran & Pompili, 2019) proposed a Joint Task Offloading and Resource Allocation (JTORA) strategy to maximize the task offloading gains, measured by a weighted sum of reductions in task completion time and energy consumption. The authors considered a multi-cell MEC system, where each Base Station (BS) is equipped with a MEC server providing offloading services to the end mobile users. The underlying optimization problem was formulated as a Mixed-Integer Non-linear Program (MINLP), which is very difficult to solve. To tackle the complexity, they decomposed the original problem into a resource allocation and a task offloading subproblem.

They addressed the resource allocation using convex and quasi-convex optimization techniques, and propose a novel heuristic algorithm for the task offloading that achieves a suboptimal solution in polynomial time.

(Zhang, Zhang, Zhang, Shen & Wang, 2021) considered a MEC system consisting of multiple users, a MEC server and a cloud center and examined the problem of joint service caching, computation offloading, transmission and computing resource allocation. Their objective was to minimize the overall computation cost and delay cost of the system and the interdependence between service caching decisions and computation offloading decisions added to the complexity of the problem. To address this challenge, they transform the problem into a Quadratically Constrained Quadratic Programming (QCQP). They heuristically recovered the feasible binary decisions and proposed a scalable and efficient approximate algorithm based on the powerful, computationally efficient Semi-Definite Relaxation (SDR) approach with the alternating optimization method.

(Yin, Chen & Zhang, 2022) addressed the task offloading problem in a scenario involving multi-user devices and multi-core edge servers, with the aim of minimizing the total energy consumption under the deadline constraint. Since this problem is known to be NP-hard, the authors proposed two greedy heuristics, namely GH1 and GH2, to tackle it. The two heuristics start from initial task sequences generated using the well-known Long Task First (LTF) and Short Task First (STF) rules, respectively. These initial task sequences are then allocated to one of MEC servers' cores and the power for data transfer through the antenna is determined using a Task Allocation Strategy (TAS). The results demonstrated that the proposed heuristics significantly outperformed the RoundRobin approach. Furthermore, it was observed that GH1 performed better than GH2, indicating the superiority of the LTF rule over the STF for the considered problem.

1.5.2 Game theory-based solutions

Using game theory and from the perspective of Device-to-Device (D2D) communication, the mutual D2D cooperation problem among users with heterogeneous demands was investigated in a MEC environment by (Fang, Wu, Chen & Liu, 2022). Users with different demands can share their idle local resources in order to minimize their processing delay. The authors proposed the heterogeneous demands-based matching game to model the complex cooperation problem. Different from the traditional matching game, they extended it to the innovative multiple-round matching game including D2D matching and edge matching. They then proposed the Multi-Round Cooperation Matching (MRCM) algorithm to find the desired solution.

The work in (Yan *et al.*, 2019) integrates task offloading decisions and resource allocation within the MEC system. The authors conceptualized a strategic game where critical elements such as offloading decisions, CPU capacity adjustments, transmission power control, and network interference management for mobile users are treated as integral components. Each mobile user adopts a best response strategy to maximize their utility rather than prioritizing the overall system utility. Their study establishes that the proposed game is an exact potential game, affirming the existence of a Nash equilibrium (NE).

1.5.3 ML-based solutions

ML-based approaches are widely acknowledged for their effectiveness in optimization, surpassing traditional optimization schemes (Maray & Shuja, 2022). Authors in (Li, Gao, Lv & Lu, 2018), consider a MEC-enabled cellular system, allowing multiple mobile users to offload their computational tasks via wireless channels to one MEC server. They propose a two-fold optimization approach, utilizing a single-stage QL and a DQN-based scheme. Their approach aims to jointly optimize the offloading decisions and computational resource allocation, with the primary objective of minimizing the combined delay and energy consumption. However, it's important to note that the authors restrict their investigation to a simplified scenario consisting of a single cell with only one MEC server, to which all users offload their tasks.

(Chen, Gu & Li, 2022) conducted a study on dynamic task offloading for Internet of Things (IoT) devices by integrating the resources of MEC and the cloud. They considered multi-user and multi-server scenario. Their research aimed to achieve two conflicting task offloading goals, maximizing the number of completed tasks (if the processing delay exceeds the tolerable delay the task is perceived as an unfinished task), and minimizing power consumption. To balance these goals within the constraints of available resources, the authors formulated a detailed task offloading problem and then transformed it to a Markov Decision Process (MDP)-based dynamic task offloading problem. To address this problem, they designed a dynamic RL-based task offloading algorithm.

To further retain low energy consumption and end-to-end delays, it has become necessary to cooperate multiple edge sites instead of operating independently. Thus, intra-infrastructure load balancing after task offloading is being proposed.

Load balancing is closely tied to the overall performance of parallel and distributed computing systems. While problems related to communication and computation have been extensively explored in data center environments, there is a notable scarcity of studies addressing these issues in IoT edge scenarios (Liu *et al.*, 2022).

1.5.4 Load balancing after task offloading solutions

Recently, many works in the pertinent literature started exploring the cooperation between the different edge sites. (Wan, Li, Xue, Lin & Xu, 2020) proposed an edge computation offloading method for the Internet of Vehicles (IoV) under the architecture of 5G networks named COV. The purpose of their work is to solve a multi-objective optimization problem, which aims to reduce offloading delay and enhance offloading cost while achieving the load balance across edge nodes. The appropriate destination edge nodes are selected using the Strength Pareto Evolutionary Algorithm 2 (SPEA2). Then, the most balanced strategies of COV at different vehicle scales are selected out by adopting the Technique for Order of Preference by Similarity to an Ideal Solution (TOPSIS) and Multi-Criteria Decision-Making (MCDM).

In (Avgeris *et al.*, 2022), as a subsequent stage to the device-to-edge task offloading, the authors propose a second phase to better balance the previous resource management decisions, where any extra workload requests from one site are redistributed to a nearby or even further away sites. The excess workload is handled in such a way that prevents sites from becoming operational for a number of requests below the threshold of their overall capacity, which will guarantee enhanced energy efficiency. They introduced a Markov Random Field (MRF) based mechanism for the distribution of the excess workload between the different edge sites. This technique allows simple distributed decision-making and is shown to achieve energy optimization at the infrastructure while respecting QoS requirements.

(Chen *et al.*, 2021b) proposed an RL-empowered feedback control method to solve the problem of cooperative load balancing (RF-CLB) in multi-edge industrial IoT environments. First, each edge independently schedules tasks and performs load balancing between adjacent edges based on the local information using QL and Support Vector Regression (SVR) algorithms. Next, they developed a new mechanism of feedback control to find the objective load-balancing plan between adjacent edges.

(Yuan & Zhou, 2021) present a profit-maximizing collaborative computation offloading and resource allocation algorithm in a three-layer system architecture consisting of multiple MDs, multiple MEC servers and a cloud data center layers. The proposed algorithm jointly optimizes the computation offloading between the cloud data center and edge computing layers while also specifying resource allocation within the cloud layer. The system profit is maximized, while the response time and energy consumption limits of tasks are strictly met. Their approach considers the varying processing capacities of heterogeneous nodes within the edge computing layer and addresses the load balance requirements of all nodes, ensuring an equitable distribution of tasks among them. A single-objective constrained optimization problem is formulated for each time slot, and it is solved using a proposed Simulated annealing-based Migrating Birds Optimization (SMBO) algorithm.

The authors in (Xu *et al.*, 2022) investigated the partial offloading of multiple MDs by proposing a Two-Stage Game Theory Algorithm (TSGTA). They divided a two-stage game theoretic scheme to reach the goal of optimal offloading on a cooperative user device-edge-cloud environment. The existence of a Nash equilibrium is proven, that minimize energy consumption under delay constraints. The authors first obtain the edge-terminal optimal offloading decision that ignores cloud resources after that they take the offloaded MDs as gamers and seek optimal offloading decisions between the cloud and the edge for all terminal devices opting to offload. Ultimately, the aim is to achieve optimal offloading across the three components: cloud, edge, and terminal.

A similar three-tier cooperative computational offloading scheme is proposed in (Kuang, Ma, Li & Deng, 2021) as well, where the authors have considered vertical cooperation among UE devices, MEC server nodes, and cloud server nodes, and the horizontal computation cooperation between edge nodes. This time utilizing a joint iterative algorithm based on the Lagrangian dual decomposition to minimize latency under energy consumption constraints.

In this work, through a novel RL mechanism, we investigate energy consumption and processing delay minimization during computational offloading in a cooperative MEC environment. Considering a MEC infrastructure consisting of multi-users multi-interconnected edge servers, we propose a two-stage solution. First, an iterative RL-based mechanism based on SLA is introduced to enable distributed and autonomous offloading decision-making (MD-to-ES offloading). Next, we integrate an ES-to-ES cooperative offloading mechanism aiming to balance the workload in the edge infrastructure. At this level, first, we designed a QL algorithm but as the number of ESs and MDs became larger, training the algorithm was almost impossible. In order to solve this problem, we used DRL and an offline-trained DQL was proposed to achieve the load-balancing goal.

CHAPTER 2

SYSTEM MODEL & ALGORITHM DESIGN

This chapter starts by presenting the system model along with the mathematical formulation of the MEC computation offloading problem. Next, the concepts of RL in computational offloading are introduced for the decision and load-balancing mechanisms and three RL-based strategies are proposed. The SLA is used in the first stage to solve the offloading decision problem, then, the QL and DQN algorithms to balance the offloaded workload at the edge.

2.1 System Model

We consider a MEC infrastructure of multiple interconnected nodes (“multi-edge-site") as $\mathcal{S} = \{1, 2, ..., S\}$ comprising small data centers each connected to a wireless AP to provide offloading services for the resource-constrained MDs of its coverage area, where MDs are able to offload computationally intensive tasks to reduce their energy consumption. Each ES $s \in \mathcal{S}$ has available computational resources F_s and bandwidth W_s given by the vector $\{F_s, W_s\}$.

The set of MDs covered by each ES s is denoted as $\mathcal{N}_s = \{1, 2, ..., N_s\}$ and are considered to be quasi-static for the examined offloading period, thus no MD mobility between different ESs is taken into account. Each MD $n_s \in \mathcal{N}_s$ is characterized by the vector $\{f_{n_s}, p_{n_s}\}$ denoting its computing capabilities and uplink transmission power respectively. An MD has one application task for execution which can either be processed locally (“on-device execution") or offloaded at the ES of coverage (“remote execution"), and to maintain simplicity, we assume a one-to-one relationship between a device and its task for a single offloading period, and henceforth, these terms will be used interchangeably.

Depending on the type of application executed by n_s, each task is defined by the vector $\{d_{n_s}, c_{n_s}, \tau_{n_s}, e_{n_s}\}$, where d_{n_s} specifies the amount of input data to be processed, c_{n_s} represents the workload, i.e., the total number of required CPU cycles, τ_{n_s} represents the maximum acceptable end-to-end delay and e_{n_s} stands for the maximum tolerable energy consumption for the task.

2.1.1 Computation Models

A task n_s can be executed either locally on the MD or via computation offloading on an ES. We introduce the binary variable $a_{n_s} \in \{0, 1\}$ which corresponds to the execution mode for task n_s; we have $a_{n_s} = 0$ for on-device execution and $a_{n_s} = 1$ for offloading to the attached ES. Consequently, we define an offloading decision vector

$$\mathcal{A}_s = \{a_1, a_2, ..., a_{N_s}\}$$

to account for all the MD devices connected to ES s.

2.1.1.1 Local Execution

For $a_{n_s} = 0$, the task n_s will be processed locally on the MD. Let f_{n_s} denote the computing capabilities (i.e., CPU cycles per second) of MD n_s, the on-device execution time and its corresponding energy consumption are calculated as follows

$$t_{n_s}^{lc} = c_{n_s} / f_{n_s} \tag{2.1}$$

$$E_{n_s}^{lc} = \kappa c_{n_s}$$

where κ is the consumed energy per CPU cycle in joules. Following (Jiang, Zhou, Li, Liu & Xu, 2020) and (Li *et al.*, 2018), we set $\kappa = 10^{-27}(f_{n_s})^2$.

Unlike other IoT devices, MDs have other processes running in the background that can consume significant power. In order to avoid the critical state of the battery, its level $battery_{n_s}^{level}$ has been set between 30% and 100%. We assume that the phone battery with full charge might have about 10000 joules, when energy is consumed, the battery gauge decreases to x percent, as calculated in the following equation. Hence, the remaining battery of MD n_s when executing locally is given by

$$battery_{n_s}^{lc} = battery_{n_s}^{level} - \frac{E_{n_s}^{lc} \times 100}{E^{max}} \tag{2.2}$$

where E^{max} is the maximum energy of the battery. We define a cost function as the weighted sum of the execution time and energy consumption. We denote β_1 and β_2 as their weights, respectively, and it satisfies $\beta_1 + \beta_2 = 1,\ 0 \leq \beta_1 \leq 1$ and $0 \leq \beta_2 \leq 1$. The weights specify the MD n_s preference on delay and energy consumption. Then, the total cost of local computing can be calculated as:

$$C_{n_s}^{lc} = \beta_1 t_{n_s}^{lc} + \beta_2 E_{n_s}^{lc} \tag{2.3}$$

2.1.1.2 Remote Execution

For $a_{n_s} = 1$, the task n_s will be offloaded to the attached ES s. In this case, its end-to-end delay, denoted by $t_{n_s}^{of}$, comprises two parts: i) the uplink transmission time, $t_{n_s}^{tr}$ and ii) the execution time at the MEC server, $t_{n_s}^{ex}$. Since the task output is usually much smaller than the size of input data, and the downlink rate from ES to MD is very high in practice, we neglect the transmission time and energy consumption for delivering the computed results (Avgeris *et al.*, 2022). Additionally, we assume that the wireless bandwidth of ES s, W_s, is equally allocated to the MDs that choose to offload to it. Then, with r_s^{of} being the number of offloaded tasks to s, the bandwidth assigned to MD n_s to upload its task input data to s is:

$$w_{n_s} = W_s / r_s^{of} \tag{2.4}$$

and based on that, the transmission time can be calculated by:

$$t_{n_s}^{tr} = d_{n_s} / w_{n_s} \tag{2.5}$$

The corresponding energy consumption for MD n during the transmission is then:

$$E_{n_s}^{of} = p_{n_s} t_{n_s}^{tr} \tag{2.6}$$

Regarding the offloading processing part, given again that the available resources of ES s, F_s, are equally allocated to the MDs that offload to it, the computing resources that site s allocates to task n_s can be calculated as:

$$F_s / r_s^{of}$$

and subsequently, the computation execution time is given by:

$$t_{n_s}^{ex} = (c_{n_s} r_s^{of}) / F_s \tag{2.7}$$

According to the above, the total end-to-end delay experienced by MD n_s in the case of remote execution becomes:

$$t_{n_s}^{of} = t_{n_s}^{tr} + t_{n_s}^{ex} \tag{2.8}$$

Next, we introduce the binary variable $k_{n_s}^{s'} \in \{0, 1\}$ to signify the transferring of the offloaded task n_s from ES s to s' ($k_{n_s}^{s'} = 1$), with $s \neq s' \in \mathcal{S}$, as well as the transferring decision vector.

$$K_{n_s} = \{k_{n_s}^1, k_{n_s}^2, ..., k_{n_s}^S\}, \forall n_s \in \mathcal{N}_s$$

This allows for formulating the transferring decision matrix for the tasks of each ES s as:

$$\mathcal{K}_s = \{K_1, K_2, ..., K_{N_s}\}^{S \times N_s}, \forall s \in \mathcal{S}$$

Before formulating the problem, we have to redefine the remote execution delay calculation 2.8 to take into consideration the additional delay introduced by ES-to-ES task transferring. First, let us replace r_s^{of} in 2.4 with

$$\sum_{n_s=1}^{\mathcal{N}_s} a_{n_s}$$

and

$$\sum_{n_s=1}^{\mathcal{N}_s} a_{n_s} + \sum_{s=1}^{\mathcal{S}} \sum_{n_{s'}=1}^{\mathcal{N}_{s'}} k_{n_{s'}}^{s}$$

for w_{n_s} calculation in 2.5 and 2.7, respectively.

Assuming that the propagation delay for the ES-to-ES communication, $t^{pr}_{n_{s,s'}}$ is proportional to the distance (number of hops) between the edge sites s and s', the remote execution delay 2.8 becomes:

$$t^{of}_{n_s} = t^{tr}_{n_s} + \sum_{s' \in S}^{s' \neq s} [(1 - k^{s'}_{n_s}) t^{ex}_{n_s} + k^{s'}_{n_s} (t^{pr}_{n_{s,s'}} + t^{ex}_{n_{s'}})] \tag{2.9}$$

Same as (2.2), the remaining battery of MD n_s when offloading is given as:

$$battery^{of}_{n_s} = battery^{level}_{n_s} - \frac{E^{of}_{n_s} \times 100}{E^{max}} \tag{2.10}$$

The total cost of MD n_s through task offloading is given by:

$$C^{of}_{n_s} = \beta_1 t^{of}_{n_s} + \beta_2 E^{of}_{n_s} \tag{2.11}$$

Based on (2.3) and (2.11), the weighted sum cost of all MDs in terms of completion time and energy consumption can be expressed as

$$C_{all} = \sum_{n_s=1}^{N_s} (1 - a_{n_s}) C^{lc}_{n_s} + a_{n_s} C^{of}_{n_s} \tag{2.12}$$

For the convenience of understanding, the mathematical notations used in this thesis are summarized in Table 2.1.

Table 2.1 List of Notations

Symbol	Definition
$\mathcal{S}$	The set of the interconnected ESs
$\mathcal{N}_s$	The set of MDs covered by each ES s
F_s	The available computational resources
W_s	The available wireless bandwidth of ES s
f_{n_s}	The computing capabilities of MD $n_s \in \mathcal{N}_s$, i.e., CPU cycles per second
p_{n_s}	The uplink transmission power of MD $n_s \in \mathcal{N}_s$
d_{n_s}	The amount of input data to be processed
c_{n_s}	The workload, i.e., the total number of CPU cycles required
τ_{n_s}	The maximum tolerable end-to-end delay for the task
e_{n_s}	The maximum tolerable energy consumption for the task
$t_{n_s}^{lc}$	The local execution time
$E_{n_s}^{lc}$	The local energy consumption for the MD
$battery_{n_s}^{level}$	The mobile battery level
$battery_{n_s}^{lc}$	The remaining mobile battery when executing task locally
$battery_{n_s}^{of}$	The remaining mobile battery when offloading
E^{max}	The maximum energy of the mobile battery
$t_{n_s}^{of}$	The total end-to-end delay when offloading
$E_{n_s}^{of}$	The energy consumption for MD n when offloading
$t_{n_s}^{tr}$	The uplink transmission time
$t_{n_s}^{ex}$	The execution time at the MEC server
$t_{n_{s,s'}}^{pr}$	The propagation delay for the ES-to-ES communication
r_s^{of}	The number of offloaded tasks to s
w_{n_s}	The bandwidth assigned to MD n_s to upload its task input data to s
κ	The consumed energy per CPU cycle in joules
p_n^s	The uplink transmission power of MD n

Symbol	Definition
a_{n_s}	Binary variable representing the execution mode for task n_s
$k_{n_s}^{s'}$	Binary variable to signify the transferring of the offloaded task n_s from ES s to s'
$\mathcal{A}_s$	The offloading decision vector for all the MD devices connected to ES s
K_{n_s}	The transferring decision vector
$\mathcal{K}_s$	The transferring decision matrix for the tasks of each ES s
$C_{n_s}^{lc}$	The total cost of local computing
$C_{n_s}^{of}$	The total cost of remote computing
C_{all}	The weighted sum cost of all MDs in terms of completion time and energy consumption
$p_{n_s,a_{n_s}}$	Probability that an agent n_s select action a_{n_s}
$R_{n_s}^{(i)}$	Reward of each MD n_s in the i-th iteration
β_1, β_2	Weights selected based on the device configuration and the application needs
Z_1, Z_2	Gain coefficients which help bring the respective terms of reward equation

2.1.2 Problem Formulation and Analysis

Edge computing resources are limited to micro data centers typically consisting of only a few servers (Avgeris *et al.*, 2022). As a result, it is not uncommon for an ES to become overloaded when handling offloaded requests. To address this challenge and maintain a high level of QoS for MDs, an effective approach is to balance the offloaded workload by transferring tasks from overloaded ESs to underloaded ones for execution. In our work, we explore two task-offloading opportunities:

1. MD-to-ES, this refers to the typical computational task offloading between the user device and the edge site.
2. ES-to-ES, this concerns the cooperation between different sites within the infrastructure, aimed at enhancing the effectiveness of the first phase (MD-to-ES).

We formulate the joint task offloading problem for MEC infrastructures as an optimization problem. Our objective is to minimize the weighted sum cost in terms of the completion time and the energy consumption for all the MDs. Under the constraint of maximum tolerable delay and energy consumption, the problem can be formulated as follows:

$$\min_{\mathcal{A}_s, \mathcal{K}_s} \quad \sum_{s\in\mathcal{S}} \sum_{n_s\in\mathcal{N}_s} [(1-a_{n_s})(\beta_1 t^{lc}_{n_s} + \beta_2 E^{lc}_{n_s}) + a_{n_s}(\beta_1 t^{of}_{n_s} + \beta_2 E^{of}_{n_s})] \tag{2.13a}$$

$$\text{s.t.} \quad a_{n_s} \in \{0,1\}, \forall n_s \in \mathcal{N}_s, \forall s \in \mathcal{S}, \tag{2.13b}$$

$$k^{s'}_{n_s} \in \{0,1\}, \forall n_s \in \mathcal{N}_s, \forall s, s' \in \mathcal{S}, \tag{2.13c}$$

$$\sum\nolimits_{s'\in S}^{s'\neq s} k^{s'}_{n_s} \leq 1, \forall n_s \in \mathcal{N}_s, \forall s \in \mathcal{S}, \tag{2.13d}$$

$$(1-a_{n_s})t^{lc}_{n_s} + a_{n_s} t^{of}_{n_s} \leq \tau_{n_s}, \forall n_s \in \mathcal{N}_s, \tag{2.13e}$$

$$(1-a_{n_s})E^{lc}_{n_s} + a_{n_s} E^{of}_{n_s} \leq e_{n_s}, \forall n_s \in \mathcal{N}_s, \tag{2.13f}$$

where the weights $\beta_1, \beta_2 \geq 0, \beta_1 + \beta_2 = 1$, are selected based on the device configuration and the application needs. Constraint (2.13b) represents the computation offloading decision, while (2.13c) the task transferring between sites. Constraint (2.13d) ensures that a task is transferred to at most one server, while (2.13e) guarantees that the task completion time should not exceed the maximum tolerable delay τ_{n_s}, either when executed locally or remotely. Finally, constraint (2.13f) ensures that the task energy threshold is respected, with e_{n_s} being the maximum tolerable energy consumption. Since task offloading decision set $\mathcal{A}_s$ is composed of binary variables, both the feasible set and the objective function of Problem (2.13) are not convex, making it challenging to solve the problem. Fortunately, as demonstrated in the literature (Li *et al.*, 2018)

this kind of NP-hard problem can be easily solved effectively by applying RL methods rather than conventional optimization methods. (Avgeris *et al.*, 2023)

2.2 Energy-Aware Computational Offloading Mechanism

Leveraging the power of RL, a two-stage computational offloading mechanism is proposed to simultaneously minimize the delay and energy consumption at the MD layer while achieving load balancing at the distributed edge infrastructure. In the first stage, the offloading decision problem is solved using a decentralized approach based on SLA. For the second stage, we initially propose a value-iteration-based RL approach utilizing QL, to balance the offloaded workload at the edge and minimize the processing delay for the offloaded tasks. Later, to overcome the curse of dimensionality, we propose a DQN method that combines deep learning and RL to solve the same problem. These two stages are combined into a multi-round cooperative computational offloading mechanism which iteratively optimizes the offloading and transferring decisions until converging to a stable solution.

2.2.1 First Stage: MD-to-ES Task Offloading

We assume that each MD $n_s \in N_s$ acts as an agent where each offloading decision $a_{n_s} \in \{0, 1\}$ corresponds to the SLA action. Then, each agent's chosen action in iteration i is based on a probability distribution $p_{n_s,a_{n_s}} \in \mathbb{P}(\{0, 1\})$ kept over the action-set and in each iteration; $p_{n_s,a_{n_s}}(i)$ is the probability that an agent n_s select action a_{n_s} in iteration i, and $\mathbb{P}(\{0, 1\})$ is the set of probability distributions over the available action set. Initially, all action probabilities would be equal and hence the action is randomly chosen and the probabilities are updated in every iteration. Based on the problem formulation in (2.13a), we define the reward of each MD n_s in the i-th iteration as:

$$R_{n_s}^{(i)} = \beta_1 \cdot (1 - \frac{1}{1 + e^{-Z_1[(1-a_{n_s}^{(i)})t_{n_s}^{lc} + a_{n_s}^{(i)} t_{n_s}^{of} - \tau_{n_s}]}}) + \beta_2 \cdot (1 - \frac{1}{1 + e^{-Z_2[(1-a_{n_s}^{(i)})E_{n_s}^{lc} + a_{n_s}^{(i)} E_{n_s}^{of} - e_{n_s}]}}), \in [0, 1], \tag{2.14}$$

where $Z_1, Z_2 \in \mathbb{R}^+$ are gain coefficients which help bring the respective terms of Eq. (2.14) close to 1 when the total delay is less than τ_{n_s} and the energy consumption less than e_{n_s}, and close to 0 otherwise. In this way, we embody constraints (2.13e) and (2.13f) into the reward function. The update rule of the SLA is based on the idea that if an action is selected by the agent n_s in iteration i, and the reward value $R_{n_s}^{(i)}$ received is high, then the probability of choosing this action in the next iteration of the learning procedure increases, with regards to the magnitude of the perceived reward.

The commonly used update rule in the research literature is the *linear reward-inaction* (LRI) defined as follows:

$$p_{n_s,a_{n_s}}^{(i+1)} = \begin{cases} p_{n_s,a_{n_s}}^{(i)} + b \cdot R_{n_s}^{(i)} \cdot (1 - p_{n_s,a_{n_s}}^{(i)}), & \text{when } a_{n_s}^{(i+1)} = a_{n_s}^{(i)} \\ p_{n_s,a_{n_s}}^{(i)} - b \cdot R_{n_s}^{(i)} \cdot p_{n_s,a_{n_s}}^{(i)}, & \text{otherwise} \end{cases} \tag{2.15}$$

The learning rate parameter, $0 < b < 1$, controls the convergence of this stage. The system converges to a stable solution when at least one state probability is close to 1, for each n_s. Algorithm 2.1 explains the procedure of SLA-based decision-making.

Algorithm 2.1 Learning Automata Offloading Algorithm

1 Initialize the action probabilities.
2 Select an action randomly.
3 Compute the total completion time, remaining battery, and the reward using equations 2.1, 2.2, 2.9, 2.10 and 2.14.
4 Update the action probabilities.
5 **if** *probability of any action* ≈ 1 **then**
6 | The chosen action is the optimal one.
7 **else**
8 | Select the action with the highest probability. Go to step 3.
9 **end if**

2.2.2 Second Stage: ES-to-ES Transferring

As the offloading decisions of the first stage are made in a distributed fashion, chances are that an ES might become overloaded with tasks, which can potentially hinder the processing times. In this direction, exploring a cooperative solution among the ESs, in the form of task transferring, is vital to satisfy the QoS of the users. This step is performed right before calculating the reward $R_{n_s}^{(i)}$ in Eq. (2.14) and specifically produces the remote execution delay $t_{n_s}^{of}$.To solve this problem, we first employ a QL-based algorithm (Algorithm 2.2), and subsequently, we implement a DQN-based algorithm (Algorithm 2.3), incorporating the following elements:

- **State:** we define as state a vector that contains the available computational capacity in each ES, after considering the MDs' offloading decisions a_{n_s} of the first stage;

$$\sigma = \{F_s - \sum_{n_s=1}^{N_s} a_{n_s} c_{n_s} | s \in \mathcal{S}\}^{1\times S}$$

  A state is terminal if it contains only zeros or negative values.

- **Action:** as an action we use the task transferring decision matrix introduced in Section 2.1, allowing, however, only one task transferring in the infrastructure per action;

$$\alpha = \{\mathcal{K}_s | s \in \mathcal{S}, \sum\nolimits_{s\in S} \sum\nolimits_{n_s \in \mathcal{N}_s} \sum\nolimits_{s'\in S}^{s'\neq s} k_{n_s}^{s'} \leq 1\}^{S\times(S\times N_s)}$$

- **Reward:** as we opt for driving our edge infrastructure towards a balanced workload distribution, the first term of the reward in this stage is the difference between the current (σ) and the next (σ') state's sum of positive values, i.e., sites where the offloaded workload is greater than the available capacity. The second term penalizes the transferring of tasks to remote ESs, an action which increases the additional propagation delay:

$$\mathcal{R}_{\sigma,\sigma',\alpha} = \delta_1 (\sum_{s\in\mathcal{S}}^{\sigma_s>0} \sigma_s - \sum_{s\in\mathcal{S}}^{\sigma'_s>0} \sigma'_s) + \delta_2 (\sum_{s\in S} \sum_{n_s\in\mathcal{N}_s} \sum_{s'\in S}^{s'\neq s} \alpha_{n_s,s'} t_{n_s,s'}^{pr})^{-2}, \tag{2.16}$$

  where $\delta_1, \delta_2 \in \mathbb{R}^+$, are properly selected weights that balance the contribution of the two terms in the reward.

2.2.2.1 QL-based Task Transferring Training

We first try to achieve load balancing at the edge layer using a QL approach. Algorithm 2.2 is executed only once and offline, it produces the matrix $Q(\sigma, \alpha)$ which contains the expectation of the long-term reward (calculated through the Bellman equation for TD learning, line 7) for each infrastructure state and transferring decision, after being trained on numerous state-action pairs; ζ is the learning rate that satisfies $0 < \zeta < 1$, while $0 < \gamma < 1$ is the discount factor, used to model the uncertainty in the future actions.

Algorithm 2.2 QL-based Task Transferring Training

```
1  Initialize with zeros: Q(σ, α).
2  for each episode do
3      Choose a random infrastructure state σ.
4      while current state is not terminal do
5          Select a task transferring action α.
6          Execute α, produce σ′ and collect R_{σ,σ′,α}.
7          Q(σ, α) ← Q(σ, α) + ζ(R_{σ,σ′,α} + γ max_{α′} Q(σ′, α′) − Q(σ, α))
8          σ ← σ′
9      end while
10 end for
```

As the number of ESs and MDs becomes larger, training Algorithm 2.2 on a sufficient number of episodes becomes a tedious and bulky task, as the possible infrastructure states grow exponentially in size. To overcome this, we propose the use of a DNN to estimate $Q(\sigma, \alpha)$, which constitutes the basic idea behind DQN and is briefly presented in Algorithm 2.3 in the following subsection.

2.2.2.2 DQN-based Task Transferring Training

DQN is a popular and foundational RL algorithm that combines the principles of QL and DNNs to approximate and learn the optimal action-value function in RL problems. We make use of experience replay to improve the sample efficiency and stability of training, while we adopt the ϵ-greedy policy for the task transferring action selection since it gives us a better balance

between exploration and exploitation. To update the weights of the Q network, we use the Stochastic Gradient Descent (SGD) algorithm.

Stochastic Gradient Descent: Gradient descent is a form of gradient-based learning in which the Neural Network (NN) learns by stepping toward a local minimum. SGD is an extension of the gradient descent algorithm, it begins by calculating the gradient of the loss function with respect to the model's parameters. The gradient essentially tells us how the loss would change if we made small adjustments to each parameter. SGD is computationally efficient as it only makes use of a single sample selected randomly from the dataset to perform gradient descent. During each iteration with a mini-batch, SGD adjusts the model's parameters to reduce the loss by subtracting a fraction of the gradient called the learning rate from the current parameter values. The learning rate controls the size of the steps taken during each update. This process is repeated for several rounds called epochs, wherein all the mini-batches are processed, allowing the model to progressively refine its parameters. (Li, 2022; Omland, 2022)

Algorithm Description: We start by randomly initializing the Q-network Q and the target network $\hat{Q}$. Additionally, we initialize an ER memory $\mathcal{D}$ to store past experiences. During the sampling phase, the exploration-exploitation trade-off is adjusted using an exploration rate ϵ, which is gradually decreased favoring exploitation of the best-known actions using an ϵ-decay strategy (line 4). A task transferring action α is chosen using an ϵ-greedy policy and the transition $(\sigma, \sigma', \alpha, \mathcal{R}, done)$ is stored in the ER memory $\mathcal{D}$.

Upon accumulating a sufficient number of experiences, we transition to the learning phase. We sample a minibatch of M transitions from the ER memory. For each transition, in the minibatch, If the agent reaches the final state, we use the immediate reward as the target value. Otherwise, we determine the target value by considering both the immediate reward and the expected future value of the next state based on the best available actions (lines 11-18).

We quantify the training progress by measuring the loss $\mathcal{L}$ as the Mean Squared Error (MSE) between the Q-values predicted by the Q-network and the target values (line 19). The Q-network parameters are updated using SGD to enhance training. Additionally, for every C step, we

Algorithm 2.3 DQN-based Task Transferring Training

1 Initialize network Q and target network $\hat{Q}$ randomly.
2 Initialize experience replay memory $\mathcal{D}$.
3 **while** *not converged* **do**
    `// Sampling Phase`
4 $\epsilon \leftarrow$ set new epsilon with ϵ-decay.
5 Select a task transferring α using ϵ-greedy policy.
6 Execute α, produce σ' and collect $\mathcal{R}_{\sigma,\sigma',\alpha}$.
7 $done \leftarrow \sum_{s\in\mathcal{S}}^{\sigma_s>0} \sigma_s == 0$
8 Store transition $(\sigma, \sigma', \alpha, \mathcal{R}, done)$ in $\mathcal{D}$.
9 **if** *enough experiences in* $\mathcal{D}$ **then**
        `// Learning Phase`
10 Sample minibatch of M transitions from $\mathcal{D}$.
11 **for** *each* $(\sigma_m, \sigma'_m, \alpha_m, \mathcal{R}_m, done_m) \in M$ **do**
12 **if** $done_m$ **then**
13 $y_m \leftarrow \mathcal{R}_m$
14 **end if**
15 **else**
16 $y_m \leftarrow \mathcal{R}_m + \gamma max_{\alpha'}\hat{Q}(\sigma'_m, \alpha')$
17 **end if**
18 **end for**
19 Loss $\mathcal{L} = \frac{1}{M}\sum_{m=0}^{M-1}(Q(\sigma_m, \alpha_m) - y_m)^2$
20 Update Q using SGD by minimizing $\mathcal{L}$.
21 Every C steps, copy weights from Q to $\hat{Q}$.
22 **end if**
23 **end while**

synchronize the Q-network and the target network by copying weights from Q to $\hat{Q}$, promoting stability in training (line 21).

2.2.3 Two Stage Cooperative MEC Computational Offloading

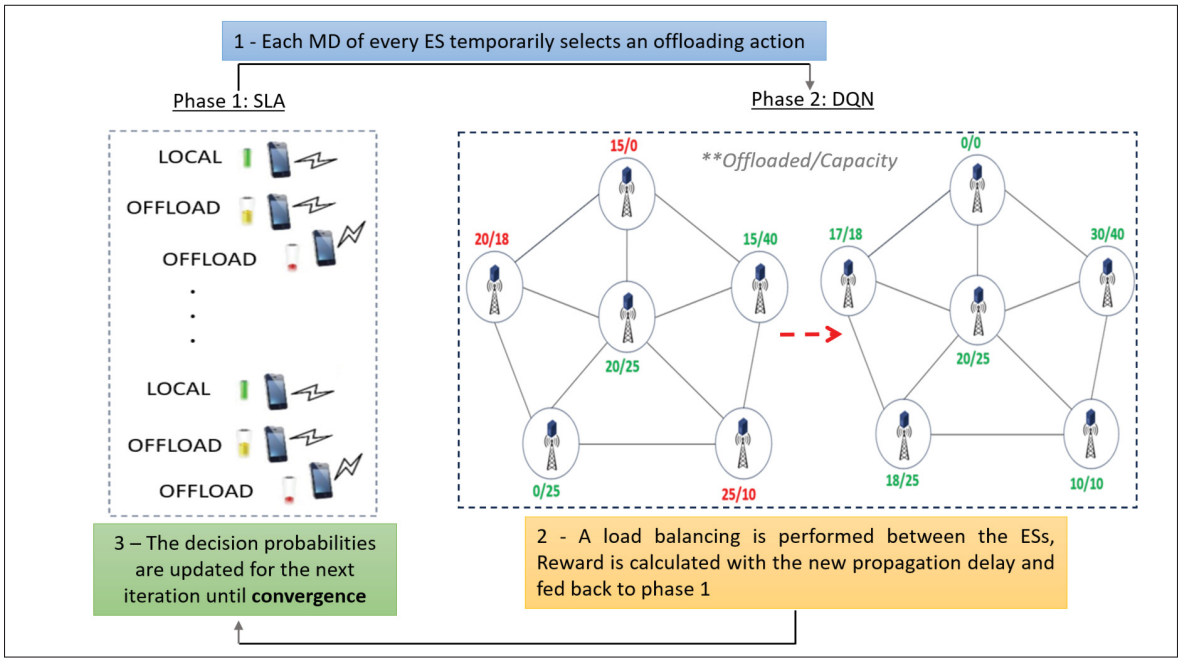


Figure 2.1 Two-stage Cooperative MEC Computational Offloading

To sum up, the proposed cooperative MEC offloading mechanism operates in two steps per iteration, as illustrated in Figure 2.1 and Algorithm 2.4. Each MD of every ES temporarily selects whether to offload their current task or execute it locally, based on the probabilities of the SLA algorithm (Section 2.2.1), and a load balancing is performed between the ESs to improve the response time of the offloaded tasks, based on an offline-trained NN (Section 2.2.2). The outcomes of this decision are fed back to the first stage to update the rewards and probabilities for the next iteration until convergence to a stable solution that satisfies the delay and energy consumption constraints is achieved.

Algorithm 2.4 Two-stage Cooperative MEC Computational Offloading

// Offline

Train the Q network using Algorithm 2.3.

// First Stage (Online)

Initialize the offloading decision probabilities.

$i \leftarrow 0$

while *not converged* **do**

  for *each ES* $s \in \mathcal{S}$ ***and*** *MD* $n_s \in \mathcal{N}_s$ **do**

    Select offloading action $a_{n_s}^{(i)}$ based on $p_{n_s,a_{n_s}}^{(i)}$.

  end for

  // Second Stage (Online)

  Calculate infrastructure state σ.

  while $\sum_{s\in\mathcal{S}}^{\sigma_s<0} \sigma_s < 0$ **do**

    Select transferring action α with the highest Q value. Execute α and produce σ'.

    $\sigma \leftarrow \sigma'$

  end while

  for *each ES* $s \in \mathcal{S}$ ***and*** *MD* $n_s \in \mathcal{N}_s$ **do**

    Calculate reward $R_{n_s}^{(i)}$ using Eq. (2.14).

    Update decision probabilities $p_{n_s,a_{n_s}}^{(i)}$.

  end for

  $i \leftarrow i + 1$

end while

This chapter presented our solution, a two-stage cooperative RL-based scheme for energy-aware computational offloading at the edge of the network. In the first stage, distributed and autonomous task offloading decision-making in the MDs is enabled, based on SLA. The second stage devises a QL approach to allow for task transferring between the edge sites and alleviate potential

overloading phenomena. In the next chapter, we will compare our solution with a well-known similar work in the literature and present the numerical results.

CHAPTER 3

PERFORMANCE EVALUATION

This chapter presents the performance evaluation of our proposed framework. We start by comparing the training time between the DQN and the plain QL algorithms used for the second stage of our solution. Next, we examine the online convergence behavior of our algorithm. Finally, we perform a comparative evaluation against a well-known work from the literature, Li et al. (Li *et al.*, 2018), and two baseline algorithms, one where all the tasks are executed on the MD ("On-device") and one where all the requests are offloaded ("Remote").

3.1 Benchmark solution

In this part, we will briefly explain the work of (Li *et al.*, 2018) which is similar to our work. The authors designed two RL-based solutions for a multi-user MEC computation offloading system with the objective of minimizing the sum cost, which is a combination of delay and energy consumption for all user equipment in the system.

They started by introducing a QL-based approach. Furthermore, the authors introduce a pre-classification step before the learning process to manage the potentially explosive growth of the action space as the number of User Equipment (UE) increases. For each UE, the system checks if in the case of local computing the local execution delay is less or equal to the maximum tolerable delay of the task. If this constraint cannot be satisfied locally, the UE is classified as an "offloading UE" for the current decision period and the constraint ensures that the allocated resource for each UE is sufficient to meet the computation and communication requirements of the user without violating the latency constraints. This approach helps manage the complexity of the RL problem, especially when dealing with a large number of UEs in the system. It reduces the possible value of the offloading decision vector and computational resource allocation to limit the action space of the RL agent.

However, even by applying this pre-classification, the authors recognized that the number of possible actions can become unwieldy as the number of users increases, making it challenging to

compute and store action values (Q-values). To address this issue, they introduced the utilization of DQN which relies on DNN to estimate the Q-values, offering a more efficient way to manage the complexities associated with a large action space.

3.2 Evaluation parameters

The two algorithms training were performed on a MacBook Air with a 16-core Neural Engine M2 chip and 16GB of RAM. We examined a MEC infrastructure composed of $2-25$ ESs, each connected to a varying number of $5-10$ MDs. The available resources of each ES range between $\{8GHz, 48Mbps\}$ and $\{10GHz, 50Mbps\}$, while the MDs possess capabilities ranging from $\{1GHz, 498mW\}$ and $\{2GHz, 502mW\}$. Each MD executes a single application with the following attributes: $\{1000kbits, 2000Mcycles, 1080ms, 4J\}$. The energy consumption per CPU cycle is set to $\kappa = 10^{-27}(f_{n_s})^2$ following (Li *et al.*, 2018).

In the first stage of the algorithm, we strive for a balanced reward between energy consumption and delay (Eq. (2.14)) achieved by setting the parameters β_1 and $\beta_2 = 0.5$ to 0.5, and Z_1 and Z_2 to 1. The learning parameter is assigned a value of $b = 0.6$ and we assume that convergence is achieved when one action probability for every MD is ≥ 0.9.

Moving on to the second stage, we establish reward weights δ_1 as 0.0005 and δ_2 as 50 to balance the contribution of the load balancing and propagation delay minimization factors. The learning rate is set to $\zeta = 0.1$ and the discount factor to $\gamma = 0.9$. We employ a neural network consisting of 2 hidden layers, each with 700 and 600 neurons, respectively. The experience replay memory size is set to 10^5 and the minibatch size $M = 256$. The target network's weights, denoted as $\hat{Q}$ are updated every 4 steps. We assess the training convergence by monitoring the average improvement in the reward (Eq. (2.16)) over the last 100 episodes, concluding that convergence has been reached when this improvement is less than 1%.

3.3 Numerical Results

3.3.1 Training time comparison

As seen below in Fig. 3.1, the QL algorithm training duration grows exponentially with the number of ESs, which makes it an unrealistic alternative when this number is greater than 5 (equivalent to more than 7 days of real-time training). On the other hand, DQN provides a far more tractable training process, with its duration showcasing a linear behavior to the number of ESs. Although the training is performed only once and offline, scalability is still important for applying our framework to larger infrastructures, which makes DQN a far more preferable choice.

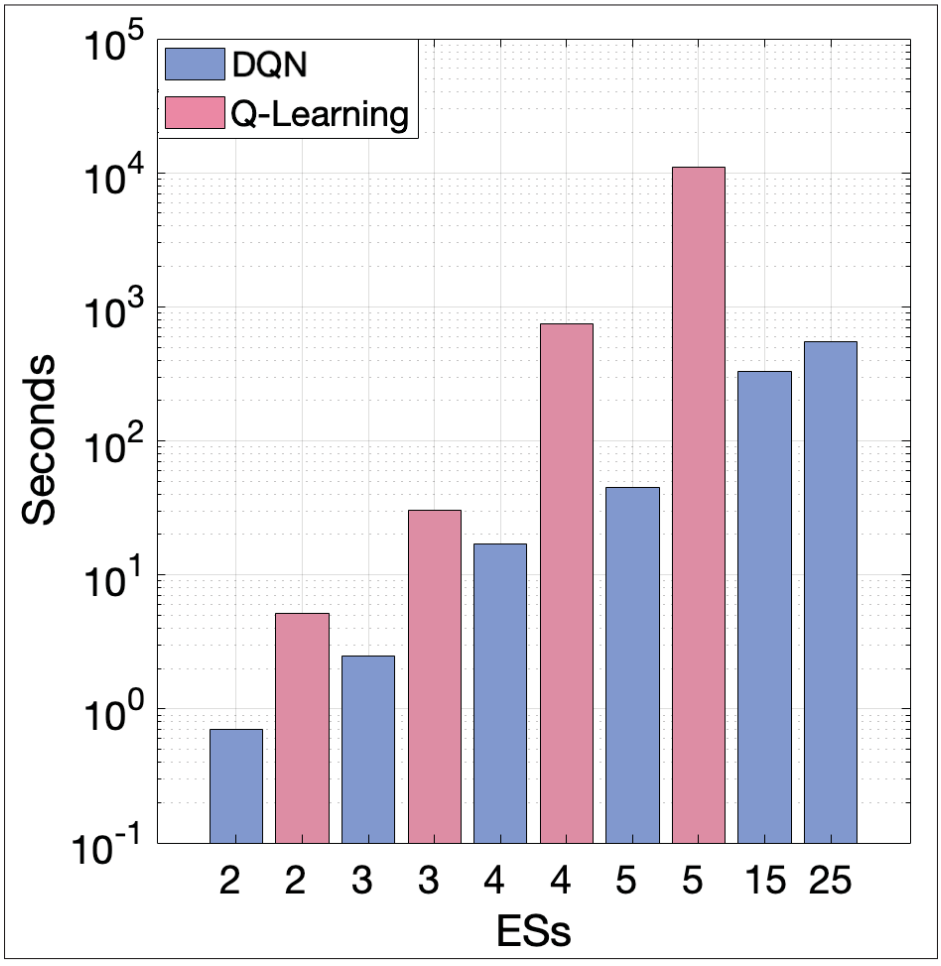


Figure 3.1 Training time

3.3.2 Online convergence behavior

Fig. 3.2 illustrates the average collected reward (Eq. (2.14)), for various infrastructure settings over 25 iterations, for 100 experiment repeats. We observe that when fixing the average number of MDs per site, increasing the number of ESs tends to yield a higher average reward for each MD, as more computational resources become available in the infrastructure. On the other hand, when fixing the number of ESs, increasing the number of average MDs per ES naturally results in lower rewards, as the competition for the available computational resources becomes stiffer. In any case, convergence to a stable solution is reached after 25 iterations which translates to less than $1sec$ of execution time, making our framework effectively a real-time decision-making tool.

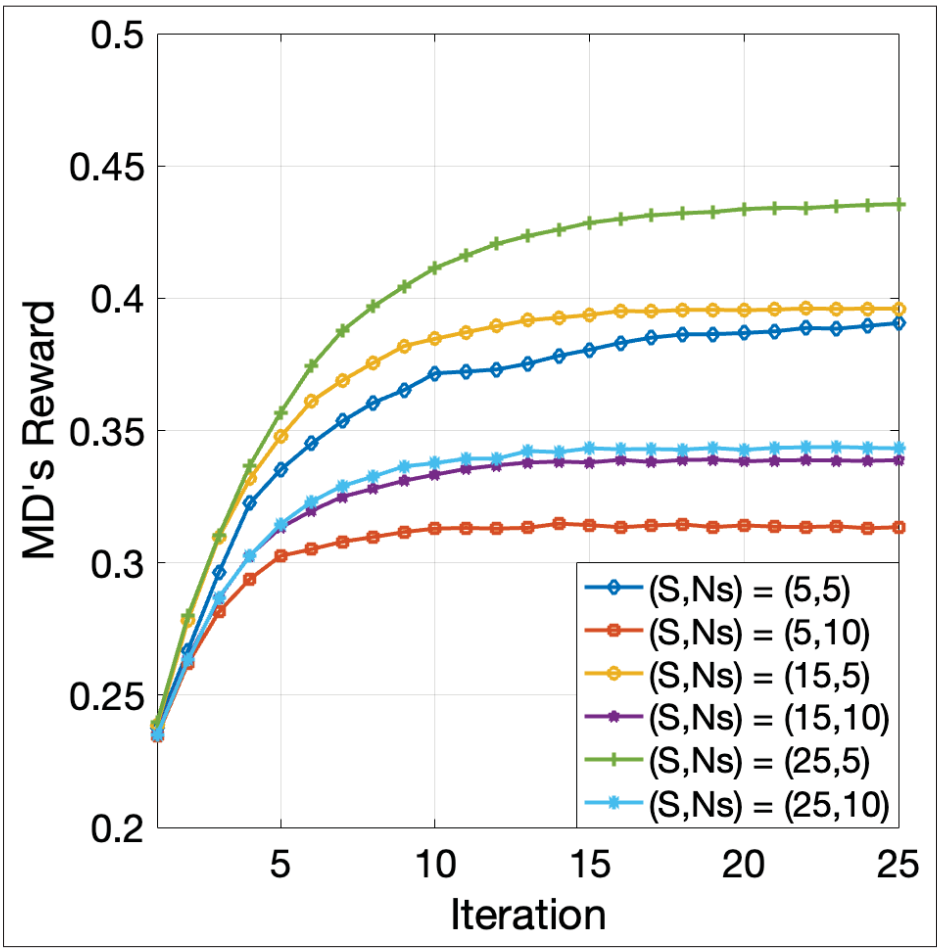


Figure 3.2 Convergence Behavior

3.3.3 Comparative evaluation

In this scenario, 25 ESs were considered, each one having an average of 7 MDs connected to it, which made some ESs overloaded when fully offloading. That is why we can see in Fig. 3.3 that the Remote's average achieved delay is the highest. We observe that our proposed solution manages to overcome this issue by transferring requests from the overloaded to the underloaded ESs. On the other hand, the algorithm in (Li *et al.*, 2018) selects the on-device execution for some MDs connected to the overloaded ESs, resulting in a slightly higher average delay.

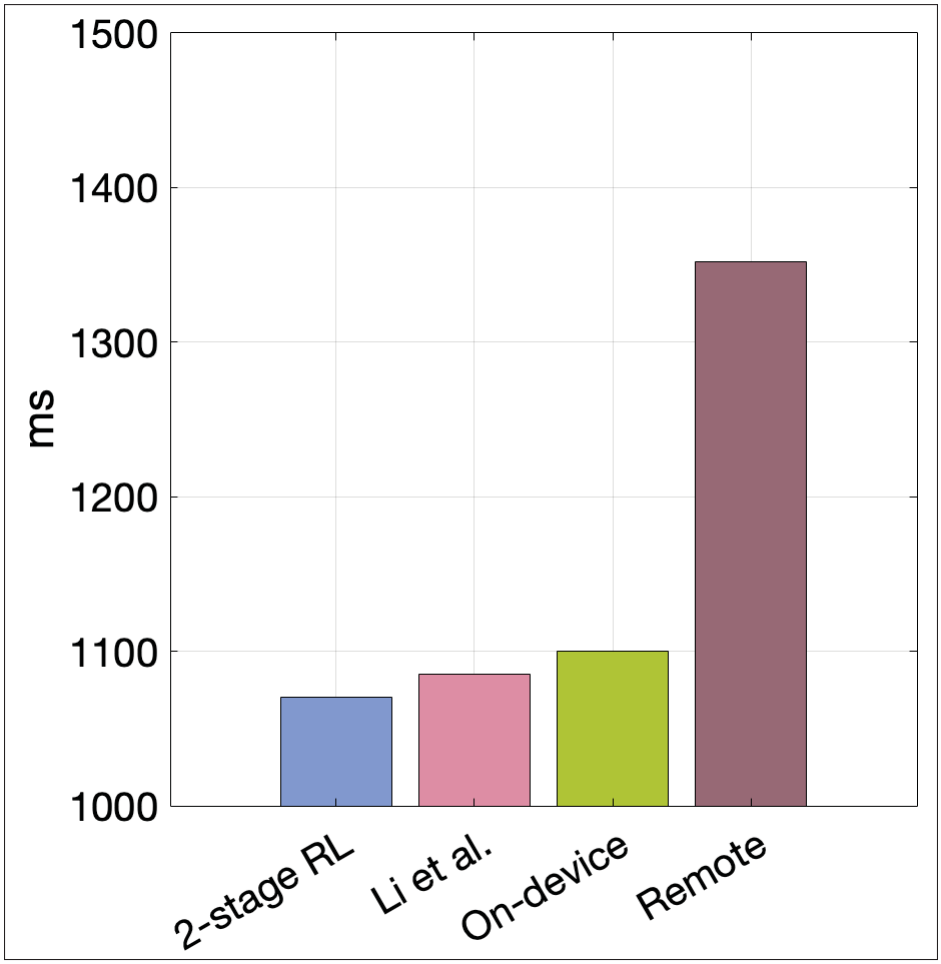


Figure 3.3 Benchmarking: Delay

Regarding energy consumption, as we can see in Fig. 3.4, the On-device execution performs the worst, as expected, and the proposed solution being as energy efficient for the MDs as the

Remote algorithm. The algorithm in (Li *et al.*, 2018) again scores slightly worse in this metric, as instead of transferring some tasks to underloaded sites, it selects the on-device execution. All in all, the presence of the load balancing mechanism in our framework, makes it capable of exploring the execution alternatives in the infrastructure, for MDs connected to overloaded ESs, achieving a better delay and energy consumption compared to typical on-device and/or remote execution solutions.

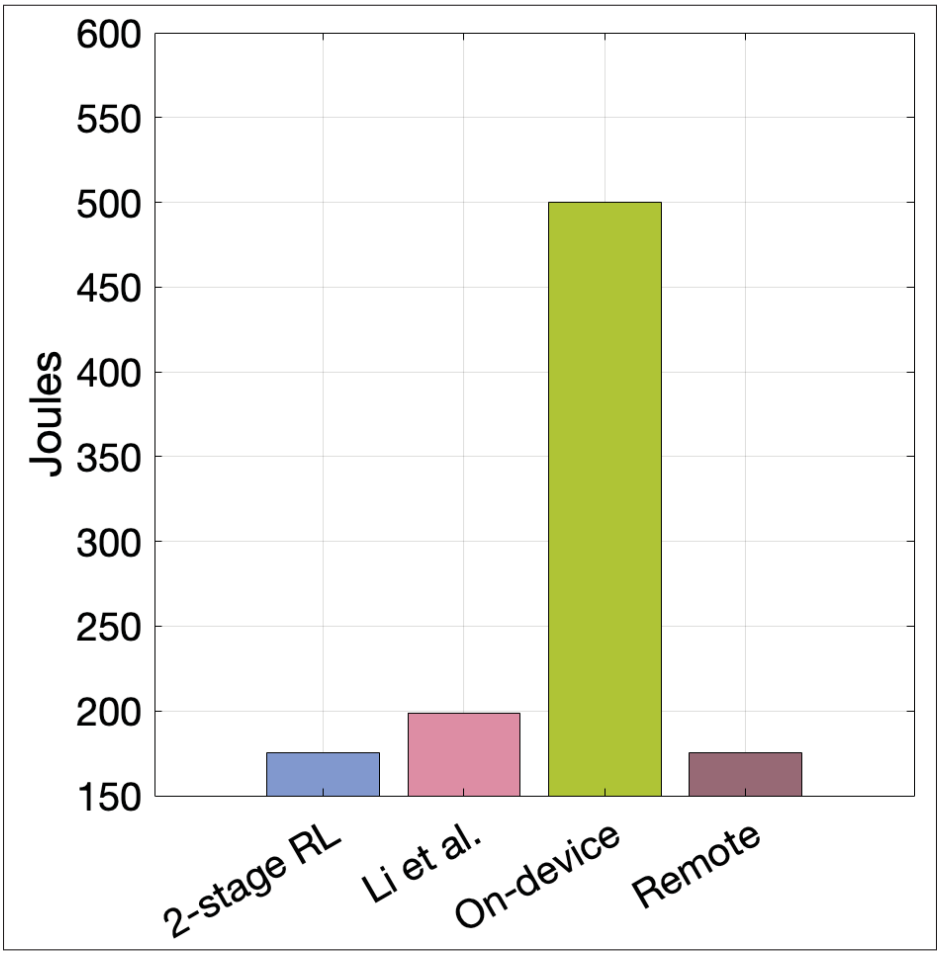


Figure 3.4 Benchmarking: Energy

3.4 Result discussion

Through rigorous experimentation, we showed that the proposed framework outperforms the baseline solutions including one where all tasks are executed on the MDs and another where all requests are offloaded to a remote server, and a well-known similar work in the literature in terms of both delay and energy efficiency. Our investigation into training time comparisons between the DQN and plain QL algorithms has illuminated the critical role of scalability in decision-making processes within MEC infrastructures. As we navigate the complexities of decision-making in the context of MEC infrastructures, it becomes evident that the number of possible states and actions can increase exponentially with the expansion of ESs and MDs. This leads to what is often termed the "curse of dimensionality," a scenario where the computational requirements for implementing QL become exceedingly burdensome.

The linear and tractable nature of DQN training process, even when dealing with an increasing number of ESs, positions it as a compelling choice for real-world implementation. Moreover, our examination of online convergence behavior has underscored the remarkable efficiency of our algorithm. It rapidly converges to stable solutions, this solidifies its position as an indispensable real-time decision-making tool for managing computational resources in dynamic MEC environments.

The comparative evaluation against a well-established similar work and baseline algorithms has emphasized the pragmatic efficacy of our proposed solution. Significant enhancements were observed when it came to reducing both end-to-end delay and energy consumption in contrast to typical on-device and remote execution solutions. Conversely, the algorithm of the prior literature work prioritizes on-device execution for specific MDs connected to overloaded ESs over task redistribution to the underloaded ones resulting in a slightly higher average delay and energy consumption.

CONCLUSION AND RECOMMENDATIONS

This thesis has addressed the critical challenge of task-offloading decision-making in MEC environments adopting the RL and DRL-based techniques. In the context of our work, our focus extends to a multi-user, multi-site MEC infrastructure with the core aim to simultaneously diminish both the end-to-end delay of applications and the energy usage of MDs while they perform computationally demanding tasks. Together, these two metrics form the sum cost of the entire system and support the fundamental user experience. Orchestrating these objectives to ensure a seamless fusion of performance optimization, user satisfaction, and energy efficiency is challenging.

To address this challenge, we introduced a novel two-stage cooperative, RL-based mechanism, designed to address the critical challenges of energy-aware computational offloading at the network edge. In the first stage of our approach, MDs autonomously make distributed task-offloading decisions using an iterative RL-based mechanism where each MD functions as SLA. In each iteration of this process, individual MDs autonomously determine whether to offload their tasks to the attached ES or execute the tasks locally. This stage enabled intelligent, on-device decision-making for offloading, aiming to optimize its energy consumption.

The second stage of our framework ensures efficient and dynamic task allocation across the edge infrastructure. We integrated an ES-to-ES cooperative offloading mechanism aiming to balance the workload across the infrastructure by proactively transferring tasks from overloaded ESs to underloaded ones. To this end, we first implemented a QL mechanism, but when the number of ESs and MDs increased, training QL with a substantial number of episodes became a laborious and cumbersome process, due to the exponential expansion in the number of potential infrastructure states. To overcome this, an offline-trained DQN was employed at the end of each iteration in the first stage. The updated processing delays are subsequently fed back to the MDs, enabling them to reconsider their offloading decisions.

The culmination of our work is the merging of these two stages into a multi-round collaborative computational offloading mechanism. This mechanism iteratively enhances the decisions made by both the MDs and the ESs, leading to the stable convergence of the optimization problem. The collective actions of MDs and ESs are coordinated to achieve the most favorable trade-off between processing delay and energy consumption.

As discussed in the section 3.4, our research has yielded compelling results showcasing that the proposed framework outperformed both the baseline solutions a well-known prior work in the field. Noteworthy improvements in terms of reducing end-to-end delay and energy consumption were observed. Our findings reveal that the incorporation of the load balancing mechanism in our framework empowers it to explore various execution alternatives within the infrastructure for MDs linked to overloaded ESs, ultimately achieving superior outcomes in terms of both delay and energy consumption. This accomplishment validates the effectiveness of our two-stage cooperative RL-based approach in tackling the crucial challenges related to computational offloading.

However, despite our significant outcomes, a notable limitation of our work lies in the static nature of our current framework. We must acknowledge that our approach failed to consider the dynamic aspects associated with users' movements within the network, a vital aspect of real-world scenarios where mobile users are in constant motion. This can significantly impact the MEC environment's state and resource allocation requirements. To enhance the effectiveness of our framework, it would be beneficial to integrate users' mobility patterns into our decision-making process. Additionally, we must address the challenge of increased dimensionality in the state space. In future work, we aim to mitigate these limitations by incorporating user mobility patterns into our decision-making process. Additionally, we intend to delve into the exploration of alternative learning techniques capable of handling the expanding complexities of state spaces, ensuring that our solutions remain adaptable and practical in dynamic MEC

environments. By doing so, our objective is to enhance the adaptability of our approach to the dynamic and practical nature of edge computing scenarios. Furthermore, we intend to delve into the exploration of alternative learning techniques capable of managing the expanding complexities of state spaces in these continuously evolving environments. These efforts will ensure that our research stays at the cutting edge of addressing the evolving demands related to energy-conscious computational offloading at the network's edge

In summary, our work delivers a threefold contribution by introducing an innovative two-stage RL-based mechanism for computational offloading in a cooperative MEC environment. Through this work, we not only tackle the challenges of multi-site MEC infrastructures, but we also address the issue of workload balancing through cooperative offloading from one ES to another. The amalgamation of these stages forms a holistic approach that optimizes decision-making and converges towards stable solutions, as underscored by the promising results obtained through simulation studies. In the future, our ongoing endeavors will be geared towards further enhancing the flexibility and efficiency of our approach.

The work described in this thesis has been published in the following paper:

Publications

The main content of this thesis was published in the 2023 IEEE 24th International Conference on High-Performance Switching and Routing (HPSR):

Marios AVGERIS, Meriem MECHENNEF, Aris LEIVADEAS, and Ioannis LAMBADARIS. A Two-Stage Cooperative Reinforcement Learning Scheme for Energy-Aware Computational Offloading. In : 2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR). IEEE, 2023. p. 179-184. https://doi.org/10.1109/HPSR57248.2023.10147932

LIST OF REFERENCES

Akhlaqi, M. Y. & Hanapi, Z. B. M. (2023). Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions. *Journal of Network and Computer Applications*, 212, 103568.

Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

Avgeris, M. (2021). *Dynamic resource allocation and computational offloading at the network edge for Internet of things applications*. (Ph.D. thesis, National Technical University of Athens, Athens, Greece).

Avgeris, M., Spatharakis, D., Dechouniotis, D., Leivadeas, A., Karyotis, V. & Papavassiliou, S. (2022). ENERDGE: Distributed energy-aware resource allocation at the edge. *Sensors*, 22(2), 660.

Avgeris, M., Mechennef, M., Leivadeas, A. & Lambadaris, I. (2023). A Two-Stage Cooperative Reinforcement Learning Scheme for Energy-Aware Computational Offloading. *2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR)*, pp. 179–184.

Biosca Caro, J. (2020). *A deep reinforcement learning approach for optimization and task-offloading of mobile edge computing in virtual radio access networks*. (Master's thesis, Universitat Politècnica de Catalunya, Barcelona, Espagne).

Bouhoula, S., Avgeris, M., Leivadeas, A. & Lambadaris, I. (2022). Computational offloading for the industrial internet of things: A performance analysis. *2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pp. 1–6.

Chen, X., Qu, G., Tang, Y., Low, S. & Li, N. (2021a). Reinforcement learning for decision-making and control in power systems: Tutorial, review, and vision. *arXiv*.

Chen, X. & Liu, G. (2021). Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks. *IEEE Internet of Things Journal*, 8(13), 10843–10856.

Chen, X., Hu, J., Chen, Z., Lin, B., Xiong, N. & Min, G. (2021b). A reinforcement learning-empowered feedback control system for industrial internet of things. *IEEE Transactions on Industrial Informatics*, 18(4), 2724-2733.

Chen, Y., Gu, W. & Li, K. (2022). Dynamic task offloading for internet of things in mobile edge computing via deep reinforcement learning. *International Journal of Communication Systems*, e5154.

Chu, W., Jia, X., Yu, Z., Lui, J. C. & Lin, Y. (2023). Joint Service Caching, Resource Allocation and Task Offloading for MEC-based Networks: A Multi-Layer Optimization Approach. *IEEE Transactions on Mobile Computing*.

Fang, T., Wu, D., Chen, J. & Liu, D. (2022). Cooperative task offloading and content delivery for heterogeneous demands: A matching game-theoretic approach. *IEEE Transactions on Cognitive Communications and Networking*, 8(2), 1092–1103.

Jiang, K., Zhou, H., Li, D., Liu, X. & Xu, S. (2020). A q-learning based method for energy-efficient computation offloading in mobile edge computing. *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7.

Ku, H. (2022). *DYNAMIC TASK OFFLOADING FOR LATENCY MINIMIZATION IN IOT EDGE-CLOUD ENVIRONMENTS*. (Ph.D. thesis, Purdue University Graduate School, West Lafayette, Indiana, USA).

Kuang, Z., Ma, Z., Li, Z. & Deng, X. (2021). Cooperative computation offloading and resource allocation for delay minimization in mobile edge computing. *Journal of Systems Architecture*, 118, 102167.

Li, J., Gao, H., Lv, T. & Lu, Y. (2018). Deep reinforcement learning based computation offloading and resource allocation for MEC. *2018 IEEE wireless communications and networking conference WCNC)*, pp. 1–6.

Li, S. E. (2022). *Reinforcement learning for sequential decision and optimal control*. Springer.

Lin, H., Zeadally, S., Chen, Z., Labiod, H. & Wang, L. (2020). A survey on computation offloading modeling for edge computing. *Journal of Network and Computer Applications*, 169, 102781.

Liu, Q., Xia, T., Cheng, L., van Eijk, M., Ozcelebi, T. & Mao, Y. (2022). Deep Reinforcement Learning for Load-Balancing Aware Network Control in IoT Edge Systems. *IEEE Transactions on Parallel & Distributed Systems*, 33(6), 1491–1502.

Maray, M. & Shuja, J. (2022). Computation offloading in mobile cloud computing and mobile edge computing: survey, taxonomy, and open issues. *Mobile Information Systems*, 2022.

Omland, S. E. (2022). *Deep Reinforcement Learning for Computation Offloading in Mobile Edge Computing*. (Master's thesis, The University of Bergen, Bergen, Norway).

Rasouli, N., Razavi, R. & Faragardi, H. R. (2020). EPBLA: energy-efficient consolidation of virtual machines using learning automata in cloud data centers. *Cluster Computing*, 23, 3013–3027.

Rezagholizadeh, A. (2022). *Source rate control in videoconferencing application using state–action–reward–state–action temporal difference reinforcement learning*. (Ph.D. thesis, École de technologie supérieure, Montreal, CANADA).

Sadiki, A. (2022). *Deep Reinforcement Learning For The Computation Offloading In MIMO-based Edge Computing*. (Ph.D. thesis, Concordia University, Montreal, CANADA).

Sadiki, A., Bentahar, J., Dssouli, R., En-Nouaary, A. & Otrok, H. (2023). Deep reinforcement learning for the computation offloading in MIMO-based Edge Computing. *Ad Hoc Networks*, 141, 103080.

Saeik, F., Avgeris, M., Spatharakis, D., Santi, N., Dechouniotis, D., Violos, J., Leivadeas, A., Athanasopoulos, N., Mitton, N. & Papavassiliou, S. (2021). Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195, 108177.

Sewak, M. (2019). *Deep reinforcement learning*. Springer.

Shakarami, A., Ghobaei-Arani, M. & Shahidinejad, A. (2020). A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks*, 182, 107496.

Singh, R., Sukapuram, R. & Chakraborty, S. (2023). A survey of mobility-aware Multi-access Edge Computing: Challenges, use cases and future directions. *Ad Hoc Networks*, 140, 103044.

Stoica, F., Popa, E. M. & Pah, I. (2008). A NEW REINFORCEMENT SCHEME FOR STOCHASTIC LEARNING AUTOMATA. *ICE-B 2008*, 45.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tran, T. X. & Pompili, D. (2019). Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks. *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, 68(1).

Unsal, C., Kachroo, P. & Bay, J. S. (1999). Multiple stochastic learning automata for vehicle path control in an automated highway system. *IEEE Transactions on Systems, Man, and Cybernetics-part A: systems and humans*, 29(1), 120–128.

Wan, S., Li, X., Xue, Y., Lin, W. & Xu, X. (2020). Efficient computation offloading for Internet of Vehicles in edge computing-assisted 5G networks. *The Journal of Supercomputing*, 76, 2518–2547.

Winder, P. (2020). *Reinforcement learning*. O'Reilly Media.

Xu, F., Xie, Y., Sun, Y., Qin, Z., Li, G. & Zhang, Z. (2022). Two-stage computing offloading algorithm in cloud-edge collaborative scenarios based on game theory. *Computers & Electrical Engineering*, 97, 107624.

Yan, J., Li, N., Zhang, Z., Liu, A. X., Martinez, J. F. & Yuan, X. (2019). Game theory based joint task offloading and resources allocation algorithm for mobile edge computing. *arXiv preprint arXiv:1912.07599*.

Yang, L., Zhang, H., Li, M., Guo, J. & Ji, H. (2018). Mobile edge computing empowered energy efficient task offloading in 5G. *IEEE Transactions on Vehicular Technology*, 67(7), 6398–6409.

Yin, G., Chen, R. & Zhang, Y. (2022). Effective task offloading heuristics for minimizing energy consumption in edge computing. *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pp. 243–249.

Yuan, H. & Zhou, M. C. (2021). Profit-Maximized Collaborative Computation Offloading and Resource Allocation in Distributed Cloud and Edge Computing Systems. *IEEE Transactions on Automation Science and Engineering*, 18(3), 1277–1287.

Zarandi, S. (2021). Resource Allocation in Multi-access Edge Computing (MEC) Systems: Optimization and Machine Learning Algorithms. Toronto, Ontario, Canada.

Zhang, G., Zhang, S., Zhang, W., Shen, Z. & Wang, L. (2021). Joint service caching, computation offloading and resource allocation in mobile edge computing systems. *IEEE Transactions on Wireless Communications*, 20(8), 5288–5300.

Zhang, K. (2020). *Task Offloading and Resource Allocation using Deep Reinforcement Learning*. (Ph.D. thesis, University of Ottawa, Ottawa, Ontario, Canada).

Zhang, X. (2023). *Resource Management in Mobile Edge Computing for Compute-intensive Application*. (Ph.D. thesis, City University of New York, New York, USA).

Zhao, H., Geng, J. & Jin, S. (2023). Performance research on a task offloading strategy in a two-tier edge structure-based MEC system. *The Journal of Supercomputing*, 1–39.

Zhou, S., Jadoon, W. & Khan, I. A. (2023). Computing Offloading Strategy in Mobile Edge Computing Environment: A Comparison between Adopted Frameworks, Challenges, and Future Directions. *Electronics*, 12(11), 2452.