Resource Allocation in Multi-access Edge Computing (MEC) Systems: Optimization and Machine Learning Algorithms

Sheyda Zarandi

A thesis submitted to the
Department of Electrical Engineering and Computer Science
in conformity with the requirements for
the degree of Master of Applied Science

YORK UNIVERSITY
TORONTO, ONTARIO

April 2021

Abstract

With the rapid proliferation of diverse wireless applications, the next generation of wireless networks are required to meet diverse quality of service (QoS) in various applications. The existing one-size-fits-all resource allocation algorithms will not be able to sustain the sheer need of supporting diverse QoS requirements. In this context, radio access network (RAN) slicing has been recently emerged as a promising approach to virtualize networks' resources and create multiple logical network slices on a common physical infrastructure. Each slice can then be tailored to a specific application with distinct QoS requirement. This would considerably reduce the cost of infrastructure providers. However, efficient virtualized network slicing is only feasible if network resources are efficiently monitored and allocated.

In the first part of this thesis, leveraging on tools from fractional programming and Augmented Lagrange method, I propose an efficient algorithm to jointly optimize users' offloading decisions, communication, and computing resource allocation in a sliced multi-cell multi-access edge computing (MEC) network in the presence of interference. The objective is to minimize the weighted sum of the delay deviation observed at each slice from its corresponding delay requirement. The considered problem enables slice prioritization, cooperation among MEC servers, and partial offloading to multiple MEC servers.

On another note, due to high computation and time complexity, traditional centralized optimization solutions are often rendered impractical and non-scalable for real-time resource allocation purposes. Thus, the need of machine learning algorithms has become more vital than ever before. To address this issue, in the second part of this thesis, exploiting the power of federated learning (FDL) and optimization theory, I develop a federated deep reinforcement learning framework for joint offloading decision and resource allocation in order to minimize the joint delay and energy consumption in a MEC-enabled internet-of-things (IoT) network with QoS constraints. The proposed algorithm is applied to an IoT network, since the IoT devices suffer significantly from limited computation and battery capacity. The proposed algorithm is distributed in nature, exploit cooperation among devices, preserves the privacy, and is executable on resource-limited cellular or IoT devices.

Acknowledgments

I would like to thank my advisor, Prof. Hina Tabassum, for her continuous support and encouragement. It was a great privilege and honor to work and study under her guidance as a member of in research group.

I am also extremely grateful to my parent for their unconditional love, caring, and sacrifices. They have always been my source of hope and love and I feel truly blessed for having them in my life.

Contents

List of Figures

Chapter 1

Introduction

1.1 Beyond 5G and 6G networks

The need for fast, reliable, and ubiquitous wireless connections continues to grow even after many of the technologies proposed in 5G were successfully implemented and exploited in the existing network infrastructure [2]. While 3GPP is continuously working on the evolution of 5G with Release 16 being finalized early 2020, industry and academia has started looking towards the next generation of mobile networks, 6G, that is targeted for 2030 and aims at addressing challenges not easily achievable in 5G evolution [3]. Some examples of these challenges are meeting distinct Quality-of-Service (QoS) requirements for diverse applications, accommodating tremendous number of devices connected to cellular networks, and providing devices with sufficient computation capability to perform advanced and resource exhaustive tasks [4].

Fueled by the emergence of IoT networks [5] and dense cellular network deployments, these requirements that are also given in Table 1.1, became more vital than ever before and still challenging to achieve. For instance, even a moderate delay might be intolerable for connected autonomous vehicles and many of the smart home

applications. The massive connectivity, ultra low latency requirements, and the need for vast infrastructure resources have created a new range of challenges of critical importance not only for networks, but for the constantly changing society, which calls for new network architecture and resource management solutions in cellular networks.

In recent years, many cutting-edge technologies have been developed targeting some of the aforementioned challenges [6]. Three of the most promising technologies in this regard are cloud computing and multi-access edge computing (MEC), network slicing, and intelligent networks. In cloud and MEC computing, the ability to offload resource intensive tasks to the remote servers provides a platform to process the tasks that exceed the capability of simple mobile devices. As such, this technology enables devices to enjoy versatile and cutting-edge services without worrying about their limited resources. In network slicing, isolation of resources and virtually sharing the network infrastructure, help service providers to guarantee the desired QoS of their subscribers in a cost efficient manner and without over-provisioning [7].

Furthermore, intelligence in wireless networks has become one of the most attractive research areas in recent decades. With the proliferation of IoT devices, nowadays more data is generated by geographically distributed and widespread IoT and mobile devices. Based on a forecast published by Ericsson, by 2024 more than 45% of the 40-ZB global Internet data will be generated by the IoT devices [8]. This tremendous available data which is fundamental for training accurate learning models, emergence of cloud computing paradigm that facilitates remote processing of tasks, and the inability of traditional resource allocation tools in addressing the scalability challenges of massive IoT networks, have given rise to the popularity of deep learning and machine learning methods.

Table 1.1: Major requirements of the 5G and B5G networks [1, 9]

Type	Data Rate	Latency		Reliability	Connectivity
		E2E	Radio-only		
4G	$\sim$ 1 Gbps (SE: bps/Hz/m^2)	$\sim$ 50 msec	$\sim$ 5 msec	four-nines	2000 connected devices per .38 square miles
5G	$\sim$ 10-20 Gbps (SE: bps/Hz/m^2)	$\sim$ 5 msec	$\sim$ 100 nsec	five-nines	1 million connected devices per .38 square miles
B5G/6G	$\sim$ 1 Tbps (SE: bps/Hz/m^3)	1 msec	$\sim$ 10 nsec	seven-nines	1 trillion connected devices per .38 square miles

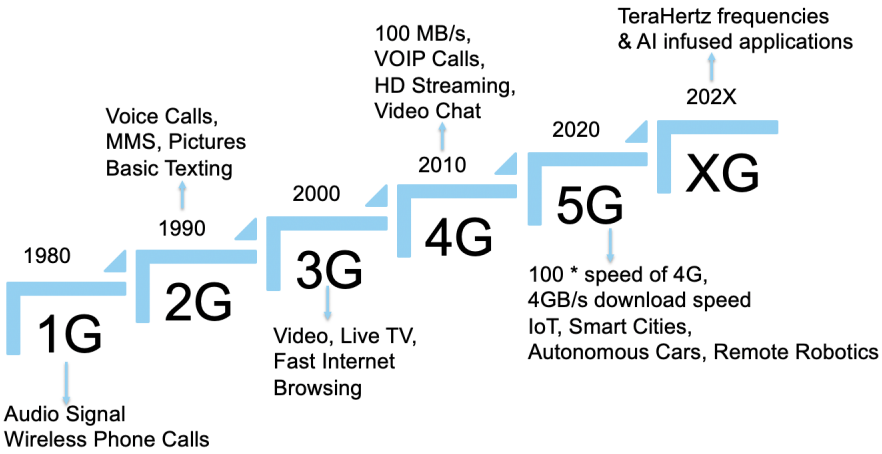

Figure 1.1: Evolution of wireless networks from both research and commercialization perspective [1]

In what follows, we will dive deeper into the above mentioned technologies and outline some of their primary advantages and challenges.

1.2 Introduction to Cloud and Edge Computing

Driven by the visions of IoT and low latency communications, recent years have seen a paradigm shift in mobile computing, from the centralized mobile cloud computing (MCC) toward MEC. The main feature of MEC is to push mobile computing, network control, and storage to the network edges (e.g., base stations and access points) so as

to enable computationally-intensive and latency-critical applications on the resource-limited mobile devices [10]. In other words, devices are able to offload their tasks to MEC servers if enough resources are not locally available to process them timely. As the name suggests, MEC servers are located at the edge of the network and close to devices, so devices do not necessarily have to deal with high latency to access their services. Subsequently, MEC promises dramatic reduction in latency and energy consumption of devices, while tackling some of the key challenges for successful roll-out of 6G.

1.2.1 Benefits

MEC systems provide wireless devices with a reliable and low latency platform for their resource-exhaustive tasks to be efficiently processed. Some of the major advantages of MEC systems compared to traditional wireless networks (without offloading capability) or cloud computing systems are [3]:

- *Low Latency:* In traditional networks, devices had to process their tasks themselves. Needless to say, as services became more sophisticated and computationally exhaustive, limited local computation capabilities became a bottleneck that rendered mobile devices completely incapable of executing cutting-edge services. Emergence of MCC partially addressed this issue. However, MCC requires the information to pass through several networks including the radio access network, backhaul network and Internet, where traffic control, routing and other network-management operations can contribute to excessive delay. Being deployed at the network edge, MEC circumvents these time-consuming transmissions and shortens the service response time considerably [11]. Also,

for the computation latency, a cloud has a massive computation power that is several orders of magnitude higher than that of an edge device. However, the cloud has to be shared by a much larger number of devices than an edge device.

- *Mobile Energy Savings*: Due to their compact forms, IoT devices have limited energy storage but are expected to cooperate and perform sophisticated tasks such as surveillance, crowd-sensing and health monitoring. Powering the tens of billions of IoT devices remains a key designing challenge given that frequent battery recharging/replacement is impractical, if not impossible. By effectively supporting computation offloading in an energy efficient manner, MEC stands out as a promising solution for prolonging battery lives of IoT devices.

- *Privacy/Security Enhancement:* The capability of enhancing the privacy and security of mobile applications is also an attractive benefit brought by MEC compared to MCC. In MCC systems, the Cloud Computing platforms are public large data centers, such as the Amazon EC2 and Microsoft Azure, which are susceptible to attacks due to their high concentration of information. On the other hand, due to the distributed deployment, small-scale nature, and the less concentration of valuable information, MEC servers are much less likely to become the target of a security attack.

1.2.2 Challenges

Just like any emerging technology, many unanswered challenges exist in MEC systems. The benefits obtained from this paradigm, completely depends on whether these issues are properly addressed or not. Some of these challenges are listed in what follows [12].

- *Communication Delay*: Unlike local computation for which a deterministic estimation of service delay can be easily obtained based on the computation capability of local device, a significant part of the delay a device would experience in MEC systems depends on the stochastic network environment. As an example, if the channel condition of devices to their associated MEC server is not good or the level of interference they have to cope with is too high, the communication delay they experience would be significant and their process may not be finished in the desired time frame. Therefore, precise physical layer resource allocation (e.g. transmit power and subchannel allocation) becomes a necessity in MEC networks and plays a significant role in minimization of this delay.

- *Offloading Decision Optimization*: While MEC can help devices with the processing of their tasks in a timely manner, offloading to edge servers is not always the best available option. For instance, if the channel quality of a given user is not desirable, the offloading data rate can be low or the delay required for the transmission of the task to the MEC server can surpass the delay required for local processing. Under such conditions, it would be better for devices to perform their task locally or find a more suitable server to offload their tasks to. This observation emphasizes the substantial impact of accurate offloading policy on the service delay devices' would experience.

- *Computation Resource Allocation*: Even after devices offload their task in an acceptable time span, they have to wait for their tasks to be processed by the edge server. Given that the computation capacity of an edge server is limited when compared to the cloud server, the proportion of computation resources allocated to the task of each user plays a significant role in their computation

delay. This highlights the vital importance of optimal computation resource allocation in the edge servers for efficient resource utilization and to ensure quality service is provided to as many devices as possible.

- *MEC Server Association and Partial Offloading*: Unlike MCC, in MEC systems usually more than one edge server exists in the proximity of user. Another challenge in MEC that needs to be carefully investigated, is how devices' tasks are assigned partially to different MEC servers. This is not only about the destination of the offloaded tasks, but also the proportion of task that is executed by each server. To address these issues, many factors such as the available computation resources in each server and the nature of devices' task (whether it can be partially offloaded or not), should be taken into account.

1.3 Introduction to Sliced Virtual Networks

The economic sustainability of future mobile networks will largely depend on the strong specialization of its offered services. The traditional business model for mobile networks is centred on operators acquiring licence to use available spectrum, building their own infrastructure, and control the resource allocation according to their needs. This model is now being challenged by a number of economic, regulatory, and technical circumstances, which are expected to change the mobile landscape in future.

The first well known factor that is challenging the traditional business model is the exponential growth of mobile traffic that is pushing operators to rapidly expand their networks with technological upgrades and extensive deployment of resources. Unfortunately, the average revenue per user is not growing with the same pace and these new requirements pose a heavy financial burden on service providers, making

them unable to provide a qualified service for devices of cellular networks [13].

On another note, the next generation of mobile networks is expected to become a dominant General Purpose platform on which millions of increasingly diversified services will be hosted. As a consequence, network operators will need to support heterogeneous QoS requirements. These trends are driving the design of cellular networks toward a strong differentiation of guarantees, separating services into macroscopic categories such as Mobile Broadband (eMBB), Ultra Reliable Low Latancy Communications (URLLC), and Massive Machine Type Communication (mMTC), based on their QoS requirements. To address such varied type of services on the same infrastructure is another challenge service providers are struggling with.

To tackle the aforementioned challenges, it is crucial to rely on the diffusion of software-defined networking (SDN) solutions, which enable network virtualization. Using this virtualization approach, called *Network Slicing*, the traditional hard box-based infrastructure can be evolved into a cloudified architecture. Network virtualization enables the deployment of multiple virtual instances of the complete network, named network slices. Slices are logical networks created on top of the physical infrastructure, each tailored to accommodate fine-tuned Service Level Agreements (SLAs) reflecting the needs of different service providers.

1.3.1 Benefits

Network slicing is an indispensable technique to support heterogeneous services in beyond 5G networks [14]. Using network slicing, multiple logical network slices can be created on a common physical infrastructure. Each slice can be tailored to a specific application with distinct QoS requirement. Some of the advantages of network slicing

that turn this new architecture into a necessity for delivering the promises in 6G are:

- *Cost Effectiveness*: One of the fundamental challenges observed by service providers is their inability to address massive connectivity without strengthening their resources and infrastructure. On the other hand, via network slicing, the role of service providers would be different from infrastructure providers (InPs). InPs would deploy hardware resources, then exploiting resource virtualization they will create logical end-to-end networks, called slices. InPs lease these slices to the service providers. Clearly, this capability to separate logical and physical network plays a significant role in reducing both the capital (CAPEX) and operational (OPEX) expenditures of service providers.

- *Service Provisioning*: With network slicing technology, service providers can circumvent the need for deployment of physical networks while provisioning new services and instead lease already available resources from InPs. This would significantly decrease the time to market for service providers and as such facilitates the provisioning of new services.

- *Dynamic Resource Sharing among Slices*: As slices are logically separated from one another and the resources allocated per slice are defined through SDN-based approaches, dynamic resource sharing is possible which would enhance the overall efficiency of resource utilization in the network. This would instigate continuous and seamless resource allocation in the whole network.

1.3.2 Challenges

Some of the major challenges in sliced networks are listed in the following [15]:

- *Resource Isolation*: Resource isolation is a fundamental property of network slicing that assures performance guarantees and security for each tenant, even when different tenants use network slices for services with conflicting performance requirements. However, isolation may come at the cost of reduced multiplexing gain, which may result in inefficient network resource utilization. A comprehensive method that not only isolates slices from each other, but also enables maximum multiplexing in the network is needed to balance the isolation versus resource multiplexing in sliced networks.

- *End-to-End Resource Allocation*: End-to-end slicing is crucial to facilitate a service delivery all the way from the service providers to the end-user/customer(s). Such a property has two extensions, (i) it stretches across different administrative domains, i.e., a slice that combines resources belonging to distinct infrastructure providers, and (ii) it unifies various network layers and heterogeneous technologies, e.g., RAN, core network, transport layer, and cloud. In particular, an end-to-end network slicing consolidates diverse resources and enables an overlaid service layer which provides new opportunities for efficient networking and service convergence [16].

- *Slice Prioritization*: To support a variety of QoS requirements, efficient slice prioritization is also crucial. Slice prioritization refers to the mechanism in which some slices are selected and are given privileges in the network. For instance, often times these prioritized slices have access to a more considerable proportion of resources in the network, and in any resource allocation strategy, they would be prioritized in allocation of available resources. In this context, it is also important to understand the impact of slice prioritization and how it

can be incorporated in the wireless resource allocation problems [17].

1.4 Introduction to Intelligent Networks

The stringent QoS requirements in many of the emerging applications such as autonomous driving and virtual reality have increased the necessity of having a problem-solving approach that not only is online and can produce the result in real-time fashion, but also can handle the dynamic environment of wireless networks. On another note, given the steep increase in the number of connected devices to cellular network following the emergence of IoT and dense networks, it is now vital for any practical resource allocation framework to be distributed and scalable. In what follows, I focus on two types of learning methods that can best address the requirements of a distributed and scalable wireless resource allocation platform.

1.4.1 Reinforcement Learning

Reinforcement learning is one of the three main machine learning categories, with the other two being supervised learning and unsupervised learning.

Supervised learning needs all the data to be labeled and labeling massive data loads can be extremely time-consuming and resource-exhaustive [18]. This type of learning may also fail in dynamic environments as the pace of change in such systems are so fast that there would not be enough time to gather sufficient data, label them, and then train the supervised model using this new dataset.

Different from supervised learning, unsupervised learning does not need labeled data and the data (without any label) would be directly fed into the learning model. Application of unsupervised learning is mostly to find hidden patterns in the datasets

so that we can either classify similar points together or find a relationship between different features of data.

Different from supervised and unsupervised learning, reinforcement Learning enables an agent to learn from repeatedly interacting with the environment and evaluating the feedback it receives that is modeled as a reward or loss signal. By this evaluation, the agent would learn which action is more beneficial (maximizes its accumulative reward or minimizes its loss) in any given state.

Note that while both unsupervised learning and reinforcement learning target finding a mapping between input and output (actions) without use of labels, instead of mathematical properties of data points, reinforcement learning uses rewards and punishments as signals for positive and negative behavior.

Furthermore, unsupervised learning and reinforcement learning differ in their objectives. While the objective of unsupervised learning is to find similarities and differences between data points, reinforcement learning aims to find an accurate action model that would maximize/minimize the total cumulative reward/loss of the agent.

The ability of reinforcement learning to learn by interacting with the environment, turns it into an ideal problem-solving approach for the extremely dynamic cellular networks. Also, as reinforcement learning does not need an input dataset for learning, the need for saving and updating massive amount of data would be obliterated.

1.4.2 Federated Learning

Recently, federated learning (FDL) has emerged as a new paradigm for cooperative learning, where multiple nodes contribute in training a single global model. The devices use their local limited datasets to train a local model and then offload their

models to a central unit for global aggregation that is obtained by applying an aggregation function, such as average, on the models. This new paradigm is beneficial from the following aspects [19].

- *Scalability:* As explained above, in FDL, at each training round, a small subset of agents send their pretrained local models at the centralized controller for aggregation. Therefore, the need for having a central controller that is responsible for the whole learning process would be removed. Therefore, this approach would not be effected by exponentially growing number of devices in the system.

- *Privacy:* In FDL, devices are not required to share their data or experiences with any external entity. As such, while their knowledge would be eventually combined together in the global model, their privacy would be preserved [20].

- *Spectrum efficiency:* In FDL, there is no need to upload huge blocks of information to an external unit at every training round, thus the available bandwidth would not be burdened by constant transfer of tremendous amount of data from all devices to the external controller [21].

- *Cooperative learning:* While data sharing is avoided in FDL, through integration of local models, the global model would be a combination of all the knowledge gathered through local models. By doing so, agents help each other to achieve a more comprehensive model that can work well even in situations that are not encountered by all the agents in the network. Therefore, using FDL, we will still obtain a cooperative learning framework that is also privacy preserving [22].

1.5 Thesis Contributions

In this thesis, I investigated the problem of joint offloading, communication, and computation resource allocation in a MEC system and develop novel resource management algorithms leveraging on tools from optimization and machine learning theory. My contributions can be listed as follows:

- Leveraging on tools from optimization theory, I develop a novel a framework for joint subchannel, power, and computation resource allocation and offloading decision optimization that aims at minimizing the devices' delay in a virtually sliced multi-cell MEC system with cooperative edge servers. The devices belong to different slices and as such have different QoS demands. Also, I consider a partial offloading scheme in which devices can perform part of their task locally and offload the rest to either of the MEC servers. Particularly, the objective is to minimize the gap between the experienced delay of devices and the maximum tolerable delay threshold of their specific slice.

- Combining the benefits of FDL and reinforcement learning, I develop a machine learning framework that aims to minimize the delay and energy consumption of IoT devices by optimizing their offloading decisions, transmit powers and local computation resource allocation. To address this problem we exploit the power of RL and directly use the locally developed deep double Q-network (DDQN) models as the input of federating process. We also employ optimization theory to accurately estimate the loss function of each agent given any state/action pair. By combining DDQN, FDL, and traditional optimization, we will obtain a novel cooperative, scalable, and privacy-preserving framework that works well

in the dynamic and dense IoT-based networks.

1.6 Publications

The outcome of this thesis, were the following two papers:

- The work outlined in the first bullet of section 1.5 is published in *IEEE Communications Letters* [23].
- The work outlined in the second bullet of section 1.5 is accepted in *IEEE ICC 2021 Workshop on Wireless Networking Innovations for Mobile Edge Learning* [24].

Chapter 2

Mathematical Background and Preliminaries

In this chapter, an overview of fractional programming and reinforcement learning, specifically Q-learning and double deep Q-learning is provided. These mathematical preliminaries are provided so that one has enough knowledge to follow the discussions and calculations in the following chapters. Thus, if the reader is familiar with these tools, the reader can move to the next chapter.

2.1 Fractional Programming

Fractional programming (FP) refers to a family of optimization problems that involve ratio term(s) [25]. Multi-ratio FP belongs to one of the major categories of FP that is applicable in the context of wireless networks, since the wireless data rate calculations are a function of signal-to-interference-plus-noise ratio (SINR) which has a fractional form. In the following subsections, we will first explore Dinklebach method that is one of the classical techniques of addressing FP and then present a unique approach to address multi-ratio FP in cellular networks (adopted from [25]) that is later on employed to address the problem in chapter 3.

2.1.1 Dinkelbach's Transform

Let us consider a single-ratio problem FP problem. Given a nonempty constraint set, such problem can be modeled as follows:

$$\max_{x} \frac{A(x)}{B(x)},$$
$$\text{subject to: } x \in \mathcal{X}. \tag{2.1}$$

The above single-ratio FP is in general not convex and thus difficult to address.

The conventional method of dealing with such problems is to reformulate the problem by decoupling the numerator and denominator from each other, whereby the joint optimization of $A(x)$ and $B(x)$ becomes simpler. One of the most common techniques belonging to this approach is called Dinkelbach's transform.

Dinkelbach transform was first proposed in [26]. In this method, any single-ratio FP in the form given in (2.1) can be reformulated as:

$$\max_{x} A(x) - \Psi B(x),$$
$$\text{subject to: } x \in \mathcal{X}, \tag{2.2}$$

where Ψ is an auxiliary variable whose value, at any time step t, is calculated as:

$$\Psi_{t+1} = \frac{A(x_t)}{B(x_t)}. \tag{2.3}$$

Given that $A(x)$ is concave and $B(x)$ is a convex function with respect to variable x, it is proven that (2.2) converges to the optimal solution of problem (2.1).

2.1.2 Quadratic Transform

Classic approaches such as Dinkelbach transform work well for single-ratio problems, but they cannot be readily generalized to multiple-ratio problems. This is due to the fact that even though classic transforms have the property that the original and the transformed problem have the same optimal solution, the optimal value of the objective function in the transformed problem and the original FP maybe different from one another. Therefore, in case of multiple ratios, we cannot apply the transform to each individual ratio, separately.

Let us assume we have a sum of ratios FP problem given by:

$$\max_{x} \sum_{m=1}^{M} \frac{A(x)}{B(x)},$$
$$\text{subject to: } x \in \mathcal{X}. \tag{2.4}$$

In this optimization method, unlike (2.1), we have a summation in the objective function that encompasses the fractional form.

To solve the aforementioned problem, the quadratic transform is an efficient approach. Quadratic transform algorithm proposed in [25] is motivated by Dinkelbach method, but with a new constraint added to the problem that the value of the objective function must remain the same. It is called quadratic transform because it uses the properties of quadratic programming to re-state and then solve the FP.

To solve this problem, using the quadratic transform, we can restate (2.4) as:

$$\max_{x,y} \sum_{m=1}^{M} \big(2y_m\sqrt{A_m(X)} - y_m^2 B_m(X)\big) \tag{2.5}$$

$$\text{subject to: } x \in \mathcal{X}, y_m \in \mathbb{R}. \tag{2.6}$$

Note that as proven in **Appendix A** of [25], (2.5) is equivalent to (2.4) and is not an approximation of it. Through this transform, we can also observe that a new type of auxiliary variable y is added to the problem that refers to a collection of variables $\{y_1, \cdots, y_M\}$. In fact, the quadratic transform can be further extended to a more general sum-of-functions-of-ratio problem.

Given non-decreasing functions $f_m(.)$ and a sequence of ratios in the form of $\frac{A_m}{B_m}$, let us consider the following problem:

$$\max_{x} \sum_{m=1}^{M} f_m(\frac{A(x)}{B(x)}),$$

$$\text{subject to: } x \in \mathcal{X}. \tag{2.7}$$

Once more using Quadratic transform, we can state (2.7) in its equivalent form as:

$$\max_{x,y} \sum_{m=1}^{M} f_m\big(2y_m\sqrt{A_m(X)} - y_m^2 B_m(X)\big) \tag{2.8}$$

$$\text{subject to: } x \in \mathcal{X}, y_m \in \mathbb{R}. \tag{2.9}$$

The ability to restate a sum-ratio FP into an equivalent form without approximation is a very strong tool that is made possible through Quadratic transform.

2.2 Reinforcement Learning (Q-Learning)

Q-learning is a form of model-free reinforcement learning. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of their actions, without requiring them to build maps of the domain. As such, it is a very powerful tool for solving problems in wireless networks, where finding a complete model of the system is difficult if not impossible all together. The two primary methods of Q-learning are deep Q-networks and double deep Q-networks, on which we will focus in the following subsections.

2.2.1 Deep Q-Learning

As previously explained, the goal of agent in reinforcement learning is to maximize its cumulative reward over time. To this end, state/action value function denoted by $Q(s, a)$ for state s and action a is constantly estimated to evaluate the merit of choosing action a given that we are currently at state s. If this action is beneficial and agent's reward is increased, then agent learns to choose this action when state s is met, otherwise, it learns to avoid the action and even state in future. Therefore, it is evident that the state/action value function and the accuracy of estimating it over the learning process, plays a significant role in the learning speed and its accuracy [27].

In deep Q-learning (DQL), $Q(s, a)$ is calculated as below:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a'). \tag{2.10}$$

where r represents the immediate reward that is obtained from being in state s and choosing action a, γ denotes the discount factor whose value is between 0 and 1, and

s' and a' are the state and action in the next time step, respectively. By setting discount factor, γ, to larger values, we encourage agent to put more emphasis on future rewards rather than the immediate reward.

To estimate the value of future action-state pairs, in deep Q-learning, we train a neural network at every round on a batch of data saved in a buffer that contains previous experiences of the agent. By using this neural network, the need for knowing the transition probability (the probability of moving to state s' given that we were in state s and chose action a) is not evident which is often impossible all-together.

Even with the above mentioned advantage, Q-learning still suffers from two major shortcomings listed in the following:

- *Over-estimation of Q-value:* it can be clearly seen in (2.10) that, to estimate the future rewards that can be obtained when we are at state s and choose action a, max operator is used. Doing this, we are in fact using overestimated values since the maximum value of estimations is taken into account. This systematic overestimation introduces a maximization bias into learning process. As Q-learning is based on bootstrapping — learning estimates from estimates — this kind of overestimation can be problematic.

- *Chasing non-stationary target:* another observation from (2.10) is that, we are bootstrapping the value of $Q(s', a')$ to estimate $Q(s, a)$, whereas $Q(s', a')$ by itself is updated continuously. So we are using a value that is constantly changing and this fact gives rise to the problem of non-stationary target at can slow down the learning process significantly.

To solve these two challenges in conventional DQL, Double deep Q-networks (DDQN) are proposed.

2.2.2 Double Deep Q-Learning

The solution of the two shortcomings of deep Q-network is to use two separate Q-value estimators/neural networks, each of which is later on used to update the other one. Employing these independent neural networks, we can obtain a more stable and unbiased learning process. In DDQN, we train the model using an online network and a target network. While the parameters of online network (its weights and biases) are changed after every learning step, target network is kept mostly unchanged, and only after multiple time steps, its parameters are updated with those of online network. By doing so, we are effectively addressing the problem of chasing non-stationary targets in DQL. Subsequently, the updated rule in DDQN is given by:

$$Q(s,a) = r + \gamma Q(s_{t+1}, \arg\max_{a'} Q'(s,a)). \tag{2.11}$$

where $Q(s_{t+1}, \arg\max_{a'} Q'(s,a))$ is estimated using the online network and $Q'(s,a)$ is estimated using the target network.

Given the dynamic nature of the wireless networks, DDQN is a potential approach to effectively deal with non-stationary environments and avoid over-estimation.

2.3 Summary

In this chapter, I provide an overview of the mathematical tools that will be used in this thesis. I also explained the motivations behind selecting specific tools to address our problems and their advantages over other existing approaches. In the following chapter, we will introduce our first problem and our proposed solution to address it.

Chapter 3

Delay Minimization in Multi-cell Sliced MEC Systems

Network slicing is an indispensable technique to support heterogeneous services in fifth generation (5G) networks [28]. Using network slicing, multiple logical network slices can be created on a common physical infrastructure. Each slice can be tailored to a specific application with distinct QoS requirement. On another note, resource-intensive and latency sensitive services necessitate MEC, that brings computational resources to the Radio Access Network (RAN) edge. Thus, users would use both RAN and computation resources to offload and process their tasks at the MEC servers. On the other hand, in a sliced network, resources are restricted for each slice based on a service level agreement (SLA) with infrastructure provider (InP). Subsequently, joint optimization of RAN resources (e.g., subchannel and power) and computation resources (e.g., CPU cycles of MEC servers) with optimal computation offloading in a sliced network becomes imperative.

3.1 Literature Review

Recently, the problem of delay minimization in a multi-cell MEC network was solved through communication and computation resource allocations (RAs) *without network slicing* [29–31]. However, in all these works, the interference was either ignored [29,30] or simplified [31]. Also, in [29], offloading decisions were not optimized, [30] did not consider RAN RA, and [31] considered a binary offloading scheme.

A handful of research studies considered RA in sliced cellular networks [28, 32–36]. In [28], the authors minimized a weighted combination of energy consumption and delay through subchannel and computation RA. This work considered two slices on a single base station (BS) with no interference. In [32], the authors minimized delay through computation RA, considering multiple BSs, and in [33], the authors maximized the offloaded workload that can be supported in a given time at each fog node through energy optimization and server allocation. However, in both [32] and [33], the inter-cell interference was ignored and offloading decisions and RAN RA were not considered. The authors in [34] optimized the traffic allocation in a multi-tier sliced architecture, while preventing over-provisioning. However RAN and computation RA were considered abstractly, i.e., neither subchannel, power, and computation RA were considered, nor offloading decisions were optimized. Similarly, in [35], an abstract view of 'resource' was adopted to minimize the weighted system delay, i.e., RAN and computation RA were not addressed.

Recently, using stochastic optimization, joint subchannel, power and computation RA was considered in a multi-cell sliced network to minimize system energy consumption in [36], while ignoring offloading decisions. It should be noted that energy

consumption is a convex function of transmit power and subchannel allocation variables, and is different from delay, which at its simplest form, is a function of inverse of non-convex data rate. Also, when all users offload, as in [36], the delay can be easily restated in terms of the users' data rate. However, with offloading decision optimization, such simplifications are not applicable.

3.2 Novelty and Contributions

To our best knowledge, the problem of *delay minimization with joint offloading, computation, and communication RA in a cooperative multi-cell MEC network with or without slicing* is not investigated in the literature. Our contributions are:

- I jointly optimize users' offloading decisions, RAN and computing RA in a multi-cell MEC network to minimize the weighted sum of the difference between the delay observed at each slice and its corresponding desired delay. The fractional form of the objective function, discrete subchannel allocation, the partial offloading scheme, and the interference incorporated in the rate function, turns this problem into a mixed integer non-linear programming problem (MINLP) for which I proposed an efficient and novel algorithm.

- I decouple the original problem into two sub-problems: (i) offloading decision-making and (ii) joint computation resource, subchannel, and power allocation. I solve the first sub-problem optimally. For the second sub-problem, I propose an efficient algorithm with polynomial computational complexity, leveraging on tools from fractional programming and Augmented Lagrangian method (ALM). Using alternating optimization, I solve these two sub-problems iteratively until convergence. Complexity analysis is also presented.

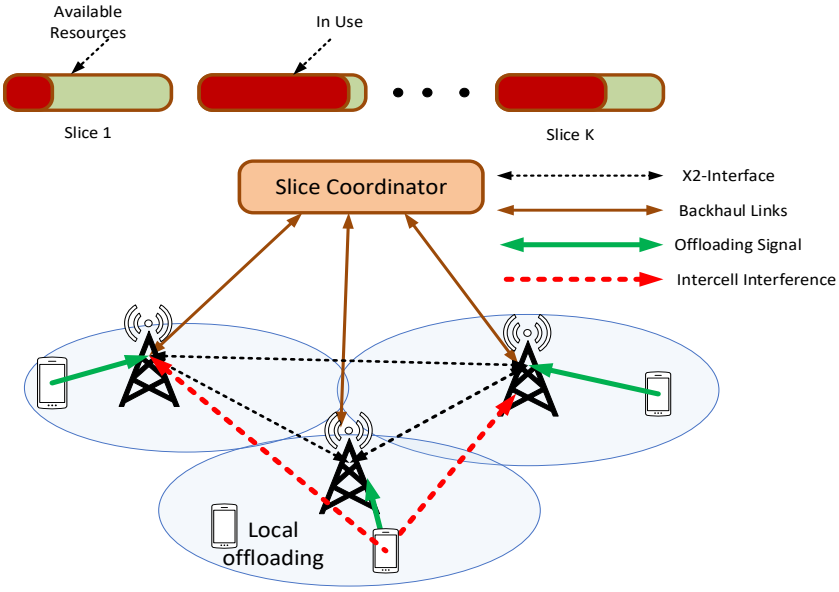

Figure 3.1: System Model

- Simulation results demonstrate the efficacy of my proposed algorithm compared to existing schemes and provide insights related to the impact of interference, slice prioritization, and cooperative MEC offloading, while demonstrating the convergence in a few iterations.

3.3 System Model and Assumptions

I consider a MEC network with M edge points (or BSs) with co-located servers[1] The set of MEC servers is denoted as $\mathcal{M} = \{1, 2, \cdots, M\}$. The available spectrum at each cell is divided into N subchannels each with bandwidth B. Network resources are sliced to accommodate $\mathcal{K} = \{1, 2, \cdots, K\}$ tenants each of which provide one specific type of service. Furthermore, the set of users for each tenant k is denoted by $\mathcal{U}_k$

[1]The edge nodes can connect to each other using any type of topology such as full-mesh or star topology.

and the set of all users is $\mathcal{U} = \{1, 2, \cdots, U\}$. Each tenant k has a SLA with InP in which the proportion of computation capacity, β_k^E, and available bandwidth, α_k, reserved for its users is determined. The task of each user u is represented by the tuple (L_u,C_u), with L_u as the size of the task and C_u as the computational demand (CPU cycles) to process each bit.

To facilitate slice resource management, I consider a software-defined network (SDN) controller referred to as slice coordinator (SC). The SC keeps track of resource utilization in each slice and ensures that service providers (SPs) follow resource constraints in SLA and do not exceed their share of resources. This network architecture is given in Fig. 3.1.

I denote $y_{u,j}$ as the proportion of the task of user u executed on the MEC server j. Thus, I have $\sum_{j\in\{\mathcal{M}\cup 0\}} y_{u,j} = 1,\ \forall u \in \mathcal{U}$, where index 0 denotes local computation.

3.3.1 Communication model

I consider that if a user offloads its task, it first sends its task to its assigned server denoted by m_u, and then the remaining communication (possible hand-offs between servers) would be done over the high speed backhaul links. Denoting $\hat{\mathcal{U}}_j$ as the set of users associated to server j, the data rate of each user u over subchannel n is:

$$r_{u,n} = B \log\left(1 + \frac{x_{u,n}\ p_{u,n} h_{u,m_u,n}}{\sigma^2 + I_{u,n}}\right) \tag{3.1}$$

where $p_{u,n}$, $I_{u,n}$, and σ^2 represent the transmit power of user u over subchannel n, its inter-cell interference calculated as $I_{u,n} = \sum_{j\in\{\mathcal{M}\setminus m_u\}} \sum_{u'\in\hat{\mathcal{U}}_j} x_{u',n} p_{u',n} h_{u',m_u,n}$, and receiver noise power, respectively. Also, $h_{u,j,n}$ is the path-gain between user u and BS j over subchannel n, and $x_{u,n}$ denotes the binary subchannel allocation variable

which is equal to one if subcarrier n is assigned to user u, and zero otherwise.

Now, I can calculate the total data rate of each user u as $R_u(\mathbf{X}, \mathbf{P}) = \sum_{n \in \mathcal{N}} r_{u,n}$, where $\mathcal{N}$, $\mathbf{X}$, $\mathbf{P}$, denote the set of N subchannels, subchannel allocation matrix, and transmit power allocation matrix, respectively. Denoting $\mathbf{Y}$ as the matrix of offloading decisions, the communication delay of user u is:

$$T^{\text{comm}}_{u,m_u}(\mathbf{X}, \mathbf{P}, \mathbf{Y}) = \frac{\sum_{j \in \mathcal{M}} y_{u,j} L_u}{R_u(\mathbf{X}, \mathbf{P})}. \tag{3.2}$$

3.3.2 Computing model

As a partial offloading scheme is adopted here, users' task may be partly processed locally. Denoting the computation capability of local device for user u as f^L_u, the local computation delay would be:

$$T^{\text{L}}_u(\mathbf{Y}) = \frac{y_{u,0} L_u C_u}{f^L_u}. \tag{3.3}$$

With $\mathbf{F}$ representing the matrix of all computation resource allocation variables, since the task of user u might be processed by servers other than its assigned server, the computation delay of user u is:

$$\begin{aligned} T^E_u(\mathbf{X},\mathbf{P}, \mathbf{F}, \mathbf{Y}) &= T^{\text{comm}}_{u,m_u}(\mathbf{X}, \mathbf{P}, \mathbf{Y}) \\ &+ \sum_{j \in \{\mathcal{M} \setminus m_u\}} y_{u,j} T^{\text{ho}}_{m_u,j} + T^{\text{comp}}_u(\mathbf{F}, \mathbf{Y}). \end{aligned} \tag{3.4}$$

where $T^{\text{ho}}_{m_u,j}$ denotes the hand-off delay, including the time for communicating with SC and the average round trip time for task transfer between m_u to the j^{th} server.

Moreover, T_u^{comp} denotes the offloading computation delay of user u. If tasks' fragments are processed sequentially (one after the other), T_u^{comp} would be the summation of delays of user u in each server j as in (3.5). In case of parallel processing, the computation delay of user would be equal to the delay in the slowest server. However, in order to retain a tractable form for the objective function, I consider an upper-bound and calculate the computation delay in both cases as follows:

$$T_u^{\text{comp}}(\mathbf{F}, \mathbf{Y}) = \sum_{j \in \mathcal{M}} \frac{y_{u,j} L_u C_u}{f_{u,j}}, \tag{3.5}$$

where $f_{u,j}$ represents the computation resource that is allocated to user u in server j. Note that even when parallel computation of the tasks is possible , due to 1) positivity of computation delay and 2) the independence between $f_{u,j}$ for different servers ,j, this upper bound would not significantly effect the optimized value of computation resource allocation in the slowest server, as minimizing the sum translates into minimizing each component separately. Due to the typically small size of response, I ignore the downlink transmission delay. Thus, the total delay of each user u is:

$$T_u(\mathbf{X}, \mathbf{P}, \mathbf{F}, \mathbf{Y}) = T_u^L(\mathbf{Y}) + T_u^E(\mathbf{X}, \mathbf{P}, \mathbf{F}, \mathbf{Y}). \tag{3.6}$$

3.4 Problem Formulation

In this section, I formulate the problem of minimizing the weighted sum of the difference between the delay observed at a given slice and its corresponding delay requirement (*or weighted sum of the delay deviation at each slice*), through jointly optimizing users' offloading decisions, RAN and computing RA in a cooperative multi-cell MEC network.

Remark: This problem offers SPs a valuable insight into the adequacy of their leased resources to meet the QoS requirement of their subscribers and the average delay they would experience under the existing SLA. Analyzing the results obtained, SPs can better plan their future strategies. If the value of objective function is negative for any slice, then it can be interpreted that the SP is over-provisioning resources and thus increasing its expenditure unnecessarily. Therefore, the SP can either increase the number of its subscribers or reduce the amount of leased resources. Otherwise, if the objective function is positive, it means some of the users in the slice are not obtaining their QoS requirement. In this case, depending on the type of service that the slice offers, the SP should decide to whether maintain the current SLA, invest more on leasing resources, or to modify the subscription policy to decrease the number of users allocated to the slice.

Now, I formally state the optimization problem as follows:

$$\mathbf{P}: \min_{\mathbf{X},\mathbf{P},\mathbf{F},\mathbf{Y}} \sum_{k\in\mathcal{K}} \sum_{u\in\mathcal{U}_k} \lambda_k (T_u(\mathbf{X},\mathbf{P},\mathbf{F},\mathbf{Y}) - \bar{T}_k)$$

Subject to:

$$\begin{aligned}
&C_1: \sum_{u\in\hat{\mathcal{U}}_j} x_{u,n} \leq 1, && \forall n \in \mathcal{N}, \forall j \in \mathcal{M},\\
&C_2: x_{u,n} \in \{1,0\}, && \forall n \in \mathcal{N}, \forall j \in \mathcal{M}, \forall u \in \hat{\mathcal{U}}_j,\\
&C_3: 0 \leq \sum_{n\in\mathcal{N}} x_{u,n} p_{u,n} \leq P_{\max,u}, && \forall u \in \mathcal{U},\\
&C_4: y_{u,0} L_u C_u \leq F_u^L, && \forall u \in \mathcal{U},\\
&C_5: \sum_{u\in\mathcal{U}} y_{u,j} L_u C_u \leq F_j^E, && \forall j \in \mathcal{M},\\
&C_6: \sum_{u\in\mathcal{U}_k} \sum_{n\in\mathcal{N}} x_{u,n} \leq \alpha_k MN, && \forall k \in \mathcal{K},\\
&C_7: \sum_{u\in\mathcal{U}_k} \sum_{j\in\mathcal{M}} f_{u,j} \leq \beta_k^E S^E, && \forall k \in \mathcal{K},\\
&C_8: \sum_{j\in\{\mathcal{M}\cup 0\}} y_{u,j} = 1, && \forall u \in \mathcal{U},\\
&C_9: y_{u,j} \in [0,1], && \forall u \in \mathcal{U}, \forall j \in \mathcal{M}.
\end{aligned} \tag{3.7}$$

In the above optimization problem, $\bar{T}_k$ denotes the desired delay threshold of each slice k and λ_k is the weighting factor whose value is defined in SLA and handles the precedence of slices over each other. Furthermore, constraint C_1 indicates that each subchannel can be allocated to at most one user in each cell and C_2 shows the binary nature of the subchannel allocation variable. In constraint C_3, users' transmit power is restricted between zero and a maximum threshold denoted by $P_{\max,u}$. In constraints C_4 and C_5, the limitation of local and edge computation resources are specified for

each user and server, respectively, with F_u^L and F_j^E denoting the total computation capacity of user u and server j, in that order. Constraints C_6 and C_7 ensure that resource consumption at each slice follows SLA. That is, C_6 limits the spectrum usage for each slice. Since there are M cells in the system and each cell has access to N subchannels, then in total we have NM subchannels, from which only α_k percent can be used by users of slice k. Similar to communication resources, the proportion of the total computation capacity S^E ($S^E = \sum_{j \in \mathcal{M}} F_j^E$) that is allocated to each slice k is limited to β_k^E as given in constraint C_7. Constraints C_8 and C_9 clarify the partial offloading decision scheme adopted in this work.

Also it should be noted that since resources in slices are isolated from each other, the value of objective function in one slice can not over-shadow the value of delay deviation in other slices and reduce the quality of resource allocation in them.

As the result of interference included in the rate function, the binary subchannel allocation variables, and the objective function which is in the form of summation of ratios, optimization problem (4.8) is MINLP and thus difficult to tackle. In the what follows I present my resource allocation algorithm.

3.5 Proposed Resource Allocation Framework

To tackle the difficulties of solving problem (4.8), I first take advantage of the problem structure and decompose it into the following two subproblems:

$$\begin{aligned} \mathbf{P1}: \min_{\mathbf{Y}} \ & \sum_{k \in \mathcal{K}} \sum_{u \in \mathcal{U}_k} \lambda_k (T_u(\mathbf{Y}) - \bar{T}_k) \\ & \text{Subject to: } C_4,\ C_5,\ C_8,\ C_9. \end{aligned} \tag{3.8}$$

$$\mathbf{P2}: \min_{\mathbf{X},\mathbf{P},\mathbf{F}} \sum_{k\in\mathcal{K}} \sum_{u\in\mathcal{U}_k} \lambda_k (T_u(\mathbf{X},\mathbf{P},\mathbf{F}) - \bar{T}_k)$$
$$\text{Subject to: } C_1 - C_3,\ C_6,\ C_7. \tag{3.9}$$

In problem (3.8), both the objective function and constraint set are affine with respect to the variable $\mathbf{Y}$. As such, it can be solved using standard optimization tools such as CVX toolbox. The first challenge in (3.9) is the multiplication of subchannel and power allocation variables in (1) as well as in constraint C_3. To tackle this challenge, I first replace all $x_{u,n}p_{u,n}$ terms with $p_{u,n}$ and then add the following constraint to (3.9):

$$C_{3,1}: 0 \leq p_{u,n} \leq x_{u,n} P_{max,u} \tag{3.10}$$

By using the above modification, users' transmit power would be automatically set to zero over subchannels they do not own. By adding this constraint, data rate function $R_u(\mathbf{X},\mathbf{P})$ would become a function of trasmit power only ($R_u(\mathbf{P})$). This step solves the variable multiplication issue, however discrete subchannel allocation variable is still challenging. To deal with this issue I replace C_2 with the following two constraints:

$$C_{2,1}: 0 \leq x_{u,n} \leq 1, \quad C_{2,2}: x_{u,n} - x_{u,n}^2 \leq 0. \tag{3.11}$$

Remark 2: Although I relax $x_{u,n}$ to a continuous variable in $C_{2,1}$, since the only two values in [0,1] that fit $C_{2,2}$ are 0 and 1, the binary nature of this variable would be preserved.

The fractional form of users' delay, T_u, is the next issue I focus on. After offloading decision is obtained through solving subproblem **P1**, edge computation delay,

$T_u^{\text{comp}}(\mathbf{F}, \mathbf{Y})$, in the objective function of **P2** would turn into a convex function and hand-off delay would be a constant. This leaves us with the summation of users' transmission delay, whose non-convexity can be easily proved.

Lemma 1. *Using the tools from FP, problem (3.9) can be restated as:*

$$\min_{\mathbf{X},\mathbf{P},\mathbf{F}} T(\mathbf{P},\mathbf{F}) = \sum_{k\in\mathcal{K}}\sum_{u\in\mathcal{U}_k} \lambda_k \Big[y_{u,j} L_u \frac{1}{2R_u(\mathbf{P})^2} \sum_{j\in\mathcal{M}, j\neq m_u} y_{u,j} T^{\text{ho}}_{m_u,j} + \sum_{j\in\mathcal{M}} \frac{y_{u,j} L_u C_u}{f_{u,j}} - \bar{T}_k \Big]$$
$$\text{S.to}: C_1 - C_3,\ C_6,\ C_7. \tag{3.12}$$

Proof. An optimization problem with the form $\min_{x\in C_x} \sum_{i=1}^{I} \frac{B_i(X)}{A_i(X)}$, can be restated equivalently as [37]:

$$\min_{x\in C_x, t\in\mathbb{R}^+} \sum_{i=1}^{I} t_i B_i(X)^2 + \sum_{i=1}^{I} \frac{1}{4t_i} \frac{1}{A_i(X)^2}, \tag{3.13}$$

where $t_i = \frac{1}{2B_i(X)A_i(X)}$. Using (3.13) and by setting $B_i = 1$ and $A_i = R_u(\mathbf{P})$, I restate problem (3.9) as given in Lemma 1. □

Due to the presence of interference, $R_i(\mathbf{P})$ is still a non-convex function.

Lemma 2. *I obtain an equal but convex representation of communication delay function by restating the rate as:*

$$\hat{r}_{u,n}(\mathbf{P}, z_{u,n}) = \log_2 \Big(1 + 2z_{u,n}\sqrt{h_{u,m_u,n} p_{u,n}} - z_{u,n}^2 (I_{u,n} + \sigma^2) \Big), \tag{3.14}$$

Proof. As mathematically proven in [38] and since in **Lemma 1**, I set $A_i = R_u(\mathbf{P})$, and $R_u(\mathbf{P}) = \sum_{n\in\mathcal{N}} r_{u,n}$, $r_{u,n}$ can be equally restated as (3.14). This modification, not only makes $r_{u,n}$ a concave function of $\mathbf{P}$, $\frac{1}{R_u(\mathbf{P})^2}$ would also become convex. □

In (3.14), $z_{u,n}$ is a slack variable that will be updated iteratively. Using **Lemma 2**, I convexify the complex non-convex function $T^{\text{comm}}_{u,m_u}(\mathbf{P})$, also I redefine $R_u(\mathbf{P}) = \sum_{n\in\mathcal{N}} \hat{r}_{u,n}(\mathbf{P}, z_{u,n})$. For optimizing $\mathbf{P}$, $\mathbf{X}$, and $\mathbf{F}$ I adopt ALM to obtain a locally optimal solution. For a given $z_{u,n}$, the the augmented Lagrangian function is given in (3.15).

$$\begin{aligned}
\min L(\mathbf{X},\mathbf{P},\mathbf{F},\mathbf{Z},\mathbf{\Gamma}) = T(\mathbf{X},\mathbf{P},\mathbf{F},\mathbf{Z}) + \frac{1}{2\Psi}\Bigg[\Bigg(\Bigg[\sum_{u\in\mathcal{U}}\theta_u + \Psi(\sum_{n\in\mathcal{N}} p_{u,n} - P_{max,u})\Bigg]^+\Bigg)^2 - \sum_{u\in\mathcal{U}}\theta_u^2 \\
+ \Bigg(\Bigg[\sum_{k\in\mathcal{K}}\delta_k + \Psi\big(\sum_{u\in\mathcal{U}_k}\sum_{j\in\mathcal{M}} f_{u,j} - \beta_k^E S^E\big)\Bigg]^+\Bigg)^2 - \sum_{k\in\mathcal{K}}\delta_k^2 + \Bigg(\Bigg[\sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{M}}\phi_{n,j} + \Psi\big(\sum_{u\in\mathcal{U}} x_{u,n} - 1\big)\Bigg]^+\Bigg)^2 \\
- \sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{J}}\phi_{n,j}^2 + \Big(\Bigg[\sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{M}}\sum_{u\in\mathcal{U}}\xi_{u,n,j} + \Psi\Big(\sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{M}}\sum_{u\in\mathcal{U}_j} x_{u,n} - x_{u,n}^2\Big)\Bigg]^+\Big)^2 - \sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{M}}\sum_{u\in\mathcal{U}}\xi_{u,n,j}^2 \\
+ \Bigg(\Bigg[\sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\Xi_{u,n} + \Psi(\sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}} p_{u,n} - x_{u,n}P_{max,u})\Bigg]^+\Bigg)^2 - \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\Xi_{u,n}^2\Bigg]. \qquad (3.15)
\end{aligned}$$

In the augmented Lagrangian function, Ψ is a positive constant that plays the role of an adjustable penalty coefficient and Γ is the vector of all Lagrangian multipliers Θ, Δ, Φ, ξ, and Ξ. Solving problem (3.9) can be done in three steps. In the first step, I consider Lagrangian multipliers to be fixed and minimize $L(\mathbf{X},\mathbf{P},\mathbf{F},\mathbf{Z})$ given in (3.15). In the second step, Lagrangian multipliers would be updated as:

$$\theta_u^{t+1} = \Bigg[\theta_u^t + \Psi\Bigg(\sum_{n\in\mathcal{N}} p_{u,n} - P_{max,u}\Bigg)\Bigg]^+, \qquad (3.16)$$

$$\delta_k^{t+1} = \Bigg[\delta_k^t + \Psi\big(\sum_{u\in\mathcal{U}_k}\sum_{j\in\mathcal{M}} f_{u,j} - \beta_k^E S^E\big)\Bigg]^+, \qquad (3.17)$$

$$\phi_{n,j}^{t+1} = \left[\phi_{n,j}^{t} + \Psi\big(\sum_{u\in\mathcal{U}} x_{u,n} - 1\big)\right]^{+}, \tag{3.18}$$

$$\xi_{u,n,j}^{t+1} = \left[\sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{M}}\sum_{u\in\mathcal{U}}\xi_{u,n,j}^{t} + \Psi\left(\sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{M}}\sum_{u\in\hat{\mathcal{U}}_j} x_{u,n} - x_{u,n}^{2}\right)\right]^{+} \tag{3.19}$$

$$\Xi_{u,n}^{t+1} = \left[\sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\Xi_{u,n}^{t} + \Psi\Big(\sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}} p_{u,n} - x_{u,n}P_{max,u}\Big)\right]^{+}. \tag{3.20}$$

The third step is executed after a solution is obtained for (3.15). In this last step, using the values obtained for $\mathbf{P}$ and $\mathbf{X}$, I update slack variable $z_{u,n}$ as $z_{u,n} = \frac{\sqrt{p_{u,n}^{*} h_{u,m_u,n}}}{(I_{u,n}+\sigma^2)}$. Our proposed algorithm is given in **Algorithm 1**.

Algorithm 1 Proposed Algorithm

1: Obtain the solution of problem (3.8) and initialize Z.
2: **Repeat**
3: Initialize $\Gamma = [\Theta, \Delta, \Phi, \xi, \Xi]$ with small numbers.
4: **Repeat**
5: Solve problem (3.15) considering Γ to be fixed,
6: Update Γ using (3.16), (3.17), (3.18), (3.19), and (3.20).
7: **Until** convergence.
8: Set $z_{u,n} = \frac{\sqrt{p_{u,n}^{*} h_{u,m_u,n}}}{(I_{u,n}+\sigma^2)}$ for all users and subchannels.
9: **Until** Convergence

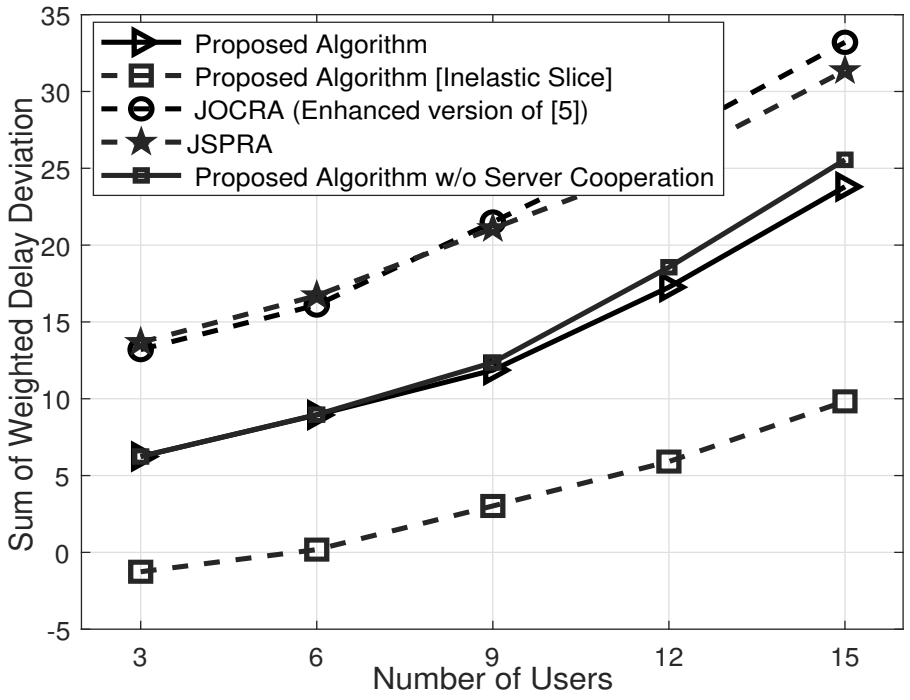

Figure 3.2: Weighted delay deviation Vs. Number of users

3.6 Computation Complexity Analysis

My proposed algorithm is divided into two sub-problems, i.e., (i) offloading decision optimization and (ii) joint computation and RAN RA. For the first sub-problem, I use interior point method in CVX whose complexity is in the order of $O(\log(\frac{C/t_0\xi}{\epsilon}))$, where C, t_0, ξ, and ϵ denote the total number of constraints, the initial point for interior point method, the stopping criterion, and a representation of the accuracy of the method, respectively. For the second sub-problem based on ALM the order of complexity at each iteration is $\mathcal{O}(KUM)^2$ which is polynomial [39].

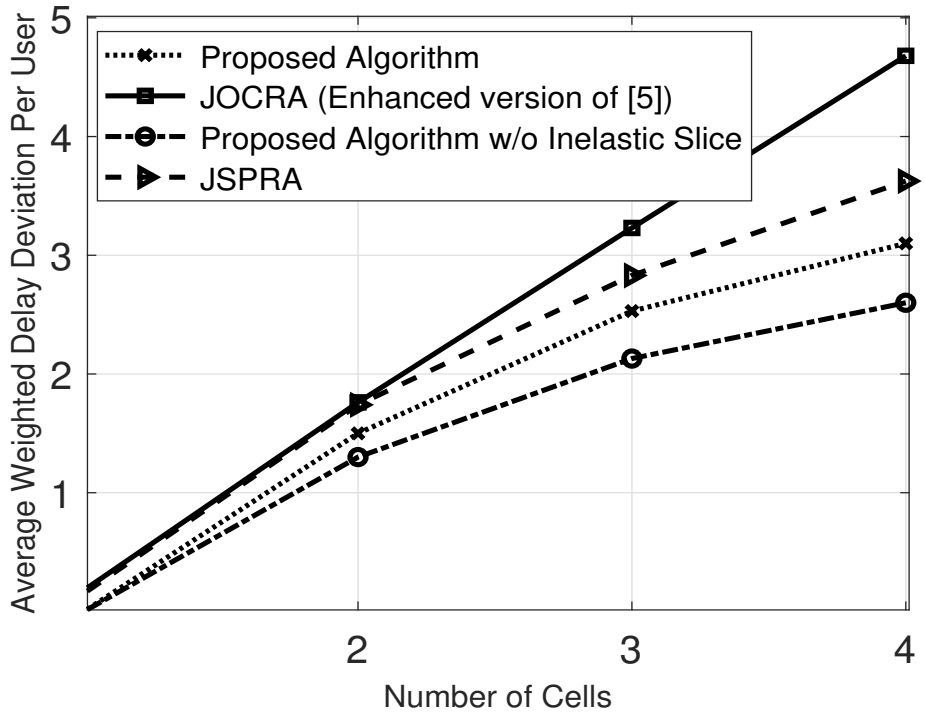

Figure 3.3: Weighted delay deviation Vs. Number of cells

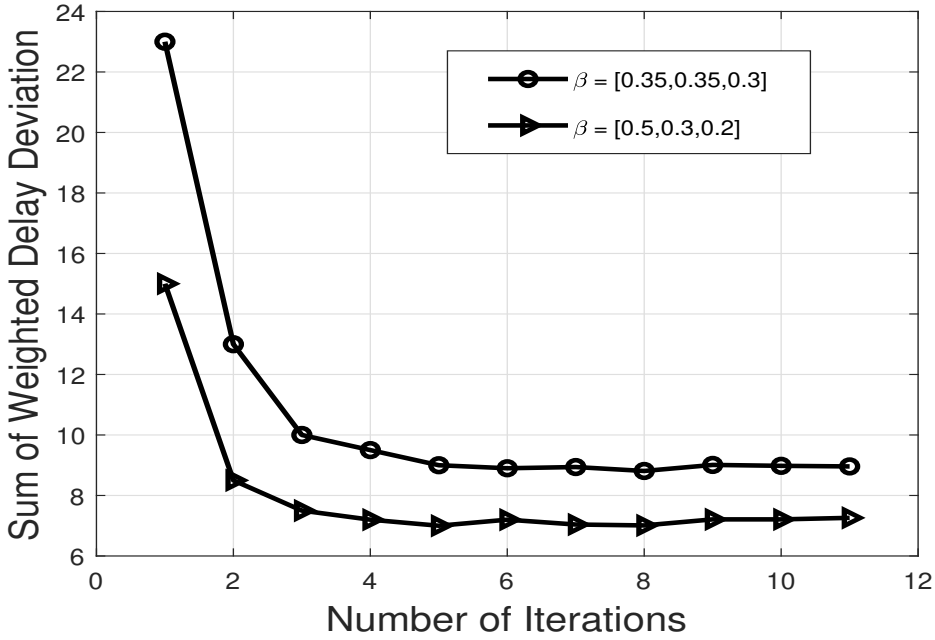

Figure 3.4: Weighted delay deviation Vs. Number of iterations

3.7 Simulation Results and Discussions

I consider a network with two cells each having 6 users and 16 available subchannels, unless stated otherwise. Similar to [35], I consider three slices/services as: *elastic services* with flexible latency constraints, *inelastic services* that require ultra-low latency, and *background services* with low latency requirement. The weighting parameter λ is set to $[3, 2, 1]$ for inelastic, elastic, and background services, with 50ms, 100ms, and 5s desired delay threshold, respectively. The value of L_u is 1 MB and the CPU cycle, C_u, is randomly chosen from $[1500, 2000, 2500]$.

Fig. 3.2 depicts the effect of number of users in each cell on the sum of weighted delay deviation at each slice. I have compared my algorithm with

1. **Joint Offloading and Computation RA (JOCRA):** where only offloading and computing RA is considered (with interference and server cooperation, this scenario is in fact an improvement on [32]),

2. **Joint offloading, Subchannel, Power RA (JSPRA):** in which only RAN RA is addressed and computation resource is equally allocated to users,

3. **Proposed scheme without server cooperation:** We can clearly observe the significance of joint computation and RAN RA in the delay that users experience. In fact, if we ignore computation RA we would have 58% and if we overlook communication RA we will have 62% increase in network delay deviation on average.

In Fig. 3.2, the impact of cooperation among cells is also illustrated. At first, when number of users is not too high, there is almost no need for cooperation. However, as the number of users increases, we observe that the effect of cooperation becomes

noteworthy (i.e., 9% reduction on average). The positive delay deviation occur when network becomes infeasible (i.e., insufficient resources in at least one slice) and satisfying the QoS of high priority services takes precedence in the network. Thus, we can preserve the QoS of slices by increasing their weight (λ_k) for prioritization of the slice or the quota of reserved resources (β and α) to avoid infeasibility. However, such modifications are often a function of the cost SPs are willing to pay.

In Fig. 3.3, I examine how increasing the number of cells impacts the delay of users. I again compare my proposed algorithm with *JOCRA* and *JSPRA*. As the number of users per cell remains constant here, we depict the average delay deviation per user. Increasing the number of cells notably increases the delay of users, however this increase is more significant when communication RA is overlooked. Since while the average amount of resources available for users remains almost the same since the number of users in each cell is constant, more cells means intensified interference in the network. To deal with the negative effect of this intensified interference, precise RAN RA becomes imperative.

The convergence of my proposed algorithm and the importance of slice resource management is numerically demonstrated in Fig. 3.4. Here, we observe that: i) my algorithm converges to its final solution after a few iterations, and ii) careful resource reservation plays a significant role in the QoS users of each slice achieve.

3.8 Summary

In this chapter, I proposed a framework to minimize the delay in cooperative MEC network by optimizing both RAN and computation resources and offloading decisions, using tools from fractional programming, convexification of rate function, and ALM.

Analyzing the results of this optimization problem provide deeper insights to service providers on the adequacy of their leased resources to meet their service quality and helps them better plan their future investment strategies to whether maintain their current SLA, invest more on leasing resources, or to modify their subscription policy.

Chapter 4

Federated DDQN for Joint Delay and Energy Minimization in IoT networks

Massive connectivity is among one of the most challenging requirements of Internet-of-Things (IoT) networks which necessitates efficient, scalable, and low-complexity network resource management. Furthermore, due to limited computation and battery capacity of the IoT devices, it is often impossible for them to process their resource-intensive tasks within a predefined deadline. In the sequel, mobile cloud computing (MCC) and MEC enable IoT devices to offload their tasks to the cloud or edge servers to access their substantial processing capabilities at the expense of having to transmit the tasks over dynamic wireless channels. Subsequently, to take full advantage of the MCC and MEC paradigms, it becomes essential to carefully optimize offloading decisions, communication, and computation resources.

For example, the amount of energy an IoT device need to spend on processing a given task can be optimized to improve the performance of device and ensure its tasks' QoS requirement.

4.1 Literature review

Most of the existing research works solved the joint offloading decision, communication, and computation resource allocation problem leveraging on tools from optimization theory [23, 40]. However, the algorithms were typically non-scalable, time-consuming, and computationally expensive.

Unlike optimization frameworks, deep reinforcement learning (DRL) enables agents to learn by interacting with the environment. This unique approach to learning, turns DRL into an ideal problem-solving tool in dynamic environments. Yet, most DRL algorithms are centralized and thus suffer from lack of scalability when the number of devices grow. Also, the computational complexity of finding an optimal policy may increase exponentially as the state space and action space grow. Furthermore, the centralized learning requires IoT devices to share their information in order to train the global model which may *violate their privacy* and create *unnecessary communication overhead* on the already scarce frequency spectrum.

Recently, federated learning (FDL) has emerged as a new paradigm for cooperative learning, where multiple nodes contribute in training a single global model. The devices use their local datasets to train and then offload their local models to the central unit for global aggregation. FDL enhances the cooperation between agents and scalability of the network resource management algorithms. Furthermore, FDL does not require local agents to share their data with any external entity, thereby preserves the privacy of each agent [41].

In [42], the problem of computation resource allocation was addressed considering an FDL system. However, FDL is only considered to formulate an optimization problem which is later on solved by using a *centralized* actor-critic agent and without

using FDL in the solution approach.

None of the aforementioned research works applied FDL to enhance the efficacy of solving a realistic wireless resource allocation problem. Very recently, [43,44] adopted FDL to facilitate the learning process in DRL, i.e., local DRL models were trained and then integrated together to cooperatively develop a comprehensive global DRL model. However, in [43], a cooperative caching scheme was proposed and offloading decisions were not considered. In [44], computational offloading was considered; however, the network was modeled as a queuing system, transmit power was modeled as an integer variable whose maximum value is equal to the maximum length of the energy queue.

Also, in [44], no explicit quality-of-service (QoS) was guaranteed for users' tasks and computation resource allocation was overlooked.

4.2 Contributions

In this part of thesis, I propose a federated DRL (FedRL) framework to solve a multi-objective optimization problem, where I consider minimizing the expected task completion delay and energy consumption of IoT devices. This is done by optimizing offloading decisions, computation resource, and transmit power allocation. Since the formulated problem is a mixed-integer non-linear programming programming (MINLP), I first reformulate my problem as a multi-agent DRL problem and address it using double deep Q-network (DDQN), where the actions are offloading decisions. The immediate cost is calculated through solving either the transmit power or local computation resource optimization, depending on the offloading decisions (actions). Then, to enhance the learning quality and speed of DRL, I incorporate FDL at the end of each episode. FDL enhances the scalability of the proposed DRL

framework, creates a context for cooperation between agents, and minimizes their privacy concerns. Numerical results demonstrate the efficacy of the federated DDQN framework in terms of learning speed compared to federated deep Q-network (DQN) and non-federated DDQN algorithms.

4.3 System Model and Assumptions

I consider a network containing one MEC server, one cloud server, and $\mathcal{N} = \{1, ..., N\}$ IoT devices with limited computation and energy resources. I consider a given time horizon $\mathcal{T}$ which is divided into T time steps. At each time t, device i needs to process one of the tasks in its queue, defined with the tuple $(L_{i,t}, C_{i,t}, \hat{T}_{i,t})$, where $L_{i,t}$ is the size of the task (in bits), $C_{i,t}$ denotes the CPU cycle requirement of the task, and $\hat{T}_{i,t}$ denotes the maximum delay threshold of the task. At any time t, devices can either execute their task locally or offload it to edge or cloud server.

Let us denote local offloading decision of device i at time t as $x_{i,t}$, where $x_{i,t} = 1$ means the task would be performed locally, and $x_{i,t} = 0$, otherwise. Similarly, I define MCC and MEC offloading variable of device i by $z_{i,t}$ and $y_{i,t}$, respectively. If device i offloads its task to the cloud $z_{i,t} = 1$ and if it offloads the task to the edge server $y_{i,t} = 1$. As a binary offloading decision is considered, I have:

$$x_{i,t} + y_{i,t} + z_{i,t} = 1. \ \forall t \in \mathcal{T}. \tag{4.1}$$

When device i offloads its task (whether to MEC server or to the cloud), the delay and energy consumption would depend on the channel condition, the size of the task, and the power with which the device transmits its task and, in case of local computation they depend on computation resource utilization. In what follows, I model the delay

and energy consumption that an IoT device would experience, given its offloading decision.

If the device i decides to offload its task, it should first transmit it to the MEC-enabled base station (BS) through wireless channels. At time step t, the transmission data rate of this user, denoted by $r_{i,t}$ is calculated as follows:

$$r_{i,t} = B \log_2(1 + \frac{p_{i,t} h_{i,t}}{\sigma^2}), \tag{4.2}$$

where B and $p_{i,t}$ denote the bandwidth and transmit power of device i at time step t, respectively. Also, $h_{i,t}$ and σ^2 represent the path-gain of device i at time t and the receiver noise. Thus, the communication delay and energy consumption of device i, while offloading is given, respectively, as follows:

$$T_{i,t}^{\text{comm}} = \frac{L_{i,t}}{r_{i,t}}, \tag{4.3}$$

$$E_{i,t}^{\text{comm}} = p_{i,t} T_{i,t}^{\text{comm}} = \frac{p_{i,t} L_{i,t}}{B \log_2(1 + \frac{p_{i,t} h_{i,t}}{\sigma^2})} \tag{4.4}$$

If device i offloads its task to the edge server the computation delay would be $T_{i,t}^{e} = \frac{C_{i,t}}{F^e}$, where F^e denotes the average computation capacity of edge server. Also, if device i offloads the task to cloud server, the computation delay would be $T_{i,t}^{c} = \frac{C_{i,t}}{F^c}$, where F^c represents the average computation capacity of the cloud server.

From the perspective of IoT device, the energy that is consumed for processing a task when it is offloaded to either of the servers, is the energy spent on the transfer of the task. Therefore, both cloud and edge computing energy utilization at step t, denoted by $E_{i,t}^{c}$ and $E_{i,t}^{e}$, would be equal to $E_{i,t}^{\text{comm}}$.

If device i chooses to perform its task locally, the local computation delay and energy consumption would depend on the amount of computation resource allocated to process the task at time t, which I denote by $f_{i,t}^L$. Thus, the local delay and energy consumption of the device is modeled as follows:

$$T_{i,t}^L = \frac{C_{i,t}}{f_{i,t}^L}, \quad E_{i,t}^L = \kappa_i (f_{i,t}^L)^2. \tag{4.5}$$

Note that higher resource utilization (transmit power or computation capacity) , decreases the task completion delay at the expense of increased energy consumption. Therefore, this trade-off must be carefully managed through efficient offloading decision making and precise optimization of $f_{i,t}$ and $p_{i,t}$ in case of local computation and offloading, respectively.

4.4 Multi-objective Problem Formulation

In this section, I formulate the multi-objective problem of jointly minimizing the long-term delay and energy consumption of an IoT device in a decentralized manner over a specified time horizon $\mathcal{T}$. The long-term expected cost (weighted sum of delay and energy consumption) for each IoT device i is formulated, respectively, as follows:

$$\begin{aligned} T_i(\mathbf{p}_i, \mathbf{f}_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i) = \mathbb{E}[\lim_{t\to\infty} \frac{1}{t} \sum_{j=0}^{t} \big(& x_{i,j} T_{i,j}^L + \\ & y_{i,j}(T_{i,j}^e + T_{i,j}^{\text{comm}}) + z_{i,j}(T_{i,j}^c + T_{i,j}^{\text{comm}} + \Psi)\big)], \end{aligned} \tag{4.6}$$

$$E_i(\mathbf{p}_i, \mathbf{f}_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$$
$$= \mathbb{E}[\lim_{t\to\infty} \frac{1}{t} \sum_{j=0}^{t} \left(x_{i,j} E^L_{i,j} + y_{i,j} E^e_{i,j} + z_{i,j} E^c_{i,j}\right)], \tag{4.7}$$

where $\mathbf{p}_i = [p_{i,1}, p_{i,2}, \cdots, p_{i,t}]$, $\mathbf{f}_i = [f_{i,1}, f_{i,2}, \cdots, f_{i,t}]$, $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \cdots, x_{i,t}]$, $\mathbf{y}_i = [y_{i,1}, y_{i,2}, \cdots, y_{i,t}]$, and $\mathbf{z}_i = [z_{i,1}, z_{i,2}, \cdots, z_{i,t}]$, represent the vectors of transmit powers, computation resource allocation, local computing, edge offloading, and cloud offloading decision of device i, respectively. As cloud server is generally located far from the IoT devices, the delay of accessing cloud is commonly more than the delay of offloading to the edge server which is located at the edge of the network. In the equation (4.6), Ψ denotes the delay of accessing the cloud server, including the time necessary to transfer the task from BS to the cloud, the possible routing in the path, and the response delay.

For any given device i at time t, I model my problem as:

$$\min_{\mathbf{p}_i, \mathbf{f}_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i} \quad T_i + \lambda_i E_i$$

subject to:

$$\begin{aligned}
&\text{C1}: f^L_{i,t} <= F_{\max,i}, \quad \forall t \in \mathcal{T}, \\
&\text{C2}: x_{i,t} E^L_{i,t} + y_{i,t} E^e_{i,t} + z_{i,t} E^c_{i,t} <= E_{\max,i}, \quad \forall t \in \mathcal{T}, \\
&\text{C3}: T_{i,t} <= \hat{T}_{i,t}, \ \forall t \in \mathcal{T}, \\
&\text{C4}: x_{i,t} + y_{i,t} + z_{i,t} = 1, \quad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \\
&\text{C5}: x_{i,t}, y_{i,t}, z_{i,t} \in \{0, 1\}, \quad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}.
\end{aligned} \tag{4.8}$$

In the above optimization problem, λ_i is a weighting factor whose value should

be carefully selected based on the heterogeneity of resources available at each individual IoT device. If device i is more restricted in the energy resource compared to computation resource, the value of λ_i should be set to a larger number. Otherwise, λ_i should be a small number. Furthermore, constraint C1 indicates the local computation capacity with the maximum threshold $F_{\max,i}$. Constraint C2 represents the restriction on the energy resource of the device and that energy utilization should not exceed $E_{\max,i}$. Furthermore, constraints C4 and C5 define the binary offloading scheme adopted in this problem. It can be proven that both equations (4.6) and (4.7) are convex with respect to the variables $p_{i,t}$ and $f_{i,t}$, respectively. However, with binary offloading variables (x_i, y_i, and z_i) included, (4.8) turns into a MINLP that cannot be solved in an acceptable time span.

4.5 Proposed Federated DDQN Algorithm

To solve (4.8) at each IoT device, I propose a DDQN algorithm, and solve the problem in the following two phases:

• **Offloading Decision Optimization:** Since each IoT device has three options to process a task (namely local, edge server, or cloud server computing), the number of possible offloading policies (from the perspective of a centralized controller) at each given time step would exponentially increase as the number of devices surges in the system. To address this problem, I apply a multi-agent DDQN framework where each IoT device would train their local DDQN models using their local data.

• **Computing and communication Resource allocation:** Given the offloading decision, I optimize computation capacity or transmit power of the devices to minimize the weighted sum of energy consumption and delay. I use optimization

theory to address this part of the problem and then feed the results into the DDQN framework as the immediate cost function. In this way, I provide the learning agent with a real sense of the quality of the adopted offloading policy that reflects many important aspects of the system model (such as limitation of resources in each device and their QoS demands).

After the DDQN agent is trained through the above mentioned process for one training round, I apply a federated learning framework where each IoT device will train their DDQN models, share their models with the centralized controller, and update their models to the central aggregating unit. This mechanism is detailed in the flowchart provided in Fig. 4.1.

In what follows, I first focus on developing local models through DDQN algorithm and then explain how FDL would be deployed.

4.5.1 Double deep Q-network for Offloading Decision-making

In the first step of my algorithm, I model my problem as a multi-agent DDQN problem. For each device (DRL agent), I have following components:

1. *State space*: the state space for each agent i, denoted by s_i, consists of the following components: the length of the task queue of device i (tasks that are not yet processed or are not successfully processed, would be kept in this queue) which is denoted by $\mathcal{L}_{i,t}$, the path gain of the IoT device $h_{i,t}$, the size of the task currently being processed $L_{i,t}$, its CPU cycle requirement, $C_{i,t}$, and available resources. Thus, $s_i = \{\mathcal{L}_{i,t}, h_{i,t}, L_{i,t}, C_{i,t}, E_{\max,i}, F_{\max,i}\}$. If a task is successfully processed under a given offloading decision policy (its QoS requirement is satisfied), it would be removed from the task queue of the device. Otherwise, it

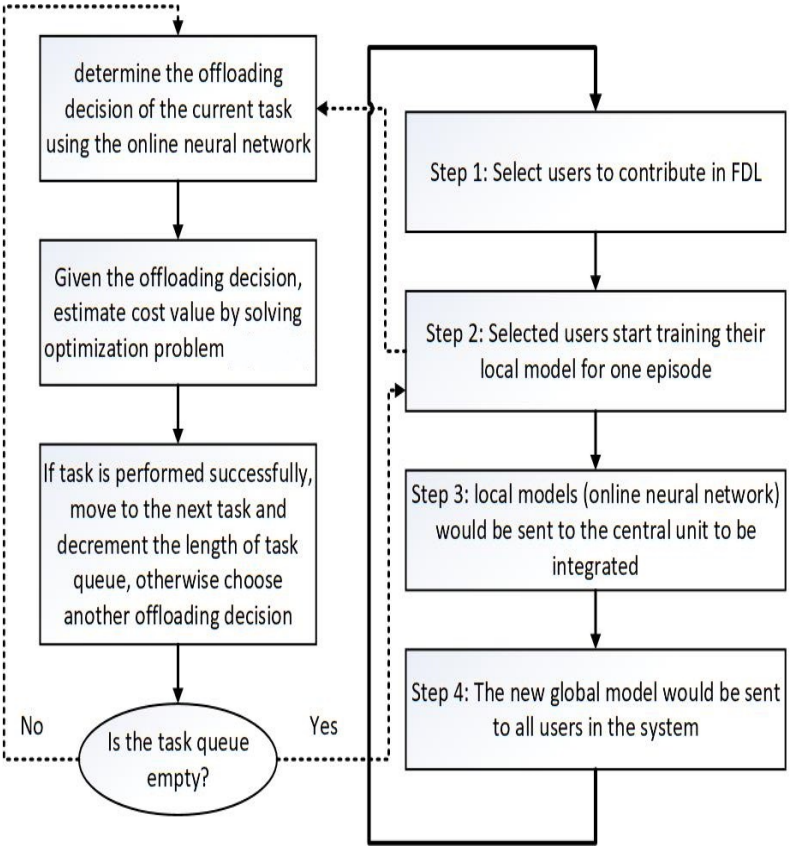

Figure 4.1: The federated reinforcement learning process

will remain at the top of the queue to be processed under another offloading decision.

2. *Action space*: The action space of agents, denoted by $\mathcal{A}$, contains possible offloading decisions, i.e., whether to process the task locally or offload.

3. *Cost*: (4.8) suggests that the cost of an agent is equal to the weighted summation of the delay and energy consumption given in the objective function. The value of this objective function and thus the cost depends on the value of $p_{i,t}$ in case of offloading and $f_{i,t}$ if local computation is selected. Therefore, to ensure that the cost function accurately reflects the benefit of a given offloading decision, these

variables should be carefully optimized. To this end, when local computation is selected ($x_{i,t} = 1$), the cost would be calculated solving the instantaneous optimization problem below:

$$\begin{aligned} &\min_{f_{i,t}} \ T^L_{i,t} + \lambda E^L_{i,t} \\ &\text{subject to: C1, C2, C3.} \end{aligned} \tag{4.9}$$

In case offloading is selected ($y_{i,t} = 1$ or $z_{i,t} = 1$), the transmit power would be optimized by solving the following optimization problem:

$$\begin{aligned} \min_{p_{i,t}} \ & y_{i,t}(T^e_{i,t} + T^{\text{comm}}_{i,t} + \lambda E^{\text{comm}}_{i,t}) \\ & + z_{i,t}(T^c_{i,t} + T^{\text{comm}}_{i,t} + \Psi + \lambda E^{\text{comm}}_{i,t}) \\ \text{subject to: C2, C3.} & \end{aligned} \tag{4.10}$$

The design of state and cost function has a significant impact on the success of DDQN in finding the optimal offloading policy, π^*. By using a multi-agent approach, I am in fact limiting the state and action space and focus on each device separately. Also, by modeling the cost function as an optimization problem not only can we optimize the local resource utilization and enforce system constraints, but also we can provide the agent with an accurate estimation of the quality of an offloading decision. Note that it can be easily proved that both (4.9) and (4.10) are convex single variable optimization problems that can be solved using standard softwares.

Let us denote the immediate cost of each device i obtained from the solution of the above mentioned optimization process as $u_i(s, a)$. Using Bellman equation, the

action-state value is:

$$Q_i(s,a) = u_i(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}(a) \max_{a' \in \mathcal{A}} Q_i(s', a'), \tag{4.11}$$

where $\mathcal{S}$, $P_{ss'}(a)$, and γ are the set of states, the transition probability function, and the discount factor, respectively. To overcome the need for having a full model of environment, calculating the transition probability function, and to acquiring a more stable learning process, DDQN is employed in this work. Each agent i has two neural networks working alongside each other, one called *online network* with parameters θ_i^{online} and the other called *target network* with parameters θ_i^{target}. At each training iteration the target value for training the online network in device i is calculated as:

$$L_i = u_i(s,a) + \gamma Q_i(s', \arg\max_{a' \in \mathcal{A}} Q_i(s', a'; \theta_i^{\text{online}}), \theta_i^{\text{target}}) \tag{4.12}$$

While θ_i^{online} is updated at every iteration, the frequency of change in θ_i^{target} is typically much lower and only once in every f_{update} rounds, θ_i^{target} would be set equal to θ_i^{online}.

As discussed before, training a DRL agent in a centralized manner can lead to critical issues related to scalability, agents' privacy, and additional communication overheads. On the other hand, training a DRL agent in a distributed manner can impact the overall performance gains (e.g., an agent might consume a longer time to train its model). As such, I consider FDL to combine the benefits of both centralized and distributed learning. FDL enables each agent to trains its own local model, using its own local data. Then these local models are sent to a central aggregation unit to be combined together. This process continues until a criterion is met.

4.5.2 Federated DRL Approach

The steps to train the FedRL agents are presented in the following:

4.5.2.1 Device selection strategy

At the beginning of each iteration of FDL (after each episode of DRL), a set of IoT agents are selected to participate in the FDL process. Let us consider that of all N devices in the network, only a small subset, denoted by $\mathcal{I} = \{1, ..., I\}$ is selected to contribute in FDL process. In this work, the device selection is done based on the following criterion:

$$\arg\max_{i \in \mathcal{N}} \ \mathrm{Var}(\frac{d_i P_{\max,i}}{F_{\max,i}}), \tag{4.13}$$

where d_i represents the distance of device from BS and the function Var stands for variance. This criterion helps in identifying devices whose experiences are more heterogeneous and thus can contribute more in the the learning process.

4.5.2.2 Training local models

As explained previously , all IoT devices use DDQN to train their local models. After this local training is finished (no more unprocessed task remains in the queue), the weights of online network, θ_i^{online}, is extracted in each agent and is then sent to the central aggregating unit.

4.5.2.3 Model aggregation

When central unit receives the models of participating IoT devices, it would aggregate the models which results in a single global model that would be then transmitted to all agents. For the purpose of aggregation, I utilize FedAvg [45], and perform model

aggregation as:

$$\theta^{\text{global}} = \frac{\sum_{i \in \mathcal{I}} \theta_i^{\text{online}}}{|\mathcal{I}|}. \tag{4.14}$$

This global model, which has integrated the experiences of all devices, is then transmitted back to IoT devices and the three steps above would be repeated. Note that the convergence of FedAvg even on Non-iid datasets is proven [46]). The details of my proposed framework is provided in **Algorithm 2** as well as the flowchart given in Fig. 4.1.

Algorithm 2 Proposed federated DDQN algorithm

Initialize the global model θ^{global}, and set maximum FDL iterations to K.
For each agent initialize online and target networks as: $\theta_i^{\text{online}} = \theta_i^{\text{target}} = \theta^{\text{global}}$.
While $(K \geq 0)$:
  Set $\theta_i^{\text{online}} = \theta_i^{\text{target}} = \theta^{\text{global}}$, $\forall i \in \mathcal{N}$,
  Select the set of participating devices, $\mathcal{I}$, based on (4.13),
  For each device i in $\mathcal{I}$ **do**:
    For each time step t if $|\mathcal{L}_{i,t}| > 0$ **do**:
      Interact with environment and calculate the cost using (4.9) or (4.10),
      Save the experience in replay memory $\mathcal{M}_{i,t}$.
      Train the local model on $\mathcal{M}_{i,t}$,
      Transmit θ_i^{online} to the central aggregation unit,
    End For
  End For
  update θ^{global} using (4.14).

4.6 Simulation Results and Discussions

Here, I present my simulation results and extract useful insights related to the performance of my proposed federated DDQN framework in comparison to federated DDQN and distributed DDQN algorithms. In addition, I investigate the impact of batch size, network layers, target network update frequency on the convergence of

the FDL. In what follows, I first focus on investigating the impact of parameters of DRL on the learning speed of my proposed algorithm and then the comparison of the proposed algorithm with benchmarks would be presented.

To simulate my system, I consider a network of 100 IoT devices among which only 20 devices are selected in each round to contribute in the FDL process. Without loss of generality and for the sake of fair comparison, I assume the maximum computation capacity and energy consumption limit of the IoT devices are 1 Gbps and 23 dBm, respectively.

Fig. 4.2 demonstrates the effect of network architecture on the convergence of my proposed FedRL algorithm. Here I have some shallow networks with up to five layers and deeper networks that are obtained by stacking multiple layers with [16,32,32] neurons on each other. I note that by increasing the number of layers, faster model training can be achieved. The reason behind this observation is that, by exploiting deeper neural networks, we can better find the patterns in data (here devices' experiences), which subsequently improves the quality of the local models. Thus, the global model is trained much faster as its underlying components, local models, are more accurate.

However, since my algorithm will be executed on IoT devices that often lack necessary resources to train a deep network, it may be infeasible to implement deeper neural networks. Therefore, in the next figure, I select a rather simple network architecture with [30,64,16,32,32] neurons in each layer and instead look for other parameters that may facilitate the learning process.

The other parameter I focus on in Fig. 4.3, is the batch size. We can observe from this figure that as the batch size increases the convergence of the proposed FedRL

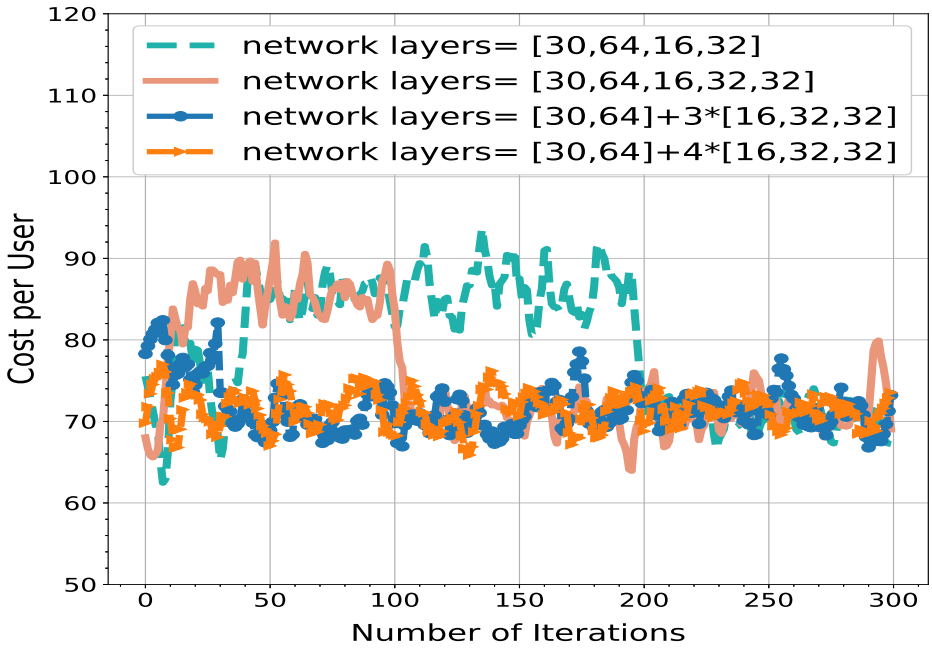

Figure 4.2: The impact of neural network architecture on the convergence of the proposed FedRL algorithm.

algorithm becomes faster. When batch size is equal to 10 and is extremely small it takes up to 200 iterations to finally converge to a relative global model, whereas in case batch size is 30, convergence is achieved almost around iteration number 40. By increasing the size of batches, we are basically training our model using more data instances. which results in enhancing the quality of local models and faster training process.

Similar to network architecture and the stated concern regarding the limited computation capacity, memory is another bottleneck in learning process of IoT devices. Larger batch size means higher memory consumption. *If the device is limited both in CPU and memory capacity, neither very deep neural networks nor increased batch size can be a proper solution to facilitate deployment of FedRL on IoT devices.*

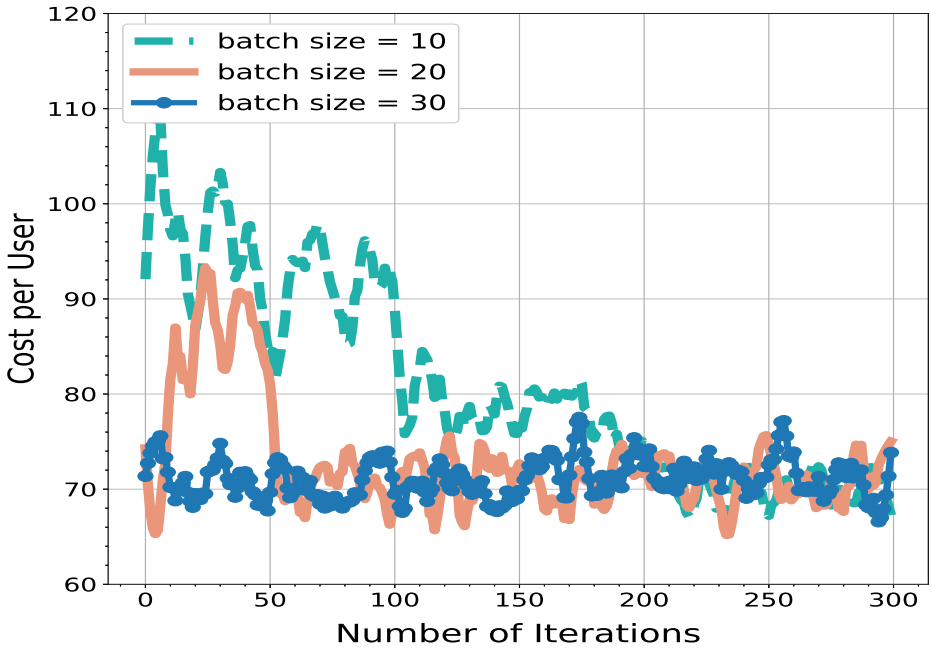

Figure 4.3: The impact of batch size on the convergence of the proposed FedRL algorithm.

To this end, in Fig. 4.4, I illustrate the effect of one of the parameters of DDQN, namely frequency of updating target network with online network. We can observe here that while the effect of this parameter on performance of DDQN algorithm is well investigated, this parameter is also considerably effective in the performance of federated DDQNs. Since many of the components in my state space, such as path-gain, QoS of tasks, and the length of the task queue, are constantly changing, efficient choice of the frequency of updating target network can stabilize the environment enough for the agent to track it better and obtain a better solution. This effect on local models is quite notable on the FDL as well.

In Fig. 4.5, I compare the performance of my proposed federated DDQN approach with those of federated DQN and simple distributed DDQN without any aggregation.

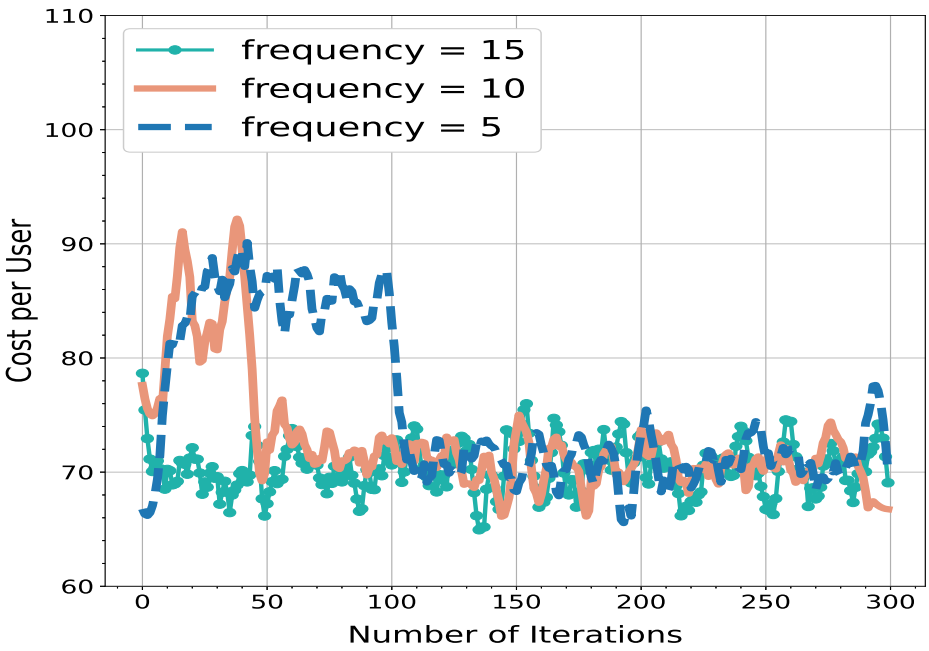

Figure 4.4: The impact of target network update frequency on the convergence of the proposed FedRL algorithm.

As can be seen, the performance of federated DDQN is superior to federated DQN in terms of learning speed. As previously explained, the main advantage of DDQN over DQN is the capability to keep target network stationary, helping with the tractability of states' values and subsequently a faster convergence to the correct estimation of them. The impact of this approach is even more notable when DRL is combined with FDL, since if the local models are not correctly trained, their errors would be propagated to other devices' local model through aggregation. Therefore, aggregation can in fact negatively effect the result.

The comparison between distributed DDQN and federated DDQN underlines that the benefits of federated DRL are not limited to its scalability and privacy preservation. The aggregation incorporated in FDL provides IoT devices a great context

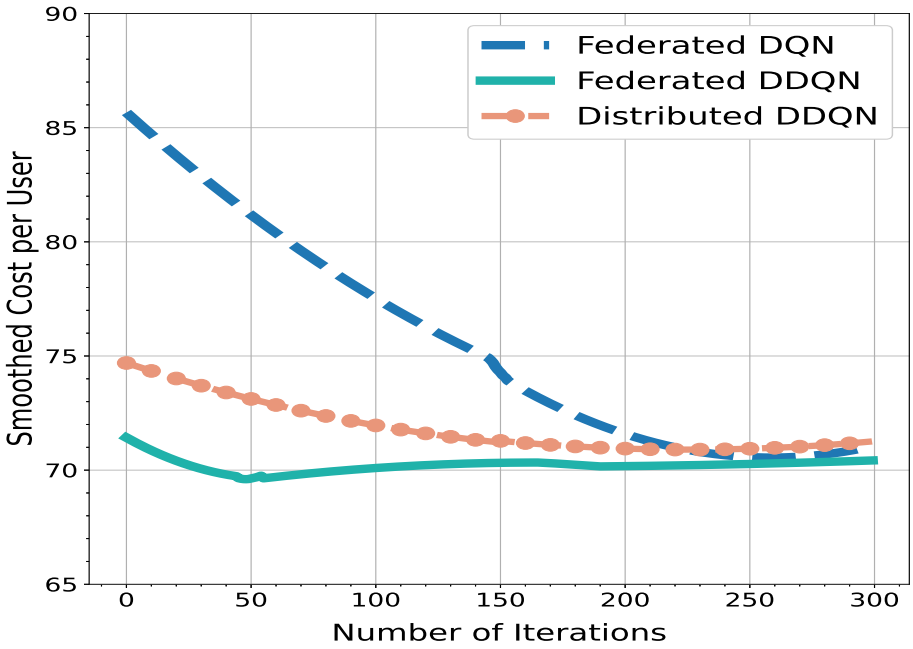

Figure 4.5: Performance of federated DDQN compared to federated DQN and distributed non-federated DDQN.

to cooperatively train their models and merge their intelligence together while preserving privacy of their information. Exploiting federated learning, at every training round is almost as if devices' models are trained with I times more data than their local information. The significance of this share of knowledge and not data is quite notable in Fig. 4.5, where as the result of this aggregation step, federated DDQN is working much better and faster than simple distributed DDQN.

4.7 Summary

In this section, I investigated the problem of joint delay and energy minimization in an IoT network with a three-tier offloading scheme. To solve this problem I combined FDL, DDQN, and optimization theory. Combination of these tools helped us to achieve a scalable, privacy-preserving, and computationally efficient framework for joint power and computation resource allocation and offloading decision optimization. In simulation results, I compared my work with those of 1) federated DQN to demonstrate the superiority of DDQN, especially in dynamic environments and 2) with distributed DDQN to signify the impact of aggregation step incorporated in FDL on the performance of the framework. As the proposed algorithm is computationally light-weight and energy efficient, it can readily be deployed for offloading decision making and resource allocation in real-world IoT networks.

Chapter 5

Conclusions and Future Directions

In this chapter, first a conclusion of this thesis is provided. Afterwards, I will present a brief overview of possible future research directions.

5.1 Conclusion

In this thesis, I investigated the problem of offloading decision making and resource allocation in two different network models.

In the first problem, I jointly optimized partial offloading decision, transmit power, subchannel, and computation resource allocation, in a two-tier virtually sliced MEC network. I minimized the weighted sum of the gap between the observed delay at each slice and its corresponding delay requirement, where weights set the priority of each slice. I observed that fractional form of the objective function, discrete subchannel allocation, considered partial offloading, and the interference incorporated in the rate function, make the considered problem a complex MINLP. Thus, I decomposed the original problem into two sub-problems: (i) offloading decision-making and (ii) joint computation resource, subchannel, and power allocation. I solved the first sub-problem optimally and for the second sub-problem, leveraging on novel tools from

fractional programming and Augmented Lagrangian method, I proposed an efficient algorithm whose computational complexity is proved to be polynomial. Using alternating optimization, I solved these two sub-problems iteratively until convergence is obtained. Simulation results demonstrated the convergence of my proposed algorithm numerically and its effectiveness compared to existing schemes.

In the second problem, I proposed a federated deep reinforcement learning framework to solve a multi-objective optimization problem, where I considered minimizing the expected long-term task completion delay and energy consumption of IoT devices. This was done by optimizing offloading decisions, computation resource allocation, and transmit power allocation. The formulated problem was a MINLP, which I first cast it as a multi-agent distributed DRL problem and addressed it using DDQN, where the actions were offloading decisions. The immediate cost of each agent was calculated through solving either the transmit power optimization or local computation resource optimization, based on the selected offloading decisions (actions). Then, to enhance the learning speed of IoT devices (agents), I incorporated FDL at the end of each episode. I observed that FDL enhances the scalability of the proposed DRL framework, creates a context for cooperation between agents, and minimizes their privacy concerns. Numerical results demonstrated the efficacy of my proposed federated DDQN framework in terms of learning speed compared to federated DQN and non-federated DDQN algorithms. Also, I investigated the impact of batch size, network layers, DDQN target network update frequency on the learning speed of the FDL.

5.2 Potential Future Directions

Some of the possible future directions are listed in what follows.

5.2.1 Wireless Connectivity between MEC Servers

As given in Section 3. my system model is based on the presence of wired links between MEC servers. Such mesh topology can be quite expensive for infrastructure providers to deploy. By lifting this assumption and allowing BSs to communicate with each other over wireless channels new opportunities would emerge.

On one hand, using wireless links, infrastructure providers can considerably cut their expenses, and on the other hand, a new set of challenges would be introduced to any resource allocation problem in such networks. To clarify, using wireless channels would necessitate precise control of transmit power between BSs, managing their interference level, and channel allocation.

This is particularly important considering BSs are far from each other and they are the intended destination of so many signals, mostly sent by users under their coverage area. As such, two major challenges need to be dealt with. First, BSs have to transmit with high transmit power to overcome the long distance between them. This high transmit power would intensify the interference level on almost all neighboring cells. Second, regarding the increased interference in the network, devices may be forced to increase their own transmit power which would shorten their already limited battery life. Such trade-offs need a thorough investigation.

5.2.2 Delay of Cooperation among MEC Servers

In the system model outlined in Section 3.2, I assumed that the delay of sending a task from a server with insufficient resources to another one with enough available computation capacity, is constant. However, there are many factors that can effect this delay and need to be considered if a precise and more accurate estimation of

delay is required. One of such factors is the size of the task. It is evident that larger tasks require more time to be transferred, therefore, this factor can have a significant impact on the total service delay of users. This is specifically true, if BSs communicate over wireless channels. In such system models, transmit power of BSs and their channel allocation should be controlled carefully while considering the size of the task they want to transmit.

In light of such delay-prone environment, it is highly possible that the optimal offloading decision would tend towards local computation as much as possible, and enable offloading only when 1) the offloading part of the task is small enough that can be processed by the users' allocated BS (avoiding hand-off) or 2) when the priority of the task or the slice it belongs to is high enough that guarantees sufficient available resources in both RAN and server for immediate transfer and processing.

5.2.3 Federated Actor-Critic Method

One of the methods to improve the resource allocation algorithm proposed in Section 4 by considering the learning for computation resource allocation in servers and use actor-critic networks to address this problem. Actor-critic method solves reinforcement learning problems by updating a parameterized policy, which is commonly known as an actor in a direction that maximizes an estimate of the expected reward known as a critic. This method works well with mixed continuous and discrete action spaces. Assuming that computation capacity of servers are accurately modeled as continuous variables and offloading decisions are binary, we can effectively use actor-critic networks to address the aforementioned problem.

5.2.4 Federated DDQN in Sliced Networks

In the first problem investigated in this thesis (introduced in Section 3), the benefits of sliced virtual networks and their impact on reducing expenses and providing resource isolation between service providers was extensively explained. Later on, in Section 4, I talked about Federated DDQN, and how it helps us to obtain a scalable, privacy-preserving, and cooperative resource allocation framework that fits well with the needs of dynamic IoT networks.

In fact, these two sections effectively point toward the fact that federated DDQN is a very efficient approach to address the resource allocation problems in sliced networks. In such networks, service providers can act as agents, deciding on how much of their available resources should be allocated to their subscribers as to maximize their cumulative reward. Such problem formulation can result in a cooperative solution for the precise control of resource utilization in slices and allows us to consider the fluctuation of networks' data load over periods of time.

The merits of such method would become even more clear if we consider the fact that service providers are businesses and as such would most probably be unwilling to share their corporal data with any external entity. subsequently, federated learning that refrains from asking agents to share their data with any other entity in the network, would turn into a very attractive problem-solving tool in such networks.

Bibliography

[1] N. Hassan, M. T. Hossan, and H. Tabassum, "User association in coexisting RF and TeraHertz networks in 6G," *IEEE Canadian Conference of Electrical and Computer Engineering*, 2020.

[2] E. Hossain, M. Rasti, H. Tabassum, and A. Abdelnasser, "Evolution toward 5G multi-tier cellular wireless networks: An interference management perspective," *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 118–127, 2014.

[3] G. Wikström, J. Peisa, P. Rugeland, N. Johansson, S. Parkvall, M. Girnyk, G. Mildh, and I. L. Da Silva, "Challenges and technologies for 6G," in *2nd 6G Wireless Summit (6G Summit)*, 2020, pp. 1–5.

[4] S. Zarandi, A. Khalili, M. Rasti, and H. Tabassum, "Multi-objective energy efficient resource allocation and user association for in-band full duplex small-cells," *IEEE Trans. on Green Commun. and Networking*, 2020.

[5] H. Ibrahim, H. Tabassum, and U. T. Nguyen, "Meta distribution of SIR in dual-hop internet-of-things (IoT) networks," in *IEEE Intl. Conference on Commun. (ICC)*, 2019, pp. 1–7.

[6] J. Sayehvand and H. Tabassum, "Interference and coverage analysis in coexisting RF and dense terahertz wireless networks," *IEEE Wireless Commun. Letters*, 2020.

[7] S. D. A. Shah, M. A. Gregory, and S. Li, "Cloud-native network slicing using software defined networking based multi-access edge computing: A survey," *IEEE Access*, vol. 9, pp. 10 903–10 924, 2021.

[8] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[9] W. Saad, M. Bennis, and M. Chen, "A vision of 6G wireless systems: Applications, trends, technologies, and open research problems," *IEEE Network*, vol. 34, no. 3, pp. 134–142, 2020.

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[11] L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, H. D. Schotten, and X. Costa-Pérez, "Laco: A latency-driven network slicing orchestration in beyond-5G networks," *IEEE Trans. on Wireless Commun.*, vol. 20, no. 1, pp. 667–682, 2021.

[12] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: Architectures, challenges, and approaches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2495–2508, 2018.

[13] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez, "Resource sharing efficiency in network slicing," *IEEE Trans. on Network and Service Management*, vol. 16, no. 3, pp. 909–923, 2019.

[14] P. Zhao, H. Tian, S. Fan, and A. Paulraj, "Information prediction and dynamic programming-based RAN slicing for mobile edge computing," *IEEE Wireless Commun. Letters*, vol. 7, no. 4, pp. 614–617, 2018.

[15] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[16] X. Fu, Q. Shen, W. Wang, H. Hou, and X. Gao, "Slice merging/spliting operations and tenant profit optimization across 5G base stations," *IEEE Access*, vol. 9, pp. 9706–9718, 2021.

[17] A. M. Escolar, J. M. Alcaraz-Calero, P. Salva-Garcia, J. B. Bernabe, and Q. Wang, "Adaptive network slicing in multi-tenant 5G IoT networks," *IEEE Access*, vol. 9, pp. 14 048–14 069, 2021.

[18] K. I. Ahmed, H. Tabassum, and E. Hossain, "Deep learning for radio resource allocation in multi-cell networks," *IEEE Network*, vol. 33, no. 6, pp. 188–195, 2019.

[19] O. A. Wahab, A. Mourad, H. Otrok, and T. Taleb, "Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems," *IEEE Commun. Surveys Tutorials*, pp. 1–1, 2021.

[20] F. Sattler, S. Wiedemann, K. R. Müller, and W. Samek, "Robust and communication-efficient federated learning from Non-i.i.d. data," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2020.

[21] S. Zhai, X. Jin, L. Wei, H. Luo, and M. Cao, "Dynamic federated learning for gmec with time-varying wireless link," *IEEE Access*, vol. 9, pp. 10 400–10 412, 2021.

[22] Y. Ye, S. Li, F. Liu, Y. Tang, and W. Hu, "Edgefed: Optimized federated learning based on edge computing," *IEEE Access*, vol. 8, pp. 209 191–209 198, 2020.

[23] S. Zarandi and H. Tabassum, "Delay minimization in sliced multi-cell mobile edge computing (MEC) systems," *IEEE Commun. Letters*, pp. 1–1, 2021.

[24] ——, "Federated double deep Q-learning for joint delay and energy minimization in IoT networks," *IEEE Intl. Conference on Commun. Workshop (ICC'21)*, 2021.

[25] K. Shen and W. Yu, "Fractional programming for communication systems—part i: Power control and beamforming," *IEEE Trans. on Signal Processing*, vol. 66, no. 10, pp. 2616–2630, 2018.

[26] W. Dinkelbach, "On nonlinear fractional programming," *Manage. Sci.*, vol. 133, no. 7, p. 492–498, 1967.

[27] Y. Huang, T. Tan, N. Wang, Y. Chen, and Y. Li, "Resource allocation for d2d commun. with a novel distributed q-learning algorithm in heterogeneous networks," in *2018 Intl. Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 2, 2018, pp. 533–537.

[28] P. Zhao, H. Tian, S. Fan, and A. Paulraj, "Information prediction and dynamic programming-based RAN slicing for mobile edge computing," *IEEE Wireless Commun. Letters*, vol. 7, no. 4, pp. 614–617, 2018.

[29] E. El Haber, T. M. Nguyen, and C. Assi, "Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds," *IEEE Trans. on Commun.*, vol. 67, no. 5, pp. 3407–3421, 2019.

[30] Y. Wang, X. Tao, X. Zhang, P. Zhang, and Y. T. Hou, "Cooperative task offloading in three-tier mobile computing networks: An ADMM framework," *IEEE Trans. on Vehicular Technology*, vol. 68, no. 3, pp. 2763–2776, 2019.

[31] J. Zhang, W. Xia, F. Yan, and L. Shen, "Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing," *IEEE Access*, vol. 6, pp. 19 324–19 337, 2018.

[32] B. Xiang, J. Elias, F. Martignon, and E. Di Nitto, "Joint network slicing and mobile edge computing in 5G networks," in *IEEE Intl. Conference on Commun. (ICC)*, 2019, pp. 1–7.

[33] Y. Xiao and M. Krunz, "Dynamic network slicing for scalable fog computing systems with energy harvesting," *IEEE Journal on Selected Areas in Commun.*, vol. 36, no. 12, pp. 2640–2654, 2018.

[34] H. Chien, Y. Lin, C. Lai, and C. Wang, "End-to-end slicing with optimized communication and computing resource allocation in multi-tenant 5G systems," *IEEE Trans. on Vehicular Tech.*, vol. 69, no. 2, pp. 2079–2091, 2020.

[35] U. Akgül, I. Malanchini, and A. Capone, "Dynamic resource trading in sliced mobile networks," *IEEE Trans. on Network and Service Management*, vol. 16, no. 1, pp. 220–233, 2019.

[36] J. Feng, Q. Pei, F. R. Yu, X. Chu, J. Du, and L. Zhu, "Dynamic network slicing and resource allocation in mobile edge computing systems," *IEEE Trans. on Vehicular Tech.*, vol. 69, no. 7, pp. 7863–7878, 2020.

[37] K. Shen and W. Yu, "Fractional programming for communication systems—part i: Power control and beamforming," *IEEE Trans. on Signal Processing*, vol. 66, no. 10, pp. 2616–2630, 2018.

[38] Z. Wang, L. Vandendorpe, M. Ashraf, Y. Mou, and N. Janatian, "Minimization of sum inverse energy efficiency for multiple base station systems," in *2020 IEEE Wireless Commun. and Networking Conference (WCNC)*, 2020, pp. 1–7.

[39] A. Khalili, S. Akhlaghi, H. Tabassum, and D. W. K. Ng, "Joint user association and resource allocation in the uplink of heterogeneous networks," *IEEE Wireless Commun. Letters*, vol. 9, no. 6, pp. 804–808, 2020.

[40] A. Khalili, S. Zarandi, and M. Rasti, "Joint resource allocation and offloading decision in mobile edge computing," *IEEE Commun. Letters*, vol. 23, no. 4, pp. 684–687, 2019.

[41] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[42] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 234–243.

[43] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9441–9455, 2020.

[44] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported internet of things," *IEEE Access*, vol. 7, pp. 69 194–69 201, 2019.

[45] D. R. H. B. McMahan, E. Moore and B. A. Y. Arcas, "Federated learning of deep networks using model averaging," *arXiv:1602.05629, 2016.*, 2016.

[46] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, 2020.