University of Windsor
Scholarship at UWindsor

Electronic Theses and Dissertations | Theses, Dissertations, and Major Papers

8-25-2022

Secure and Energy-efficient Computation Offloading in Mobile Edge Computing

Xue Qin
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation

Qin, Xue, "Secure and Energy-efficient Computation Offloading in Mobile Edge Computing" (2022). *Electronic Theses and Dissertations*. 9597.
https://scholar.uwindsor.ca/etd/9597

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Secure and Energy-efficient Computation Offloading in Mobile Edge Computing

By

Xue Qin

A Thesis

Submitted to the Faculty of Graduate Studies

through the Department of Electrical and Computer Engineering

in Partial Fulfillment of the Requirements for

the Degree of Master of Applied Science

at the University of Windsor

Windsor, Ontario, Canada

2022

© 2022 Xue Qin

Secure and Energy-efficient Computation Offloading in Mobile Edge Computing

by

Xue Qin

APPROVED BY:

L. Rueda

School of Computer Science

M. Mirhassani

Department of Electrical and Computer Engineering

H. Wu, Advisor

Department of Electrical and Computer Engineering

July 11, 2022

Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Driven by the popularity of the Internet of Things, the emergence of a multitude of applications highly require computing and storage capacity. However, the Internet of Things users are generally constrained with resources. Mobile edge computing is proposed whereby data generated by Internet of Things devices could be offloaded to nearby edge servers instead of remote cloud servers, thus their requests can be served locally and quickly.

One of the most important research issues in mobile edge computing is computation offloading. Due to the dynamic and time-varying operating environments, there is an increasing interest in applying deep reinforcement learning to decision-making in computation offloading. It integrates neural network into reinforcement learning without labeled data and formulates computation offloading problem as a decision-making process. Moreover, offloading tasks to edge servers would raise serious security concerns. Physical-layer security based on information theoretic methods could be leveraged to safeguard the data transmission. As an effective way in physical layer security, artificial jammer could be employed to greatly degrade the reception performance of eavesdroppers while keeping the legitimate users unaffected by means of beamforming techniques.

In this work, a deep reinforcement learning model is proposed for computation offloading in dynamic mobile edge computing system. Our objectives include maximizing number of completed tasks before tolerant time and minimizing energy consumption, subject to security rate requirement. The proposed solution can learn to optimize edge server selection for offloading a certain task, CPU allocation at selected edge servers for executing given task, and friendly jammer selection. Simulations show that the developed model outperforms the existing methods, including deep reinforcement learning models combined with optimization methods, traditional reinforcement learning algorithm, and greedy algorithm.

Acknowledgements

I would like to express my deep gratitude and thankfulness to Dr. Huapeng Wu. His patience, vision, passion, guidance, aspiration and motivation immense knowledge throughout my graduate study and chiseled me from a curious student to a competent researcher. He gave me enough freedom in terms of approaching the research problem we have been working on and encouraged me to take courses relevant to my research. His supervision has helped me not only during master program studying period but will inspire me in future as well.

I would like to thank my committee members, Dr. Rueda and Dr. Mirhassani. I am very grateful for all the helpful comments and suggestions that they provided in my presentation.

Last but not least, I will thank my mother for all the struggles and hard work she put into raising me. Thanks for all her sacrifices to educate me. I also want to thank my husband for being supportive to our family constantly and encouraging me during the time of completing my further study continuously.

Table of Contents

List of Tables

List of Figures

List of Abbreviation

AI Artificial intelligence. 12

ANN Artificial Neural Networks. 13

CNN Convolution Neural Networks. 13

DL Deep Learning. 7

DQL Deep Q-learning. 16

DQN Deep-Q network. 16

DRL Deep Reinforcement Learning. 7

EH Energy Harvest. 10

IoT Internet of Things. 1

ITS Intelligent Transportation System. 1

LODCO Lyapunov Optimization-based Dynamic Computation Offloading. 10

LSTM Long Short-term Memory. 14

MCC Mobile Cloud Computing. 3

MDP Markov Decision Process. 7, 28

MEC Mobile Edge Computing. 4

MEC ENV MEC Network Environment. 11

ML Machine Learning. 12

MSE Mean Square Error. 35

OFDMA Orthogonal Frequency-division Multiple Access. 11

PLS Physical Layer Security. 7

QoS Quality of Services. 3

RL Reinforcement Learning. 7

RNN Recurrent Neural Networks. 13

SGD Stochastic Gradient Descent. 41

SVM Support Vector Machine. 12

TDMA Time-division Multiple Access. 10

Chapter 1

Introduction

1.1 Internet of Things

Internet of Things (IoT) is a new paragon that is defined as a network of physical "things" which are embedded with some technologies such as software, sensor, processing ability for exchanging data and connecting with other systems or devices over Internet [1–3]. With IoT, data can be collected and processed efficiently to support decision making for different applications. Over the past few years, IoT has become one of the most emerging markets globally. Driven by popularity of IoT, we have not only witnessed a rapid proliferation of wireless devices and exponential growth of wireless data traffic, but also the emergence of a multitude of useful applications.

With IoT technologies, many applications can be enabled, e.g., smart home, smart city, intelligent transportation system, smart grids and smart factory. In smart home [4], there is an internet-connected center with the responsibility to control various IoT devices, which can be under control of a mobile application. With various control technologies, smart home is becoming affordable and can be applied to control electrical and electronic home devices including fire alarm, lights timer, conditioner and so on. Another advanced IoT application is intelligent transportation system (ITS) [5, 6]. This technology is employed worldwide for improving road efficiency and driving safety. It ensures a safer, more efficient and coordinated utilization of transport networks in a variety of situations such as road

transport, mobility and traffic management. For example, emergency calling services aim to carry out the traffic signs or laws for indicating speed limit changes, as appropriate. By employing IoT and other technologies, smart grid can better monitor the status of the grid and provide an integral solution to ensure security, efficiency and reliability of electric grids.

According to the statistical data from statista (shown in Fig. 1.1), the number of connected IoT devices worldwide continues to grow, and is predicted to exceed 29 billion by 2030, more than three times of the 8.6 billion in 2019. Moreover, based on the statistics from DataProt, the consumer IoT market is expected to get to $142 billion by 2026 with a CAGR of 17%. 94 percents of retailers approve IoT implement benefits far outweigh any risks. By 2025, IoT devices are estimated to produce a volume of data as much as 73.1 ZB.

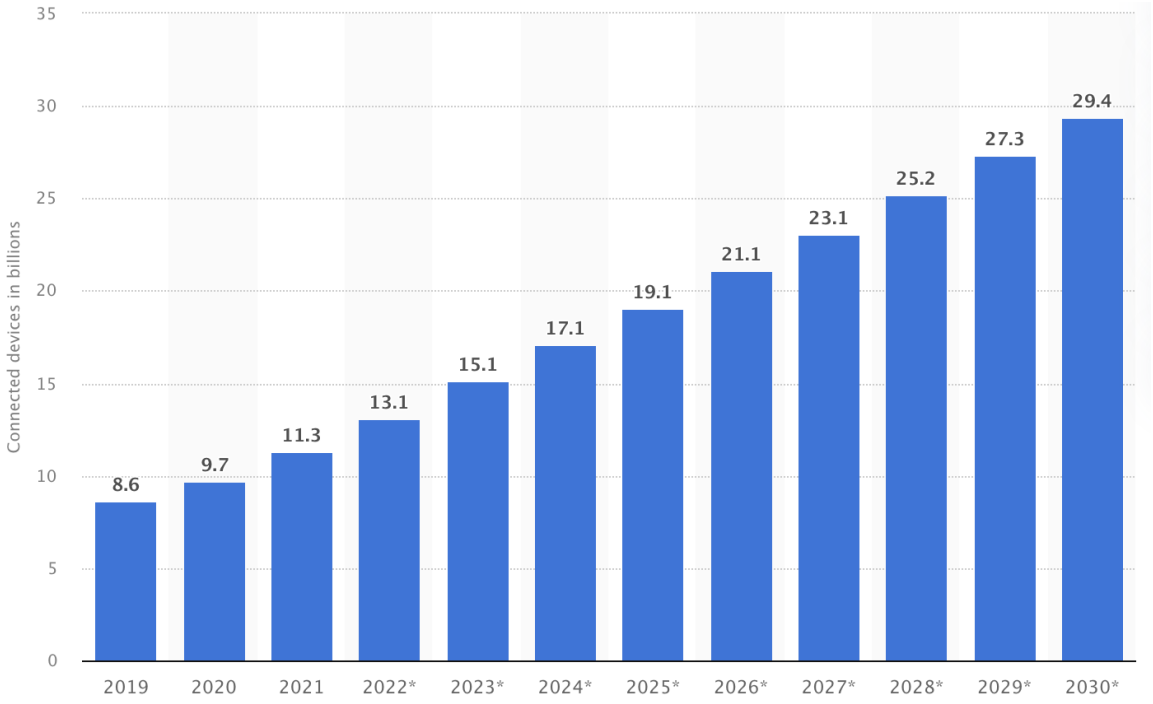

FIGURE 1.1: Number of IoT connected devices worldwide 2019-2030

IoT can provide many advantages, including reducing the need for human intervention and efforts, improving data collection, promoting automation, making efficient use of resources, and so on. However, IoT still faces many challenges such as privacy and security concerns, high internet dependency, and limited devices' capacity (storage, computational power, battery).

1.2 Mobile Cloud Computing

In order to support different quality of services (QoS) requirements of IoT applications, the data collection and procession should be conducted efficiently. The computing and storage capacity are highly demanded by most IoT applications. However, the end devices with limited physical size are generally constrained in resources, such as energy, computational power and storage.

To process the data collected by the IoT devices and meet the computing and storage requirements, mobile cloud computing (MCC) residing in core networks can be leveraged to better accommodate the emerging services [7]. From the MCC architecture provided by DataFlair (Fig. 1.2), we can see MCC rents access to a virtualized computer in a remote data center and uses cloud computing to deliver applications to mobile devices. Mobile cloud applications are built or revised using cloud services. The computation and resources are stored in the remote cloud where data processing and analysis are conducted for decision making, so devices are directly connected to the cloud which is far away from data sources.

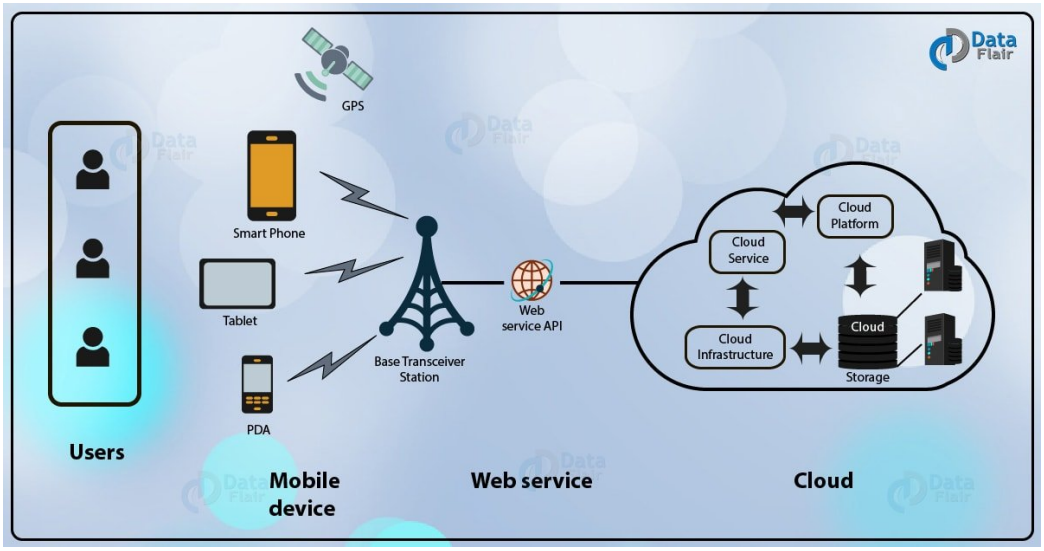

FIGURE 1.2: mobile cloud computing architecture

Although MCC has a lot of benefits such as data backup, cost efficiency, disaster recovery, integrated data and multiple platforms, there are still some shortages in MCC:

Firstly, because of long distance, moving a large volume of data into and out of the core network wirelessly requires substantial spectrum resources, and can incur long service

latency, which might cause service failure to support real-time applications, such as autonomous driving. High latency has problems such as slow performance, high error rates, increased traffic, and more insecurity.

Secondly, moving massive data from IoT devices to the remote servers can cause network congestion. When network congestion occurs, the QoS is deteroriated. There are some typical effects such as packet loss, queuing delay, and new connections blocking.

1.3 Mobile Edge Computing

To address the above MCC issues, mobile edge computing (MEC) emerges, which is a new paradigm where computing resources are deployed at network edge in close proximity of end devices [8–10]. The architecture of MEC is shown in Fig. 1.3. By leveraging MEC, computation-intensive tasks can be offloaded to edge servers for execution and the results can be feedback to the end devices. Rather than the remote cloud through backhaul, mobile edge could serve IoT users' demands locally and quickly. The service latency for IoT users and pressure on core network could be reduced and mitigated significantly [11–13].

MEC enables most of the market drivers such as business transformation, technology integration, and industry collaboration. Moreover, it supports new innovative markets with a wide variety of use cases such as e-Health, connected vehicles, industry automation, IoT services. MEC performs computation-rich tasks from resource-constrained IoT devices, has Lower-Latency computing, and could minimize backhaul congestion. Also, its location awareness could make the services that are provided by applications much better suited to device and user location. MEC is expected to improve response times and user experience.

Along with those benefits, MEC is still facing many challenges. Firstly, MEC usually has less capacity than MCC. The coordination of multiple edge servers is required for serving IoT devices' tasks efficiently. Secondly, there are stringent latency requirements in some IoT services, which depend on transmission and computation power allocation in offloading. Also, with various service requirements, IoT users' tasks are generated dynamically. Moreover, the MEC operating environment changes over time, such as the varying channel, edge workload, CPU allocation conditions. Therefore, the decision making in MEC system

has to be adjusted accordingly. Finally, IoT devices are energy-constrained, and thus the energy consumption should also be considered.

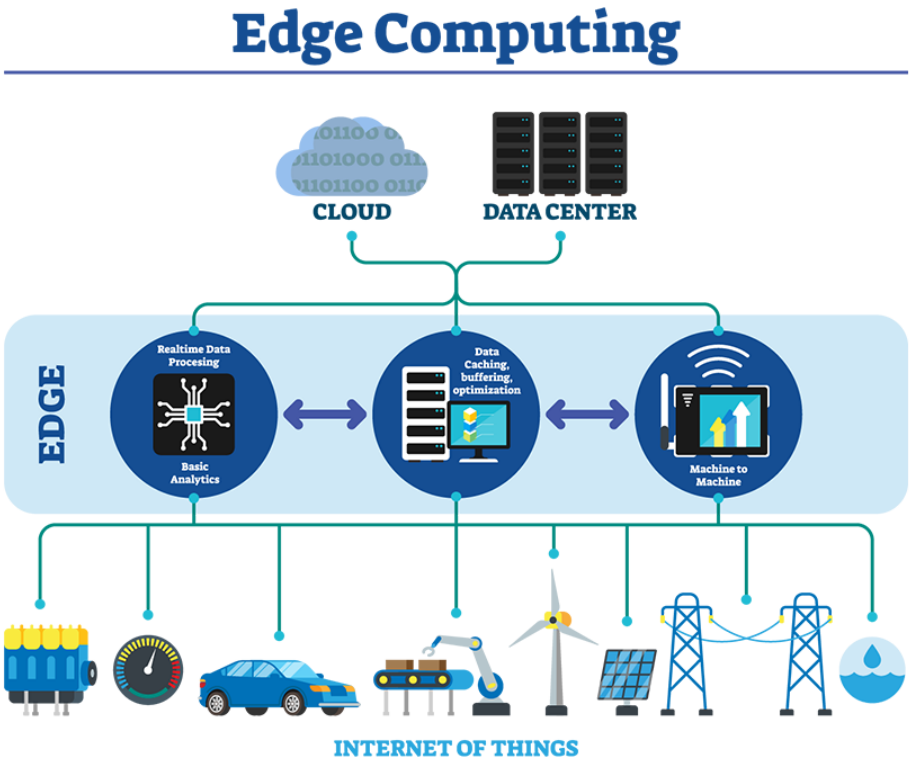

FIGURE 1.3: mobile edge computing architecture

1.4 Computation Offloading in Mobile Edge Computing

Limited computing resources and battery of IoT users may cause performance limitations in MEC networks. Yang et al. presented the task offloading scenario as shown in Fig. 1.4. In the computation offloading of MEC, resource-constrained IoT user's compute-intensive tasks are offloaded to edge servers with more computing resources for QoS enhancement. Compared with being transferred to cloud in MCC, tasks can be transferred to mobile edge server which are close to the end devices such that the latency can be reduced, and network congestion can be mitigated [14].

As service latency is critical to applications for making real-time decision or control, it is important to achieve low-latency computation offloading in MEC. The service latency mainly consists of both the communication and computation latency, where the communi-

cation latency is the latency of transmitting data required by computation tasks to edge servers, and the computation latency is the time spent in processing the computation tasks in edge servers [15]. Another, energy consumption consists of both communication and computation energy consumption. Therefore, communication and computing resource management need to be jointly allocated. Compared to MCC, the advantage of proximity to end-users in MEC greatly reduces the transmission delay and energy consumption of offloading computing tasks to edge servers.

Nonetheless, the computation offloading mechanisms in MEC are still facing several challenges. Firstly, edge servers have time-varying network conditions and limited computing resources, so the IoT users make offloading decision may not achieve lowest cost. Secondly, computation offloading in MEC can also bring security challenges. Some privacy-sensitive IoT users may be deterred away from using the MEC without proper protection mechanisms.

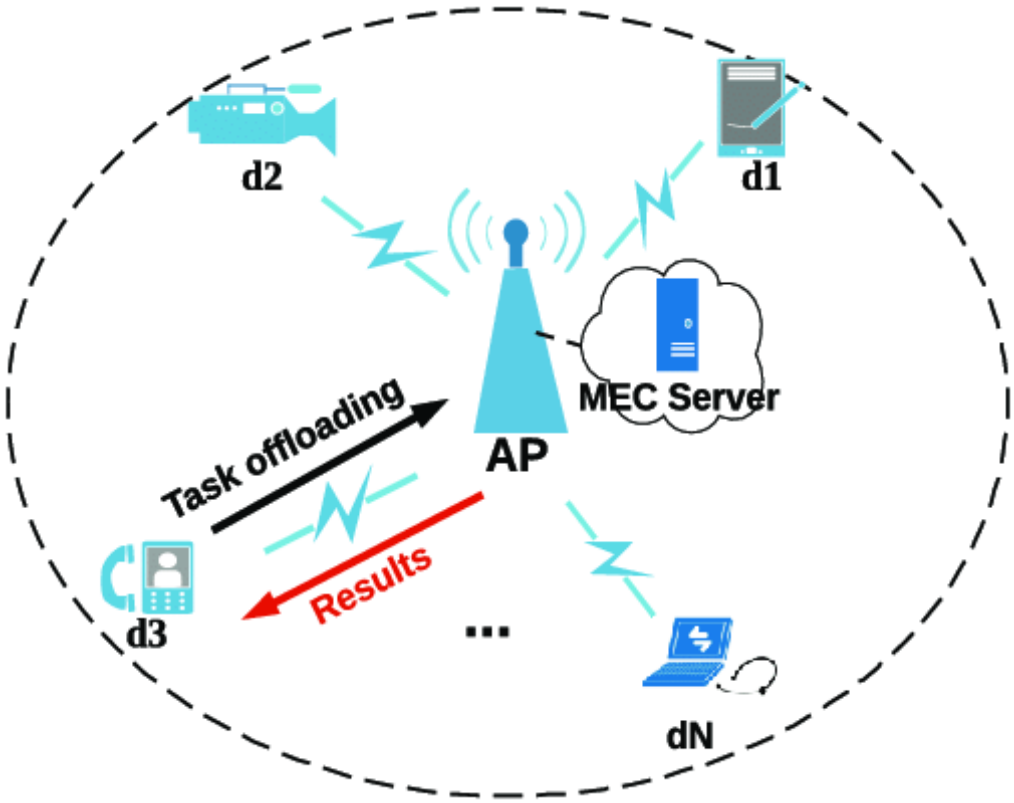

FIGURE 1.4: Computation Offloading in MEC

1.5 Motivation and Objective

As mentioned above, the computation offloading in MEC has two major challenges: security and energy efficiency. Although there are lots of existing works done to minimize energy cost and maximize the number of offloading tasks in offloading process in edge computing, there are many limitations with the current approaches such as traditional approach, deep learning (DL) or reinforcement learning (RL) methods. The detailed drawback of these approaches is discussed in the literature review section later. Moreover, they seldom focus on how to offload tasks to edge server in a secure and timely manner. Due to the broadcast nature of wireless communications, there are security concerns during computation tasks being offloaded from IoT users to edge servers. Without secure offloading, IoT users' private data such as health information and identity information could be overheard by the nearby eavesdroppers.

Therefore, this thesis aims to design a novel method to enhance secure transmission which is on physical layer assists other secure methods. A novel solution using deep reinforcement learning (DRL) approach is proposed to meet both security and energy efficiency requirements in mobile edge computation offloading with multiple users, multiple MEC servers, and an eavesdropper.

Specifically, physical layer security (PLS) [16, 17] built on information theoretic methods, is be leveraged to safeguard the data transmission, where relays and artificial jammer will be employed to greatly degrade the reception performance of eavesdroppers while keeping the legitimate users unaffected. In addition, the offloading decisions in terms of which edge servers to offload the task and how to allocate computing resources and communication resources (e.g., transmission power) will be determined to meet the latency requirement as well. Due to heterogeneity, high dynamics and multi-dimensional randomness in the system, e.g., time-changing wireless channel conditions, varying workloads of edge servers, it is of great challenge to find optimal or near optimal solution to this multi-objective optimization problem.

The problem is formulated to a Markov Decision Process (MDP) with a continuous state and large action spaces. The reward function is a weighted summation of multiple objectives related to security issues, energy consumption, and the completed number of tasks before

deadlines. Artificial intelligence, especially an end-end DRL is adopted to deal with such a complex problem based on dynamic network environment and current tasks information. DRL is able to achieve all optimization targets in one step and the proposed learning scheme could jointly optimizes long-time utility includes both maximizing the number of completed tasks before their deadlines, reducing energy consumption and satisfying security requirement.

1.6 Contributions and Organization of Thesis

In a nutshell, the main contributions of this work can be summarized as follows:

- A MDP with a continuous state and large action spaces is formulated. The reward function can be adjusted according to network providers' and users' demands regarding security concerns, energy consumption, and the completed number of tasks.
- An End-to-End DRL model is proposed to optimize multiple objectives, including maximizing the number of completed tasks before expiration and saving energy consumption constrained by security transmit rate.
- Extensive simulations show that the developed model outperforms the existing methods, including DRL models combined with optimization methods, traditional reinforcement learning algorithm, and the greedy algorithm.

The rest of the paper is organized as follows:

- Chapter 2 reviews the existing works.
- Chapter 3 presents our system model and problem formulation.
- Chapter 4 proposes deep reinforcement learning method and training processes.
- Chapter 5 provides the simulation and results analysis.
- Chapter 6 concludes this work.

Chapter 2

Literature Review

By offloading computations to edge servers with more resources in the proximity, IoT users are able to support computation-intensive and energy-hungry mobile applications in MEC systems. The energy-efficient MEC has attracted significant attention. In the past years, a number of computation offloading frameworks in MEC have been proposed with optimization approaches for IoT applications. Also, different optimization problems are formulated and solved based on optimization techniques. Nonetheless, there are several limitations with convectional approaches as these approaches for computation offloading are either classic approach or optimization based approach which do not take intelligent decisions and suffer from the fact that require lots of iterations. With the development of various AI techniques, researchers are motivated to turn to solve these above problems by using some standard supervised ML, DL or RL methods and they are successful in optimizing resource allocation. However, they are still facing challenges in the relatively dynamic and complex MEC network environment.

2.1 Overview of Classic Approaches

By offloading resource-constrained IoT users' compute-intensive tasks to edge servers nearby with potential computation capability, MEC is an emerging computing pattern to augment computational capabilities of IoT users. However, edge clouds still have limited computation

resources and energy in general. In order to ensure satisfactory IoT users experience, achieving energy efficiency is usually considered as the criterion for performance evaluation.

Energy-efficient computation offloading and wireless communication to be jointly designed. Recent years have seen a number of classic methods with optimization techniques developed for MEC resource allocation. The research progress focused on this topic is for both single-user and multi-user MEC systems. For single user scenario, Mao et al. [18] investigated MEC systems with Energy Harvest (EH) mobile devices and deployed a Lyapunov optimization-based dynamic computation offloading (LODCO) algorithm which is an online algorithm with low complexity and little prior knowledge. Some others focused on multi-user scenarios. In the research of Ren et al. [19], joint communication and computation resource allocation for a time-division multiple access (TDMA)-based multiple users MEC system was investigated for QoE improvement of IoT users by using a joint communication and computation resource allocation algorithm for minimizing all devices' weighted-sum delay. Munoz et al. [20] reduced the MDs' energy cost by jointly optimizing the time of transmission and the amount of offloading data to a femto AP. For multi-server MEC Networks, Tran et al. [21] addressed resource allocation problem and proposed a heuristic algorithm using convex and quasi-convex optimization techniques for optimal solution in polynomial time. A game theoretic method was adopted by Chen et al. [22] for energy and latency minimization at mobile devices. They formulated computation offloading decision making problem as a multiple users' game, and a distributed offloading algorithm was designed to achieve a Nash equilibrium. In the works of Sardellitti et al. [23], the radio and computation resources were jointly allocated for multi-cell MEC system to reduce the mobile energy consumption under offloading latency constraints. An iterative algorithmic framework was proposed based on successive convex approximation method to converge to an optimal local solution for original non-convex problem. Also, a parallel and distributed implementation was achieved with limited cloud coordination/signaling requirement.

In the work of Wang et al. [24], computation offloading problem was formulated as a cost minimization and energy efficiency problem with completion time consideration. They proposed a distributed algorithm includes allocating transmission power, configuring clock frequency, scheduling channel rate, and selecting offloading strategy to solve this optimization problem. A resource allocation policy based on TDMA and orthogonal frequency-division

multiple access (OFDMA) was proposed by You et al. [25] to minimize mobile energy consumption for computation offloading in MEC system. Mazouzi et al. [26] formulated the offloading cost minimization problem as a binary programming which is NP-hard, and adopted a distributed linear relaxation based heuristic approach using Lagrangian decomposition method to solve it. Moreover, for maximizing the sum of computation efficiency among users with weighting factors, an optimization problem was formulated by Sun et al. [27] and was solved efficiently with iterative and gradient descent methods. Guo et al. [28] proposed a novel distributed eDors algorithm includes computation offloading selection, clock frequency control and transmission power allocation sub-algorithms to reduce energy cost under completion time deadline and required task precedence.

For the tradeoff between execution latency and energy consumption, some research works have been reported in the literature. In the work of Dinh et al. [29], both tasks' execution latency and MD's energy consumption by jointly optimizing the task offloading decision and MD's CPU-cycle frequency have been minimized. Deng et al. [30] aimed to propose an adaptive sequential offloading game approach in multi-cell MEC system for offloading decision where the weighted sum of energy and computational time as the optimal objective function. In order to investigate execution latency and energy consumption minimization, Wang et al. [31] jointly optimized the computation speed and transmission power of mobile devices, respectively. Hong et al. [32] developed a weighting factor for defining weighted sum of energy cost and formulated a dynamic programming problem for data offloading scheduling. By jointly optimizing channel resource allocation, transmission power and CPU-cycle frequency, Zhang et al. [33] developed an energy-aware offloading scheme to exploit the tradeoff between execution latency and energy consumption, which focused on defining the weighting factor based on the residual energy of MD battery.

In practice, as MEC environment (MEC ENV) is extraordinarily dynamic and complex, it is hard to model it. Almost all the optimization problems mentioned above are mainly formulated based on network snapshots and have some shortcomings. Firstly, when the environmental conditions change over time, it has to be reformulated. Secondly, these classical optimization methods require many iterations, and it is hard to find optimal solution in a complex environment.

2.2 Overview of Deep Learning Approach

In order to address above issues, some machine learning (ML) based solutions were introduced. In the paper of Ismail et al. [34], with the Support Vector Machine (SVM) regression model, a machine learning-based offloading algorithm was proposed for energy optimization of the edge-cloud computing platform.

Although ML is helpful to solve offloading problems, there are more ongoing human intervention requirements. Also, structured data and conventional algorithms are required and used. Compared with it, DL methods [35, 36] with neural networks can be adopted, which can perform better than some ML methods in almost all the artificial intelligence (AI) fields, such as speech recognition, computer vision and natural language processing. Compared with conventional ML based offloading schemes, DL scheme carries extreme calculation speed for test and has accuracy capacity for making decision. In MEC system, DL was used for resource demands prediction and resource allocation optimization with requiring minimal intervention thereafter.

2.2.1 Neural Networks

AI aims to make computers able to think in a way mimic how humans learn new information. ML focuses on how to make computers learn without being explicitly programmed. As a subset of ML, DL creates more complex hierarchical models to mimic how humans do.

In the context of AI and ML, a model is a mathematical algorithm that is trained to have the same result or prediction that a human expert would when provided the same information. Many recent advances in AI were made possible by DL.

As a series of algorithms, a neural network aims to recognize underlying relationships in a set of data. The algorithms are inspired by the structure of the human brain and known as systems of neurons with multiple layers that do not require preprocessing the input data in order to produce a result. As a working system at the heart of a DL algorithm, a neural network helps to process raw data which is fed into the algorithm. System analyzes raw data based on what it already knows and what it can infer from the new data, and makes a prediction. There are three important types of neural networks with different

characteristics that form the basis for most pre-trained models in deep learning (Shown in Fig. 2.1): Artificial Neural Networks (ANN), Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN).

	ANN	CNN	RNN
Basics	Simple neural network	Popular neural network	Advanced neural network
Date Type	Tabular and text data	Image data	Sequence data
Structural Layout	Information flows in one direction only	Multiple convolutional layers	Information flows in different directions
Feature	Work with incomplete knowledge and high fault tolerance	Accuracy in recognizing images	Memory and self-learning
Complexity	Simple	More powerful	Fewer features, powerful
Recurrent Connection	No	No	Yes
Spatial Recognition	No	Yes	No
Uses	Solve complex problem	Computer vision	Nature language processing
Shortages	Hardware dependence	Large training data requirement	Slow and complex training and gradient issues

FIGURE 2.1: Characteristics Comparison of ANN, CNN and RNN

ANNs are based on artificial neurons which model neurons in a biological brain loosely and signal can be transmitted to other neurons by each connection. The knowledge in ANNs is gathered by detecting data relationships and learning through experience. Hundreds of single units, artificial neurons, connected with weights constitute ANN neural structure. Specifically, an artificial neuron has weighted inputs, transfer function and one output. It receives signals which are real numbers at a connection called edge, and then processes the signals. Some non-linear functions of sum of inputs compute the output of each neuron. Neurons are aggregated into different layers that perform different transformations. Neurons and edges have a weight that increases or decreases the strength of the signal at a connection and adjusts as learning proceeds. A signal is sent only if the aggregate signal crosses the threshold of neurons. Both literature-based and experimental data can be combined and incorporated by ANNs to solve problems. As a promising modeling technique, especially for data sets having non-linear relationships, ANN requires large training sets.

As a specialized DNN models, CNNs appropriately handle high resolution and large images [37], by using the mathematical convolutions for nearby samples weighted average computation and data summary in a region with max-pooling or a similar operation [38].

RNNs models are used for function approximation and temporal sequences. The network makes decisions for each step depending on the output of the previous time step [38]. Using memory to recall information learned in previous steps, RNN performs gradient descent-based training with "exploding gradients" issues that were corrected by LSTM models with additional information flow structures.

2.2.2 Deep Learning-based Energy-efficient Computations Offloading

As a subset of ML, DL has been extensively researched and employed in almost every field of life [39], especially, in fields of computer vision and natural language processing. It has surpassed the performance of conventional ML (shallow learning) methods [35, 40].

In order to predict the time-series user requests and shorten service latency for users, Ale et al. [41] developed a DL model based on bidirectional RNN for reducing the computational cost and addressing the sparsity of data. A novel performance and energy efficient DL-based offloading algorithm was proposed by Gong et al. [42] based on remaining energy and an optimal part of components for offloading was selected. A DL algorithm for an offloading decision-making process was designed by Ali et al. [43]. They used a trained deep neural network with data generated by mathematical model to compute the energy efficient offloading scheme. Yu et al. [44] formulated offloading decision problem as a multiple label classification problem in MEC environment and developed a deep supervised learning method to for reducing system cost. Also, some researchers in the paper [45] adopted a secure and energy-efficient computational offloading scheme, and a long short-term memory (LSTM) algorithm is used for a task's prediction-based computation offloading strategy.

Although DL methods are markable successful in many supervised learning and optimize resource allocation challenges. However, due to vast labeled datasets are required for training models, DL performs not well in MEC networks. Thus, generating and labeling data from a dynamic and complex MEC ENV using DL is considered a big challenge.

2.3 Overview of Reinforcement Learning Approach

As mentioned above, it is challenging to generate and label data from dynamic and complex MEC network with either classical methods which require a large volume of iterations, or DL approach that requires labeled datasets for model training. In contrast, another ML called reinforcement learning [46] is remarkably successful to be employed in computing offloading control using Q-learning with Q-table in MEC networks [47] without labeled data requirement for training.

As another branch of the AI field, RL is the subset of ML method. In RL, by interacting with the environment continuously, an agent learns to take actions to have the most rewards through dynamic learning process. Compared with other ML methods, RL does not rely on labelled data, and it involves more objects, such as action, environment, state transition probability and reward function. RL performs better to solve problems for dynamic systems.

2.3.1 Q-learning

As a model-free (no requirement of environment model) RL method, Q-learning learns the value of an action in a particular state. Unlike standard ML, RL method has an agent to learn from the evaluative and sequential feedback from MEC network environment through interaction. The learning results are stored in a Q-table with tuples including the states, actions, and values. It does not require to know the stochastic transitions or rewards functions. For maximizing the expected value of total reward over successive steps from the current state, an optimal action-selection policy can be identified during MDP. The function computed by this algorithm called "Q" - the expected rewards for an action taken in a given state.

2.3.2 Reinforcement Learning-based Energy-efficient Computation Offloading

For maximizing the long-term utility, RL agent can adjust strategies based on the reward feedback from environment in future state [48]. RL is adopted by MEC researchers to

address many issues for solving control problems without knowing MEC internal transition mechanism.

In [49], a model-free Q-learning approach was adopted by Pan et al. It learns the offloading decision and energy consumption optimization through interaction with MEC network environment to minimize energy consumption and execution time. Liu et al. [50] developed a resource allocation scheme based on RL for making request acceptance and resources allocation decisions. However, because of the successive complex states and actions of dynamic environment in offloading process, storing all the state-action value pairs in a Q-table is considerable challenging and even impossible. Therefore, standard RL methods cannot perform well.

2.3.3 Deep Reinforcement Learning Approach

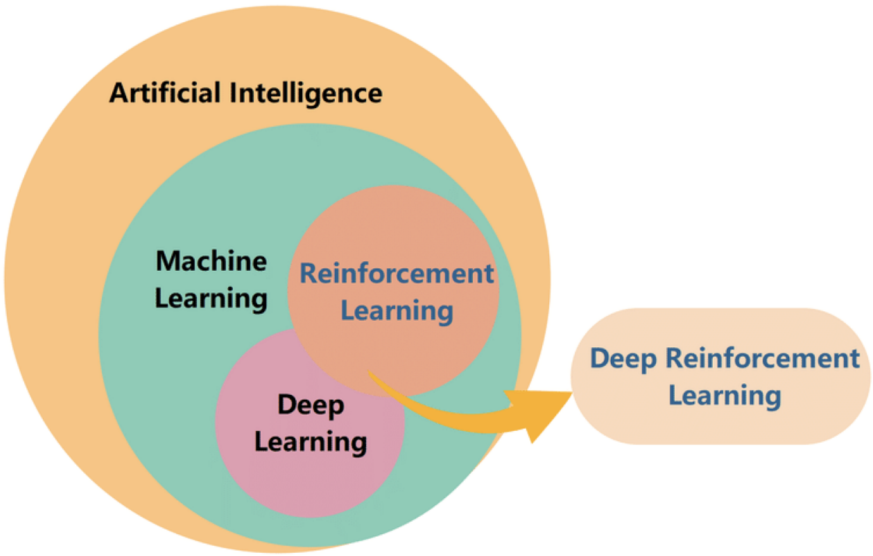

FIGURE 2.2: The relationship between AI, ML, RL, DL and DRL

To address the above issues in offloading process in MEC, deep reinforcement learning based method [51] is introduced by integrating deep Q-network (DQN) into reinforcement learning and has been regarded as a suitable method for searching asymptotically optimal solutions in time-varying MEC environment [52]. Ji et al. [39] shows the relationship between AI, ML, RL DL and DRL in Fig. 2.2.

As one of the most effective types of DRL, Deep Q-learning (DQL) has its Q-function

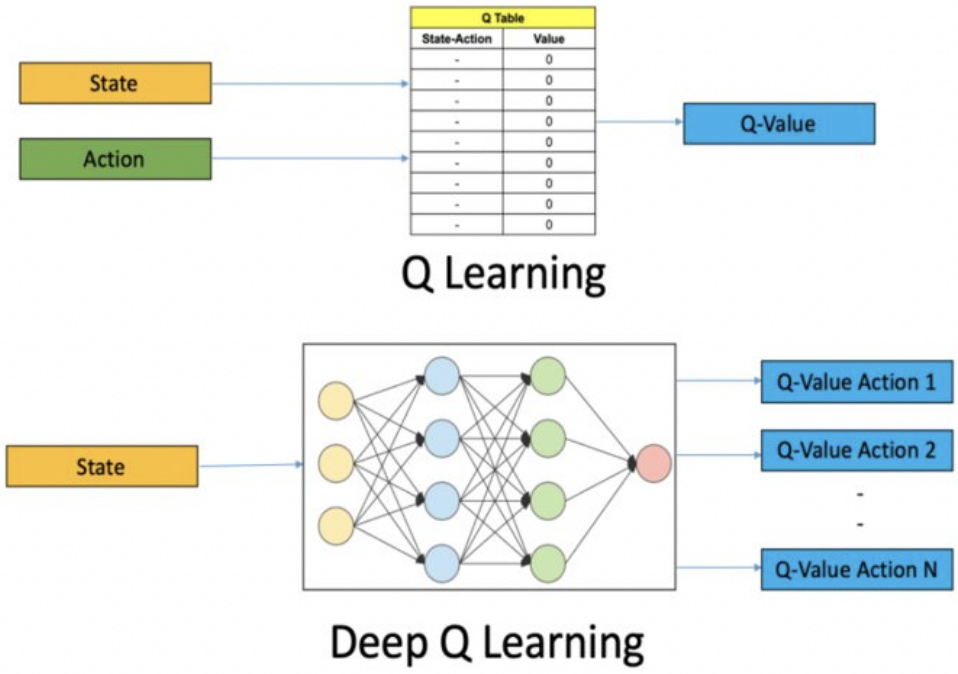

FIGURE 2.3: The Difference between Q learning and Deep Q-learning

that integrates with DQN (a deep convolutional neural network) and layers of tiled convolutional filters for function value estimation. Critically, the DQN works as an approximator of the state-action values in DRL, instead of establishing a Q table. With a random sample is used of previous actions (not the most recent action) for proceeding, experience replay is used in DQL for smoothing data distribution changes and removing observation sequence correlations. Adjusted Q values are updated to target values with periodical updating, and correlations with the target are reduced further. The difference between Q-learning with Q-table and DQL with DQN is shown in Fig. 2.3).

In DQL, some parameters including model state s, action taken by agent a, and reward value r are compositions of the Q-function. The agent in DQL can be represented as Q(s,a;θ) and selects an action a and receives a reward for it correspondingly, where θ is the weights related to each layer at time t in DQN. Moreover, the next state in DQL model is represented by s'. The DQL agent may move to s' depending on action a performed in previous state s. DQN constructs a loop of feedback for estimation and target Q-value prediction so that the target weights are periodically updated. Moreover, obtained on current and previous

states, the Q-values determine the loss function of each learning activity.

2.3.4 Deep Reinforcement Learning-based Energy-efficient Computation Offloading

The environment's hidden dynamics can be captured better by enhancing the intelligence of mobile edge networks and the best long-term goal could be realized by agent learning strategies through repeated environment interaction [53, 54]. With the features and advantage, DRL exhibits its particular potential for computation offloading and resource allocation schemes designation in dynamic MEC system.

For an efficient computation offloading strategy, Zhan et al. [55] applied game theory in designing the decentralized algorithm and adopted a policy gradient DRL approach to solve the offloading decision-making problem. In order to greatly reduce computational complexity due to dynamic and complex channel conditions, Huang et al. [56] proposed a DRL-based online offloading approach to optimize the task offloading decisions and wireless resource allocations. Alfakih et al. [57] developed a DRL-based state-action-reward-state-action algorithm for making optimal offloading decision to save energy consumption and computing time. In [58], DRL is used to minimize energy consumption by controlling computation offloading for Internet of Vehicles. Li et al. [59] proposed a DRL based resource allocation approach to improve the Quality-of-Service and reduce service delay. Moreover, DQN is adopted by [60] in MEC system for minimizing energy consumption and response latency. In [61], an online DRL-based offloading control method in MEC is developed. A double DQN (DDQN)-based method was adopted, and a mixed-integer nonlinear programming (MINLP) problem was formulated by Zhou et al. [62] to save energy consumption in MEC system with delay constraint and uncertain resource requirements of heterogeneous computation tasks. Also, DRL was utilized to design an optimal computation offloading strategy in the work of Dai et al. [63] for minimizing system energy consumption. By exploiting non-orthogonal multiple access for computation offloading in MEC, Qian et al. [64] adopted distributed online algorithm based on DRL to solve a joint optimization problem of the multi-access multi-task computation offloading, NOMA transmission, and computation-resource allocation, in order to save IoT user energy cost for completing tasks with required

service latency. The Q-learning and DRL based schemes are proposed in the work of Li et al. [65] to minimize sum energy consumption and jointly optimize the offloading decision and computational resource allocation in a multi-user wireless MEC system. Moreover, in the work of Yan et al. [66], a MEC system where an access point assists an IoT user to execute applications was considered. They developed a DRL structure based on actor-critic learning to jointly make offloading decision of tasks following a general task call graph and the resource allocation under dynamic wireless channels and stochastic edge computing capability, to achieve energy-time consumption minimization goals.

2.4 Security Challenges

In existing literature, researchers mainly focus on delay-optimal [67] or energy-optimal offloading schemes in computation offloading in MEC systems [68–72], and the security issue is seldom considered. However, due to the broadcast nature of wireless communications, security provisioning is essential [73]. The computation offloading in MEC may face security challenges when the computation tasks could be overheard by malicious eavesdroppers nearby during offloading process from IoT users to edge servers through wireless channel [74, 75]. For example, the users' private data (e.g., health information, identity information) could be eavesdropped. The more devices are connected, the more data is attractive for cybercriminals.

In order to address above problem, some conventional cryptography at upper layer techniques [76] can be used, such as Elliptic Curve Cryptography [77], a steganographic protocol using HTTP "control" messages [78]. As cryptosystems at upper layer are built on hardness of computing problems such as Factoring (RSA) and Discrete logarithm (Diffie-Hellman), and the assumption of limited computational capability of eavesdroppers. They have some shortcomings: perfect secure channel for key exchange required, algorithms could be compromised and high computation complexity. They usually require high computational power and need key management. Therefore, they might not be applicable to IoT network with very limited capacity. In addition, they can only provide computational security and they are vulnerable to various attacks as the adversary's power keeps increasing. Although there are some cryptographic techniques that can be employed, it is considered challenging to

implement the traditional cryptographic techniques in IoT users with limited computing capacities [79]. Moreover, those techniques often add extra-computational overheads to the resource-limited IoT users inevitably. Novel solutions are urgently needed to protect the security while not incurring too much overheads.

2.4.1 Physical Layer Security with Friendly Jammer

Without replacing conventional security schemes, extensive studies have explored advantages of one potential solution which is PLS [80]. As a viable solution to guarantee wireless offloading security, PLS explores the physical layer characteristics (e.g., wireless channels) and exploits the differences in channel conditions between each legitimate user and eavesdropper to enhance communication systems security [81]. PLS achieves perfect secrecy or information-theoretical security during data transmission between legitimate terminals.

Unlike encryption-based algorithms, the secrecy level provided by PLS is not compromised by the computation resources limitation of mobile devices and does not depend on any encryption technologies which are computationally expensive [82].

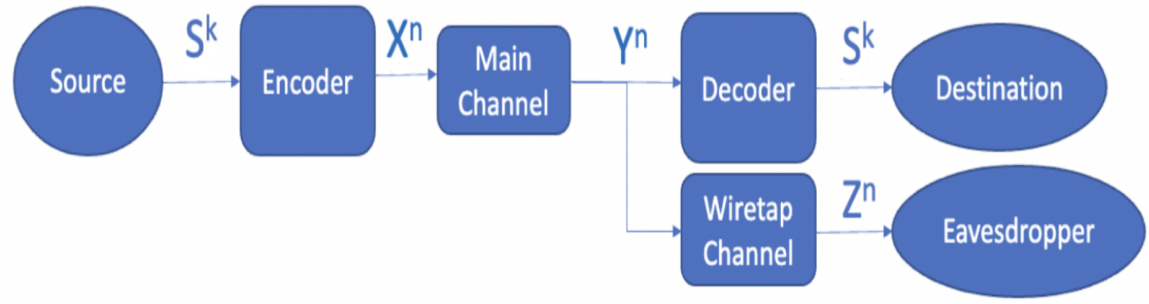

FIGURE 2.4: Wiretap Channel Model

See the wiretap channel model shown in Fig. 2.4, in PLS, cryptosystem cannot be broken even when the adversary has unlimited computing power. Transmitted signal could be protected from being decoded by eavesdroppers with maximizing secrecy capacity. Secrecy capacity is the maximum rate at which the source sends information securely to the destination which can recover information without errors, while the eavesdropper gains nothing

[80]. It is calculated by main channel capacity minus wiretap channel capacity as:

$$C_{Sec} = C_M - C_W \tag{2.1}$$

Only when $C_M > C_W$, secrecy capacity becomes positive. However,in the case that channel capacity of the destination is less or close to wiretap channel capacity ($C_M \lesssim C_W$), signal could not be securely transmitted to destination.

To solve this problem, researchers have investigated some PLS techniques with jammer [83, 84] for reinforcing the privacy and security of wireless communication systems. When the source is sending signal, a jammer can generate artificial noise to degrade the reception performance of the eavesdropper, thus improving the secrecy rate. In practical computation offloading in MEC, eavesdropping attacks have been considered and wireless channels nature in PLS has been exploited to enhance security [85, 86]. Moreover, IoT users have great concerns in performance experiences regarding energy minimizing under the premise of security provisioning. It is a nature idea to consider PLS applications in secure computation offloading to MEC system [74]. It is critical to focus research on resource management in MEC for optimizing offloading energy efficiency and ensuring offloading security [85]. Therefore, a novel DRL approach was proposed in this work with the goals of minimizing energy cost and enhancing security of offloading tasks using PLS technologies with friendly jammer in a MEC system with an eavesdropper.

Chapter 3

System Model and Problem Formulation

Motivated by pioneering research, a jammer assisted secure offloading scheme is proposed in this work for MEC. In the system model, a friendly jammer is employed to assist IoT users, which sends jamming signal to degrade the eavesdropper's reception and enhances the secure transmissions (increasing security rate) in offloading process. When the main channel capacity is less or close to the eavesdroppers' capacity, jammer generates noise signals, which is transmitted to confuse or degrade the decoding capability of eavesdroppers (unintended receivers). Therefore, it could significantly improve the secrecy capacity and enhance secure communications between loT users and selected edge servers, while malicious eavesdroppers obtain zero useful information.

In this study, similar to many existing works on PLS [74], the eavesdropping attack is considered where an eavesdropper is a passive attacker. The eavesdropper's interest is to intercept the signal and learn the information transmitted between IoT devices and edge servers. The jammer attack is not considered in this work. For potential jamming attack from the adversaries, there are some anti-jamming techniques that can be used. One anti-jamming technique is frequency hopping whereby the legitimate users rapidly change the carrier frequency in a predetermined order (only known to the legitimate users) to avoid jamming attack.

3.1 System Model

Fig. 3.1 shows the system model, where IoT users offload their computation tasks to MEC server with the aid of a friendly jammer and in presence of an eavesdropper.

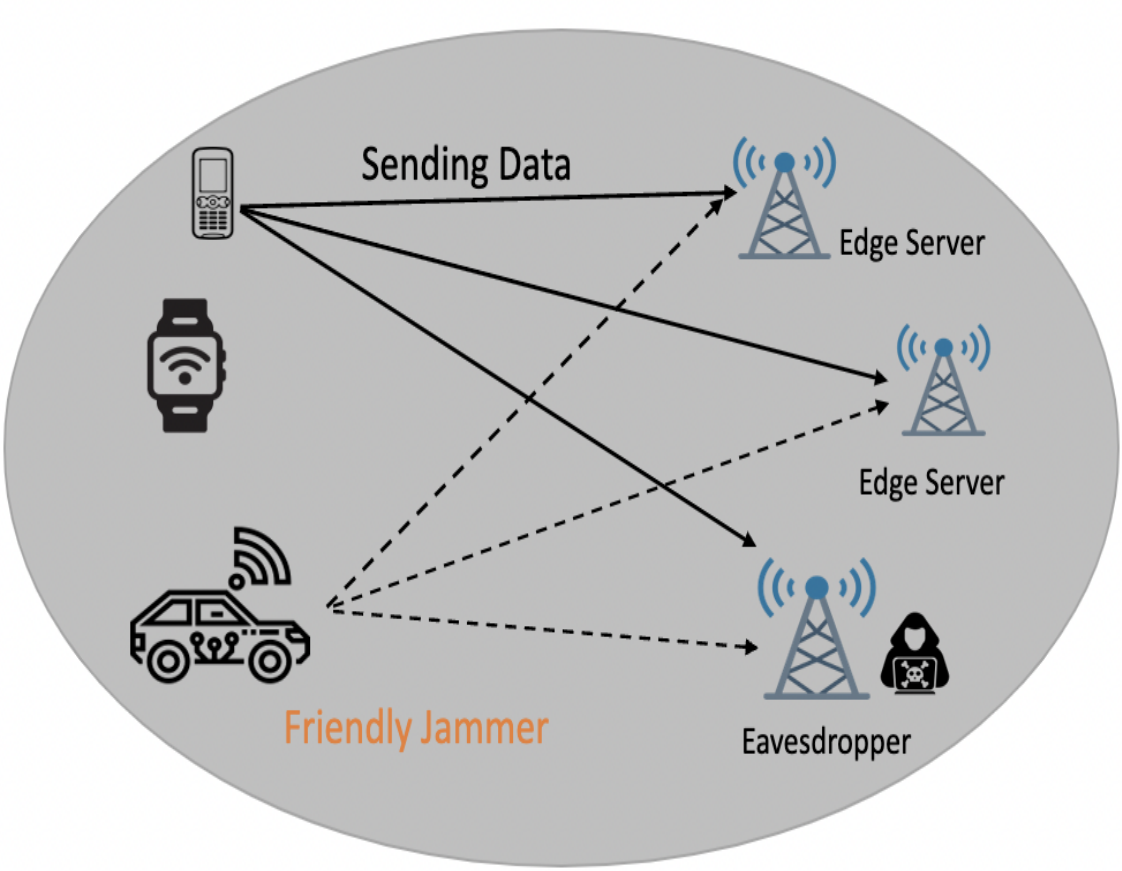

FIGURE 3.1: System Model

Suppose that there exists an eavesdropper that can eavesdrop and receive the data offloaded to edge servers. In order to reduce the data leakage, we choose other IoT user as a friendly jammer to broadcast artificial noise called friendly jamming. The jammer produces jamming signal to degrade the reception of the eavesdropper and thus improve the security.

For computation offloading, suppose there is a control agent which can communicate with any of edge servers and coordinate MEC system. The system state information such as tasks profile, each edge server status, and output of a learning model for optimizing decisions for executing tasks, can be collected by agent. The action includes edge server selection for offloading a certain task, recommended CPU frequency at edge servers for

TABLE 3.1: Summary of Main Notations

$u_n^{(t)}$	n^{th} IoT user at at t time slot
$m_k^{(t)}$	k^{th} MEC server at t time slot
Q_k	k^{th} MEC server m_k tasks queue
$f_{max}^{(k)}$	maximum recommended CPU frequency of k^{th} edge server m_k
$T_{i,j}^{(t)}$	j^{th} task of i^{th} user at t time slot
$d_{i,j}$	task $T_{i,j}^{(t)}$ data size
$c_{i,j}$	requested CPU cycles to process task $T_{i,j}^{(t)}$
$t_{i,j}^{max}$	maximum tolerate time of task $T_{i,j}^{(t)}$
t_k^{res}	current task residual running time in selected k^{th} MEC server
t_k^{que}	task waiting time in queue before being serviced
$t_{i,j}^{tran}$	security transmission time for offloading task $T_{i,j}$ to MEC server
$t_{i,j}^{comp}$	computing time of task $T_{i,j}$
$r_{i,k}$	transmission rate at k^{th} MEC server
$\alpha_{i,l}^{(B)}$	bandwidth
P_i	transmission power of i^{th} IoT user
P_i'	transmission power of jammer
$h_{i,l}$	Rayleigh fading
$L_{i,l}$	path loss
$r_{i,e}$	eavesdropper's transmission rate
$r_{i,k}^{sec}$	security transmission rate to target k^{th} MEC server
$E_{i,j}$	total energy consumption of offloading task $T_{i,j}$
$E_{i,j}^{tran}$	security transmission energy consumption
$E_{i,j}^{comp}$	computation energy consumption

executing task and friendly jammer selection. Tasks could be offloaded to the selected edge servers under the aid of the selected friendly jammer for security purposes, and then tasks are processed by servers with recommended CPU frequencies by following the agent's instructions.

In this system, a set of IoT users is denoted as $\mathcal{U}_t = \{u_1, u_2, \ldots, u_n\}$, where n is the number of users at a given time slot t. A set of MEC servers is denoted as $\mathcal{M}_t =$

$\{m_1, m_2, \ldots, m_k\}$ with k as the number of edge servers at the given t time slot. Specifically, as MEC servers have different capacities, their heterogeneity is captured by $\mathcal{M}_t = \{(Q_1, f_{max}^{(1)}), (Q_2, f_{max}^{(2)}), \ldots, (Q_k, f_{max}^{(k)})\}$, where Q_k is task queue and $f_{max}^{(k)}$ is maximum CPU frequency of k^{th} edge server. At a given time slot, more than one task could be generated for offloading, and tasks also could be chosen to process remotely by an edge server or locally. Each IoT user has computation tasks to be completed within a certain delay constraint. $T_t = \{T_{0,0}, T_{0,1}, \ldots, T_{i,j}, \ldots, \}$ is the set of tasks and the i^{th}user's j^{th} task is denoted as $T_{i,j}$. Each computation task can be described in three terms as $T_{i,j} = \{d_{i,j}, c_{i,j}, t_{i,j}^{max}\}$ with $d_{i,j}$ denotes the data size, $c_{i,j}$ defines the requested CPU cycles and $t_{i,j}^{max}$ represents the maximum tolerant time of task $T_{i,j}$.

The task $T_{i,j}$ service time includes t_k^{res} denotes current task residual running time in selected edge server, the transmission time for offloading task to MEC server $t_{i,j}^{tran}$, the waiting time in the queue before service t_k^{que}, and the computing time $t_{i,j}^{comp}$. When total task service time is less than task maximum tolerant time $t_{i,j}^{max}$, a task is processed successfully. Otherwise, the task fails. It is formed as

$$T_{i,j} = \begin{cases} 1, & \text{if } t_k^{res} + t_k^{que} + t_{i,j}^{tran} + t_{i,j}^{comp} \leq \mathrm{t}_{i,j}^{max}; \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

t_k^{res} is the current executed-task's residual running time in the target k^{th} server and can be computed by current running task's total computing time t_{curr}^{comp} minus start running time t_{run}. Namely,

$$t_k^{res} = t_{curr}^{comp} - t_{run}. \tag{3.2}$$

In the queue of k^{th} server, the task waiting time t_k^{que} is the aggregate computing time of all the tasks before the current task. Namely,

$$t_k^{que} = \sum_{j=0}^{M} \frac{c_{*,j}}{f_{*,j}}. \tag{3.3}$$

where $c_{*,j}$ is the required CPU cycles[33] and $f_{*,j}$ is the recommended frequency.

$t_{i,j}^{tran}$ is the security transmission time which depends on the data size and the security rate. It is the time spent to transmit the data to the edge server securely (i.e., the eavesdropper cannot gain nothing). It can be obtained by dividing the task data size $d_{i,j}$ by the security rate $r_{i,k}^{sec}$, as

$$t_{i,j}^{tran} = \frac{d_{i,j}}{r_{i,k}^{sec}}, \tag{3.4}$$

The security transmission rate $r_{i,k}^{sec}$ to the target k^{th} server is given by

$$r_{i,k}^{sec} = r_{i,k} - r_{i,e}, \tag{3.5}$$

where $r_{i,k}$ and $r_{i,e}$ are the transmission rate at the k^{th} edge server and the eavesdropper's rate, respectively.

The transmission rate $r_{i,k}$ to the target k^{th} server can be given as

$$r_{i,k} = \alpha_{i,l}^{(B)} log_2(1 + \frac{P_i h_{i,k} L_{i,k}}{N_0 + P_i' h_{i,k} L_{i,k}'}), \tag{3.6}$$

where $\alpha_{i,l}^{(B)}$ is the bandwidth, P_i is i^{th} IoT user's transmission power, P_i' is jammer's transmission power, $h_{i,l}$ is Rayleigh fading and and $L_{i,l}$ is path loss. Similarly, the eavesdropper's rate $r_{i,e}$ can be given by

$$r_{i,e} = \alpha_{i,l}^{(B')} log_2(1 + \frac{P_i h_{i,e} L_{i,e}}{N_0 + P_i' h_{i,e} L_{i,e}'}) \tag{3.7}$$

The computing time $t_{i,j}^{comp}$ of task which is offloaded to selected edge server is given as

$$t_{i,j}^{comp} = \frac{c_{i,j}}{f_{i,j}^{k}}, \tag{3.8}$$

$c_{i,j}$ denotes the requested CPU cycles for task $T_{i,j}$ computation and $f_{i,j}^{k}$ represents the recommended CPU frequency.

Therefore, we can get the sum energy consumption $E_{i,j}$ as summation of security transmission energy consumption and computation energy consumption which are denoted by denoted by $E_{i,j}^{tran}$ and $E_{i,j}^{comp}$, respectively

$$E_{i,j} = E_{i,j}^{tran} + E_{i,j}^{comp}. \tag{3.9}$$

3.2 Problem Formulation

The long-term utility of computation offloading process in MEC system is considered in this work, and our objective is maximizing number of completed tasks before deadlines, minimizing energy consumption and data leakage with enhanced transmission security in the long run.

The optimization problem can be formulated theoretically as following:

$$\begin{aligned}
\min_{a_\tau} \quad & \sum_{\tau=0}^{N} E_\tau^{tran} + E_\tau^{comp} \\
& = \sum_{\tau=0}^{N} \left[\frac{d_{i,j}}{r_{i,j}^{sec}} P_{i,j} + 10^{-26} (f_{i,j}^k)^2 c_{i,j} \right] \\
\text{s.t.} \quad & t_k^{res} + t_k^{que} + t_{i,j}^{tran} + t_{i,j}^{comp} \leq \mathrm{t}_{i,j}^{max}, \\
& r_{i,j}^{sec} \geq \mathcal{P}, \\
& f_{i,j}^k \leq f_k^{max}, \\
& a_\tau = \{k, f_{i,j}^k, J_s\}
\end{aligned} \tag{3.10}$$

where $\mathcal{P}$ is the security threshold, J_s is selected jammer and f_k^{max} is maximum frequency of k^{th} MEC server, respectively. The E_τ^{tran} and E_τ^{comp} are security transmission energy consumption and computation energy consumption which are calculated by

$$E_{i,j}^{tran} = t_{i,j}^{tran} P_{i,j} = \frac{d_{i,j}}{r_{i,j}^{sec}} P_{i,j}. \tag{3.11}$$

$$E_{i,j}^{comp} = 10^{-26} (f_{i,j}^k)^2 c_{i,j}, \tag{3.12}$$

where $f_{i,j}^k$ is the frequency for computing task $T_{i,j}$ in k^{th} MEC server.

However, the above reward formulation has some drawbacks. Firstly, sometimes, the

constraint $t_k^{res} + t_k^{que} + t_{i,j}^{tran} + t_{i,j}^{comp} \leq \mathrm{t}_{i,j}^{max}$ might not be satisfied. Moreover, when there is no solution in the feasible areas, control agent has to deal with it. Secondly, balancing the energy cost and response delay is not flexible. Moreover, the computational cost may grow dramatically due to the increase in variables and the problem scale.

3.2.1 Markov Decision Process Formulation

As mentioned previously, the long-term utility of the system is mainly considered in this work. Therefore, the optimization problem is formulated as a Markov Decision Process (MDP) for maximizing the expected long-term rewards. The interaction of agent and environment in a MDP is shown in Fig. 3.2 [46].

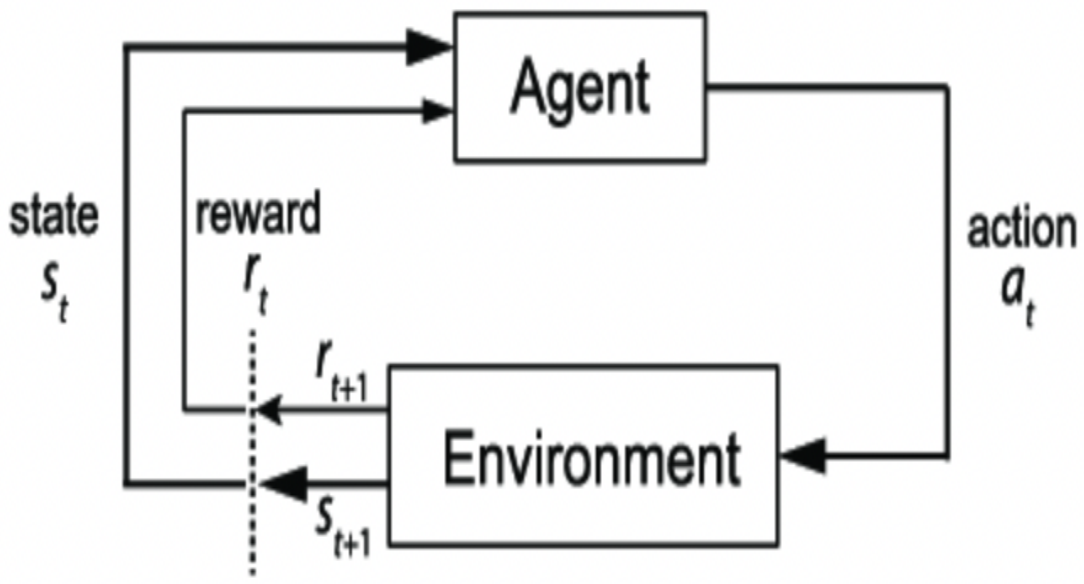

FIGURE 3.2: Typical Reinforcement Learning Cycle

For RL problem, MDP is an idealized and mathematical form for evaluative feedback, choosing different actions according to situations. In the agent-environment interface, MDP is adopted for learning from interaction between agent and MEC network environment to achieve the long-term goals. In this goal-directed interaction process, the agent is the learner and decision maker, and environment is almost all the things agent tries to interact with outside. Informally, we could consider that anything outside of the agent and cannot be

arbitrarily altered is the environment. When actions are selected by the agent, responses will rise for the actions accordingly, and then new situations in the environment will be provided to the agent. The responses passing to the agent from the environment are called rewards. In RL, the formulating goal of agent is maximizing its received cumulative rewards over time instead of immediate ones. Simply, by choosing actions and receiving environment responses, the cumulative numerical values kind of rewards have been maximized in a long run by the agent.

As a discrete time stochastic control process, MDP models the computation offloading decision process. The interaction between agent and environment in MDP is considered as an episode that equals to an offloading period. We call the sub-sequences as episodes where interactions naturally break into. An episode contains several discrete time slots denoted by t. At each t time slot, a control agent is in some s_t state. The MDP episode starts at a a standard random initial state or a standard distribution sample and ends at a special state named terminal state. An episode could start independently whenever the previous episodes ended at. Therefore, we can consider that almost all the episodes end at the same terminal state. with different rewards for the different outcomes. Specifically, in this work, an episode is considered to terminate at the time when one or more edge servers are overloaded. when the MEC network environment current state is s_t, as a decision maker, the agent has to choose any action a_t. Therefore, by forwarding the agent to the next new state and rewarding the agent correspondingly, the MDP provides responses at the next time step. When the MEC network environment current state is s_t, the action a_t taken by the agent brings rewards. A_t, S_t and R_t represent the action space, state space and reward, respectively. Our proposed MDP guarantees the memory-less Markov property. The future states only rely on current state, not former ones.

Concretely, with time step in given time slot t, the states, actions, and reward of the MDP formulation in the MEC network environment are formally defined with details as below.

States: The states of the environment include the current arrived tasks, the workload of edge servers, the wireless channel conditions.

- States={Tasks Info, Edge Servers, Security Speed Matrix}

- Task ={user ID, data size, CPU cycles, maxi tolerant time}
- Edge Server ={server configuration, task queue}
- Security Speed Matrix: depends on channel and pass loss

At t time slot, states s_t is:

$$s_t = \{s_0, s_1, s_\tau, \dots, s_\delta | \sum_{\tau=0}^{\delta} \tau << t\}, \tag{3.13}$$

where the δ is the threshold of number of tasks received by the MEC servers at t. If indices of tasks are larger than δ, tasks are assumed to be processed in the next time slot. For $s_\tau = \{T_\tau, M_\tau, r_\tau^{tran}, r'_\tau\}$, $m_k^{(\tau)}$ is the state of k^{th} edge server in M_τ, r_τ^{tran} denotes the transfer speed to r_τ^{tran} edge servers, and r'_τ is eavesdropper transmit speed.

Moreover, when the model takes action, tasks are added into queues at edge servers. Before the task being served, its waiting time in target edge server is updated. To run the task, its queues are derived in selected k^{th} edge server with tasks information and recommended CPU frequency represented as $f_r^{(k,\tau)}$:

$$Q_k^\tau = \{(T_{1,1}^{(1,\tau)}, f_r^{(1,\tau)}), \dots, (T_{i,j}^{(k,\tau)}, f_r^{(k,\tau)}), \dots\}, \tag{3.14}$$

where $T_{i,j}$ is the j^{th} task of i^{th} IoT user.

Actions: the actions the agent can choose include which edge server to offload the tasks, which intermediate node is selected as the friendly jammer among all possible nodes, and how to allocate CPU cycles at the edge servers to execute the task. When the agent chooses to take actions, factors in the actions includes task offloading target edge server's index, edge servers' recommended CPU frequency, and selected friendly jammer, as following:

Action={⟨ Target Server ID, Recommended CPU Frequency, Selected Jammer ⟩, ...}

An action can be denoted as $a_\tau = (k, f_{i,j}^k, J_s)$ with target offloading edge server k, k^{th} server's recommended maximum CPU frequency for j^{th} task of i^{th} IoT user executing $f_{i,j}^k$ which has range between 0% and 100%, and selected jammer J_s.

Within t, a set of actions can be taken by agent is:

$$a_t = \{a_0, a_1, a_\tau, \ldots, a_\delta | \sum_{\tau=0}^{\delta} \tau << t\}. \tag{3.15}$$

Ideally, at a time slot t, received tasks can be distributed by a single action. When the multi-tasks are distributed from multi-users to multi-edge servers with other action control parameters, some RL model may be diverged due to action space explosion [46]. In order to address those challenges, the states transition could be assumed depend on the control agent actions for collecting rewards of the tasks received in a task queue within $\tau << t$. During t time slot, we assume there is no change in some other MEC parameters and channel distribution. Based on state transition with time step τ instead of t in MDP, $p(s'|s,a)$ presents the transition probability and can be formed follows:

$$p(s'|s,a) \doteq \Pr\{s_\tau = s' | s_{\tau-1} = s, a_{\tau-1} = a\}. \tag{3.16}$$

$$\sum_{r \in R} p(s'_\tau, r_\tau | s_\tau, a_\tau) = 1 \tag{3.17}$$

Rewards: The reward function reflects the multiple objectives that I want to achieve, including maximizing the number of completed tasks before deadlines and saving energy consumption, while satisfying the security requirement. I plan to use a weighted sum method for multi-objective optimization. Different weights can be assigned to the energy consumption and number of completed tasks, so that they can be converted to the same scale. The weights can be adjusted based on the interests of the operators or users in the system.

A flexible reward function of DRL model is designed in our work for solving all aforementioned optimization problems in an end-to-end manner at one time.

Rewards Function:

$$R_t(s_t, a_t) = \sum_{\tau=0}^{\delta} R_\tau(s_\tau, a_\tau) = \sum_{\tau=0}^{\delta} (1-\eta)\beta_1 T_{i,j} - \eta\beta_2 \log_2(E_\tau) + \mathcal{C}, \tag{3.18}$$

with $\sum_{\tau=0}^{\delta} \tau << t$.

As reward is obtained in t time slot, all the rewards $R_\tau(s_\tau, a_\tau)$ at τ time step are collected in $R_t(s_t, a_t)$. $R_\tau(s_\tau, a_\tau)$ indicates completed tasks number and energy cost with β_1 is a term for reward normalization of completed tasks before deadline and β_2 is for energy consumption. For balancing the completed tasks number and energy cost, the weight $\eta \in [0, 1]$ is adjustable in different applications. A positive constant number $\mathcal{C}$ is used for accumulative increasing.

Value Function for Policy: As the functions of state–action pairs or states, value functions are used for agent performance estimation such as action choice from experience in states. The agent performs good or not depends on the expected future rewards or return which depends on the chosen actions. Therefore, the value functions are restricted with particular acting approach which is called policy. Concretely, as an ordinary function that maps states to each possible selected action's probability, $\pi(a|s)$ (distribution is over $a \in A(s)$ for each $s \in S$) is followed by agent at time t. In RL methods, the agent changes its policy based on the experiences.

Under a policy π, as an expected return, $\upsilon_\pi(s)$ denotes the value of state s, so it is defined as state-value function for all $s \in S$ in MDP. Value $\upsilon_\pi(s)$ can be presented as,

$$\upsilon_\pi(s) \ = \mathbb{E}_\pi \Big[G_t \mid S_t = s\Big] = \mathbb{E}_\pi \Big[\sum_{k=0}^{\infty} \lambda^k \ R_{t+k+1} \mid S_t = s\Big]. \tag{3.19}$$

Where, the expected value $\mathbb{E}_\pi$ is from a random variable, and indicated that policy π is followed by the agent.

Moreover, under the policy π, $q_\pi(s, a)$ is called action-value function and presents the value of choosing action a in state s. It is defined as,

$$\mathrm{q}_\pi(s, a) \ = \mathbb{E}_\pi \Big[G_t \mid S_t = s, A_t = a\Big] = \mathbb{E}_\pi \Big[\sum_{k=0}^{\infty} \lambda^k \ R_{t+k+1} \mid S_t = s, A_t = a\Big]. \tag{3.20}$$

Optimal Value Function for Optimal Policy: In RL, for achieving the goal that obtains

as much reward as possible over long run, finding the optimal functions includes optimal state-value function and optimal action-value function for policies are significant. As there always exists a policy that is not worse than others. Theoretically, the number of this kind of policy is more than one. We call them all the optimal policies which share the same optimal state-value function and optimal action-value function. Separately, these two optimal value functions are denoted as,

$$\upsilon_*(s) \ = \max_{\pi} \ \upsilon_\pi(s) \tag{3.21}$$

and

$$\mathrm{q}_*(s,a) \ = \max_{\pi} \ \mathrm{q}_\pi(s,a) \tag{3.22}$$

Theoretically, an optimal policy can be learned very well. However, it can not be computed by solving Bellman optimality equation simply in practice, even if there is an accurate and complete model for the dynamics in environment. Thus, the agent can only approximate to varying degrees. For approximating the value functions and policies, a large amount of memory is required. It is possible to build up the optimal policies approximation in RL with on-line nature to make better decisions. This makes RL distinguished from other traditional approaches.

Chapter 4

Proposed Method

4.1 Proposed Deep Reinforcement Learning Model

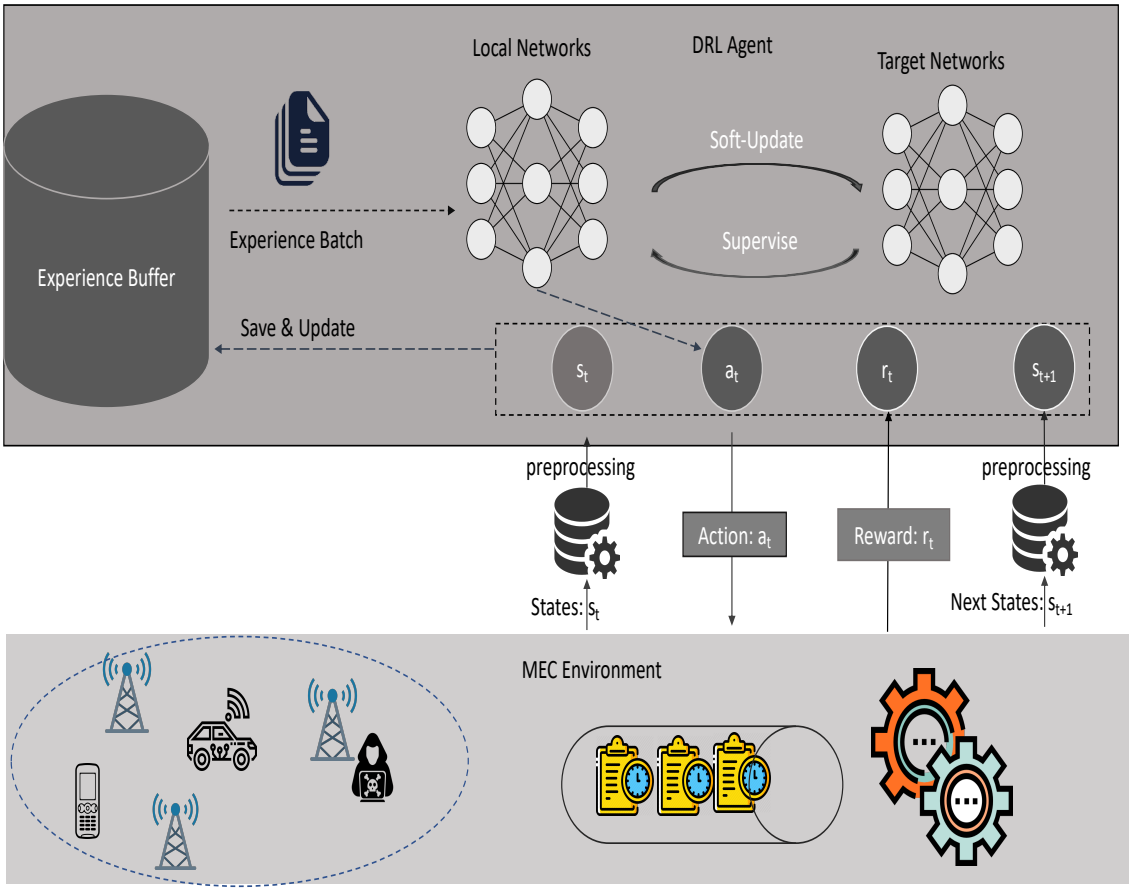

FIGURE 4.1: Developed end-to-end Deep Reinforcement Learning Model

In this work, we propose a developed end-to-end DRL method (shown in Fig. 4.1) concerned with how the agent takes actions in the MEC environment in order to maximize the long-term reward (achieve multiple objectives) includes maximizing the number of completed tasks before deadline and saving energy consumption constrained by security transmit rate at the same time.

Without assuming the knowledge of any MDP mathematical models exactly, the MEC network environment is considered as a RL environment and a MDP is satisfied with internal transition probability $P(r, s\prime|s, a)$. We consider the proposed DRL model as a learning agent that can learn from the experience through interacting with MEC network.

First, we assume that the decision of the n^{th} IoT user for choosing an action in a given state s is determined by a policy:

$$\pi(s) = a, \forall s \in S, a \in A. \tag{4.1}$$

In the offloading system of MEC network, a coordinator plays a role for collecting and placing offloading tasks profiles into the queue in MEC ENV firstly. Then, input states are represented by DQN for probabilities computation $P_A = \{p_1, \ldots, p_k\}$ of all possible actions at one time. As the backbone of the proposed method, DRL agent takes actions depending on this probabilities and environment interactions. The tasks offloading is executed by coordinator to the target server, in where tasks are run with recommended CPU frequencies. The experience replay buffer works for storing data information includes action, current and next states, and reward. The data is fed into the DRL model for training purposes. Then, the sample data is withdrawn from the buffer by DRL agent. The learning network is trained by reducing the loss function which is determined by Mean Square Error (MSE). After every episode, the target network is updated accordingly, finally.

In this work, in order to learn from an unknown MEC ENV, balancing exploration and exploitation (Fig 4.2) with a $\epsilon - greedy$ algorithm is critical for the DRL agent (the greedy algorithm is shown in Fig 4.3), where ϵ is designed for decreasing over episodes. The reason ϵ is decay over time because the agent needs to explore and learn more about the environment in the earlier episodes; further, it can leverage those explored data and learn to take better actions over the episodes. As the agents collect more data and learn enough about the MEC

ENV, we should decrease the explore actions because those explore may take actions that leads to poor performance. Exploration is the agent takes actions randomly for knowledge acquiring and learn from the unknown MEC network environment, and exploitation allows the agent to leverage what it has learned by exploring the search space associated with Q value [**?**]. In other words, the exploration process is a phase to collect training data for the agent to train the model, whereas the exploitation is to choose optimal policies based on the collected information. In the early episodes, due to no knowledge of MEC network environment for agent, it should take actions randomly to explore the MEC ENV. When the agent acquires enough knowledge gradually, learned knowledge will start to be exploited for generating the goal which is optimal policies, namely,

$$\pi(s) = a = \arg\max_{a' \in A} Q\left(s, a'; \theta\right), \tag{4.2}$$

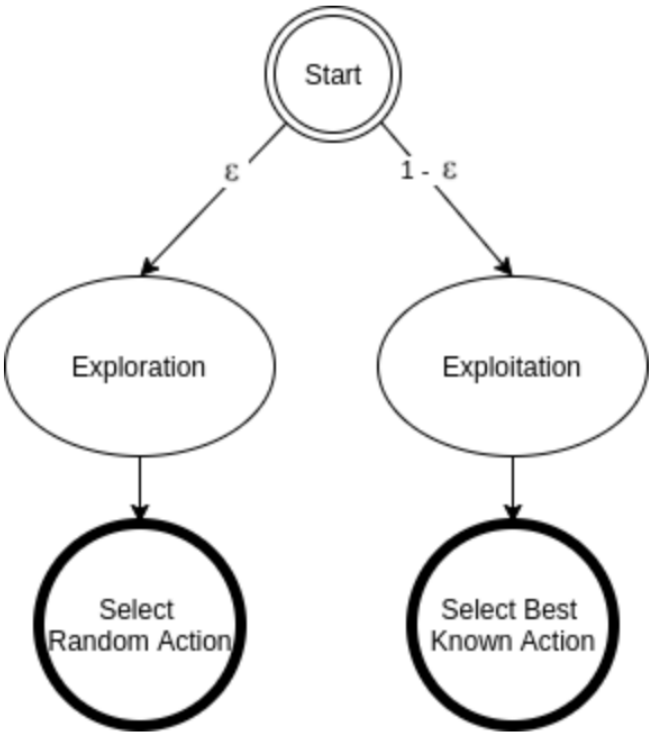

FIGURE 4.2: Epsilon-Greedy Action Selection

The optimal policy π^* is the policy for optimizing the action-value $Q^*(s, a; \theta)$ to maximize the long-term accumulative rewards. Agent takes action a with state s, and following policy π to generate action-value $Q(s, a; \theta)$. It is computed with Bellman equation (Eq.4.4),

Algorithm 1 An $\epsilon - greedy$ algorithm

Initialization:
$t \leftarrow 0; s \leftarrow 0, \forall s \in S; Q(s, a; \theta) \leftarrow 0, \forall s \in S, a \in A$
while t $\leq$t^{max} (t^{max} is the is the maximum number of iterations) **do**
  for each user u **do**
    if exploration **then**
      chooses action a with $\epsilon \in (0, 1)$;
    else if exploitation **then**
      chooses an action $\pi(s) = a = \arg\max_{a' \in A} Q(s, a'; \theta)$;
    end if
    perform a and get a reward r(s,a) and a successor state s$'$;
    update the Q-value function;
    s $\leftarrow s'$;
  end for
  t $\leftarrow t + 1$;
end while

FIGURE 4.3: Greedy Algorithm for Balancing Exploration and Exploitation

which is the summation of an immediate reward R and the expected return with discounted by the factor λ. $Q^*(s, a; \theta)$ denotes the optimal action-value which is the maximum value of all possible values of $Q(s, a; \theta)$. To derive optimal action-value, the DRL gent takes action a' from all possible actions, and this is for the next step of $Q(s, a; \theta)$ in order to maximize the $R + Q(s', a'; \theta)$. This is a repeat step for generating the optimal policies over all the states. The optimal action-value could be derived by iterative updating of Bellman equation, and the value of $Q(s, a; \theta)$ has continuous improvement in iterations. $Q_\tau \rightarrow Q^*$ with $\tau \rightarrow \infty$ is shown as below,

$$Q^*(s, a; \theta) = \max_{\pi} \mathbb{E}\left[R_\tau | s_\tau = s, a_\tau = a, \pi\right]. \tag{4.3}$$

$$Q(s, a; \theta) = \mathbb{E}_{s'}\left[R + \lambda \max_{a' \in A} Q\left(s', a'\ ; \theta\right) | s, a\right]. \tag{4.4}$$

$$Q_{\tau+1}(s, a; \theta) = \mathbb{E}_{s'}\left[R + \lambda \max_{a' \in A} Q_\tau\left(s', a'\ ; \theta\right) | s, a\right], \tag{4.5}$$

In this proposed DRL model, a Deep Q-network with parameter θ is the approximate function and introduces fixation methods for oscillation mitigation and divergence prevention in training process. Specifically, there is a copy with fixed parameters θ^- from main network at the same intervals named target network of the approximator neural network (weights keep unchanged). The other copy called the primary network with copied weights to target network. The target network is considerably important to stabilize the training process. Unlike the standard deep learning training processes have labels of the training data, DRL models only have feedback from the environments, which indicate the performances of the decisions made by the DRL agent but not label data. In addition, the feedback from the environments could extremely noisy due to the environment and random actions taken by the agents. Therefore, the weights of target network are fixed for certain number of episodes to avoid noisy weights update for the Q-Network. The primary network is trained with data sampled from the memory buffer. Thus, through back propagation, only the main network is trained truly. (s, a, r, s') represents a special structure called experience replay. It is used for storing agent's every step. In the process of each network training iteration, a cluster of experience will be extracted from experience replay randomly for learning purpose. As an off-policy algorithm, Q-learning could learn from both past and current experience. In the process of learning, the neural network will be more efficient due to the random addition of previous experience, and the correlation between training samples will be broken down also. The loss function for updating DQN updates with iteration i is as below,

$$L_\tau(\theta_\tau) = \mathbb{E}_{(s,a,r,s')}\left[(F' - Q\left(s, a; \theta_\tau\right))^2\right] + \mathbb{E}_{s,a,r}\Big[\mathbb{V}_s[F']\Big]. \tag{4.6}$$

where θ_τ^- is the MSE of current action-value Q(s, a; θ_τ^-) and optimal value $Q^*(s', a')$. For minimizing the loss function, θ_τ^- is updated in previous iterations, and can be replaced with F'=$r + \lambda \max_{a'} Q^*\left(s', a'; \theta_\tau^-\right)$ which is fixation term.

By keeping interacting with the environments and receiving the feedback (rewards), the DRL agent can learn optimal decisions (i.e., for a given task, which edge server to offload the task, how to select the friendly jammer, and how to allocate CPU cycles at the edge servers) without knowing its internal transition to maximize the long-term rewards which

is the accumulative rewards over time. The reward function is defined as a weighted sum method for multi-objective optimization includes maximizing the number of completed tasks before deadlines and saving energy consumption, while satisfying the security requirement. The security rate is used in the data transmission phase from IoT devices to the edge servers to calculate the transmission time. According to the physical meaning of security rate, the data size of the task divided by the security rate equals to the time needed to offload the data to the edge server securely without any information leakage to the eavesdropper. In this work, the security rate is improved by using a friendly jammer which is selected from one of the IoT users. Different weights are assigned to the number of completed tasks and energy consumption so that they can be converted to the same scale. The weights can be adjusted based on the interests of the operators or users in the system.

4.1.1 Data Prepossessing

In this proposed DRL method, the DRL agent does not require any dataset prior to training the model, which is different from supervised learning. The agent learns to make optimal decisions through interacting with the MEC network environment and gain some experience. Those experiences will be put into a so-called replay buffer. The replay buffer is used to train the neural network. Those noisy and complicated raw data would increase the training effort for this DRL model due to the dynamic raw features such as users' tasks, channels and servers' workload. Raw data (real world data) is always incomplete and cannot be sent through a model.

To deal with the above presence of unformatted real-world data issues, data prepossessing is necessary and critical. Without data prepossessing, the essential features with raw data may be overlooked by DRL agent, and optimal or non-optimal solutions may be converged too slowly. Our data has variable scales, and its feature distribution does not follow a Gaussian (bell curve) distribution. In order to improve the accuracy and performance of our model, we adopt a normalization method to concatenate and re-scale the features in a desirable format in this work. Consisting of hierarchical components, the features are stored in a tree-like data structure. Specifically, there is a root node, and features from MEC servers are presented by a branch with three sub-branches such as states of edge server M_τ, secure transfer rate matrix r_τ^{sec}, and tasks T_τ. Further, every sub-branch involves several

components in leaf-level that should be normalized and concatenated together.

First, the Frobenius norm (Eq.4.7) is computed as a matrix norm of an $m \times n$ matrix A, and determined as a square root of the sum of the absolute squares of its elements. It is for all leaf-level components in feature $\mathbb{R}^{m \times n}$.

Then, in an element-wise manner, the normal is divided for computing the matrix $\mathbb{R}_N^{m \times n}$ (Eq.4.8) normalization.

Finally, a single feature from the concatenating of all sub-features is used to feed the learning model.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}, a_{ij} \in \mathbb{R}^{m \times n} \tag{4.7}$$

$$\mathbb{R}_N^{m \times n} = \frac{\mathbb{R}^{m \times n}}{\|A\|_F}. \tag{4.8}$$

4.1.2 Training Process and Implementation Details

This subsection presents the DRL model training process with training process algorithm shown in Fig 4.4 for the proposed DRL method to accomplish maximizing the cumulative reward (achieve multiple objectives).

In our DRL model, as neural networks are used, RL is more unstable to represent the action-values. Moreover, a lot of data is required for the network training, and converging on the optimal value function is still not guaranteed. Due to there is high correlation between actions and states, the network weights have oscillation or divergence risks.

For addressing above issues, we will introduce two techniques in this section: Experience Replay and Target Network in DQN:

Experience Replay and Experience Replay Buffer: Q(s, a), a nonlinear function with a neural network, is the function we are trying to approximate. Therefore, targets should be calculated using Bellman equation. However, the independent and identically distributed training data is one of the key requirements of Stochastic gradient descent

(SGD) optimization. When the agent interacts with environment and data is fed into the model sequentially, the experience tuples sequence can have high correlation. The effects of the high-correlation may affect and vacillate the Q-learning algorithm when it learns from each of those experiences tuples in a sequential order.

The experience replay is used to store the past experiences, and then the Q-network is updated by a subset of those experiences instead of just considering the single most-recent experience. By using these past experiences and sample training data, we can prevent action values from calamitous oscillation or divergence. This large buffer for storing is called experience buffer or replay buffer. A number of experience tuples are collected in this buffer and denoted by (S, A, R, S'). During the MEC ENV interaction, these tuples are added into the buffer gradually. When the buffer with fixed size is full, the new data is added to the buffer end, and the oldest data is deleted from buffer in order to make room for later new ones. The training data is drawn uniform randomly from experiences buffer. For learning purposes, as an act of sampling a small batch of tuples from the replay buffer, experience replay makes model to sample multiple data samples for leveraging batch normalization and reducing the swaying.

The actual loss function with random samples and experiences replay is shown as

$$L_\tau(\theta_\tau) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\Big[\Big(F' - Q(s,a;\theta_\tau)\Big)^2\Big]. \tag{4.9}$$

There are several benefits that experience replay and experience replay buffer brings to the training process. First, not only break harmful correlation down, experience replay also make it available for learning more from the individual tuples with multiple times. Secondly, it can recall rare occurrences for model get rid of over fitting and the affect from bias of training sample distribution, and it could make the use of experience better in general. Thirdly, instead of feeding the sample one by one, the sampled training data from the experience buffer could decouple the sequential order correlation. Third, by using batch samples, oscillation or divergence could be mitigated.

Target Networks: In Q learning, updating guess by guess may cause harmful correlations potentially. Although, the value of Q(s, a) is provided by Bellman equation via Q(s',

a'). There is only one step between the states s and s', and it is hard for a Neural Network to distinguish between s and s' due to their similarity. In order to make Q(s, a) closer to the desired result, the Neural Networks' parameters are updated gradually. Actually, other nearby states and the value produced for Q(s', a') can modified indirectly. However, this could lead to unstable training.

To solve this problem, the stability of the training could be enhanced by employing the Q values of target network for main Q-network training. By target network, primary network is used to Q(s', a') value in the Bellman equation to back-propagate through main Q network training. The parameters of the target network are synchronous with main Q-network parameters regularly. Note that they are not trained.

Algorithm 2: Deep-Q Learning Training Algorithm

Input: $epoch\_no, \epsilon_{start}, \epsilon_{end}$
Output: $loss, gradients$
//1. Initialization:
Initialize replay memory D with capacity N;
Initialize action-value $Q(s, a; \theta)$ with random weights θ;
Initialize target action-value Q' weights $\theta^- \leftarrow \theta$;
Initialize scores with window size;
$\epsilon \leftarrow \epsilon_{start}$;
for *episode* $\leftarrow$ *1 to max episode* M **do**
  Initialize input raw data x_1;
  Prepossess initial state: $S \leftarrow \delta(< x_1 >)$;
  for *time step:* $\tau \leftarrow 1$ *to* T_{max} **do**
    // 2. Training data Generating:
    Select action A from state S: $\pi \leftarrow \epsilon - Greedy(Q'(S, A, \theta))$;
    Take action A, observe reward R, get next input $s_{\tau+1}$;
    Prepossessing next state: $S' \leftarrow \delta(s_{\tau+1})$;
    Store experience tuple (S, A, R, S') in replay memory D;
    $S' \leftarrow S$;
    // 3. Learning:
    Obtain mini-batch of (s_j, a_j, r_j, s_{j+1}) from D randomly;
    if *episode terminate at step* $j + 1$ **then**
      Set target $F'_j \leftarrow r_j$;
    else
      Set target $F'_j \leftarrow r_j + \lambda \max_{a'} Q'(s_j, a, \theta^-)$;
    Update: $\theta \leftarrow \theta + \alpha \nabla_{\theta_j} L(\theta_j)$ with $Adam$;
    Every $\mathcal{N}$ steps, update: $\theta^- \leftarrow \theta$;
  $\epsilon \leftarrow max(\epsilon_{end}, \epsilon * decay)$;
  Store score for current episode;

FIGURE 4.4: Deep Q-Learning Training Algorithm

There are two main phases (not directly dependent on each other) that are interleaved in my proposed algorithm. First, environment is sampled by actions taken, and then a replay memory D stores the observed experienced tuples. Second, D selects a small batch

of tuples randomly, and a SGD update step is used for learning from that batch. Multi-sampling steps and one/multi-learning steps could be performed with different batches. In practice, the learning step could not be run immediately untill D stores enough experiences tuples. Simplify, the process starts with an initialization and preparation. Then, the agent fully explores environment and training data is generated. Finally, it moves to the learning process.

Initialization

From the Fig 4.4, it can be seen that the initializing of an empty replay memory D and the exploration proportion ϵ, and also neural networks (i.e., the target and learning network) establishment are required at the beginning of algorithm. When the DRL agent interacts with MEC network environment, all experiences obtained by agent are stored in replay buffer. The replay memory D is finite, so a circular queue is used to retain the most recent experience tuples d. Also, a learning network with random weights is initialized and copied to target network. Then, in the algorithm, data is generated and a DRL model is trained over all episodes. An episode may end when there is at least one overloaded MEC servers (indication of a finish flag returned by MEC ENV), or at the time step which equal or over time threshold T_{max}. After each episode, the memory is not cleared out to recall and build batches of experiences from across episodes.

Data Collection

The DRL agent does not require any dataset prior to training the model. Instead, the training dataset is generated by the interaction between DRL agent and the MEC network environment. Concretely, by using $\epsilon - greedy$ method, DRL agent obtains the knowledge. In epsilon-greedy action selection, the agent uses both exploration to look for new options and exploitation to take advantage of prior knowledge. With a probability of ϵ, greedy actions are explored and produced by agent randomly. Then, the optimal action is taken most of the time with probability of $1 - \epsilon$. In each interaction, a tuple contains current state s_τ, action a_τ, reward r_τ and next state $\mathrm{s}_{\tau+1}$ are generated as data that be stored into the experience buffer for further learning purposes. Although the training samples are stored in queue-like memory buffer, they will be sampled randomly to reduce the sequential correlation during the training process.

Learning

Unlike training a conventional deep learning model requires the availability of a large amount of pre-existing human-created, labeled, or verified data, which is difficult to obtain for real-life scenarios, the motivates DRL technique lets the system learn by itself and solves sequential decision-making problems relies on "trial and error".

When a policy is executed in an environment, the experience replay buffer is used to store trajectories of experience. In the training process, replay buffer is to be solicited for a subset of the paths (a sequential subset) to "replay" the experience of agent. In the forward propagation firstly, a number of training samples are drawn by DRL model from the buffer and then fed to the local learning neural network and also the target network. Then, we compute the loss (Eq. 4.6) between the learning neural network Q values and target network Q values. During the learning process, the error estimated by the loss function is minimized by optimizing the weights θ. The standard neural network computes the loss function as the error between labels and outputs for evaluating how well algorithm model's dataset is. However, as evaluative feedback rather than true label data is learned by agent, the DRL model loss function is computed by the difference outputs from the learning network and target network. The partial derivative with respect to θ is computed as Eq. 4.10. The term $\mathbb{E}_{s,a,r}\Big[\mathbb{V}_s[F']\Big]$ in the loss function could be ignored when computing partial derivative as it does not depend on learning network parameters θ. Therefore, we got the simplified partial derivative as Eq. 4.11.

Further, the gradient of $L(\theta_\tau)$ could be derived by employing chain rule and replacing F' with $F' = r + \lambda \max_{a'} Q^*(s', a'; \theta_\tau^-)$. Therefore, we got the gradient of $L(\theta_\tau)$ as Eq. 4.12, where $\theta_\tau^- = \theta_{\tau-1}$.

In the back propagation, the parameters θ of local learning neural network are updated based on the received rewards to produce an output that matches the target Q values as close as possible. They are updated in Adam optimization function by the multiplication of learning rate α and partial derivative loss function about θ (Eq.4.13). We use the target network's predicted Q values to back propagate through and train the main Q-network for improving the stability of the training. Note that, the target network's parameters are not trained but are synchronous with the main Q-network's ones regularly. In every $\mathcal{N}$

episodes, updated parameters θ in learning network are replicated to the parameters in target network to overwrite the θ^{-}.

$$\nabla_{\theta_\tau} L\left(\theta_\tau\right) = \nabla_{\theta_\tau} \mathbb{E}_{s,a,r,s'}\left[\left(F' - Q\left(s,a;\theta_\tau\right)\right)^2\right] + \nabla_{\theta_\tau} \mathbb{E}_{s,a,r}\Big[\mathbb{V}_s[F']\Big]. \tag{4.10}$$

$$\nabla_{\theta_\tau} L\left(\theta_\tau\right) = \nabla_{\theta_\tau} \mathbb{E}_{s,a,r,s'}\left[\left(F' - Q\left(s,a;\theta_\tau\right)\right)^2\right]. \tag{4.11}$$

$$\nabla_{\theta_\tau} L\left(\theta_\tau\right) = \mathbb{E}_{s,a,r,s'}\Big[\Big(r + \lambda \max_{a'} Q^*\left(s,', a';\theta_\tau^-\right) - Q\left(s,a;\theta_\tau\right)\Big)\nabla_{\theta_\tau} Q\left(s,a;\theta_\tau\right)\Big]. \tag{4.12}$$

$$\theta \leftarrow Adam(\theta, \alpha \nabla_{\theta_\tau} L\left(\theta_\tau\right)). \tag{4.13}$$

Reward Clipping

As a common practice in DQN, rewards can be clipped for promoting the convergence and optimal policies generating successfully. While applying DQN to the complex and uncertain MEC network environment setting, where reward points are not on the same scale, the training becomes inefficient. Even with a small alter of feature, rewards obtained are significantly different. Also, due to the high variance in training samples' features, such as channel distributions, DRL model may be misrepresented back and forth. Converging to the optimal polices in the learning process may slow down or never happen. As a practical but straightforward technique, the clip can adress and mitigate above issues. With clips, min will replace the element in $\mathbb{R}^{m\times n}$ that less than it, and max will replace those ones greater than it (Eq.4.14). Clipping the rewards to lie in the [min, max] interval reduces the impact of extreme observations, so to make the model more robust. Also, this avoids large weight updates and allows our DQN to update parameters smoothly.

$$\mathbb{R}^{m\times n}_{clipped} = clip(\mathbb{R}^{m\times n}, min, max). \tag{4.14}$$

Chapter 5

Simulation Results

In this section, the simulation results are presented in detail to evaluate the performance of our proposed DRL model for secure computation offloading with multiple IoT users, multiple edge servers, and one eavesdropper in MEC. The simulation environment is mainly based on Python as the programming language. PyTorch (a deep learning library) and NumPy are chosen to build the DRL model to reduce implementation efforts. In this simulation, we compare our proposed DRL model with a classical algorithm, the greedy algorithm, canonical reinforcement learning, and the existing deep reinforcement learning. To construct the simulation as close as a real-world scenario, we simulate the MEC network elements including various IoT users, MEC servers, control agent and DRL agent as independent processors.

According to the the proposed DRL model architecture in the previous proposed methods part, there are two major parts in our simulation system which are the MEC network environment and DRL agent.

As shown in Fig. 5.1, the MEC ENV contains three components: IoT users (include friendly jammer), coordinator and MEC servers as local-based stations. Various states include signal channel distribution, CPU allocation and transmission rate distribution. First, as the task generator, each of IoT users creates multiple tasks with a random waiting time which is around 0.001 seconds. Second, the multiple MEC servers are responsible for status information maintaining, tasks processing and rewards computing. They are generated by

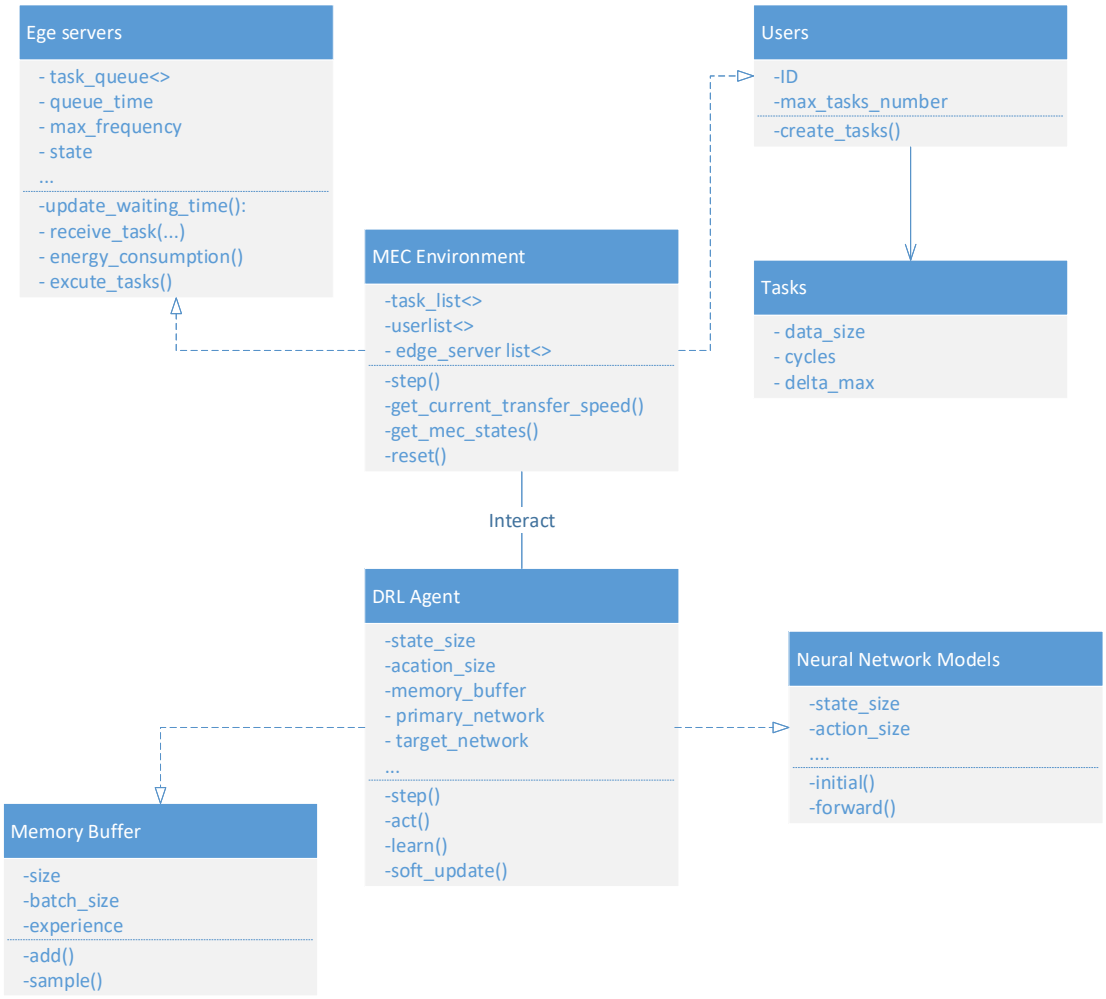

FIGURE 5.1: Summary of simulation UML

simulator based on parameter settings such as minimum and maximum frequencies, size of task queues, and overload thresholds. When the task queue is empty, MEC status is idle with minimum frequency. When a MEC server receives a task, it runs with recommended frequency. The reward includes the number of tasks finished before the deadline and consumed energy. Both of them can be computed based on required CPU cycles and this recommended frequency. The environment provides interface that allows the DRL agent to control the MEC networks and offloading the tasks. Specifically, the environment is the main process in the simulator, and it generates other processes that present MEC servers and users. The users create tasks given the configuration and put the task profiles into a queue maintained by the environment process so that the DRL agent can access those

information. Further, the DRL or other agents can take action according to the observed information which is also simulated by a process. Once the environment receives the agent's decision, the agent distributes the tasks according to recommended actions. In other words, the environment process transfers the tasks from its queue to the target servers with recommended frequencies. And then, the MEC servers start to process tasks from the tasks queue. However, the environment will be restarted and all of the task queues in the environment and MEC servers will be clean when one of the server are overloaded, which indicates end of an episode.

The DRL agent maintains the experience replay buffer. The DRL agent learns and generates the optimal policy and actions to interact with the MEC network environment through the coordinator and rewards it for every action it takes, then saves the result into the experience replay buffer. In this simulation, we construct the local learning network and the target network with three types of layers: input layer, hidden layer and output layer. Input layer works for data initialization for neural network; all the computation will be done in the hidden layer which is used as the inter-media between input layer and output layer; and the result of given inputs is produced in output layer. Each network has an input layer with the numbers of neurons that equal to the sizes of state space. The five hidden layers with the number of neurons are 256, 512, 512, 512 and 256 from layer one to layer five. For the output layer, the size of action space is equal to the numbers of neurons. We set the discount rate as 0.9 and learning rate as 5×10^{-4}; the IoT users number is considered as 500, the batch size is 256 and signal to noise ratio is 100 dB; the range of some other main parameters such as max frequency of server, data size and computing size of task are set as from 2×10^9 to 8×10^9 Hz, from 2×10^5 to 2×10^7 bits and from 8×10^6 to 1×10^7 cycles.

Fig. 5.2 shows the rewards performance of our proposed DRL (DQN-E2E) model, and other methods, including the existing DRL model (DQN-CVX), standard RL (Q-learning) model and traditional greedy algorithm. As we can see from the figure, the Q-learning converges faster than other DRL models because Q-learning method has to discretize continuous space into discrete Q-table, and it has required less episodes to explore to its optimal policy. The DQN-CVX model converges to optimal policy around $10,000$ episodes, which is relatively faster than end-to-end model because partial of the decisions are optimized CVX. Although the proposed model converges relative slower than other models because it has

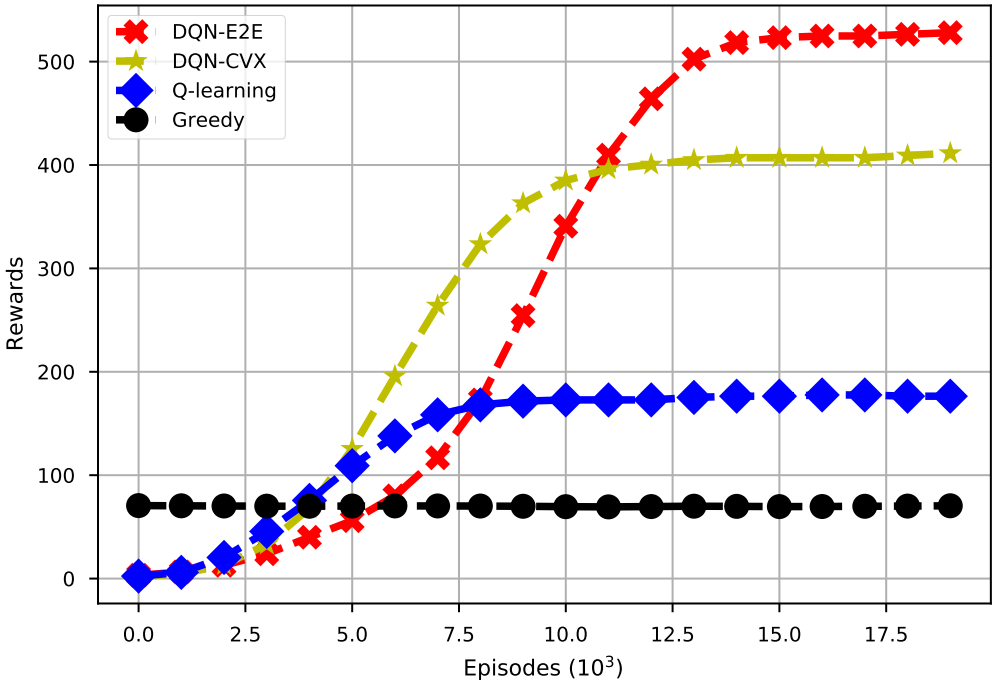

FIGURE 5.2: Rewards

to learn optimize all the decisions respect to maximize long-term expected reward, it can achieve the best performance. We can also observe all learning methods are performs poorer than conventional at the early episodes because they trail random actions to explore and collect training data. In contrast, the greedy algorithm does not learn over the episodes. Overall, compared with existing DRL models and other models, the end-to-end model we adopted in the proposed method has more freedom to take actions. Therefore, our proposed end-to-end DRL approach presents a much better performance and results.

Note that the completed task means the security transmission rate is greater than the pre-setting security threshold, considered safe enough to transmit the tasks to the targeted edge server as defined problem formulation (Eq.3.10). The security constraint holds throughout the simulation. For the sake of simplicity of the augment, we will not repeat the security constraint in the following analysis.

As shown in Fig. 5.3, with respect to the number of tasks completed, the proposed DRL model is compared with the greedy algorithm, RL and existing DRL model. In terms of

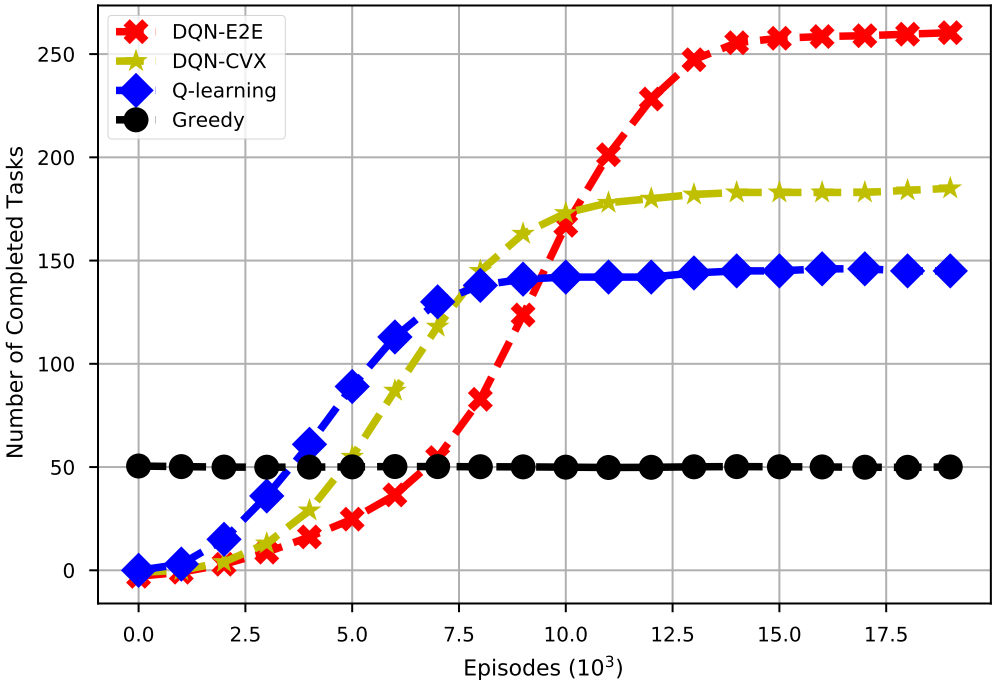

FIGURE 5.3: Completed Number of Tasks

the rewards, our proposed method outperforms the other models because the deep learning models can seize the complex MEC networks states and make decisions via RL. Due to the historical data which is kept in buffer is learned by the DRL models continuously, performance of DRL models and the Q-learning is improving until reach the convergence of optimal policies. Although the proposed end-to-end model takes the most episodes to learn and find its optimal polices, it can process the most tasks before the expired time while minimizing the energy consumption. Otherwise, the greedy algorithm keeps no change because it does not have a learning process to improve its performance over episodes. The Q-learning method has a limited capacity to hold the vast states of the MEC networks, and it only provides considerably low resolution to capture the states and learn to optimize the policy; therefore, it is outperformed by the DRL models.

For energy consumption, the Q-learning and DRL models have more energy cost in the early episodes due to the $\epsilon - greedy$ which model take actions with and parameters are initialized randomly in those episodes (shown in Fig. 5.4). As the learning models learn from the acquired data and decrease random actions, it can reduce more energy

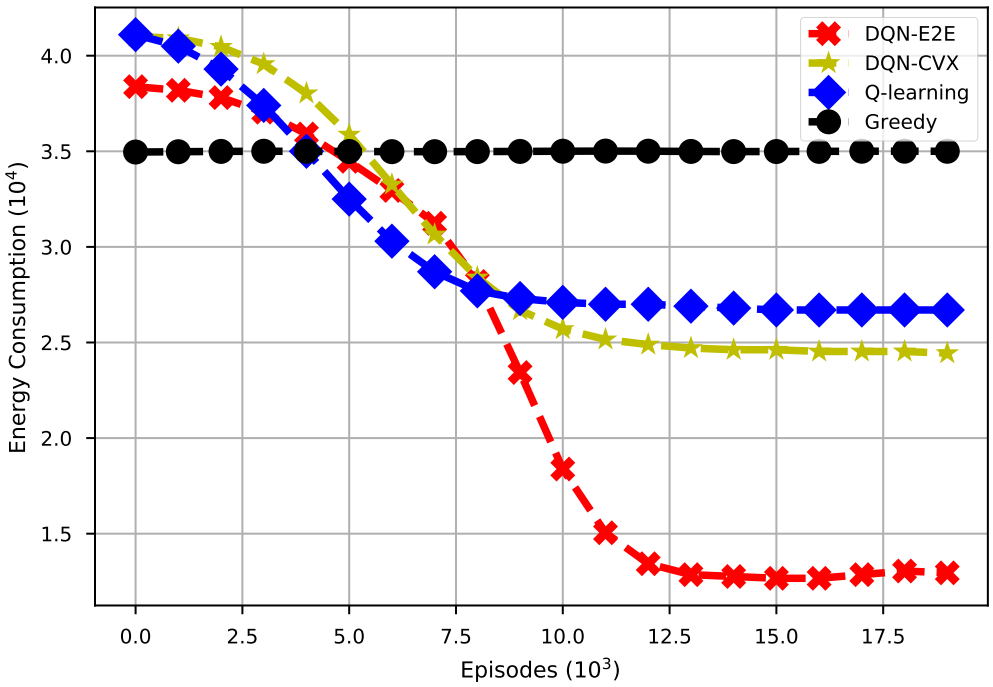

FIGURE 5.4: Energy Consumption.

consumption. Although the goal of the proposed DRL model is maximizing long-term accumulated rewards, this model also learns how to reduce the energy consumption over all time steps. The number of tasks can also be increased by the DRL model with CVX significantly. However, it does not save energy consumption as much as the proposed model due to only part of its actions space can be controlled, and the CVX optimization methods optimize the other parts of the actions space.

Similarly, Fig. 5.5 presents the completed tasks number and energy cost ratio, and the results are very similar to the rewards shown in Fig. 5.2 or completed number of tasks in Fig. 5.3. However, we can see that DQN-CVX outperforms the Q-learning and proposed methods because the CVX optimization starts work in the early episodes and does not learn over the episodes. As we can see from Fig. 5.5, the DQN models can use the energy more efficiently than traditional RL models, and all the learning models can do save more energy than the greedy algorithm. In Fig. 5.5, we can see the proposed model can complete a greater number of tasks than the rest of the models when consuming the same amount of energy because the end-to-end model can access and control over the optimization process

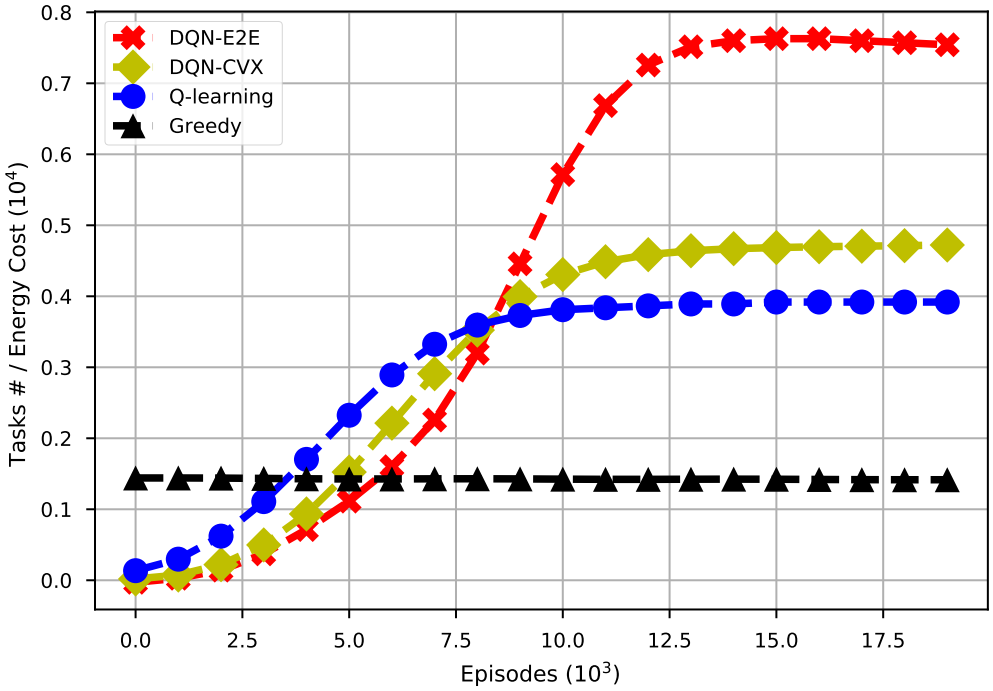

FIGURE 5.5: Ratio of Completed Tasks Number to Energy Cost (10^4)

so that it can optimize the decision process.

Further, to compare the capacity of the model, Fig. 5.6 shows the performance of the model with respect to the number of users. The learning method can complete more tasks before expiration than the greedy algorithm. The DRL model can perform better than the Q-learning model because it can capture the MEC network states and make better decisions than the Q-learning model when more tasks are offloaded quickly. The MEC networks always have limited resources compared to the resource demanded from the IoT users. In addition, the learning agents need to choose targeted server according to the task information and security status in the network, which pose more restrictions on exploiting the limited resource. Fortunately, the proposed method can choose optimal polices that can use limited resources maximally and serve maximum number of users without delay.

Finally, we compared the proposed model with configuration different values of exploration. Fig. 5.7 indicates that when the ϵ is smaller, there is less exploitation but faster convergence to local optimal for the model. In this simulation, we can see that the model

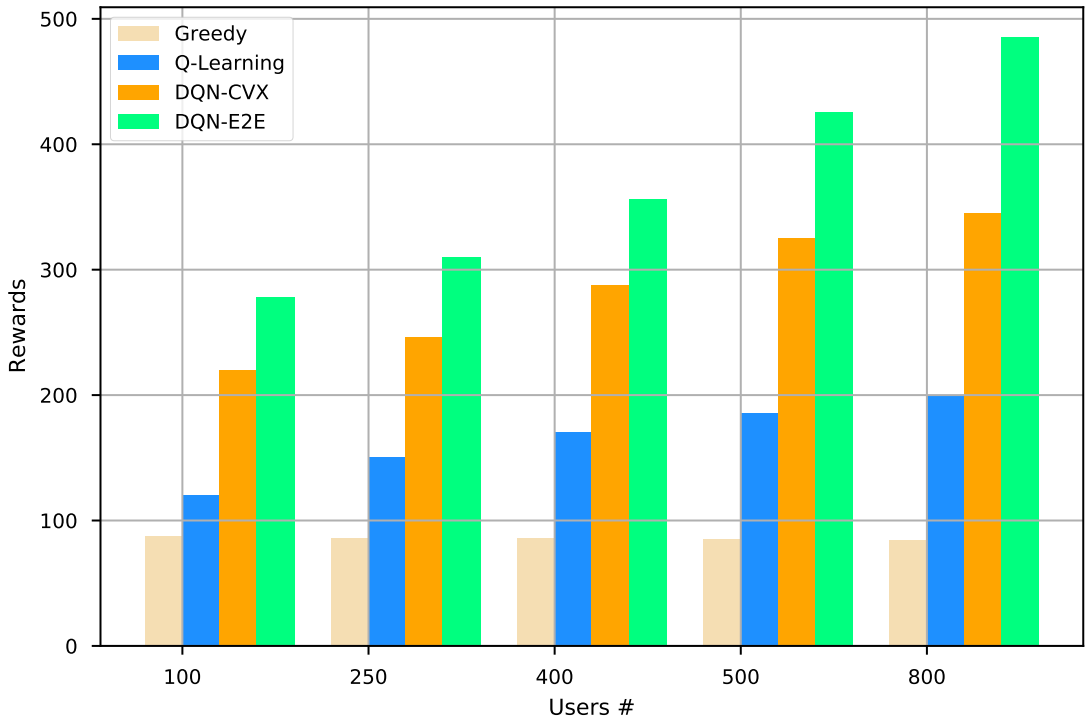

FIGURE 5.6: Rewards Increasing Respect to Users Number

catch the highest performance at $\epsilon = 0.99$. On the contrary, the model ends up with the lowest rewards when $\epsilon = 0.7$. These as results of with larger ϵ values, model should take more time for environment exploration compared with other ones with smaller ϵ. However, when ϵ is larger, it is not the best choice usually. This is because more exploration may take longer time for convergence to the optimal policies, and more computational power may be cost also. For conclusion, $\epsilon = 0.99$ is chosen in our work due to its best rewards collection performance.

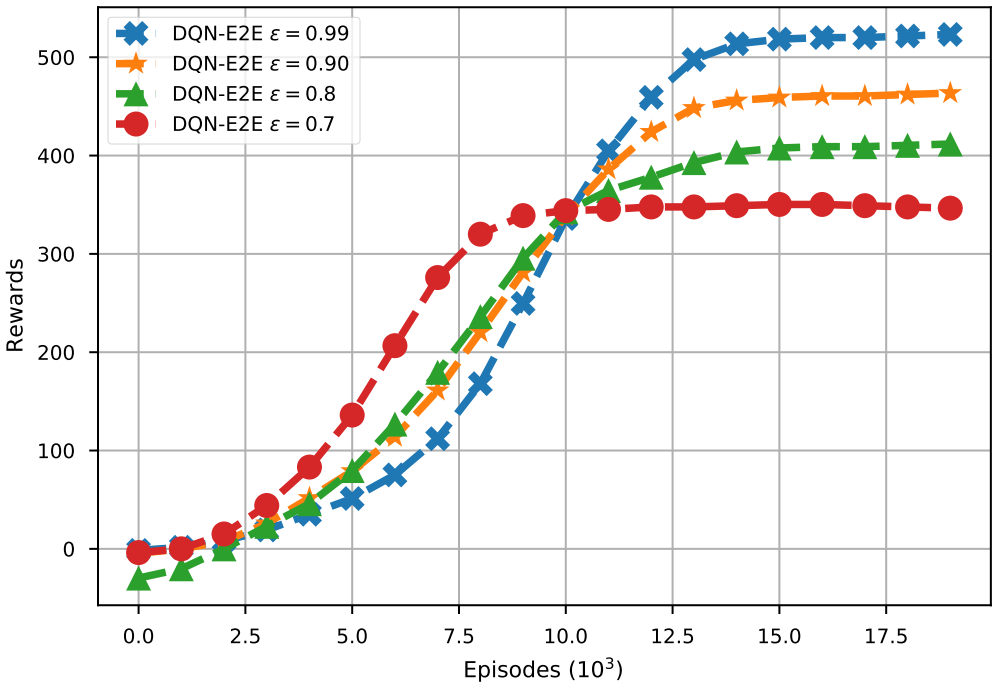

FIGURE 5.7: Proposed Model with Different Exploration Ratio (ϵ)

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work, we investigated the computation offloading problem in a dynamic MEC network in a secure and timely manner. An end-to-end deep reinforcement learning method was proposed to optimize multiply objectives such as number of completed tasks maximization before their deadlines and saving energy consumption, subject to the security transmit rate requirements. The proposed developed method focuses on maximizing the accumulated long-term rewards. To prevent the DRL model from being oscillated and diverged, experience replay with buffer and clip techniques are adopted to boost the training process in model. Finally, simulation results are provided, which demonstrate the proposed DRL method is effective and outperforms other methods.

6.2 Future Work

In the future work, two aspects of task offloading will be considered: (1). Green energy powered MEC systems. The MEC servers are powered by renewable energy. Due to the randomness of energy harvesting and task arrivals, it is crucial to study offloading in such a system to ensure service quality and make efficient use of renewable energy. (2). Task partitioning in dynamic MEC systems. A computation task can be partitioned into multi-

independent sub-tasks and each sub-task can be offloaded to an edge server for processing. In a multi-edge server environment, it is necessary to study how to partition the tasks and coordinate edgers to meet the service requirements efficiently.

Bibliography

[1] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa. Fogflow: Easy programming of iot services over cloud and edges for smart cities. *IEEE Internet of Things Journal*, 5(2):696–707, 2018.

[2] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris. A review of machine learning and iot in smart transportation. *Future Internet*, 11(4):94, 2019.

[3] J. Ren, Y. Zhang, N. Zhang, D. Zhang, and X. Shen. Dynamic channel access to improve energy efficiency in cognitive radio sensor networks. *IEEE Transactions on Wireless Communications*, 15(5):3143–3156, 2016.

[4] Y. Dong and Y. Yao. Secure mmwave-radar-based speaker verification for iot smart home. *IEEE Internet of Things Journal*, 8(5):3500–3511, 2020.

[5] S. Muthuramalingam, A. Bharathi, N. Gayathri, R. Sathiyaraj, B. Balamurugan, et al. Iot based intelligent transportation system (iot-its) for global perspective: A case study. In *Internet of Things and Big Data Analytics for Smart Generation*, pages 279–300. Springer, 2019.

[6] N. Lu, N. Zhang, N. Cheng, X. Shen, J. W. Mark, and F. Bai. Vehicles meet infrastructure: Toward capacity–cost tradeoffs for vehicular access networks. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1266–1277, 2013.

[7] E. J. Ghomi, A. M. Rahmani, and N. N. Qader. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*, 88:50–71, 2017.

[8] J. Zhao, Q. Li, Y. Gong, and K. Zhang. Computation offloading and resource allocation

for cloud assisted mobile edge computing in vehicular networks. *IEEE Transactions on Vehicular Technology*, 68(8):7944–7956, 2019.

[9] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[10] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[11] J. Feng, Q. Pei, F. R. Yu, X. Chu, and B. Shang. Computation offloading and resource allocation for wireless powered mobile edge computing with latency constraint. *IEEE Wireless Communications Letters*, 8(5):1320–1323, 2019.

[12] J. Wang, D. Feng, S. Zhang, J. Tang, and T. QS Quek. Computation offloading for mobile edge computing enabled vehicular networks. *IEEE Access*, 7:62624–62632, 2019.

[13] M. Zeng, W. Hao, O. A Dobre, Z. Ding, and H V. Poor. Massive mimo-assisted mobile edge computing: Exciting possibilities for computation offloading. *IEEE Vehicular Technology Magazine*, 15(2):31–38, 2020.

[14] B. Yang, X. Cao, J. Bassey, X. Li, T. Kroecker, and L. Qian. Computation offloading in multi-access edge computing networks: A multi-task learning approach. pages 1–6, 05 2019.

[15] Y. Mao, C. You, J. Zhang, K. Huang, and K. B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials*, 19(4):2322–2358, 2017.

[16] J. Zhang, S. Rajendran, Z. Sun, R. Woods, and L. Hanzo. Physical layer security for the internet of things: Authentication and key generation. *IEEE Wireless Communications*, 26(5):92–98, 2019.

[17] S. Atapattu, N. Ross, Y. Jing, and M. Premaratne. Source-based jamming for physical-layer security on untrusted full-duplex relay. *IEEE Communications Letters*, 23(5): 842–846, 2019.

[18] Y. Mao, J. Zhang, and K. B Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016.

[19] J. Ren, G. Yu, Y. Cai, and Y. He. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 17(8):5506–5519, 2018.

[20] O. Muñoz, A. Pascual-Iserte, and J. Vidal. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Transactions on Vehicular Technology*, 64(10):4738–4755, 2015.

[21] T. X Tran and D. Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, 68(1): 856–868, 2018.

[22] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2016.

[23] S. Sardellitti, G. Scutari, and S. Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[24] Q. Wang, S. Guo, J. Liu, and Y. Yang. Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing. *Sustainable Computing: Informatics and Systems*, 21:154–164, 2019. ISSN 2210-5379.

[25] C. You, K. Huang, H. Chae, and B. Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3): 1397–1411, 2017.

[26] H. Mazouzi, N. Achir, and K. Boussetta. Dm2-ecop: An efficient computation offloading policy for multi-user multi-cloudlet mobile edge computing environment. *ACM Trans. Internet Technol.*, 19(2), apr 2019. ISSN 1533-5399.

[27] H. Sun, F. Zhou, and R. Q. Hu. Joint offloading and computation energy efficiency maximization in a mobile edge computing system. *IEEE Transactions on Vehicular Technology*, 68(3):3052–3056, 2019.

[28] S. Guo, B. Xiao, Y. Yang, and Y. Yang. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[29] T. Q. Dinh, J. Tang, Q. D. La, and T. QS Quek. Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[30] M. Deng, H. Tian, and X. Lyu. Adaptive sequential offloading game for multi-cell mobile edge computing. In *2016 23rd international conference on telecommunications (ICT)*, pages 1–5. IEEE, 2016.

[31] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Transactions on Communications*, 64(10):4268–4282, 2016.

[32] S. Hong and H. Kim. Qoe-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2016.

[33] J. Zhang, X. Hu, Z. Ning, E. C-H Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu. Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. *IEEE Internet of Things Journal*, 5(4):2633–2645, 2017.

[34] L. Ismail and H. Materwala. Machine learning-based energy-aware offloading in edge-cloud vehicular networks. *Procedia Computer Science*, 191:328–336, 2021.

[35] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[36] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

[37] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In

D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[38] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[39] G. Bradski, A. Keahler, and V. Pisarevsky. Learning-based computer vision with intel's open source computer vision library. intel. *Technology Journal*, 9:119–130, 01 2005.

[40] R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML '08*, 2008.

[41] L. Ale, N. Zhang, H. Wu, D. Chen, and T. Han. Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network. *IEEE Internet of Things Journal*, 6(3):5520–5530, 2019.

[42] Y. Gong, C. Lv, S. Cao, L. Yan, and H. Wang. Deep learning-based computation offloading with energy and performance optimization. *EURASIP Journal on Wireless Communications and Networking*, 2020(1):1–8, 2020.

[43] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf. A deep learning approach for energy efficient computational offloading in mobile edge computing. *IEEE Access*, 7:149623–149633, 2019.

[44] S. Yu, X. Wang, and R. Langar. Computation offloading for mobile edge computing: A deep learning approach. *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6, 2017.

[45] M. Arif, F. Ajesh, S. Shamsudheen, and M. Shahzad. Secure and energy-efficient computational offloading using lstm in mobile edge computing. *Security and Communication Networks*, 2022:13, 2022.

[46] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[47] B. Dab, N. Aitsaadi, and R. Langar. Q-learning algorithm for joint computation offloading and resource allocation in edge cloud. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 45–52. IEEE, 2019.

[48] D. Wang, X. Tian, H. Cui, and Z. Liu. Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware mec network. *China Communications*, 17(8):31–44, 2020.

[49] S. Pan, Z. Zhang, Z. Zhang, and D. Zeng. Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach. *IEEE Access*, 7:134742–134753, 2019.

[50] H. Liu, S. Liu, and K. Zheng. A reinforcement learning-based resource allocation scheme for cloud robotics. *IEEE Access*, 6:17215–17222, 2018.

[51] F R. Yu and Y. He. Deep reinforcement learning for wireless networks. 2019.

[52] W. Zhang, Z. Zhang, S. Zeadally, H. Chao, and V. C. M. Leung. Masm: A multiple-algorithm service model for energy-delay optimization in edge artificial intelligence. *IEEE Transactions on Industrial Informatics*, 15(7):4216–4224, 2019.

[53] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang. Learning-aided computation offloading for trusted collaborative mobile edge computing. *IEEE Transactions on Mobile Computing*, 19(12):2833–2849, 2020.

[54] D. Wang, H. Qin, B. Song, X. Du, and M. Guizani. Resource allocation in information-centric wireless networking with d2d-enabled mec: A deep reinforcement learning approach. *IEEE Access*, 7:114935–114944, 2019.

[55] Y. Zhan, S. Guo, P. Li, and J. Zhang. A deep reinforcement learning based offloading game in edge computing. *IEEE Transactions on Computers*, 69(6):883–893, 2020.

[56] L. Huang, S. Bi, and Y. A. Zhang. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing*, 19(11):2581–2593, 2020.

[57] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. *IEEE Access*, 8:54074–54084, 2020.

[58] Z. Ning, P. Dong, X. Wang, L. Guo, J. J. Rodrigues, X. Kong, J. Huang, and R. Y. Kwok. Deep reinforcement learning for intelligent internet of vehicles: An energy-

efficient computational offloading scheme. *IEEE Transactions on Cognitive Communications and Networking*, 5(4):1060–1072, 2019.

[59] J. Li, X. Zhang, J. Zhang, J. Wu, Q. Sun, and Y. Xie. Deep reinforcement learning-based mobility-aware robust proactive resource allocation in heterogeneous networks. *IEEE Transactions on Cognitive Communications and Networking*, 6(1):408–421, 2019.

[60] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu. iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks. *IEEE Internet of Things Journal*, 6(4):7011–7024, 2019.

[61] L. Huang, S. Bi, and Y. A. Zhang. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing*, 19(11):2581–2593, 2019.

[62] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung. Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing. *IEEE Internet of Things Journal*, 9(2):1517–1530, 2022.

[63] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang. Edge intelligence for energy-efficient computation offloading and resource allocation in 5g beyond. *IEEE Transactions on Vehicular Technology*, 69(10):12175–12186, 2020.

[64] L. Qian, Y. Wu, F. Jiang, N. Yu, W. Lu, and B. Lin. Noma assisted multi-task multi-access mobile edge computing via deep reinforcement learning for industrial internet of things. *IEEE Transactions on Industrial Informatics*, 17(8):5688–5698, 2021.

[65] J. Li, H. Gao, T. Lv, and Y. Lu. Deep reinforcement learning based computation offloading and resource allocation for mec. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2018.

[66] J. Yan, S. Bi, and Y. J. A. Zhang. Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 19(8):5404–5419, 2020.

[67] J. Liu, Y. Mao, J. Zhang, and K. B Letaief. Delay-optimal computation task scheduling for mobile-edge computing systems. In *2016 IEEE international symposium on information theory (ISIT)*, pages 1451–1455. IEEE, 2016.

[68] Y. Wu, L. P. Qian, K. Ni, C. Zhang, and X. Shen. Delay-minimization nonorthogonal multiple access enabled multi-user mobile edge computation offloading. *IEEE Journal of Selected Topics in Signal Processing*, 13(3):392–407, 2019.

[69] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen. Toffee: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing. *IEEE Transactions on Cloud Computing*, 9(4):1634–1644, 2019.

[70] S. Mao, S. Leng, S. Maharjan, and Y. Zhang. Energy efficiency and delay tradeoff for wireless powered mobile-edge computing systems with multi-access schemes. *IEEE Transactions on Wireless Communications*, 19(3):1855–1867, 2020.

[71] A. Samanta and Z. Chang. Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint. *IEEE Internet of Things Journal*, 6(2): 3864–3872, 2019.

[72] L. Rui, Y. Yang, Z. Gao, and X. Qiu. Computation offloading in a mobile edge communication network: A joint transmission delay and energy consumption dynamic awareness mechanism. *IEEE Internet of Things Journal*, 6(6):10546–10559, 2019.

[73] B. Li, W. Wu, W. Zhao, and H. Zhang. Security enhancement with a hybrid cooperative noma scheme for mec system. *IEEE Transactions on Vehicular Technology*, 70(3):2635–2648, 2021.

[74] X. He, R. Jin, and H. Dai. Physical-layer assisted secure offloading in mobile-edge computing. *IEEE Transactions on Wireless Communications*, 19(6):4054–4066, 2020.

[75] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji, and Y. Zhang. Reinforcement learning-based mobile offloading for edge computing against jamming and interference. *IEEE Transactions on Communications*, 68(10):6114–6126, 2020.

[76] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu. Data security and privacy-preserving in edge computing paradigm: Survey and open issues. *IEEE access*, 6:18209–18237, 2018.

[77] F. De Rango, G. Potrino, M. Tropea, and P. Fazio. Energy-aware dynamic internet of things security system based on elliptic curve cryptography and message queue teleme-

try transport protocol for mitigating replay attacks. *Pervasive and Mobile Computing*, 61:101105, 2020.

[78] A. Castiglione, M. Nappi, F. Narducci, and C. Pero. Fostering secure cross-layer collaborative communications by means of covert channels in mec environments. *Computer Communications*, 169:211–219, 2021.

[79] Z. Ding, P. Fan, and H V. Poor. Impact of non-orthogonal multiple access on the offloading of mobile edge computing. *IEEE Transactions on Communications*, 67(1): 375–390, 2018.

[80] A. D Wyner. The wire-tap channel. *Bell system technical journal*, 54(8):1355–1387, 1975.

[81] L. Qing, H. Guangyao, and F. Xiaomei. Physical layer security in multi-hop af relay network based on compressed sensing. *IEEE Communications Letters*, 22(9):1882–1885, 2018.

[82] M. Kamel, W. Hamouda, and A. Youssef. Physical layer security in ultra-dense networks. *IEEE Wireless Communications Letters*, 6(5):690–693, 2017.

[83] G. Zheng, I. Krikidis, J. Li, A. P Petropulu, and B. Ottersten. Improving physical layer secrecy using full-duplex jamming receivers. *IEEE Transactions on Signal Processing*, 61(20):4962–4974, 2013.

[84] R. Zhao, X. Tan, D. Chen, Y. He, and Z. Ding. Secrecy performance of untrusted relay systems with a full-duplex jamming destination. *IEEE Transactions on Vehicular Technology*, 67(12):11511–11524, 2018.

[85] J. Xu and J. Yao. Exploiting physical-layer security for multiuser multicarrier computation offloading. *IEEE Wireless Communications Letters*, 8(1):9–12, 2018.

[86] Y. Wu, J. Shi, K. Ni, L. Qian, W. Zhu, Z. Shi, and L. Meng. Secrecy-based delay-aware computation offloading via mobile edge computing for internet of things. *IEEE Internet of Things Journal*, 6(3):4201–4213, 2018.

Vita Auctoris

NAME: Xue Qin

PLACE OF BIRTH: Shanxi, China

YEAR OF BIRTH: 1985

EDUCATION: 2004-2008

North University of China

Taiyuan, Shanxi, China

2019-2020

Texas A&M University-Corpus Christi

Corpus Christi, TX, USA