TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

THESIS NO: PUL075MSCSK019

Self managed cloud computing and edge computing system using deep reinforcement learning

by

Sushil Shakya

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL

September, 2021

Self managed cloud computing and edge computing system using deep reinforcement learning

by

Sushil Shakya

Thesis Supervisor

Prof. Dr. Subarna Shakya

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer System and Knowledge Engineering

Department of Electronics and Computer Engineering

Institute of Engineering, Pulchowk Campus

Tribhuvan University

Lalitpur, Nepal

September, 2021

COPYRIGHT©

The author has agreed that the library, Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk Campus, may make this thesis freely available for inspection. Moreover the author has agreed that the permission for extensive copying of this thesis work for scholarly purpose may be granted by the professor(s), who supervised the thesis work recorded herein or, in their absence, by the Head of the Department, wherein this thesis was done. It is understood that the recognition will be given to the author of this thesis and to the Department of Electronics and Computer Engineering, Pulchowk Campus in any use of the material of this thesis. Copying of publication or other use of this thesis for financial gain without approval of the Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk Campus and author's written permission is prohibited.

Request for permission to copy or to make any use of the material in this thesis in whole or part should be addressed to:

Head
Department of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus
Pulchowk, Lalitpur, Nepal

DECLARATION

I declare that the work hereby submitted for Master of Science in Computer System and Knowledge Engineering(MSCSK) at IOE, Pulchowk Campus entitled **Self managed cloud computing and edge computing system using deep reinforcement learning** is my own work and has not been previously submitted by me at any university for any academic award.

I authorize IOE, Pulchowk Campus to lend this thesis to other institution or individuals for the purpose of scholarly research.

Sushil Shakya

PUL075MSCSK019

Date: 2021-09-01

RECOMMENDATION

The undersigned certify that they have read and recommended to the Department of Electronics and Computer Engineering for acceptance, a thesis entitled **Self managed cloud computing and edge computing system using deep reinforcement learning**, submitted by **Sushil Shakya** in partial fulfillment of the requirement for the award of the degree of **"Master of Science in Computer System and Knowledge Engineering"**.

....

Supervisor: Prof. Dr. Subarna Shakya,
Department of Electronics and Computer Engineering,
Institute of Engineering, Tribhuvan University

....

External Examiner: Dr. Pradip Paudyal,
Assistant Director,
Nepal Telecommunications Authority (NTA)

....

Committee Chairperson: Associate. Prof. Dr. Nanda Bikram Adhikari,
Department of Electronics and Computer Engineering,
Institute of Engineering, Tribhuvan University

Date:

DEPARTMENTAL ACCEPTANCE

The thesis entitled **Self managed cloud computing and edge computing system using deep reinforcement learning**, submitted by **Sushil Shakya** in partial fulfillment of the requirement for the award of the degree of **“Master of Science in Computer System and Knowledge Engineering”** has been accepted as a bonafide record of work independently carried out by him in the department.

....

Dr. Ram Krishna Maharjan
Head of Department,
Department of Electronics and Computer Engineering.
Institute of Engineering,
Tribhuvan University,
Nepal.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Prof. Dr. Subarna Shakya for his guidance and support throughout the conception, experimentation and conclusion of this research work. I would also like to thank Associate. Prof. Dr. Nanda Bikram Adhikari, Dr. Aman Shakya, Dr. Basanta Joshi, Dr. Pradip Paudyal and all the faculty members of the Department of Electronics and Computer Engineering for their valuable comments and suggestions for the improvement of this research.

Sincerely,
Sushil Shakya (PUL075MSCSK019)

ABSTRACT

In order to tackle the latency problems in sensitive applications implemented in IoT devices, the edge computing came into existence. On top of that, the idea of a mobile edge computing (MEC) network, in which such computationally heavy activities are computed in many edge servers deployed adjacent to mobile devices, has recently gained popularity. Unlike cloud server, the edge device has a finite computation capacity and it can't handle massive computation tasks. There needs to be a smart task offloading scheme in order to decide whether to execute the task in the local device itself, or offload it to the edge device and process there or again send it further to the cloud server for processing. Also the algorithm should be able to utilize the available network bandwidth efficiently. The optimization of joint offloading decision and bandwidth allocation in a multi user, multi task, multi server environment can be formulated as a mixed integer non linear programming problem (MINLP) in MEC to preserve energy and maintain quality of service for wireless devices. MINLP is a NP-hard problem and the time complexity to solve it grows exponentially. In this research work, the power of deep learning and reinforcement learning has been applied to solve this MINLP problem under a fraction of a second which makes it suitable for real world usage. Further an end to end integrated edge and cloud computing system has been proposed that switches from one to another whenever required, and leverages the benefits of both paradigm.

Keywords: Edge computing, Cloud computing, Deep learning, Reinforcement learning

TABLE OF CONTENTS

COPYRIGHT iii

DECLARATION iv

RECOMMENDATION v

DEPARTMENTAL ACCEPTANCE vi

ACKNOWLEDGEMENT vii

ABSTRACT viii

TABLE OF CONTENTS ix

LIST OF FIGURES xi

LIST OF TABLES xii

LIST OF ALGORITHMS xiii

LIST OF ABBREVIATIONS xiv

1 INTRODUCTION 1

1.1 Background and Motivation 1

1.2 Problem Definition 2

1.3 Objectives 3

1.4 Scope of the Work 3

1.5 Originality of this Work 3

1.6 Organisation of thesis work 4

2 LITERATURE REVIEW 5

3 METHODOLOGY 8

3.1 Preliminary Work 8

3.2 Data Collection 9

3.3 Data Analysis 10
3.4 Implementation 10
3.4.1 Local Computing 12
3.4.2 Edge Computing 13
3.4.3 Solving the optimization problem 15
3.4.4 Algorithm 16
3.4.5 Edge server to cloud data center offloading 18

4 RESULTS, ANALYSIS AND COMPARISON 21
4.1 Experimental Setup 21
4.2 Training in multi user, multi task, single server environment 22
4.3 Execution Time under different number of DNNs 26
4.4 Training in multi user, multi task, multi server environment 27
4.5 Comparison of DiDQN with Local, Edge Processing and DDLO model 28

5 CONCLUSION AND RECOMMENDATION 30
5.1 Conclusion 30
5.2 Limitation 30
5.3 Recommendation 31

REFERENCES 34

A TURNITIN PLAGIARISM REPORT 35

LIST OF FIGURES

3.1.1 Task distribution and power management in cloud data center [1] 8
3.3.2 User workload size distribution 10
3.4.3 Multi user multi task multi server MEC network architecture . . . 11
3.4.4 Distributed Deep Q-learning Network training architecture 17

4.2.1 Gain ratio under different number of DNNs [N=3, M=3, K=1] . . 22
4.2.2 Training loss [N=3, M=3, K=1, Z=3] 23
4.2.3 Gain ratio [N=3, M=3, K=1, Z=3 with different learning rates] . 23
4.2.4 Gain ratio [N=3, M=3, K=1, Z=3 with different memory sizes] . . 24
4.2.5 Gain ratio [N=3, M=3, K=1, Z=3 with different batch sizes] . . . 25
4.2.6 Gain ratio [N=3, M=3, K=1, Z=3 with different train intervals] . 25
4.4.7 Gain ratio under different number of DNNs [N=3, M=3, K=3] . . 27
4.4.8 Training loss [N=3, M=3, K=3, Z=10] 28
4.5.9 Gain ratio from different models [N=3, M=3, K=1, Z=3] 28
4.5.10 Gain ratio from didqn and ddlo models [N=3, M=3, K=1] 29

A.0.1 Similarity index report from Turnitin 35

LIST OF TABLES

3.1 Task offloading RL agent description 11
3.2 Notations Table 15

4.1 AWS EC2 Server Specification 21
4.2 Parameter Values used in the simulation 21
4.3 Gain ratio on test set under different learning rates 24
4.4 Gain ratio on test set under different memory sizes 24
4.5 Gain ratio on test set under different batch sizes 25
4.6 Gain ratio on test set under different train intervals 26
4.7 Best Hyperparameter values 26
4.8 Execution time under different number of DNNs 27
4.9 Execution Time for different models 29

LIST OF ALGORITHMS

1 DiDQN Algorithm 18
2 Edge to Cloud Offloading Algorithm 19

LIST OF ABBREVIATIONS

AWS	Amazon Web Services
CPU	Central Processing Unit
DDLO	Distributed Deep Learning Offloading
DROO	Deep Reinforcement Online Offloading
DiDQN	Distributed Deep Q-Network
DNN	Deep Neural Network
DPM	Dynamic Power Management
DQN	Deep Q-Network
EBS	Elastic Block Storage
EC2	Elastic Cloud Compute
ECU	EC2 Compute Unit
GB	Gigabytes
GPU	Graphics Processing Unit
IoT	Internet of Things
IT	Information Technology
K	Thousands
MB	MegaBytes
MDP	Markov Decision Process
MEC	Mobile Edge Computing
MINLP	Mixed Integer Non Linear Programming
MIP	Mixed Integer Programming
NP	Non Polynomial
QoS	Quality of Service
RL	Reinforcement Learning
RNN	Recurrent Neural Network
TB	TeraBytes
VM	Virtual Machine
WD	Wireless Device

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

The number of smart devices such as mobile phones, wearables, speakers, security systems, and other devices that fall under the Internet of Things category has exploded in recent years. The Internet of Things (IoT) is a network of physical items that are equipped with sensors, software, and other technologies that enable them to connect and exchange data with other devices and systems over the internet. According to the statistics published in [2], the total number of IoT connected devices has reached 8.74 billion by the year 2020 and it is projected to triple by the year 2030.

Complementing desktop PCs and laptops, mobile devices are increasingly being used as platforms to help users complete more online work. Not every smart electronic devices need to be equipped with large CPU cores and memory resources. This only makes them expensive and is also against the Green IT initiative under which people are trying to minimize the adverse impact of IT operations on the environment by designing, manufacturing, operating and disposing computers in an environment friendly manner. Recent researches are focusing on how to manage the available resources efficiently. In the mean time, cloud computing is becoming more and more popular where computationally expensive jobs are uploaded to remote servers, which subsequently transport the execution results back to, due to restricted execution capabilities and the battery constraints of IT devices.

Due to data propagation and routing between mobile devices and faraway cloud servers, the approach suffers from high latency and unpredictable service quality, despite the fact that it saves mobile devices from doing computationally intensive activities. The idea of a mobile edge computing (MEC) network, in which such computationally heavy activities are computed in many edge servers deployed adjacent to mobile devices, has recently gained popularity. As the Internet of

Things (IoT) grows in popularity, more devices are being attached to MEC networks. Edge computing allows vast amounts of measured data to be offloaded to low-latency edge servers, extending the computation capacity of IoT sensors. The MEC networks' QoS varies depending on the offloading decisions made. As a result, carefully designing a compute offloading method for MEC networks is critical. Congestion occurs when multiple jobs are uploaded to the same edge server and that results in longer execution times for the processes. As a result, just uploading a task to the nearest edge server is surely not the best option.

1.2 Problem Definition

An exponential growth in the number of IoT devices has resulted in an tremendous increase in data processing, storage, and communication requirements. Although the servers in the cloud have the capacity for infinite storage and computations, it suffers from the latency problems which affects the time sensitive applications and also degrades the user experience. As a solution, the concept of edge computing has been proposed which carries out the computations in the edge devices instead of the cloud servers. Unlike cloud server, the edge device has a finite computation capacity and it can't handle massive computation tasks. There needs to be a smart task offloading scheme in order to decide whether to execute the task in the local device itself, or offload it to the edge device and process there or again send it to the cloud server for processing.

The decision of whether or not to offload depends on the available execution capacities at local mobile device, edge servers, and cloud servers, along with transmission capacity. This problem can be viewed into two phases, the first being offloading the task from local devices to edge servers. If the edge servers have enough resources, the task can be successfully executed there. But in case the resources are in full use or not enough, then only the task is sent to the cloud servers for processing. The optimization of joint offloading decision and bandwidth allocation is formulated as a mixed integer non linear programming problem in MEC to preserve energy and maintain quality of service for wireless devices (WD). The curse of dimensionality, on the other hand, places a computational restriction

on the problem, which cannot be tackled effectively and efficiently by ordinary optimization tools, especially for large-scale WDs.

1.3 Objectives

The objectives of this research work are:

- To develop a smart task offloading scheme for mobile edge computing in multi user, multi server environment
- To save energy and ensure uncompromised service quality for wireless devices in mobile edge computing

1.4 Scope of the Work

Although, there are a lot of problems and challenges in cloud computing and edge computing specifically, the scope of this research work is limited to resource allocation and power management in general cloud servers and edge computing only. The mobility of the wireless devices isn't considered during the experimentation of this research work.

1.5 Originality of this Work

In [3], the authors have implemented a distributed deep learning framework for modeling a smart decision offloading scheme for a single edge server. Although, the system architecture in [4] has implemented multi user, multi task, multi server architecture, it considers the cloud servers as a single remote entity which isn't necessarily true. Extending on the architecture and results from [3, 4], this research tries to model a cluster of remote servers and integrate the whole cloud and edge computing together to create a complete end to end system. Looking into all the past research works, none have implemented such a complete end to end multi user, multi task, multi edge server, multi cloud server environment.

1.6 Organisation of thesis work

This research work has been carried out in two major phases. In the first phase, an adaptive RL based cloud computing system was implemented that could learn to better manage the latency and power consumption in the cloud cluster. The satisfactory results obtained in the first phase allowed the further progress in the second phase where a smart decision task offloading scheme has been implemented for edge computing scenario. Finally, the system architectures from phase 1 and phase 2 have been integrated together using a rule based algorithm to model an end to end integrated edge to cloud computing system. The extensive experiments carried out in both phases and the results thus obtained show that the reinforcement learning can in fact play a significant role in better task allocation and power management in both edge and cloud computing.

CHAPTER 2

LITERATURE REVIEW

According to the statistics published in [2], the total number of IoT connected devices has reached 8.74 billion by the year 2020 and it is projected to triple by the year 2030. In order to handle the ever-growing data storage and processing requirements of IoT devices, a cloud platform has been proposed [5] but it is unusable for time sensitive applications which has led to the advent of edge computing. In edge computing, the data processing and storage takes place in the edge devices which are one hop away from the end devices. Edge computing, though promising has it's own limitations. Unlike cloud servers, the edge devices are resource constrained and can handle only limited computations. To make the best use of edge resource and avoid network congestion, a dynamic task offloading scheme is necessary which decides whether to execute the task on the local devices or offload it to the edge devices. Several wireless devices (WDs) can offload their processing tasks to an edge server using mobile edge computing (MEC) networks. The subject of computation offloading is commonly characterized as a mixed integer non linear programming problem (MINLP), and it has been extensively researched utilizing convex optimization and linear relaxation approximation. These traditional optimization approaches would take a very long time to discover the ideal solution in MEC networks. For example, assuming 30 wireless devices, there will be a total of 2^{30}options to pick from. As a result, traditional optimization approaches suffer from the dimensionality curse.

Recent researches [6, 3] have shown that applying deep reinforcement learning for solving such computation offloading problems can yield better performance and QoS than the traditional optimization algorithms. Huang et al. [3] developed a distributed deep learning-based offloading algorithm (DDLO) for MEC networks with multiple WDs and a single edge server, which can successfully deliver almost optimum offloading decisions. DDLO generated offloading decisions using multiple concurrent Deep Neural Networks (DNNs) and stored the freshly gener-

ated offloading decisions in a shared replay memory that was then used to train and upgrade all DNNs. The distributed deep learning offloading technique can deliver near-optimum offloading decisions for MEC networks in few milliseconds, according to extensive numerical data from [3]. Deep Reinforcement Learning for Online Computation Offloading (DROO) [6] proposed an online algorithm that optimally adapts task offloading decisions and wireless resource allocations to the time-varying wireless channel conditions. The framework learns from the past offloading experiences under various wireless fading conditions, and automatically improves its action generating policy. As such, it completely removes the need of solving complex Mixed Integer Programming (MIP) problems, and thus, the computational complexity does not explode with the network size. DROO used an adaptive procedure to adjusts the parameters of the algorithm online. Results show that compared to tradition optimization methods, DROO can produce near-optimal performance while significantly decreasing the execution time.

Most of the previous research works in the field of MEC have been done without the consideration of cloud servers in the network. Huang etl.al [4] considered the remote cloud server as part of the whole MEC network and proposed a heterogeneous DDLO that achieved better convergence as compared to DDLO [3]. The authors also presented the list of past research works in MEC on the basis of the number of users, tasks, edge servers and remote cloud servers. The system architecture in [4] hasn't taken some important facts of cloud computing into consideration. First, the cloud servers aren't necessarily idle all the time and the computation task offloaded from edge server to cloud server mightn't necessarily be processed immediately. Second, although there mightn't be a significant communication latency, there would definitely be some computation latency which can adversely affect the performance. Third, the cloud computing service consists of multiple cloud servers rather than just a single one. Shakya [1] investigated on the idea of using a hierarchical reinforcement learning framework for distributing the tasks in the cloud servers so as to minimize the task latency and the power consumption. The hierarchical architecture consisted of a global RL model for task dispatching and a local RL model for deciding when to turn the servers on and off. Building upon the idea of [3, 4, 1], this research work has implemented an end to end

integrated edge and cloud computing system setting up a multi user, multi task, multi edge server, multi cloud server environment. Unlike previous researches, here, the task offloading scheme in edge computing has been formulated as a k-armed bandit problem and has tried to solve it using the reinforcement learning algorithms. The use of reinforcement learning algorithms makes the setup adaptive to the system changes. The results obtained in this research are comparable to that of [3] and also show that the setup can learn to take near optimal decisions within few milliseconds thus showing it's feasibility in real world applications.

CHAPTER 3

METHODOLOGY

IoT devices generally have finite battery life and computing capability due to their tiny form factor and strict production cost constraints. Recent developments in mobile-edge computing (MEC) technology can effectively increase the computational capabilities of devices. By placing computing devices at the user's side and preventing relaying traffic produced by apps to a remote data center, MEC gives an effective option to bridge the user and edge server. For those delay-sensitive cloud-computing applications like online gaming, real-time media streaming, and virtual/augmented reality, it reduces latency in compute operations and saves energy. It's necessary to evaluate whether or not to offload a wireless device's compute task to a MEC server. The uplink wireless channels become extremely crowded if computing jobs are aggressively offloaded to the edge server, resulting in significant delays in completing computation activities. A joint management system is needed for offloading choices and radio bandwidth allocation to take use of computation offloading. Enumerating all of the possible options is computationally costly because to the binary nature of offloading decisions.

3.1 Preliminary Work

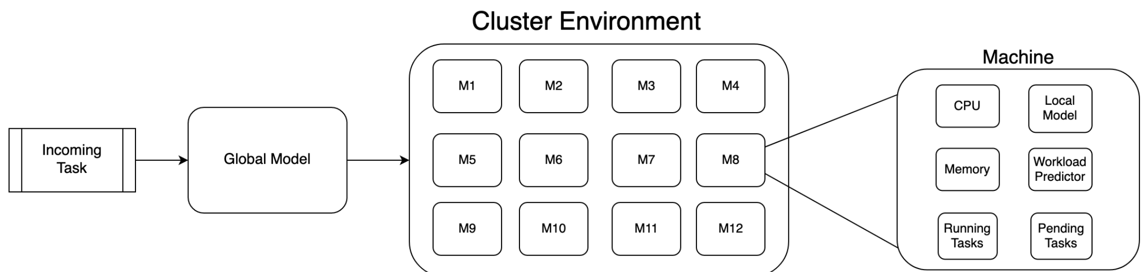

Figure 3.1.1: Task distribution and power management in cloud data center [1]

As a preliminary work for this research work, a hierarchical RL model based system was proposed in [1] for the optimal task distribution and power management in cloud servers. The proposed hierarchical model system shown in Figure 3.1.1

consisted of two RL models named as global and local model. The global model was responsible for task dispatching and the local model was responsible for the power management in each individual server. Both RL models were trained and validated on the Alibaba cluster trace dataset and compared against round robin and greedy approach as the baseline algorithm. The hierarchical setup outperformed both round robin and greedy approach by a huge margin in terms of power consumption but the approach suffered from high latency issues. In order to improve the latency without degrading the power performance, the global model was replaced by round robin algorithm. This modified hierarchical model setup outperformed the round robin and greedy approach in terms of latency as well as the power consumption but the improvement in latency wasn't relatively significant.

Although, the works presented in [1] validates the feasibility and positive impact of using hierarchical RL model architecture for task distribution and power management in cloud computing, there are some limitations unaddressed in it. To start with, both the final power consumption and latency improvements don't show relative significance. Next, the experiments were carried out only with Alibaba cluster traces and the proposed system didn't have any graphical simulation of the servers. Also, the difference combination of algorithms in global and local models weren't considered. So, definitely there is a lot of further experiments and achievements possible on top that work. Moving a step further, the solution proposed in [1] can be adapted for the sub categories under cloud computing such as edge computing, fog computing, mist computing and so on.

As an improvement and extension of the previous work [1], to the field of edge computing for handling the dynamic task offloading problem, the methodology of the proposed solution has been described in the following sub-sections:

3.2 Data Collection

A multi user, multi task, multi server MEC network architecture is being considered in this research. The data consists of the workload size for each M tasks of N users / wireless devices. The workload size varies from minimum of 10 megabytes (MB) to a maximum of 30 megabytes (MB). The dataset consists of 20,000 records

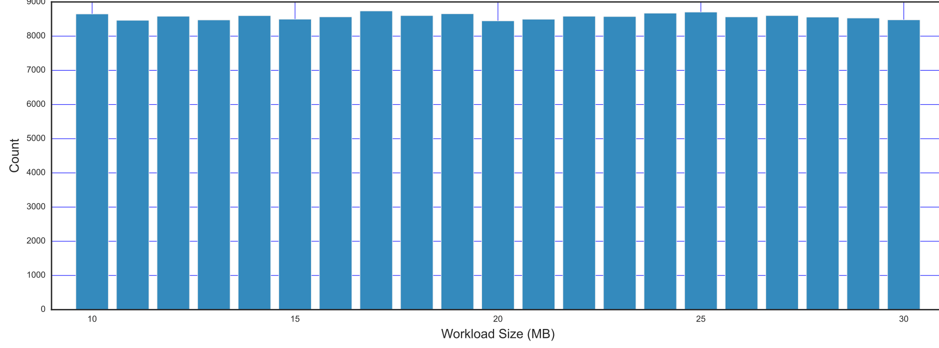

Figure 3.3.2: User workload size distribution

each consisting of NXM workload sizes.

3.3 Data Analysis

The dataset consists of the workload sizes for three tasks of three users. There are a total of 20,000 records in the dataset. The minimum workload size is 10 MB and the maximum is 30 MB. The distribution of the workload sizes in shown in Figure 3.3.2. The dataset also consists of the most optimal offloading decision vector and optimal gain value for each NXM workload record. The optimal gain value will be used to calculate the gain ratio of the predictions obtained from the training.

3.4 Implementation

In this research work, an MEC network is considered which is composed of K edge servers, K access point (AP), and N wireless devices (WD), indicated by a set $N = 1, 2, ..., N$, as shown in Figure 3.4.3. To keep the notations general, let's denote the set of servers as $K = 0, 1, 2.......K$, where server 0 denotes the WD itself. An optical fiber connects the AP and the edge server, and the transmission delay can be ignored. Each WD has a number of duties that must be completed locally or sent to the edge server via the AP. It is assumed that each WD has M impartial tasks, indicated by a set $M = 1, 2, ..., M$. The load of the mth task of user n is denoted as d_{nm} . Each WD n can evaluate whether to send its task m to the edge server or not, and the offloading decision is indicated by a binary variable $x_{nmk} \in \{0, 1\}$. Generally, $x_{nmk} = 1$ denotes that user n decides to offload the task

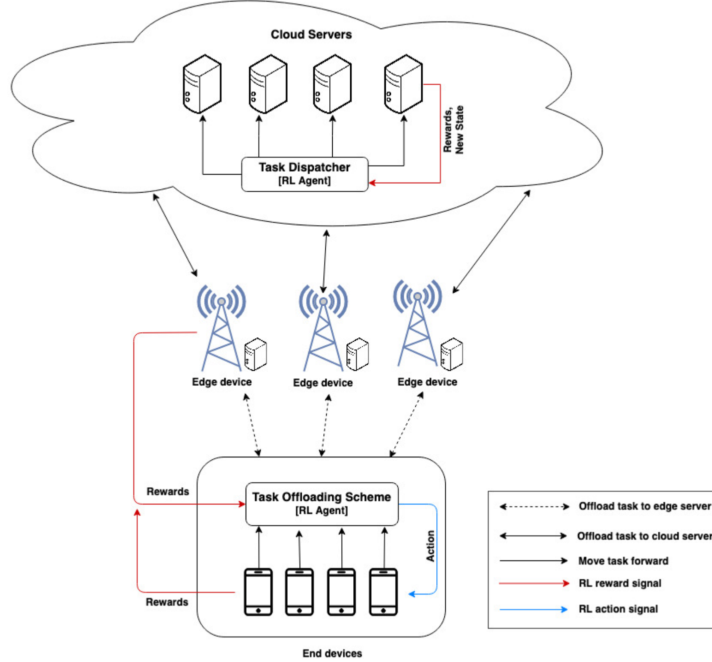

Figure 3.4.3: Multi user multi task multi server MEC network architecture

m to the edge server k, and $x_{nmk} = 0$ means that user n decides to execute the task m in local device itself. Just to avoid confusion the users and wireless devices (WDs) mean the same thing here.

Let's discuss about the task offloading RL agent shown in Figure 3.4.3. Since it is an RL agent, it is important to describe the state, action, reward and environment.

Table 3.1: Task offloading RL agent description

RL Component	Description
State	Current user task workload and available bandwidth
Action	Offloading decision vector
Reward	System utility cost
Environment	Multi user multi task multi server environment

In this research, the offloading decision optimization is modeled as a k-armed bandit problem [7], so the state of the system doesn't play much of a role here. The k-armed bandit problem is one in which a limited number of resources must be allocated among competing options in such a way that their expected gain is maximized, even though each option's properties are only partially known at the time of allocation and may improve with time or by allocating resources to the option. In Figure 3.4.3, the offload decision vector is represented by **Action** line

going from task offloading RL agent to end devices. The RL agent consists of pretrained distributed deep q-learning network model that outputs this offload decision vector / action. The model is trained in offline mode and the details of the training are explained in Section 3.4.4. The agent regularly collects the **Rewards** signal from end user devices and edge servers which is basically the total system utility cost calculated using the Equation 3.8. In reinforcement learning, rewards are the system feedbacks that help the agent to learn from it's past experiences and take better decisions in the future. The goal of RL agent is to maximize this reward value. One of the major benefit of using reinforcement learning is that it automatically helps the agent adapt to the system changes and doesn't require manual adjustments of the configuration.

In Figure 3.4.3, there are two computing modes that are being dealt: **Local computing mode** and **Edge computing mode**. The detailed operation of these modes is discussed next.

3.4.1 Local Computing

Let's first start with the case where the user n evaluates to execute it's m^{th} task in local device. The energy consumed by local device per data bit of user n can be denoted as e_n^l. Thus, the energy consumed by a user n to execute it's m^{th} task locally can be calculated as:

$$E_{nm}^l = d_{nm} e_n^l \tag{3.1}$$

For the latency, let's indicate user $n's$ local processing time per data bit as t^l, so the overall processing time for n^{th} user to complete it's m^{th} task can be calculated as:

$$T_{nm}^l = d_{nm} t^l \tag{3.2}$$

Finally, assuming user $n's$ offloading decision $\{x_{nm0}\}$, the total latency for n^{th} user to compelte all of it's tasks locally is given by:

$$T_n^l = \sum_{m=1}^{M} T_{nm}^l (1 - x_{nm0}) \tag{3.3}$$

3.4.2 Edge Computing

Whenever a job is offloaded to the edge server, the WD n sends its task load d_{nm} to the AP which is then transmitted to the edge server and processed. Because the data size of feedback information is minimal in general, the energy consumption and latency when the edge server communicates the compute results back to WDs can be ignored.

Let's start with modeling the energy consumption in edge computing mode. E_{nmk}^t indicates the energy used by WDs for sending its task load to the kth edge server. The energy requirement for data processing at edge servers can be modeled as a linearly varying function of task load d_{nm}. The total cost for user n to offload its task m to the kth edge server is computed as:

$$E_{nmk}^c = E_{nmk}^t + \alpha_k d_{nm} \tag{3.4}$$

where α_k indicates the weight of energy usage at kth edge server. When $\alpha_k = 0$, the energy usage at the WD is only taken into account.

Next, let's compute the offloading latency in edge computing mode. Consider c_{nk} as the assigned bandwidth for user n for sending it's offloaded task to the kth edge server. Thus, the transmission delay when the nth user offload it's m_{th} task to the edge server k is computed as:

$$T_{nmk}^t = \frac{d_{nm}}{c_n} \tag{3.5}$$

Given, the kth edge server's processing rate as f_k^c, the edge processing delay is computed as:

$$T_{nmk}^c = \frac{d_{nm}}{f_k^c} \tag{3.6}$$

Assuming the offloading decisions x_{nm}, the overall delay for user n can be calculated

as:

$$T_{nk}^{c} = \sum_{m=1}^{M} (T_{nmk}^{t} + T_{nmk}^{c}) x_{nmk} \tag{3.7}$$

OBJECTIVE FUNCTION

The objective function formulated in Equation 3.8 and it's solution in Section 3.4.3 have been derived from the research works of [3] and adjusted for the case of multiple edge servers. To minimize both the overall latency of completing all users' tasks and the respective energy usage, let's introduce a system utility $Q(d, x, c, k)$ described as the weighted sum of task completion latency and energy usage, as

$$Q(d, x, c, k) = \sum_{n=1}^{N} \sum_{m=1}^{M} (E_{nmk}^{l}(1 - x_{nmk}) + E_{nmk}^{c} x_{nmk}) + \beta max\{T_{nk}^{l}, T_{nk}^{c}\} \tag{3.8}$$

where,

$d = \{d_{nm} | n \in N, m \in M\}$,

$x = \{x_{nmk} | n \in N, m \in M, k \in K\}$,

$c = \{c_{nk} | n \in N, k \in K\}$,

β indicates the energy weight of task completion and energy usage

To minimize $Q(d, x, c, k)$ by collectively optimizing each user n's offloading decisions $\{x_{nmk}\}$ and the allocations of bandwidth c_{nk} for user n's task transmission, $(P1)$ is formulated as an optimization problem:

$$(P1) : Q^{*}(d) = \underset{x,c,k}{\text{minimize}}\ Q(x, c, k) \tag{3.9a}$$

$$\text{subject to} : \sum_{n=1}^{N} c_n \leq C, \tag{3.9b}$$

$$c_{nk} \geq 1,\ c_{n0} = \infty \tag{3.9c}$$

$$x_{nmk} \in \{0, 1\} \tag{3.9d}$$

Here, constraint (3.9b) indicates that the overall uplink bandwidth allocation can't surpass the maximum bandwidth C. The assigned bandwidth for each user in each edge server c_{nk} is either 0 or positive as indicated in (3.9c). The binary constraint on x_{nmk} is indicated in (3.9d). $(P1)$ is a mixed-integer programming (MIP) optimization problem, which is very hard to solve.

Table 3.2 indicates the crucial notations of this research work.

Table 3.2: Notations Table

Notation	Definition
x_{nmk}	$x_{nmk} = 1$ if user n offloads its task m to edge server k. Otherwise, $x_{nmk} = 0$
d_{nmk}	Data size of mth task for n^{th} user
E^t_{nmk}	Data transmission energy consumption of mth task for n^{th} user
c_{nk}	User n assigned bandwidth
C	Overall bandwidth
T^t_{nmk}	Upload time of user n's m-th task
T^c_{nmk}	Execution time of user n's m-th task in edge server
f^c_k	Execution rate of edge server
T^c_{nk}	Time consumption of user n for edge Execution
e^l_{n0}	Energy usage per data bit of user n in local device
E^l_{nm0}	Energy usage of user n's m-th task in local device
t^l	Execution time per data bit in local device
T^l_{nm0}	Execution time consumption of user n's m-th task in local device
α	System utility cost weight
β	Weight between energy usage and Execution delay
E^c_{nmk}	Energy Usage of the edge sever for executing user n's m-th task
T^l_n	Execution delay of user n in local device

3.4.3 Solving the optimization problem

As the optimization problem is a mixed integer programming problem and there are two possible values of decision variable $x_{nmk} \in \{0, 1\}$ being considered, the total search space is given by 2^{NM} where N is the number of users and M is the number of tasks per user. The search space grows exponentially with respect to MN and thus, this is in fact a NP-hard problem, it can't be solved effectively and efficiently using ordinary optimization methods.

Branch-and-bound techniques and dynamic programming are commonly employed to solve MIP problems, however these are ineffective in large-scale MEC networks due to high execution cost. Convex relaxation [8, 9] and heuristic local search

[10, 11] are used to reduce computing complexity however, both of them need a large amount of computational time to reach the local optimum, rendering them unsuitable for real-time offloading decisions in rapidly fading channels.

In recent years, researchers have been successful in solving such optimization problem within desired time, with the help of reinforcement learning. Inspired from the works in [6, 3, 4], the following algorithm 3.4.4 has been developed in order to solve the optimization problem formulated in 3.9a.

3.4.4 Algorithm

Given the input data vector d_{NxM} of size NXM where N is the number of users, M is the number of tasks per users and taking K as the total number of available edge servers, the target is to find a policy function π for offloading to produce the optimal offloading action vector $x^* \in \{0,1\}^{NMK}$ for $(P1)$ defined in Equation 3.9a, as

$$\pi : d \longrightarrow x^* \tag{3.10}$$

For each input vector d, Z offloading deep learning networks (DNNs) are used to generate Z candidate offloading action vector with one action vector from each DNN, as shown in Figure 3.4.4. Basically, here each DNN is trying to model the desired policy described in Equation 3.10 as a parameterized function f_{θ_z}, i.e.

$$f_{\theta_z} : d \longrightarrow x_z \tag{3.11}$$

where θ_z indicates the parameters of the z-th DNN. All those Z DNNs differ in parameter values θ_z but have the same structure.

After the binary offloading decision vector x_z has been computed, the original optimization problem $(P1)$ of Equation 3.9a transforms into a bandwidth allocation

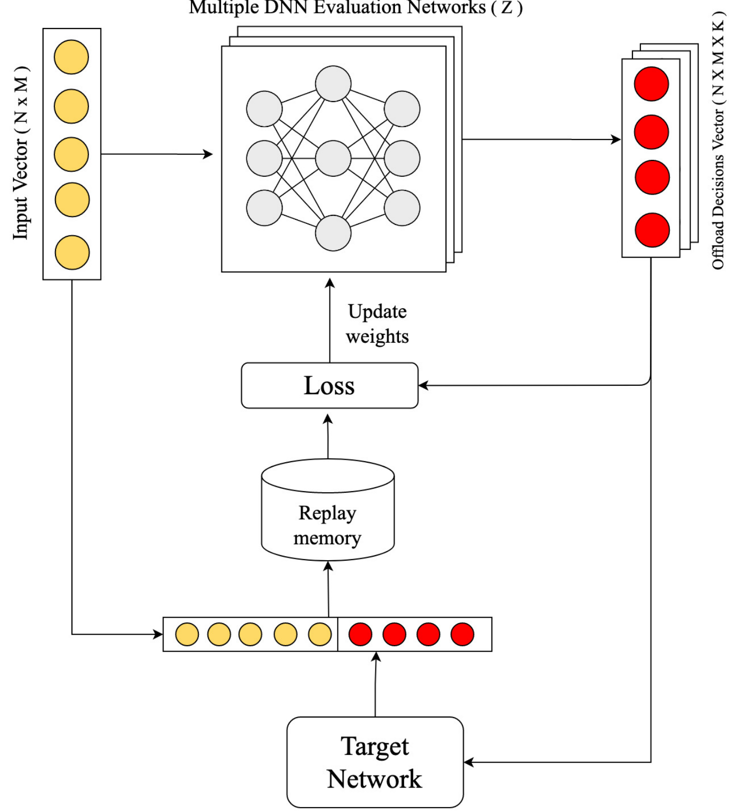

Figure 3.4.4: Distributed Deep Q-learning Network training architecture

problem $(P2)$, as

$$(P2): Q^*(d,x) = \underset{c}{\text{minimize}}\ Q(d,x,c) \tag{3.12a}$$

$$\text{subject to}: \sum_{n=1}^{N} c_n \leq C, \tag{3.12b}$$

$$c_{nk} \geq 1,\ c_{n0} = \infty \tag{3.12c}$$

Since $(P2)$ is a convex optimization problem, standard optimization tool can be used to easily solve it. Once all Z $(P2)$ are solved, the offloading action vector with least $Q^*(d, x_z)$ is selected among all Z candidates, as

$$x^* = \underset{z \in Z}{argmin} Q^*(d, x_z) \tag{3.13}$$

This chosen x^* output is the binary offloading decision for the input d vector and

it is saved as a new labeled data, (d, x^*), in the limited-size memory structure, where the recent labeled data replaces the oldest one if the memory is full. All Z DNNs are trained using those produced labeled data as shown in Figure 3.4.4. The DNNs aren't trained on the whole data, instead a batch of random data samples is extracted from the memory and a gradient descent algorithm is used to optimize the DNN parameter θ_z by optimizing the cross entropy loss, as

$$L(\theta_z) = -x^T log f_{\theta_z}(d) - (1 - x)^T log(1 - f_{\theta_z}(d)) \tag{3.14}$$

The full algorithm is shown in Algorithm 1. At the beginning, all Z DNNs are set with random parameter values θ_z and the memory store is empty.

Algorithm 1 DiDQN Algorithm

1: **Input:** Input task vector d_{nXm}
2: **Output:** Optimal offloading decision vector x^*_{nxmxk}
3: **Initialization:**
4: Randomly initialize the weight parameters θ_z of Z DNNs, $z \in Z$;
5: Set N, M, K
6: Reset the memory
7: **for** $i = 1, Z$ **do**
8: Share the task vector d_i across to all Z DNNs.
9: Generate z^{th} offloading decision vector x_z from z^{th} DNN evaluation networks in parallel, $x_z = f_{\theta_{z,i}}(d_i)$;
10: Solve all Z optimization problem (P2) in parallel using x_z
11: Choose the best offloading decision using target network $x^*_i = \underset{z \in Z}{argmin} Q^*(d_i, x_z)$
12: Store (d_i, x^*_i) into the replay memory
13: Sample batches of training data from replay memory randomly
14: Train DNNs and update weight parameters $\theta_{z,i}$
15: **end for**

3.4.5 Edge server to cloud data center offloading

Even though the edge servers are powerful and are placed very close to the mobile user devices, not all computationally extensive tasks can be executed in the edge server itself. As show in Figure 3.1.1 when there isn't enough resources available for executing the task at edge servers, then these are offloaded to the cloud data center for processing. The edge to cloud data center offloading doesn't need to be

as smart the user mobile device to edge server offloading scheme. It can simply be a rule based approach that checks for specified criteria and decides whether to offload the task to the cloud server or execute it in the edge server.The edge servers aren't aware of all the servers available in the cloud data center, so it is impractical to decide the exact cloud server to which the task should be offloaded. Instead the edge servers pass the task to the task placement manager in cloud which then assigns the task to specific cloud server and later returns the result back to the edge server. The edge server and cloud data center are generally connected through the fiber optics cable and the communication latency between them is small.

Although it mightn't be the case that the communication latency can be completely ignored, the edge server to cloud data center offloading only takes place only when there is absolute need to do so and thus in such cases, the latency from edge server to cloud data centers mightn't be so significant. Also, compared to the mobile user devices and edge servers, the cloud data center has very high computation capacity, so the computation latency can be ignored for most of the cases. For this research purpose, the communication and computation latency in the edge server to cloud data center offloading is set to a small value.

Algorithm 2 Edge to Cloud Offloading Algorithm

```
1: Input: Input task batch
2: Output: Offloading decision vector
3: Initialization:
4:    Set MAX_QUEUE_SIZE, MAX_TASK_SIZE
5:    Set the edge server execution batch size B
6: for i = 1, B do
7:    if i^th task > MAX_TASK_SIZE or current_queue_size = MAX_QUEUE_SIZE
      then
8:       Forward it to the cloud data center for processing
9:    else
10:      Calculate the computation time in edge server and cloud server
11:      if edge server computation time > cloud server computation time then
12:         Forward it to the cloud data center for processing
13:      else
14:         Forward it to the edge server pending queue
15:      end if
16:   end if
17: end for
```

In order to keep the edge server to cloud data center offloading agent minimalistic,

a rule based approach is followed. The offloading agent is present inside each edge server and uses a specific set of rules to decide whether to execute the task in the edge server itself or forward it to the cloud data center. The agent looks into the task size and the number of tasks in the pending queue of the edge server, makes a rough calculation of the time it would take for the task to get executed and then makes the offloading decision. The necessary steps have been described in Algorithm 2.

After the task has been forwarded to the cloud data center, it is the job of the task placement manager to dispatch the task to a specific cloud server. In order to minimize the latency and energy consumption of the cloud data center, the architecture shown in Figure 3.1.1 is used. The details have been already discussed in Section 3.1.

CHAPTER 4

RESULTS, ANALYSIS AND COMPARISON

4.1 Experimental Setup

In this section, the results obtained from Algorithm 1 is presented along with the detailed analysis of those results. The main objective was to train a DiDQN network for making near optimal task offloading decisions in a multi user, multi task, multi server environment. The model was trained on the dataset discussed in Section 3.2. The dataset consists a total 20,000 training records. Out of those, 80% were used for training and rest 20% for testing. The assumptions made regarding different parameters values during the simulation are listed in Table 4.2. All the experiments were carried out on AWS EC2 Servers with following specifications.

Table 4.1: AWS EC2 Server Specification

Instance Type	vCPU	ECU	Memory (GiB)	Linux/UNIX Usage
t2.2xlarge	4	Variable	16GiB	$0.1856 per Hour
r4.4xlarge	16	58	122GiB	$1.064 per Hour

Table 4.2: Parameter Values used in the simulation

Parameter	**Value**
Computation time of mobile device	$4.75x10^{-7} seconds/bit$
Processing energy usage in local device	$3.25x10^{-7} joules/bit$
Uplink bandwith limit	$150 Mbps$
Receiving energy usage	$1.42x10^{-7} joules/bit$
Transmission energy usage	$1.42x10^{-7} joules/bit$
Processing rate of edge servers	$10x10^{1}0 cycle/second$
System utility cost weight (α)	$1.5x10^{-7} joules/bit$
Weight between energy usage and processing delay (β)	$1 joules/second$

The performance evaluation of the algorithm was done on the basis of the Gain ratio which is calculated as:

$$Gain\ ratio = \frac{optimal\ system\ utility\ value\ from\ the\ dataset}{system\ utility\ obtained\ from\ DiDQN} \tag{4.1}$$

The model was optimized on the binary cross entropy loss of the prediction offload decision vector and actual minimum offload decision vector. Section 4.2 presents the training results obtained on a multi user, multi task, single edge server environment under different hyperparameter values. Section 4.4 presents the training results obtained on a multi user, multi task, multi edge server environment. Section 4.3 presents the execution time of the DiDQN model when the number of DNNs is increased. Finally the performance of DiDQN model is compared with Local processing and Edge processing in Section 4.5

4.2 Training in multi user, multi task, single server environment

For this experiment, three users with three different workload each were simulated in a single edge server environment. The DiDQN model was trained and tested by varying different hyperparameter values. Figure 4.2.1 presents the model performance under different number of DNNs which shows that the model with a single DNN doesn't learn anything and can't improve it's gain ratio. The figure also shows that with the increment the number of DNNs, the gain ratio performance also increases.

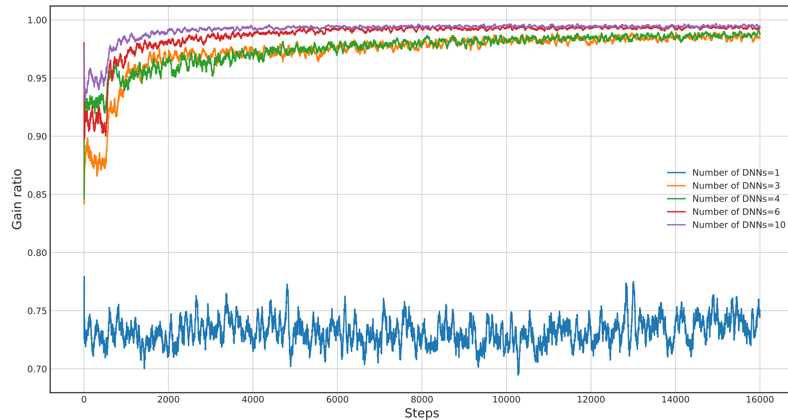

Figure 4.2.1: Gain ratio under different number of DNNs [N=3, M=3, K=1]

Figure 4.2.2 shows the training loss of the model when three DNNs were used. The model loss quickly reduces within the first 200 steps and then oscillates with some major spikes time and again.

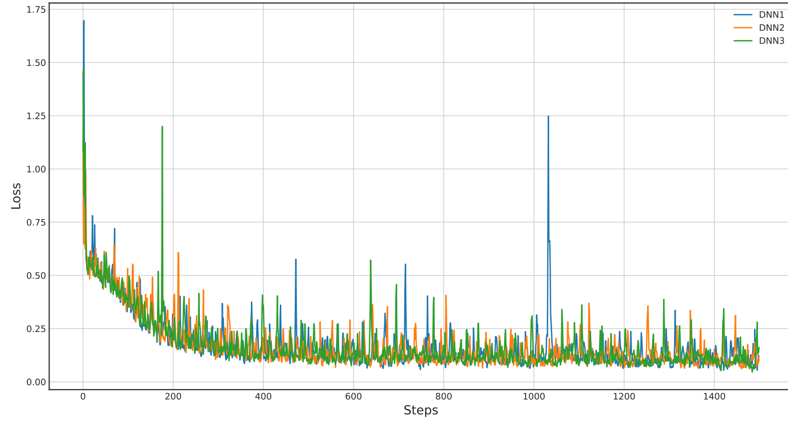

Figure 4.2.2: Training loss [N=3, M=3, K=1, Z=3]

Figure 4.2.1 also shows that with the increase in number of DNNs, the cumulative gain ratio becomes much better. For the case of three users, three tasks per user, one edge server [N=3, M=3, K=1], three DNNs (Z=3) seems to be decent enough. Results obtained by training a DiDQN model with three DNNs, and tweaking the hyperparameters like learning rate (lr), batch sizes, memory sizes and training interval are show in in Figure 4.2.3 and Figure 4.2.5, Figure 4.2.4 and Figure 4.2.4 respectively.

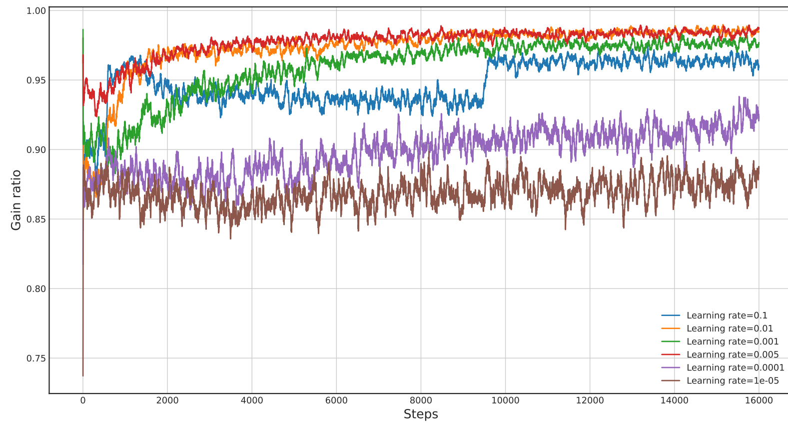

Figure 4.2.3: Gain ratio [N=3, M=3, K=1, Z=3 with different learning rates]

The best performance is observed for the case of lr=0.01 and lr=0.05. The mean gain ratio obtained in the test set under different learning rates is shown in Table 4.3. The best gain ratio on test set is obtained when lr=0.01.

Table 4.3: Gain ratio on test set under different learning rates

Learning rate	Average gain ratio on test set
lr=0.1	0.9637
lr=0.01	0.9871
lr=0.001	0.9765
lr=0.005	0.9859
lr=0.0001	0.9203
lr=0.00001	0.8778

The result obtained by varying the memory sizes has been shown in Figure 4.2.4. The best performance is observed for the case of memory size=1024. The average gain ratio obtain in the test set under different memory sizes is shown in Table 4.4. The best gain ratio on test set is obtained when memory size=1024.

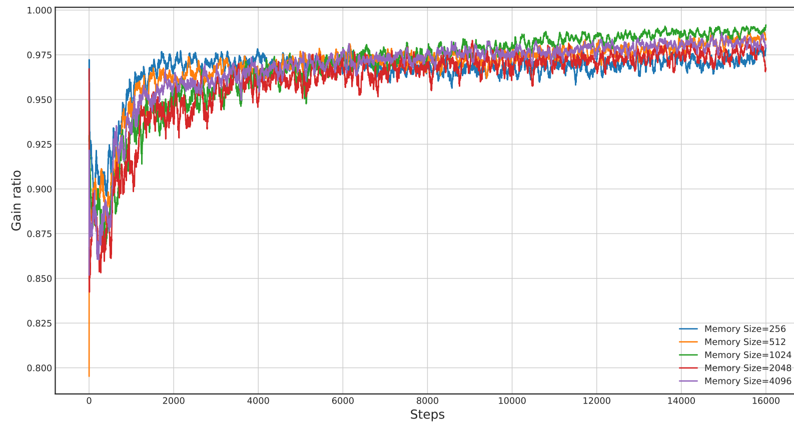

Figure 4.2.4: Gain ratio [N=3, M=3, K=1, Z=3 with different memory sizes]

Table 4.4: Gain ratio on test set under different memory sizes

Memory size	Average gain ratio on test set
memory size=256	0.9726
memory size=512	0.9861
memory size=1024	0.9896
memory size=2048	0.9769
memory size=4096	0.9842

The results obtained by varying the batch sizes has been shown in Figure 4.2.5. The best performance was observed for the case of batch=256. The average gain ratio obtain in the test set under different memory sizes is shown in Table 4.5. The best gain ratio on test set is obtained when batch size=256.

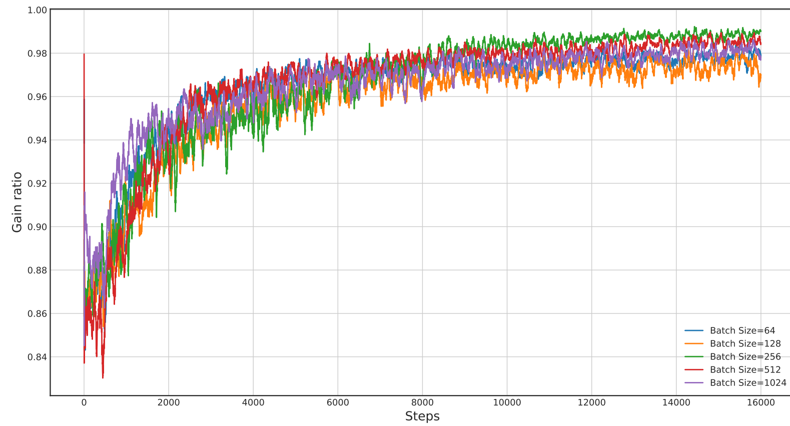

Figure 4.2.5: Gain ratio [N=3, M=3, K=1, Z=3 with different batch sizes]

Table 4.5: Gain ratio on test set under different batch sizes

Batch size	Average gain ratio on test set
batch size=64	0.9792
batch size=128	0.9776
batch size=256	0.9889
batch size=512	0.9873
batch size=1024	0.9824

The results obtained by varying the train intervals has been shown in Figure 4.2.6. The best performance was observed for the case of train interval=1. The average gain ratio obtain in the test set under different memory sizes is shown in Table 4.6. The best gain ratio on test set is obtained when train interval=1.

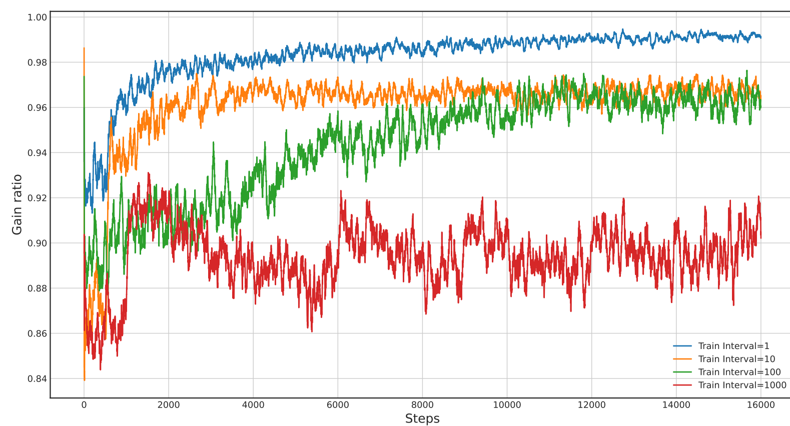

Figure 4.2.6: Gain ratio [N=3, M=3, K=1, Z=3 with different train intervals]

Table 4.6: Gain ratio on test set under different train intervals

Train interval	Average gain ratio on test set
train interval=1	0.9920
train interval=10	0.9685
train interval=100	0.9699
train interval=1000	0.8957

Different results obtained by varying the hyperparameters like learning rate, batch size, memory size, train interval shows that the change in these hyperparameters have some significant impact on the train and test performance of the model and it is very important to tune these parameters correctly. Too small learning rate makes the model learning slower whereas too high learning rate makes the model diverge more from the global optimum and reduce the performance on the test set. Similarly, the memory size, batch size and train interval, all have non-monotonic effect on the model learning. Too much or too little of these parameter values can slow down the learning and affect the performance adversely. Looking into the results obtained under different hyperparameter settings, the following hyperparameters has been selected:

Table 4.7: Best Hyperparameter values

Hyperparameter	Value
lr	0.01
memory size	1024
batch size	128
train interval	10

The reason behind not selecting the $train\ interval = 1$ even though it had superior performance is that, it made the training very slow. In case of sufficient availability of training resources, it is best to train the model with $train\ interval = 1$ but in other cases training with $train\ interval = 10$ would be a better option.

4.3 Execution Time under different number of DNNs

Table 4.8 presents the execution / prediction time of DiDQN model with different number of DNNs. It is observed that with the increase in the number of DNNs, the prediction time increases as well but even for a model with 10 DNNs the prediction

time is still under $5ms$ which is acceptable.

Table 4.8: Execution time under different number of DNNs

Number of DNNs	Execution time (ms)
1	0.46
2	0.88
3	1.32
4	2.00
5	2.17
10	4.38

4.4 Training in multi user, multi task, multi server environment

As the optimization problem that is being dealt here is of NP-hard case. The computation complexity grows exponentially with increase in number of either users, tasks or servers. With a three users, three tasks, three edge servers, the problem search space reaches to 2^{27} and thus a larger number of DNNs are required to get better performance in such environments. Figure 4.4.7 shows that the gain ratio performance of the model increases with the increase in number of DNNs. It also reflects that the gain ratio performance has reduced significantly due to the exponential increase in the problem space but still the performance is superior than that of Local processing and Edge processing as shown in Figure **??**.

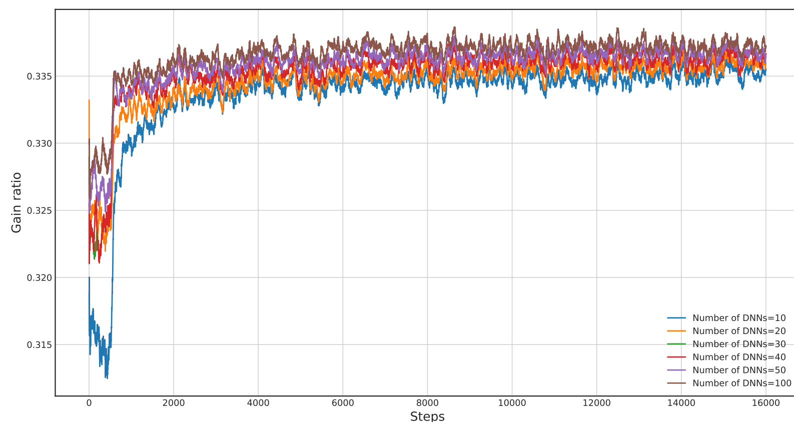

Figure 4.4.7: Gain ratio under different number of DNNs [N=3, M=3, K=3]

In order to verify that the model is actually learning something, the loss curve for a DiDQN model with ten DNNs is presented in Figure 4.4.8 which shows that the

training loss of the DNN models gradually decreases with some minor spikes and few major spikes.

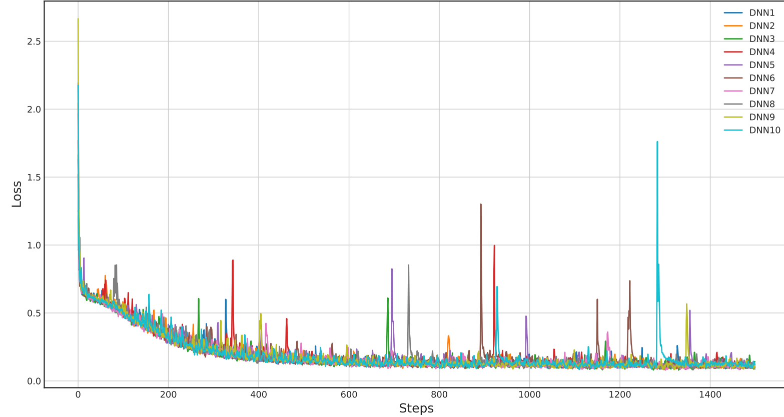

Figure 4.4.8: Training loss [N=3, M=3, K=3, Z=10]

4.5 Comparison of DiDQN with Local, Edge Processing and DDLO model

In this section the performance of DiDQN model is compared with the performances of naive Local processing and Edge Processing model, and DDLO model. In Local processing model, all the tasks are assigned to the local mobile devices and in case of Edge processing model, all the tasks are assigned to the edge server. Figure 4.5.9 present the performances of these models which clearly shows that DiDQN model performance is far superior than Local processing and Edge processing models.

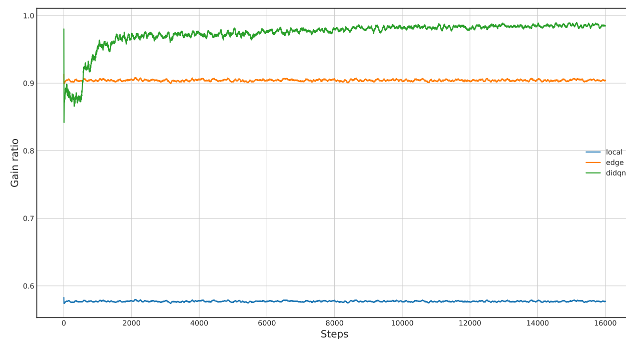

Figure 4.5.9: Gain ratio from different models [N=3, M=3, K=1, Z=3]

Local and Edge processing are just naive algorithms and don't learn anything from system feedbacks. In contrary, even though the RL agent initially has lower performance than that of Edge processing as shown in Figure 4.5.9, gradually it learns to make better decisions and keeps on improving it's performance with more training.

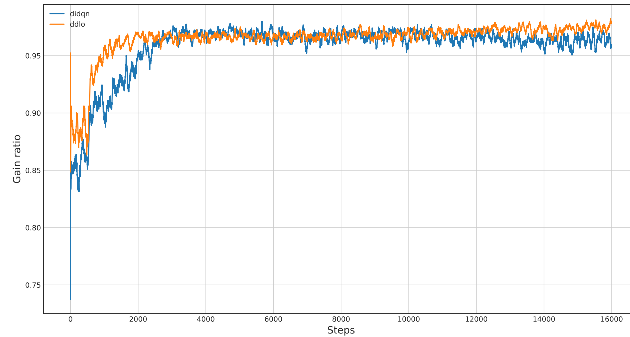

Figure 4.5.10: Gain ratio from didqn and ddlo models [N=3, M=3, K=1]

Figure 4.5.10 shows the comparison between DiDQN model and the model proposed by the researchers in [3] in terms of gain ratio. The performance of DiDQN model is comparable, although slightly lower than DDLO model. In terms of execution time, both DiDQN and DDLO models have similar performance with DDLO few milliseconds faster than DiDQN. Furthermore, it also shows that the reinforcement learning algorithm used in this research can actually learn from the system feedbacks and gradually improve it's performance.

Table 4.9: Execution Time for different models

Model	Execution Time
Local Processing	4.5 μs
Edge Processing	5.96 μs
DiDQN Processing	1.34 ms
DDLO Processing	1.29 ms

The execution time for different models under multi user multi task single server environment is listed in Table 4.9. As expected, it shows that the execution time of DiDQN model is slower than that of Local processing and Edge processing, but a few milliseconds of the prediction time is still tolerable for task offloading decision if it gives far superior performance which is the case here.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

In this research work, a distributed deep q-learning networks (DiDQN) model has been implemented for multi user, multi task, multi server MEC networks with the objective of finding an optimal offloading decisions that minimizes the overall system utility cost and produces near optimal results. Experiment results show that the DiDQN model can achieve superior performance than Local processing and Edge processing, and has comparable performance to that of DDLO models. Furthermore, the DiDQN algorithm has been shown to produce near-optimal offloading decisions in few milliseconds even after using a model with ten DNNs. The offloading decisions optimization is an NP-hard problem and it's search space grows exponentially and can't be solved using traditional optimization methods as it is very time consuming. This research shows that a distributed deep learning network can be used to handle such optimization problems and obtain better results under the desired time. Furthermore, combining the resource allocation architecture presented in [1] and the multi user, multi task, multi server mobile edge computing network, this research work presents a completed automated end to end task offloading and management scheme for integrated edge and cloud computing.

5.2 Limitation

The results obtained in this research looks very promising, the models are learning and gradually improving it's gain ratio performance. But there are few limitations in the research. To start with, the problem formulation doesn't consider the mobility of the wireless devices. The change in the distance from wireless devices to edge servers definitely affects the available bandwidth and eventually the offloading decision. If the mobility rate of the wireless devices is lower than the model

execution time (few milliseconds), then there shouldn't be any problem but if it starts moving too quickly then the decisions can't be relied upon. The future research should definitely consider this and carry out more experiments on it.

In case of multi edge server setup, the datasets aren't available yet. The same dataset created for a single edge server research has been used for varying number of edge servers. The optimal gain value given in the dataset was computed for the one edge server scenario where the upload and download bandwidth is always constant. This isn't always true, especially in case of wireless channels where the bandwidth keeps varying with time. The model can sure dynamically adapt in the fluctuating bandwidth environment but there isn't a proper dataset to validate this.

5.3 Recommendation

The results obtained from this research work can help to build better IoT devices ecosystem where all the IoT devices can be equipped with only necessary computation power and rest of the resources extensive jobs are smartly offloaded to the edge and cloud servers. There are some important areas of this research work that can be improved further in the future. For example, in this research, the same dataset of single edge server setup has been reused for training in multi edge servers setup assuming that the optimal gain value that can be achieved should still remain the same. The results show that the **DiDQN** model can still learn, improve it's gain ratio and perform better than **Local processing** and **Edge processing** even in this setup but the overall gain ratio remains low. One of the reason behind this is the exponential increase of the search space with each new addition of the edge servers. With $N = 3, M = 3, K = 1$, the search space is 2^9 but after increasing the number of edge servers $K = 3$, the problem space increases exponentially and reaches 2^{27}. In such scenario, the models have to be trained much longer and should be allowed to make enough exploration. The future research should focus on preparing a robust dataset for multi user multi task multi server environment. Moreover, a different approach towards solving this optimization problem can be also carried by using the ideas from Genetic algorithm which are considered a

better alternative for solving NP-hard problems.

REFERENCES

[1] Sushil Shakya. Self-managed cloud computing system using deep reinforcement learning. Technical report, Department of Electronics and Computer Engineering, Institute of Engineering, Tribhuvan University, Lalitpur, Nepal, 2021. 3rd semester project report.

[2] IOT Stats. `https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/`. Accessed: 2021-04-02.

[3] Liang Huang, Xu Feng, Anqi Feng, Yupin Huang, and Li Ping Qian. Distributed deep learning-based offloading for mobile edge computing networks. *Mobile Networks and Applications*, pages 1–8, 2018.

[4] Liang Huang, Xu Feng, Luxin Zhang, Liping Qian, and Yuan Wu. Multi-server multi-user multi-task computation offloading for mobile edge computing networks. *Sensors*, 19(6):1446, 2019.

[5] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[6] Liang Huang, Suzhi Bi, and Ying-Jun Angela 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.

[7] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[8] Songtao Guo, Bin Xiao, Yuanyuan Yang, and Yang 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. IEEE, 2016.

[9] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony QS Quek. Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[10] Suzhi Bi and Ying Jun Zhang. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. *IEEE Transactions on Wireless Communications*, 17(6):4177–4190, 2018.

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

[12] Mingxi Cheng, Ji Li, and Shahin Nazarian. Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 129–134. IEEE, 2018.

[13] Xiaolan Liu, Zhijin Qin, and Yue Gao. Resource allocation for edge computing in iot networks via reinforcement learning. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[14] Fundamentals of Reinforcement Learning. `https://www.coursera.org/learn/fundamentals-of-reinforcement-learning/home/welcome`. Accessed: 2021-03-01.

[15] Tensorflow. `https://www.tensorflow.org/`. Accessed: 2021-05-28.

[16] Python. `https://www.python.org/`. Accessed: 2021-05-28.

APPENDIX A

TURNITIN PLAGIARISM REPORT

MSCKE_Final_Thesis_Report_for_SI/075Mscke_019_Sushil_S...

ORIGINALITY REPORT

19%

SIMILARITY INDEX

PRIMARY SOURCES

#	Source	Words
1	link.springer.com Internet	408 words — 5%
2	Liang Huang, Xu Feng, Anqi Feng, Yupin Huang, Li Ping Qian. "Distributed Deep Learning-based Offloading for Mobile Edge Computing Networks", Mobile Networks and Applications, 2018 Crossref	206 words — 2%
3	arxiv.org Internet	96 words — 1%
4	citeseerx.ist.psu.edu Internet	86 words — 1%
5	www.mdpi.com Internet	69 words — 1%
6	Wei-Zhe Zhang, Ibrahim A. Elgendy, Mohamed Hammad, Abdullah M. Iliyasu, Xiaojiang Du, Mohsen Guizani, Ahmed A. Abd El-Latif. "Secure and Optimized Load Balancing for Multi-Tier IoT and Edge-Cloud Computing Systems", IEEE Internet of Things Journal, 2020 Crossref	49 words — 1%
7	deepai.org Internet	43 words — < 1%

Figure A.0.1: Similarity index report from Turnitin