Value-based Reinforcement Learning Approaches for Task Offloading in Delay Constrained Vehicular Edge Computing

Abstract

In the age of blooming information technology, human-being has witnessed the emergence of the need for new paradigms that has both high computational capability and low latency. A potential solutions is Vehicular Edge Computing (VEC). From our last research, we proposed FDQO algorithm that combines Fuzzy rules and Deep Q-Network (DQN) to improve DQN's early performance by using Fuzzy Controller. However, we notice that frequent usage of Fuzzy Controller can hinder the future growth performance of DQN. To overcome this issue, our team then introduce two approaches: removing Fuzzy controller entirely (Baseline Deep Q-Network) and limiting the usage of Fuzzy Controller in later time steps (Baseline FDQO). The former attempts to only use a baseline to modify the exploration rate to perform better when convergence compared to other conventional Q-Learning based algorithms like Deep Q-Network. However, its early result is still very low compare to FDQO. Our second proposed algorithm b-FDQO attempts to minimize the Fuzzy Controller's support when the performance is higher than a threshold while still using baseline to modify the exploration rate like b-DQN. Through experiment, when convergence, the average QoE in each time slot of b-FDQN is as high as that of b-DQN while still benefiting from better early performance due to Fuzzy Controller (better by 80%).

Keywords: Vehicular Edge Computing, Deep Q-learning, Fuzzy logic, Quality of Experience, offloading, delay constraint

1. Introduction

In recent years, the Internet of Things (IoT) has evolved to help devices in the physical world connect and share data through the Internet. There are many applications of IoT in many aspects of life, such as smart homes, self-driven vehicles, smart wearables, intelligent transport systems, smart grids, and many more. IHS Markit estimates there are about 27 billion IoT devices in 2017 and predicts more than 120 billion devices by 2030 with a growth rate of 12% per year [1]. This will result in a significant increase in the amount of data sent around the world. Furthermore, a wide range of IoT applications and services are constantly evolving and gaining traction. IoT applications and services frequently necessitate large amounts of computation and ongoing data processing. On the other hand, IoT devices frequently face data storage, processing capability, and battery capacity limitations. To address these limitations, Mobile Cloud Computing (MCC) [2] is proposed to support IoT devices by enabling the offloading of resource-intensive jobs onto a powerful cloud computing system. However, the clouds are far away from the end-user, resulting in high transmission latency. Besides, the increase in data sharing puts more pressure on the network transmission system. If all of the data is sent to the cloud for processing, the bandwidth usage and competition will be enormous.

The European Telecommunications Standards Institute (ETSI) introduced Mobile Edge Computing - MEC [3] (an implementation of Fog Computing) as a way to bring intelligence and higher processing and storage capabilities to the network's edge. MEC is based on the idea of extending cloud computing capabilities to the edge of cellular networks. This paradigm of computing will not only reduce network congestion but also improve resource optimization, user experience, and overall network performance. However, the storage capacity and computation power of the MEC are often limited. Combining resources at the edge and resources in the cloud to expand the total resource capacity is a solution to overcome the resource constraints of MEC. [4, 5, 6, 7, 8].

Recently, there has been a growing population of smart vehicles around the world. According to Statista, by 2023, there will be about 350 million smart vehicles. With that, vehicles are increasingly equipped with more computing and storage resources. Therefore, it is possible to use idle computing resources to significantly reduce the computation pressure of MEC. Vehicle Edge Computing (VEC) [9] is proposed as a promising model that receives a lot of attention because it can expand the computing capacity of the MEC network by leveraging resources on smart vehicles (parked and moving). All mobile users in a traditional MEC network offload their tasks to a single MEC server at the base station. As a result, some users' tasks may fall outside of the acceptable latency threshold. Multiple VEC servers make up a vehicular network, and tasks can be offloaded directly to vehicles with idle compute resources to process. In addition, since the task completion delay is significantly affected by the mobility of the vehicles, we also consider mobility in the formulation of the problem.

In a heterogeneous and dynamic environment like VEC, there exists a challenging demand for building an effective offloading mechanism that ensures the quality of the user experience. Recently, RL, especially Value-Based method has emerged as a potential offloading approach. It has advantages in learning without prior knowledge, learning online, and up-to-date environment perception. However, there exists several disadvantages in value-based reinforcement learning, including slow learning rate for large problems, the high reliance on the initialization step, and the need to scale the trade-off between exploration and exploitation rate.

In this study, we incorporate some supporting techniques to address the drawbacks mentioned above. The main contributions of this paper are as follows:

- Formulate the task offloading problem with delay constraint in VEC as a Markov Decision Process (MDP) and design a reward function that takes the tasks' deadline into account.
- Propose a technique with a view to increase the return of Deep Q-learning by minimizing the exploration rate based on obversed rewards.
- Design an algorithm that can balance the utilization between the above technique and Fuzzy Logic to overcome their particular weaknesses while reserving the strength of each effectively.
- Conduct intensive experiments to demonstrate the efficiency of each proposed algorithms in some aspects, especially the performance at the initial stages and return in the long run.

The rest of this paper is organized as follows. In Section 2, we briefly review the literature. In section 3, we introduce the overview of Deep Q-learning and Fuzzy logic which will be exploited in our proposed algorithm. Section 4 presents the network model and formulates our targeted problem. The details of the proposal are described in Section 5. We evaluate and compare the performance of our proposal with the existing approaches in Section 6. Our paper concludes with Section 7 that shows our future work.

2. Related Work

Many studies have been conducted in recent years to solve the problem of task offloading. The methods used are generally divided into two categories: offline and online. The analysis of offline methods is discussed in the next paragraph, while online methods are discussed in the following two paragraphs.

Huang et al. [4] formulated the computation offloading problem while taking into account security, energy consumption, and workflow execution time in MEC environment. They built a GA-based (Genetic Algorithm) security overhead model and energy consumption minimization strategy for computation offloading in heterogeneous Mobile Edge servers using performance parameters such as protected data size, CPU cores, and MEC servers' computation frequency. Xu et al. [5] focused on energy-aware computation offloading strategy for smart edge computing in wireless metropolitan area networks. They adopted the Non-dominated Sorting Genetic Algorithm II (NSGA-II), which uses multi-objective optimization to reduce computing task offloading time and energy consumption. The components of an Augmented Reality (AR) task were modeled as a directed acyclic graph with dependencies in [10], which looked at the latency and the reliability of AR task offloading in MEC. An Integer Particle Swarm Optimization algorithm has been proposed to minimize the probability of AR service failure. Another scheme [11] for minimizing the task computation latency in vehicular networks is Soft-VAN: a mobility-aware task offloading scheme. Fog node selection and task offloading are two phases of their proposed scheme. The computation problem is divided into two sub-problems and solved using evolutionary algorithms due to its NP-hard nature. The authors of the majority of these papers only considered offline offloading methods. When applying these types of methods to dynamic and time-sensitive environments, such as MEC, one possible limitation is that these models only consider the effectiveness of the proposed method, leaving out the speed of the method.

Alam et al. [12] introduce an autonomic offloading framework in MEC network to reduce service computing latency. The optimal offloading decision is determined using a deep Q-learning-based algorithm based on current resource requirements, resources, and network state. In [13], the authors proposed an online learning algorithm based on multi-armed bandit theory for task offloading decisions among multiple candidate servers with the goal of minimizing the average task offloading delay. By leveraging Deep Reinforcement Learning (DRL), the work in [14] designed an intelligent VEC offloading system. The original problem was subdivided into two parts. To schedule offloading requests and assign network resources, respectively, a two-sided matching method and a DRL approach were developed. Wang et al. [15] proposed a novel regional intelligent management vehicular network with dual MEC layers, in which MEC servers in the same region collaborate to share resources. To make offloading decisions, they developed a resource allocation algorithm based on deep Q-learning that can adapt to the changing MEC environment and process highdimensional data. The problems in these studies were solved using DRL techniques. This method has the advantages of learning without prior knowledge, being able to learn online, and having an up-to-date perception of the environment. However, there are several drawbacks to using DRL, including the fact that it takes a long time to reach an optimal or near-optimal solution for large problems without prior knowledge, and the Deep Q-network must be properly initialized.

Bylykbashi et al. [16] used fuzzy logic to determine the situational awareness of the driver by collecting data from the driver and the environment, such as noise level, vehicle temperature, and the driver's heartbeat. In [17], the authors presented a fuzzy-based approach with a mobile edge orchestrator to reduce latency in IoT by splitting tasks and avoiding task allocation failures at the network's edge. Their work proposed an optimal task execution and task handling strategy for avoiding data transmission latency. A fuzzy load balancer was built with different levels of fuzzy control design and tuning in [18]. Fuzzy logic is a powerful tool for solving difficult problems. However, creating a set of fuzzy logic rules that can cover all possible scenarios in reality is difficult. When the environment changes, fuzzy logic can become obsolete, especially in dynamic systems.

The study in [19] focused the task offloading in VEC. The authors proposed a new algorithm called FDQO to improve the initial QoE performance. However, that approach has some disadvantages. This present study which extends our research in [19] will point out and analyze that drawbacks. In addition, we introduce two new algorithms, one utilizes a technique to improve the return in the long run that excludes Fuzzy Logic and one to combine that technique with the conventional FDQO to bypass its weakness. A series of experiments will also be conducted to analyze the pros and cons when using each algorithms.

3. Preliminaries

In this section, we describe the overview of two main techniques which will be used in our proposal algorithms: Deep Q-learning and Fuzzy logic.

3.1. Deep Q-learning

In reinforcement learning [20], to drive the agent toward the target, a metric called *return* will be used. Mathematically, at time step t, it is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where γ is the *discount rate*, $\gamma \in [0;1]$, R_τ is the reward at time step τ.

Note that although the return is a sum of an infinite number of terms it is still finite if each reward is up-bounded and $\gamma < 1$. The mission of the agent is to select actions that maximize the *return*, whose expected value is ambiguous.

Q-learning is a reinforcement learning technique which can be described as a system learning that addresses the above problem. This algorithm uses the so-called Q-value to estimate the return following the *ideal* optimal policy:

$$Q(s,a) = R(s,a) + \gamma \max_{a'}(Q(s',a')) \tag{2}$$

where $Q(s,a)$ is the Q-value of performing action a against state s, $R(s,a)$ is the reward received, $Q(s',a')$ is the Q-value of performing next action a' against next state s', and γ is the discount factor ($\gamma \in [0;1]$).

These Q-values are randomly initialized and then repeatedly updated towards the correct Q-value. The matrix $Q(s,a)$ can be used as a lookup table. The difference between the Q-value in the lookup table and the Q-value after completing an action a against a state s is called Temporal Difference (TD):

$$TD(a,s) = R(s,a) + \gamma * \max_{a'}(Q(s',a')) - Q_{t-1}(s,a) \tag{3}$$

The Q-value concerning action a and state s in the lookup table is updated based on TD value as follows:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(a,s) \tag{4}$$

After the lookup table is completed, the system can choose the best action for each state. Although Q-learning is a simple and powerful algorithm, it fails to deal with problems whose state and action space are significantly large. To this end, Deep Q-learning technique which exploits neural network to enhance the learning process has been introduced, with an alternative name Deep Q-Network (DQN) [21]. This algorithm follows the steps below:

Step 1: The environment inputs a state into the neural network. The outputs are the Q-values of all actions.

Step 2: The agent selects an action a using ϵ-greedy policy and executes that action.

Step 3: The environment returns a new state s' and a reward, which are the results of action a. The system saves the experience tuple $[s, a, r, s']$ into memory.

Step 4: Sampling the experiences into a few batches and training the neural network.

These steps are repeated for a certain number of episodes or until the stopping condition is encountered.

3.2. Fuzzy logic

Fuzzy logic [22] is developed from fuzzy set theory. It serves as the logic for approximate reasoning instead of precise reasoning as in classical predicate logic. Fuzzy inference is a process of mapping a set of control inputs into a set of control outputs through fuzzy rules. In general, the design of a fuzzy controller includes the following steps:

Step 1: Defining the fuzzy sets and membership functions of the input signals.

Step 2: Defining the rule base for the fuzzy controller.

Each fuzzy set is associated with a linguistic term like "fast" or "slow". Membership function $\mu_A(x)$ quantifies the degree to which the element x is in the fuzzy set A. If $\mu_A(x)$ returns 0 then x is not in A, while a return value of 1 signifies that x is a full member of A. Otherwise, a return value between 0 and 1 shows the degree of member x belonging to fuzzy set A. Each member x may belong to one or more fuzzy sets. Below is a simple example of the membership function:

$$\mu_A(x) = \begin{cases} 1 & \text{if} \quad x \leq a \\ 0 & \text{if} \quad x \geq b \\ \frac{b-x}{b-a} & \text{if} \quad a < x < b \end{cases} \tag{5}$$

4. System model

4.1. Network model

Today, the advancement of IoT has given rise to a myriad of computational tasks generated by all kinds of smart devices, from sensors to gadgets connected to the internet, etc. Each task doesn't require a system with massive computation power but it requires fast response time. When a lot of tasks are sent to a server system in a short time, the system can be overloaded and cannot give a timely response to the requests of users, hence, service quality is not guaranteed.

We follow a four-component vehicular edge computing system with a base station, a local edge server close to the base, an access point, and mobile edge servers on vehicles (Figure 1). The base station uses both wired and wireless networks to communicate to both local and vehicular edge servers (VES). VES can either communicate directly to the base within communication range or via an access point if the vehicles are further away. We make some preassumptions that the VES always prioritize the nearest access point, and the connection is stable.

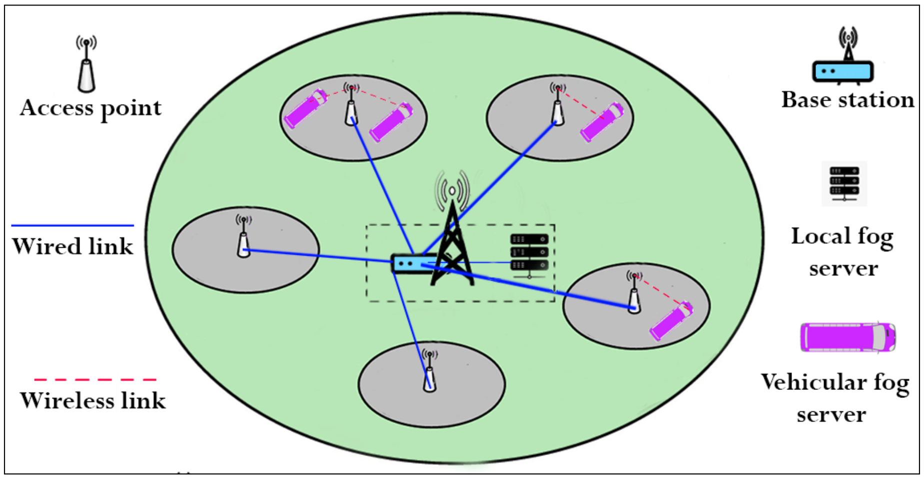


Figure 1: Illustration of the network model.

A typical process flow of tasks in the system is as follows: A task generated from the end-users is sent to the base station along with the information about the task's deadline. An offloading computer at the base station considers which computing node each task will be routed to for processing. In the case a local edge server is selected, the task is sent directly to the computing node in negligible time. In contrast, if the task is handled on a vehicular edge server, the task must first be routed to an access point closest to the vehicle egde server, then afterward, to the intended vehicle node. If the base station is closest to the vehicle, it sends the tasks directly to the vehicle. When tasks are processed in computing devices, they are put in a queue when waiting for service based on the principle "first come first serve". We assume that a computing device can only process one task at a time. After the task is completed, the vehicular edge server will transmit response data of the task to the closest access point, and it will be carried back to the base station.

In this system, every task has a soft delay constraint depending on its application. Our objective is to propose an offloading scheme to guarantee that the tasks' round-trip delay does not exceed their delay constraint.

4.2. Problem formulation

We divide the network operation time into equal time slots. We denote $M^t = \{M_1^t, ..., M_n^t\}$ as the set of tasks arriving at the base station within time slot t. Each task M_i^t is represented by a tuple $\{g_i^t, w_i^t, p_{i,out}^t, p_{i,in}^t, r_i^t\}$, where g_i^t is the timing when the task was generated; w_i^t is the computational resource needed to perform the task; $p_{i,in}^t$ and $p_{i,out}^t$ are the sizes of the packet containing the task and the packet conveying the result of the task, respectively; r_i^t is the deadline of the task. To check whether a task reaches its delay constraint, we define a QoE (i.e., quality of experience) function (denoted as f) as follows:

$$f(M_i^t) = \begin{cases} 1 & , \quad \text{if } \mathcal{T}(M_i^t) \le r_i^t \\ 0 & , \quad \text{if } \mathcal{T}(M_i^t) > 2r_i^t \\ \frac{2r_i^t - \mathcal{T}(M_i^t)}{r_i^t} & , \quad \text{otherwise,} \end{cases} \tag{6}$$

where $\mathcal{T}(M_i^t)$ is the total delay of M_i^t. Hereafter, a task is considered a good task if its QoE value is 1, and is considered a bad task if its QoE value is 0, and is considered medium otherwise. The QoE at the time slot t, denoted by $f(\mathbf{M}^t)$, is defined as the sum of the QoE of all tasks arriving at time slot t:

$$f(\mathbf{M}^t) = \sum_{i=1}^{n} f(M_i^t) \tag{7}$$

Our objective is to maximize $f(\mathbf{M}^t)$ for all time slots. In the following part, we will derive the delay of the tasks.

4.3. Delay derivation

The total delay of a task comprises three parts: (1) the transmission time from the base station to the server; (2) the processing time at the server; (3) the transmission time from the server back to the base station. The server's processing time can be further divided into waiting time in the queue and task performing time.

Let M_i^t be a task; then we denote its transmission time from the base station to the server and vice versa as $\mathcal{T}_{B2S}(M_i^t)$ and $\mathcal{T}_{S2B}(M_i^t)$, respectively. We also denote the waiting time in the queue and the time to perform the task as $\mathcal{T}_W(M_i^t)$ and $\mathcal{T}_P(M_i^t)$. Consequently, the total delay of M_i^t can be written as follows:

$$\mathcal{T}(M_i^t) = \mathcal{T}_{B2S}(M_i^t) + \mathcal{T}_W(M_i^t) + \mathcal{T}_P(M_i^t) + \mathcal{T}_{S2B}(M_i^t) \tag{8}$$

There are two types of tasks: tasks being offloaded to local edge server and tasks being offloaded to vehicular edge servers. Throughout this paper, we denote F_j ($j = 0, ..., n$) the servers, where F_0 represents the local server and $F_1, ..., F_n$ represent n vehicular servers. For a server F_j, let $\mathcal{R}(F_j)$ denote the computational resource and $\mathcal{C}(F_j)$ denote the remaining time to complete tasks assigned to F_j.

4.3.1. Type 1: Tasks that are offloaded to the local server

Since the wired network's bandwidth is significantly large, we assume that $\mathcal{T}_{B2S}$ and $\mathcal{T}_{S2B}$ are negligible. Thus, the delay of task M_i^t offloaded to local server can be represented as follows:

$$\mathcal{T}(M_i^t) = \mathcal{T}_W(M_i^t) + \mathcal{T}_P(M_i^t) \tag{9}$$

The time to perform the task is determined as:

$$\mathcal{T}_P(M_i^t) = \frac{w_i^t}{\mathcal{R}(F_0)} \tag{10}$$

The waiting time is defined as:

$$\mathcal{T}_W(M_i^t) = \mathcal{C}(F_0) \tag{11}$$

From (9), (10), and (11), we have:

$$\mathcal{T}(M_i^t) = \mathcal{C}(F_0) + \frac{w_i^t}{\mathcal{R}(F_0)} \tag{12}$$

4.3.2. Type 2: Tasks that are offloaded to the vehicular edge servers

Suppose F_j ($1 \leq j \leq n$) is the server receiving the task, and A is the base station that transfers the task from base station to F_j. The transmission time between the base station and the vehicular server consists of transmission time between the base station and the access point, and transmission time between the access point and the server. Since the wired link's bandwidth is significantly large compared to the wireless link[23], we omit the transmission time between the base station and the server. Let us denote by $d(A, F_j)$ the distance between A and F_j; then according to [24], the transmission rate between A and F_j is defined as follows:

$$v(A, F_j) = W\left(1 + \frac{P \times d(A, F_j)^{-\alpha}}{\sigma^2 + I}\right), \tag{13}$$

where W is the channel bandwidth, P is the base station's transmission power, α is the path loss exponent, σ^2 is the additive Gaussian noise, and I is the inter-cell interference. Therefore, the transmission time from the base station to the vehicular fog server can be calculated as follow:

$$\mathcal{T}_{B2S}(M_i^t) = \frac{p_{i,in}^t}{v(A, F_j)} = \frac{p_{i,in}^t}{W\left(1 + \frac{P \times d(A, F_j)^{-\alpha}}{\sigma^2 + I}\right)} \tag{14}$$

After performing the task, the vehicular server sends the result to the base station via the access point nearest to its current location, denoted as $A^{'}$. As the vehicle moves, its new location may be far from the location where it received task M_i^t, and $A^{'}$ may be different from A. Then, to address the location change, the time to transfer the result from the server to the base station is then determined as:

$$\mathcal{T}_{S2B}(M_i^t) = \frac{p_{i,out}^t}{v(A', F_j)} = \frac{p_{i,out}^t}{W\left(1 + \frac{P \times d(A', F_j)^{-\alpha}}{\sigma^2 + I}\right)} \tag{15}$$

The time to perform task M_i^t is defined as:

$$\mathcal{T}_P(M_i^t) = \frac{w_i^t}{\mathcal{R}(F_j)} \tag{16}$$

The waiting time is defined as:

$$\mathcal{T}_W(M_i^t) = \max\left(0, \mathcal{C}(F_j) - \frac{p_{i,in}^t}{W\left(1 + \frac{P \times d(A, F_j)^{-\alpha}}{\sigma^2 + I}\right)}\right) \tag{17}$$

From (14), (15), (16), and (17), we derive the total delay of M_i^t as follows:

$$\mathcal{T}(M_i^t) = \max\left(\frac{p_{i,in}^t}{W\left(1 + \frac{P \times d(A, F_j)^{-\alpha}}{\sigma^2 + I}\right)}, \mathcal{C}(F_j)\right) + \frac{p_{i,out}^t}{W\left(1 + \frac{P \times d(A', F_j)^{-\alpha}}{\sigma^2 + I}\right)} + \frac{w_i^t}{\mathcal{R}(F_j)} \tag{18}$$

5. Proposal

We exploit the complementary advantages of Deep Q-Network (DQN) with active exploration rate and Fuzzy logic to propose two approaches to optimize DQN models for offloading schemes: Baseline-based and Fuzzy-based models.

Our first baseline-based approach attempts to balance the exploration and exploitation rate of DQN. There are two baseline techniques not using Fuzzy Logic: Static baseline and Dynamic baseline. Our second Fuzzy-based approach utilizes Fuzzy logic to boost DQN's early low performance. Fuzzy Controller's fast task offloading based on a predefined set of rules can replace DQN's slow initial learning process. There are two Fuzzy-based models: Fuzzy Deep Q-Network in Offloading scheme (FDQO) and baseline-FDQO (b-FDQO).

Both approaches try to model our problem as a Markov Decision Process. After going into details of defining state space and action space in Section 5.1, the two approaches will be exposed respectively.

5.1. State space and action space

Let M_i^t be the upcoming task to be offloaded. Then the state concerning $S(M_i^t)$ is defined as a tuple representing the information of that task and the servers (both local and vehicular) as follows: $s(M_i^t) = \left\{w_i^t, p_{i,in}^t, r_i^t, \{\mathcal{C}(F_0), \mathcal{R}(F_0)\}, \{d(A, F_1), \mathcal{C}(F_1), \mathcal{R}(F_1)\}, ..., \{d(A, F_n), \mathcal{C}(F_n), \mathcal{R}(F_n)\}\right\}$, where $\{w_i^t, p_{i,in}^t, r_i^t\}$ shows information of M_i^t, $\{\mathcal{C}(F_0), \mathcal{R}(F_0)\}$ is concerned information of the local server, and $\{d(A, F_j), \mathcal{C}(F_j), \mathcal{R}(F_j)\}$ $(j = 1, ..., n)$ represents information related to vehicular edge server F_j.

Every tasks is offloaded to any of our servers. That set of selections forms the action space. Specifically, $a(M_i^t) = 0$ means that the task is offloaded to the local server, and $a(M_i^t) = j$ $(j \geq 1)$ indicates the task is offloaded to either of the vehicular edge servers F_j. In addition, reward function $r(M_i^t)$ returns the same value as QoE function described in Section 4.2 $(r(M_i^t) = f(M_i^t))$.

Because the state space is continuous and the action space is discrete, value-based reinforcement learning algorithms are reasonable approaches.

5.2. Proposed Baseline Deep Q-Network

Agents of traditional reinforcement learning in general, and Q-learning in particular, have little knowledge about the environment. Giving a set of actions to select from, it can choose to exploit, meaning to choose the action that has the highest estimated expected return temporarily or to explore for the sake of getting more information about the environment. The conventional $\epsilon - greedy$ policy helps the agent gradually improve itself, thanks to the exploration phase. However, a fixed rate of exploration can also be the factor that hinders our agent from following a better strategy, even when the expected return estimation is comparatively accurate. Therefore, we propose a new policy that can modify the exploration rate.

Our policy is based on the proposition that the exploration rate (ϵ) would decrease over time if the model performs better and increases when the model performs worst. In specific, when the environment is relatively stable, we can gradually reduce the exploration rate since the DQN network does not need much data exploration (ϵ gets smaller than its initial value). Nonetheless, that ideal condition hardly happens in real life. Thus, we need an alternative solution that ϵ may also increase when the expected-return estimation accuracy decreases due to the instability of the environment. Since the reward of an action in our problem model is always between 0 and 1, the average reward of a particular number of nearest actions is in [0;1] as well.

Some concepts need to be defined for model interpretation. Let η be the number of nearest actions to be observed, then $\bar{R}_\eta$ is the maximum average reward of the η nearest actions. $\beta \in [0;1]$ is a baseline-value that if $\bar{R}_\eta$ surpasses β, ϵ starts to drop. Vice versa, ϵ can remain or increase if $\bar{R}_\eta$ declines but never gets greater than its starting value. As a result, our agent's exploration rate would be more adaptive to the environment. More details will be provided through two algorithms named "Static-baseline Deep Q-Network (Sb-DQN)" and "Dynamic-baseline Deep Q-Network (Db-DQN)".

5.2.1. Proposed Static-baseline Deep Q-Network (Sb-DQN)

The flow of Sb-DQN is illustrated as Figure 2.

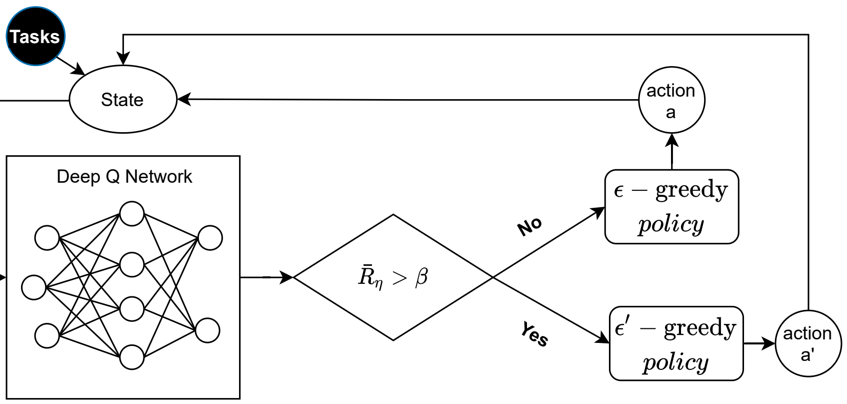


Figure 2: Static-baseline Deep Q-Network.

Step 1: Initialize η, β, ϵ, $\epsilon' < \epsilon$.

Initialize an empty queue $\bar{R}$ with a volume of η that stores rewards of some nearest actions.

Step 2: Use a deep neural network called Deep Q-Network to estimate the Q-value for each action.

Step 3: At first, check whether the queue $\bar{R}$ is full; if the answer is "yes", the value at the head of the queue must be removed. Then, the value of the nearest observed reward will be added to the tail of the queue. $\bar{R}_\eta$ is calculated as the average reward of all the values stored in the queue (The progress is made clear in the pseudo-code).

Step 4: Compare $\bar{R}_\eta$ to β to decide which policy will be chosen.

Step 5: Follow the policy selected from step 3 to choose the action. The state then changed. Observe the result and save the experience into the memory.

Step 6: Update all parameters of the Deep Q-Network and come back to step 2.

5.2.2. Dynamic-baseline Deep Q-Network (Db-DQN)

Because $\bar{R}_\eta$ can be any number between 0 and 1, we expect the exploration rate to be more flexible, that - ϵ should have a more extensive set of values and better adaptability to the changing of $\bar{R}_\eta$. One idea is creating a function $\mathcal{F}$ (exploration-controlling function) that can receive $\bar{R}_\eta$ as input and return a value of ϵ as output which must satisfy 2 conditions simultaneously:

$$0 <= \mathcal{F}(x) <= 1, \forall x \in [0; 1] \textbf{ and } \text{ if } x_1 > x_2, \mathcal{F}(x_1) <= \mathcal{F}(x_2)$$

As mentioned before, we also need a baseline β that as long as $\bar{R}_\eta \leq \beta$, ϵ stays unchanged. Our function is then defined as:

$$\mathcal{F}(\bar{R}_\eta) \doteq max(min(\epsilon_1, \epsilon_1 - \kappa(\bar{R}_\eta - \beta)), \epsilon_2) \tag{19}$$

where:

- ϵ_1 is the max value that the exploration rate can be.
- ϵ_2 is the min value that the exploration rate can be.
- κ is the coefficient determine to reduction speed of ϵ.

The flow of Db-DQN is illustrated as Figure 3.

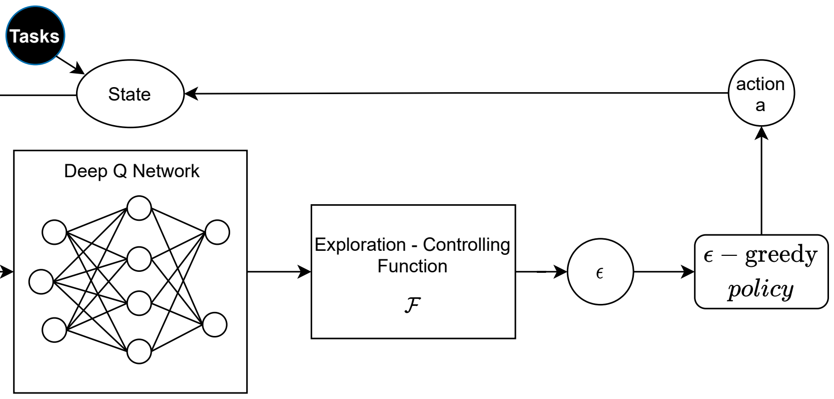

Tasks
State
action a
Deep Q Network
Exploration - Controlling Function
$\mathcal{F}$
ϵ
ϵ − greedy *policy*

Figure 3: Dynamic-baseline Deep Q-Network.

Step 1: Initialize η, β, ϵ_1, $\epsilon_2 < \epsilon_1$, κ.

Initialize a queue $\bar{R}$ with a volumn of η that stores rewards of some nearest actions.

Step 2: Same as step 1 in Static-baseline Deep Q-Network.

Step 3: Same as step 2 in Static-baseline Deep Q-Network.

Step 4: Use the exploration-controlling function to calculate ϵ.

Step 5: Follow ϵ-greedy policy with ϵ taken from step 3 to select an action. The state then changed. Observe the result and save the experience into the memory.

Step 6: Update all parameters of the Deep Q-Network and proceed to step 2.

Now, if we define the exploration-controlling function as:

$$\mathcal{F}(x) = \begin{cases} \epsilon & \text{if } x <= \beta \\ \epsilon' & \text{if } x > \beta \end{cases} \quad \forall x \in [0, 1] \tag{20}$$

The corresponding Db-DQN will be running exactly the same as Sb-DQN. Thus, Sb-DQN is a special case of Db-DQN. Therefore, only pseudocode of Db-DQN is mentioned in Algorithm 1.

Algorithm 1 Dynamic-baseline Deep Q-Network.

Input: $\mathcal{S} = \{d(A, F_1), \mathcal{C}(F_1), \mathcal{R}(F_1), ..., d(A, F_n), \mathcal{C}(F_n), \mathcal{R}(F_n), \mathcal{C}(F_0), R(F_0)\}$, $M_i^t = \{w_i^t, p_{i,in}^t, r_i^t\}$
Output: Index of server which executes the task.

```
1:  Initialize $\theta$ of the policy $\pi(a|s, \theta)$
2:  Initialize $\theta^* = \theta$ of the policy target $\pi(a|s, \theta^*)$
3:  Initialize replay memory $D$ whose capacity is $N$.
4:  K is queue of tasks awaiting scheduling
5:  Initialize $\eta \in \mathbb{N}$, $\beta \in (0; 1)$, $\epsilon_1 \in [0; 1]$, $\epsilon_2 < \epsilon_1 \in [0; 1]$, $\kappa > 0$.
6:  Queue $\bar{R} \leftarrow \varnothing$ of size $\eta$, $sum \leftarrow 0$, $t \leftarrow 0$, $\bar{R}_\eta \leftarrow 0$
```

```
7:  for each time slot $t$ do
8:      while K $\neq \varnothing$ do
9:          $M_i^t \leftarrow K.dequeue()$
10:         $s(M_i^t) \leftarrow (S, M_i^t)$        ▷ $S$ is status of system, $M_i^t$ is information of the $i^{th}$ task
11:         $\epsilon \leftarrow \mathcal{F}(\bar{R}_\eta)$
12:         $a(M_i^t) \leftarrow_a Q(s(M_i^t), a; \theta)$ & $\epsilon - greedy$ policy
13:         With $a(M_i^t)$, calculate $T(M_i^t) \rightarrow r(M_i^t)$
14:         $t \leftarrow t + 1$, $sum \leftarrow sum + r(M_i^t)$
15:         if $t > \eta$ then
16:             $temp \leftarrow \bar{R}.dequeue()$        ▷ temp stores the $\eta^{th}$ nearest observed reward
17:             $t \leftarrow t - 1$, $sum \leftarrow sum - temp$
18:         end if
19:         $\bar{R} \leftarrow \bar{R}.enqueue(r(M_i^t))$
20:         $\bar{R}_\eta \leftarrow sum/t$
21:         Update parameter[25]
22:     end while
23: end for
```

5.3. DQN combined with Fuzzy logic

5.3.1. Fuzzy Deep Q-Network in Offloading scheme

In Deep Q-Network, the Q-values are approximated by a neural network. As a result, its accuracy is severely affected by the parameters set up randomly at the start. Although the performance would improve through time due to backpropagation, generally speaking, the model would not perform well in the early stage. A new approach is using a heuristic method to prevent the models from making too many bad decisions. Fuzzy Logic is our proposal, leading to a new algorithm which we call “Fuzzy Deep Q-Network in Offloading scheme” (FDQO). Figure 4 depicts the overview of FDQO.

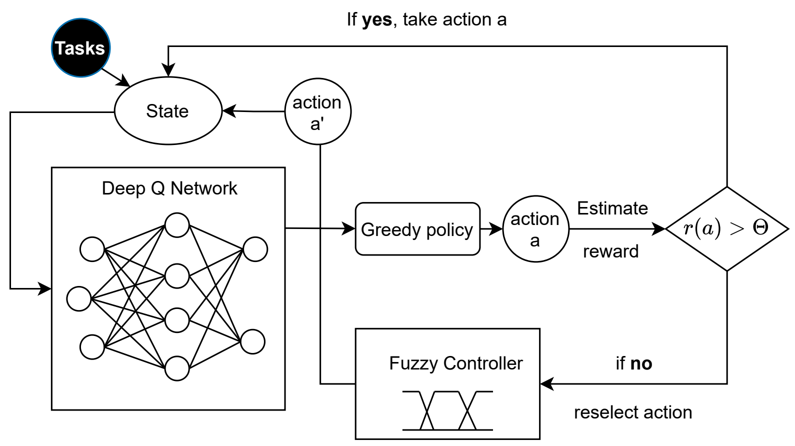


Figure 4: Fuzzy Deep Q-Network in Offloading scheme.

Fuzzy Controller (FC) will assign tasks that requires a quick response to servers with large available resources. Our FC contains two types of fuzzy sets [19]. The first type comprises three fuzzy sets that address the tasks' urgency level: top-priority (T-P), normal-priority (N-P), and low-priority (L-P) for low, medium, and high latency respectively. The second Fuzzy type addresses the servers' busy status: ”high”, ”medium”, and ”low”. From here, we will use 3 membership functions to measure the membership of fuzzy elements. We can use both the membership functions and Fuzzy rules to to calculate the probability of choosing an action quickly compare to DQN in the early stage.

There will be a threshold Θ that decides whether the ”DQN model” or Fuzzy logic's predefined rules (”Fuzzy Controller”) would be used to offload tasks. At first, DQN would suggest an action by the greedy policy. If the estimated reward of this selection is greater than threshold Θ, the action will be selected. Otherwise, the action would be replaced by another appointed by FC.

Step 1: Initialize θ.

Step 2: Use the Deep Q-Network to estimate the Q-value for each action.

Step 3: Suggest an action by ϵ-greedy policy.

Step 4: Compare the estimated reward $r(a)$ of the action in step 3 with the threshold Θ. If not surpassing the threshold, the action will be replaced by the one suggested by the Fuzzy Controller.

Step 5: Offload the task to the server in base on selected action. The state then changed. Observe the result and save the experience into the memory.

Step 6: Update all parameters of the Deep Q-Network and come back to step 2.

The details of FDQO is presented in Algorithm 2.

Algorithm 2 Fuzzy Deep Q-Learning in Offloading Scheme.

Input: $\mathcal{S} = \{d(A, F_1), \mathcal{C}(F_1), \mathcal{R}(F_1), ..., d(A, F_n), \mathcal{C}(F_n), \mathcal{R}(F_n), \mathcal{C}(F_0), R(F_0)\}$, $M_i^t = \{w_i^t, p_{i,in}^t, r_i^t\}$
Output: Index of computing devices which execute the task.

```
 1: Initialize $\theta$, $\theta^*$, $D$, $N$, $K$
 2: Queue $G \leftarrow \varnothing$
 3: for each time slot $t$ do
 4:     while K $\neq \varnothing$ do
 5:         $M_i^t \leftarrow K.dequeue()$
 6:         $s(M_i^t) \leftarrow (S, M_i^t)$        ▷ $S$ is status of system, $M_i^t$ is information of the $i^{th}$ task
 7:         $a(M_i^t) \leftarrow_a Q(s(M_i^t), a; \theta)$
 8:         With $a(M_i^t)$, estimate $T(a) \rightarrow r(a)$
 9:         if $r(a) \leq \beta$ then
10:             Use fuzzy controller to reselect $a(M_i^t)$        ▷ Refer to [19]
11:         end if
12:         System executes $a(M_i^t)$ and return reward $r(M_i^t)$
13:         if M $\neq \varnothing$ then
14:             $\big(s(M_{i-1}^t), a(M_{i-1}^t), r(M_{i-1}^t)\big) \leftarrow G.pop()$
15:             $D.add\,\big((s(M_{i-1}^t), a(M_{i-1}^t), r(M_{i-1}^t), s(M_i^t))\big)$
16:         end if
17:         $G.push\,((s(M_i^t), a(M_i^t), r(M_i^t)))$
18:         Update parameter[25]
19:     end while
20: end for
```

5.3.2. Proposed Baseline Fuzzy Deep Q-Network in offloading scheme

As we can see in figure 4, for each step, if the estimated reward is under a specific value (threshold Θ), the agent will select another action according to the FC. This reliance on FC can bring on some unique risks. Assuming that, at a specific time step, the optimal action with the highest expected return has the estimated reward less than Θ. Then, the new action taken from the FC may differ from the optimal one. This matter may hinder the improvement of the performance. To minimize the possible errors from FC, we will propose another solution that can reduce the Fuzzy Logic usage. We define this solution as a hybrid algorithm that combines FDQO and Db-DQN - baseline Fuzzy Deep Q-Network in offloading scheme (b-FDQO). The flow of b-FDQO is illustrated in Figure 5.

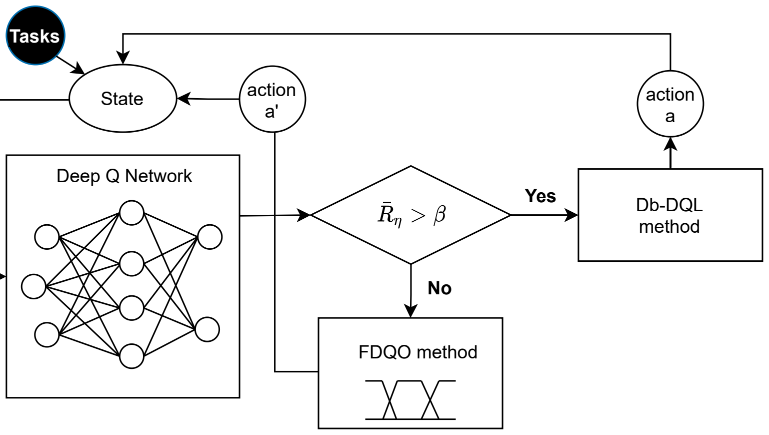


Figure 5: Baseline Fuzzy Deep Q-Network in offloading scheme.

Like Baseline Deep Q-Network, a queue $\bar{R}$ will be used to store the maximum of η nearest rewards. $\bar{R}_\eta$ is the average reward of these rewards. If the value of $\bar{R}_\eta$ is greater than the value of baseline β, the action will be chosen by Db-DQN policy; otherwise, the implemented FDQO model would select a different action with a similar process mentioned above. The details of b-FDQO is presented in Algorithm 3.

Algorithm 3 Baseline Fuzzy Deep Q-Learning in Offloading Scheme.

Input: $\mathcal{S} = \{d(A,F_1), \mathcal{C}(F_1), \mathcal{R}(F_1), ..., d(A,F_n), \mathcal{C}(F_n), \mathcal{R}(F_n), \mathcal{C}(F_0), R(F_0)\}$, $M_i^t = \{w_i^t, p_{i,in}^t, r_i^t\}$
Output: Index of computing devices which execute the task.

```
1: Initialize $\theta$, $\theta^*$, $D$, $N$, $K$
2: Initialize $\eta \in \mathbb{N}$, $\beta \in (0;1)$, $\epsilon_1 \in [0;1]$, $\epsilon_2 < \epsilon_1 \in [0;1]$, $\kappa > 0$.
3: Queue $\bar{R} \leftarrow \varnothing$ of size $\eta$, $sum \leftarrow 0$, $t \leftarrow 0$, $\bar{R}_\eta \leftarrow 0$
4: Queue $G \leftarrow \varnothing$
5: for each time slot $t$ do
6:     while K $\neq \varnothing$ do
7:         if $\bar{R}_\eta > \beta$ then
8:             {Line 9-14 in algorithm 1}
9:         else
10:            {Line 5-17 in algorithm 2}
11:        end if
12:        if $t > \eta$ then
13:            $temp \leftarrow \bar{R}.dequeue()$          ▷ temp stores the $\eta^{th}$ nearest observed reward
14:            $t \leftarrow t - 1$, $sum \leftarrow sum - temp$
15:        end if
16:        $\bar{R} \leftarrow \bar{R}.enqueue(r(M_i^t))$
17:        $\bar{R}_\eta \leftarrow sum/t$
18:        Update parameter[25]
19:    end while
20: end for
```

6. Experimental Results

6.1. Experiment methodology

We consider a simulation system that runs in practice between 8 a.m. and 7 p.m. Tasks come to the system randomly during this time. However, in order to control and evaluate the system's stability as well as detect abnormal errors, we break down the above period of time into 5-minute intervals.

Table 1: Parameters concerning the tasks

Properties	Values	Units
Number of task in a time slot	[900;1200]	
Input data size	[1000,2000]	Kb
Output data size	[100,200]	Kb
Number of CPU cycles	[500,1500]	Megacycles
Time threshold	[1,3]	second

The tasks are represented using the same feature set as proposed in [24, 26]. The base station's transmission power (P_r) is 46 dBm and the channel bandwidth (W) is 10 MHz. The background noise power (σ^2) is -100 dBm and the path loss exponent (α) is 4.

For Vehicular Edge location and route information, we use the Seattle bus trace[27]. The Seattle bus trace includes data of about 1,200 buses in Seattle, Washington, USA. In this paper, we use information about the IDs, time, and coordinates of 3 buses from 8 a.m. to 7 p.m. on October 31, 2003. The computational capacity of Vehicular Edge server 1, 2, 3, and the local server is 1 Ghz, 1.2 Ghz, 1 Ghz, and 3 Ghz, respectively. We assume that the queue of each computing device is not limited.

In the next section, our proposed algorithms with some other existing models will be taken into experiment for the simulation system stated above. The first subsection focuses on the performances of Q-Learning based algorithms which are not related to Fuzzy Logic. After that, the performance of FDQO with different threshold will be discussed. The third subsection is the most significant part, where the performance of b-FDQO is compared to other algorithms, including the baseline method and FDQO, etc in some concerned aspects. In each subsections, the average QoE on each time slot of each related algorithm are represented before any further analysis. Average QoE of the first 5 time slots, all time slots and the last 30 time slots is focused in the first subsection. They respectively demonstrate the optimism of QoE at the beginning, and overall performance of each algorithm. FDQO algorithms, however, concentrate only on the average QoE of all running time slots. The third subsection will try to answer the two questions that draw our attention the most: Which algorithms can get the most optimistic performance at the very begining steps and which algoritms bring the highest return at the long run. Besides, percentage of exploration (RL algorithms without Fuzzy Logic) and percentage of using Fuzzy Controller (FDQO and b-FDQO) helps us deeper understand how each algorithm works and its correlation with the performance.

6.2. Result and evaluation

6.2.1. Non-Fuzzy Q-learning based algorithms evaluation

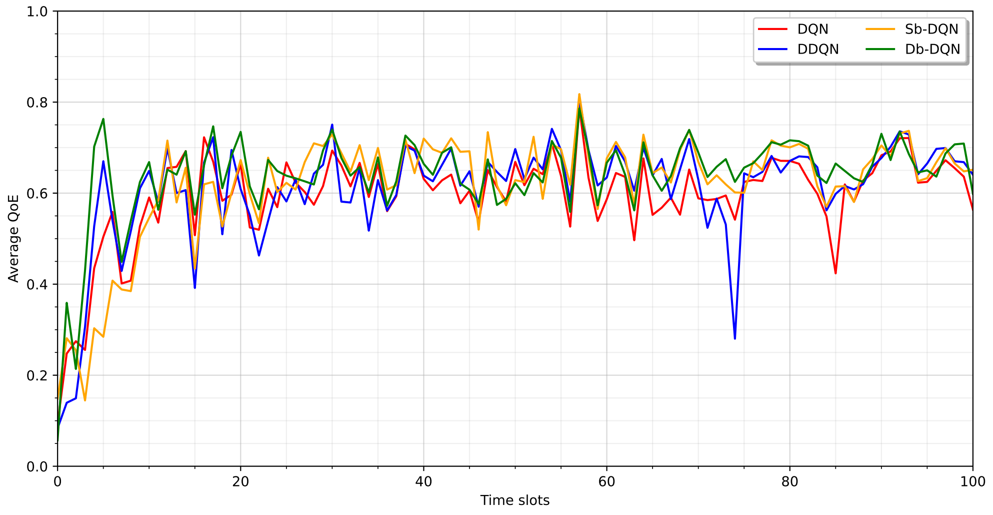


Figure 6: The average QoE through each time slot of DQN, DDQN, Sb-DQN, Db-DQN.

Figure 6 showcases the QoE performance of four Q-Learning based models without incorporating a Fuzzy Controller. Among these algorithms, the models have a common issue of low performance within the first five episodes. The average QoE scores gradually increase and fluctuate, starting from episode 15. The most unstable model is DDQN with high fluctuation magnitude and a sudden drop in average QoE at episode 74. The Sb-DQN model seems to have the lowest growth in the first 15 episodes but performs much better in later episodes. The model that performs best is the Db-DQN model with the fastest learning rate in the first ten episodes and seems to has the highest QoE score over time. Thus, Figure 6 shows the need for a new technique with relatively consistent high performance throughout all episodes instead of the low early performance.

Figure 7 indicates that Db-DQN is the best model among the non-Fuzzy models by comparing the average QoE values in three time slots period: first five episodes, all episodes, and last 30 episodes. Here, Db-DQN scores the highest in all three categories, which means Db-DQN has the best early learning rate and the highest overall performance and long-term growth potential among non-fuzzy models. Especially with the concern of low early performance, Db-DQN has better early results than other non-Fuzzy models by about 35%.

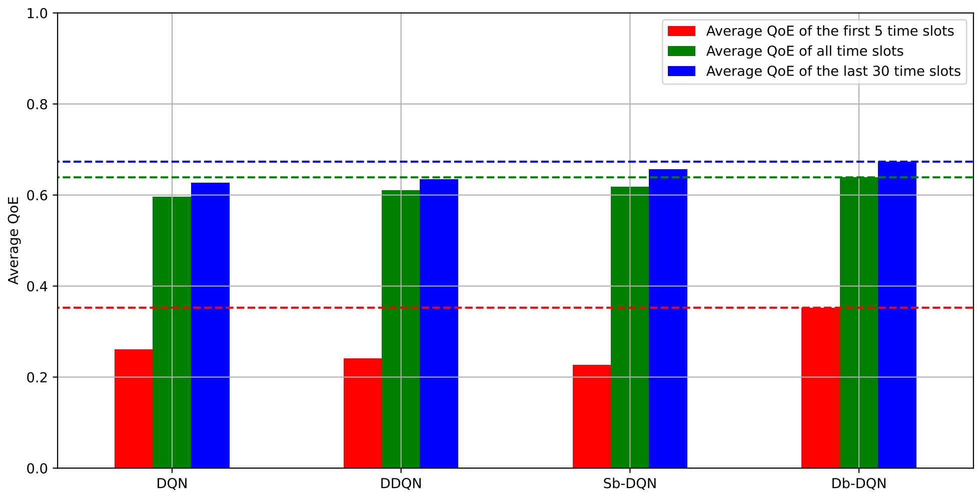
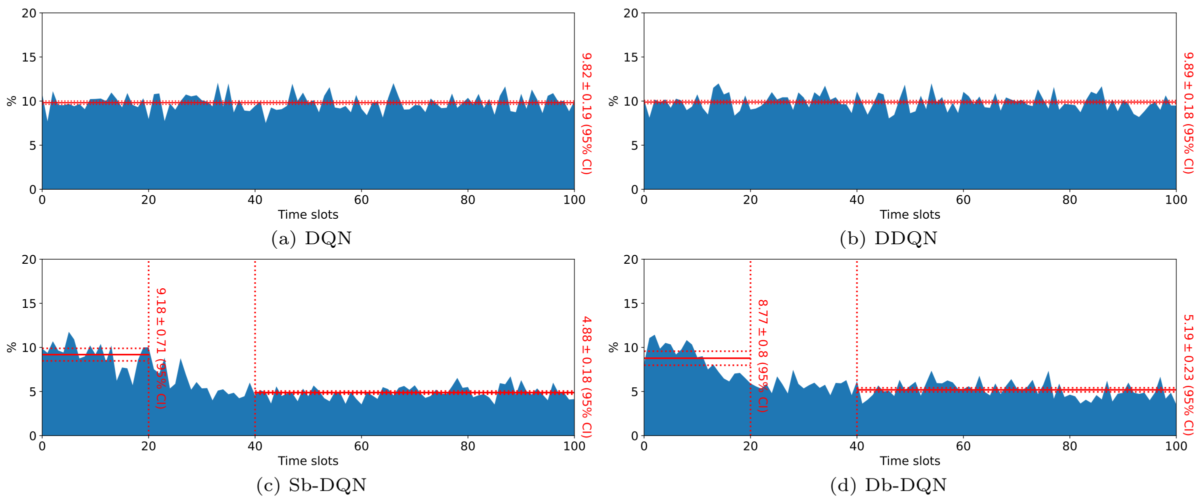


Figure 7: The average QoE of DQN, DDQN, Sb-DQN, Db-DQN in some time slot periods.



(a) DQN (b) DDQN (c) Sb-DQN (d) Db-DQN

Figure 8: Exploration rate of DQN, DDQN, Sb-DQN, Db-DQN.

Since non-Fuzzy models are data exploration intensive, Figure 8 compares the difference in exploration rates between these models. Figures 8a and 8b for DQN and DDQN show a stable exploration rate of around 10%, the margin of error for 95% confidence interval of mean QoE score for both models is small (< 0.2%). This constant small margin of error means DQN and DDQN fail to decrease their learning rate as they adapt to the environment over time. On the other hand, Sb-DQN and Db-DQN show much more promising results with clearly falling data exploration rates over 100 episodes (from 10% to 5%) with margin of error dropping from 0.8% to 0.2%. Comparing to DQN and DDQN, Sb-DQN and Db-DQN models show signs of dropping exploration rate overtime and stablize with smaller margin of error in later episodes. Combining with findings in figures 6 and

7, we conclude that Db-DQN is the best non-Fuzzy model with a faster learning rate, higher overall performance, and faster rate of adapting to a new environment (it requires less data exploration over time).

6.2.2. FDQO evaluation

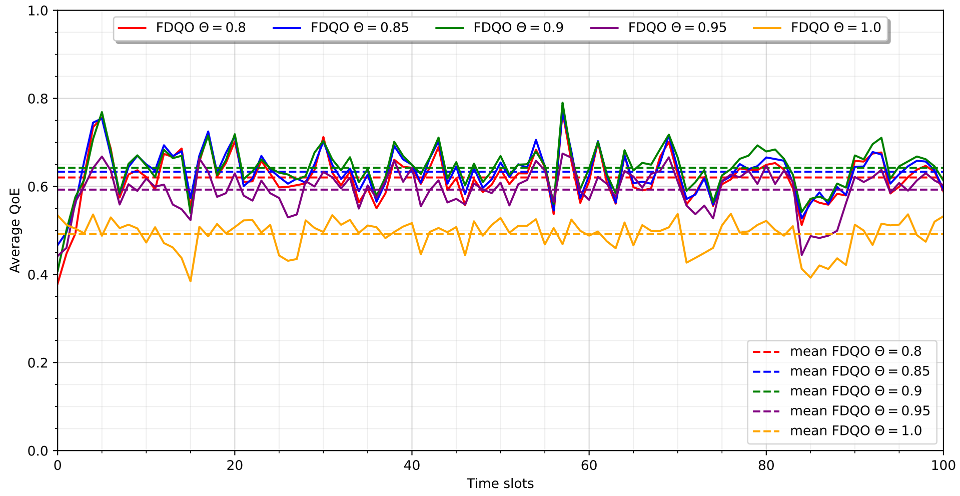


Figure 9: The average QoE of different thresholds of FDQO.

Figure 9 then manages to compare the QoE performance of above fuzzy controller thresholds. There are two major findings from this figure. First, we can see that Q-learning based algorithms should have a considerate level of data exploration since the overused of Fuzzy rules can make the whole model under-performs - thresholds of 0.95 and 1.0 have lowest overall QoE scores. On the other hand, the model shouldn't rely too much on DQN's data exploration since a lower threshold of using Fuzzy controller can still lower the model performance (the average QoE value for threshold 0.8 is lower than that of threshold 0.85 and 0.9). From Figure 9 statistics, we decide that the threshold of 0.9 for Fuzzy Controller yields the best overall performance and stresses the importance of balancing tasks allocation using DQN and Fuzzy rules. We will apply 0.9 threshold for Fuzzy controller in later experiments.

As mentioned above, the proposed FDQO algorithm utilizes Fuzzy controller in the early stage to boost the QoE performance while providing an instruction for DQN to learn. Figure 10 compares multiple tested thresholds of relying on Fuzzy controller in all episodes with higher threshold means more reliance on Fuzzy rules (mean FDQO threshold = 1.0 is relying on Fuzzy controller only). There are signs of Fuzzy controller being used more in early episodes to support the performance of DQN (60% – 90%) in the first 5 time slots. It's also clear that the higher the threshold, the bigger the blue area (the rate of using Fuzzy controller) becomes in all episodes. Despite the early benefits, there can be a potential drawback of this model that the rate of Fuzzy controller does not drop after episode 5 (constant small fluctuations from the mean with small margin of error of 1%). This constant rate of Fuzzy rules can limit the learning potential in late episodes and lower the performance of DQN in long term.

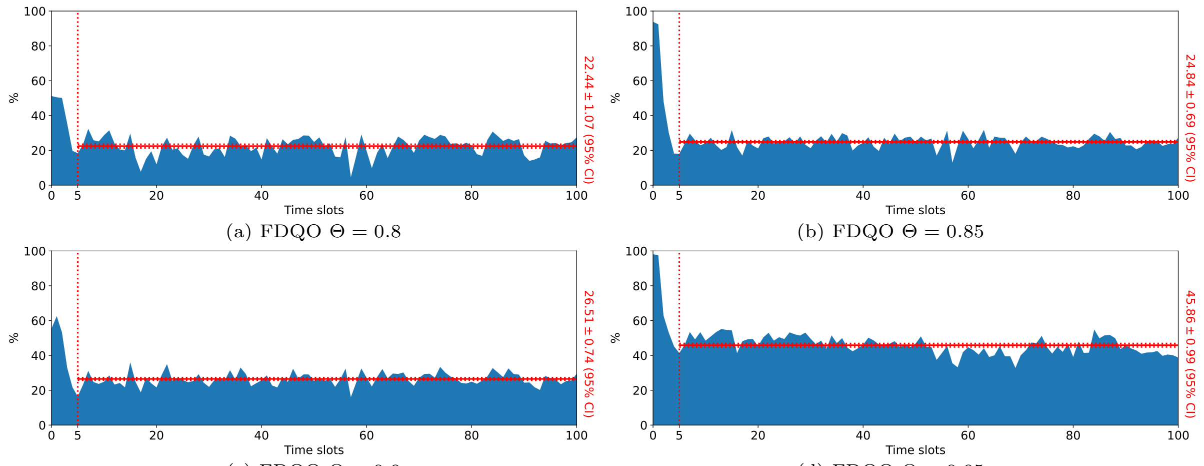


Figure 10: Percentage of using Fuzzy Controller with different FDQO thresholds.

6.2.3. b-FDQO evaluation

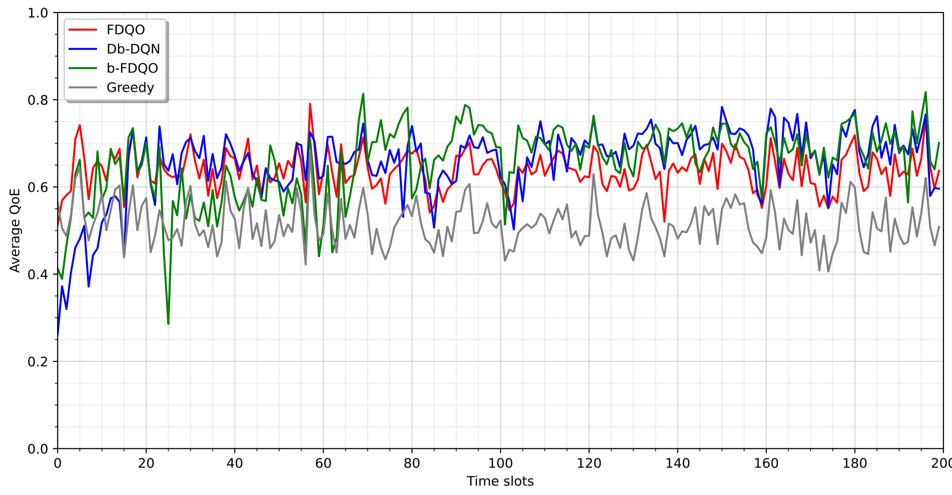


Figure 11: Average QoE through time slots of b-FDQO vs others.

After seeing the baseline addition having an effect of Fuzzy controller, our team proceed to compare the performance of two FDQO based models: FDQO and b-FDQO with two other non-Fuzzy models: Db-DQN and Greedy in Figure 11. We are comparing to these two models specifically because Db-DQN is shown to have the best overall performance among non-fuzzy algorithms but it still has weak initial learning results while Greedy is a simple, intuitive algorithm that can have relatively high initial QoE score but can not improve the performance in later episodes.

From Figure 11, we can see that both Fuzzy models have high initial QoE scores similar to the Greedy method (around 0.5 average QoE in the first 5 episodes compared to 0.3 for Db-DQN). These high starting QoE scores prove the benefit of utilizing Fuzzy Controller in the early data exploration process. Furthermore, since the Fuzzy-Logic-utilizing models have a balance threshold that help prevent the agent from taking the low-reward action, both FDQO and b-FDQO still have stable high results compare to that of Db-DQN in the long run. Specifically, the FDQO algorithm (red line) has a faster initial learning rate and the highest QoE scores for the first 15 episodes. The b-FDQO model, on the other hand, has a slower beginning learning rate but gets higher later performance, even as good as that of Db-DQN, because b-FDQO puts more limits on Fuzzy controller usage. It will later be shown in Figure 14 that b-FDQO has no Fuzzy Controller usage after time step 30, making it behaves exactly like a normal Db-DQN.

Figures 12 and 13 gives a more compacted picture of QoE performances between the multiple FDQO models with different thresholds and the non-Fuzzy models.

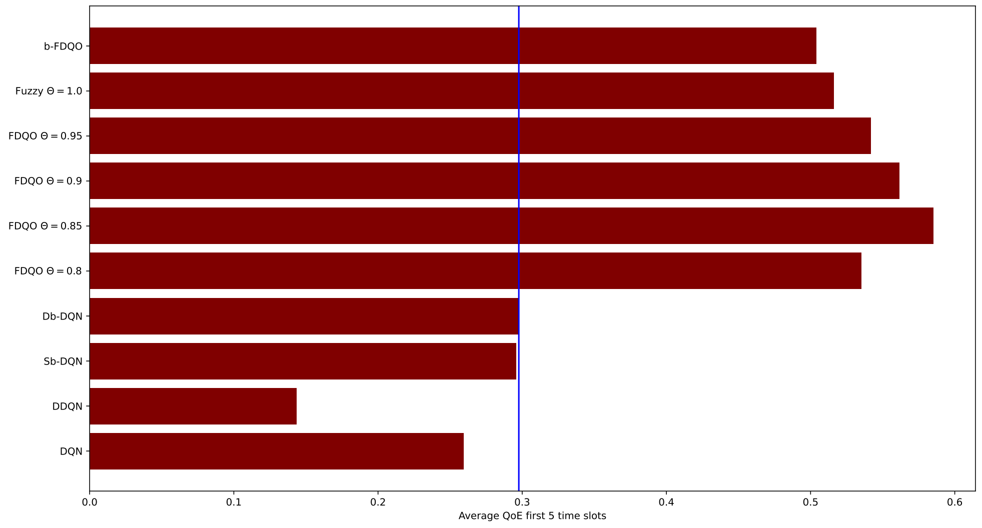


Figure 12: Performance in the first 5 time slots.

Figure 12 is a bar graph visualizing the average QoE score in the first 5 time slots. For this graph, we run each algorithm 30 times and take the mean of average QoE scores in the first 5 time steps. We can see that FDQO has the highest early average QoE rating comparing to other non-Fuzzy models with FDQO $\Theta = 0.85$ has the highest score of (0.57) compared to that of Db-DQN

(0.30), which again shows that Fuzzy-based models have much better initial performance compare to pure DQN. It's also interesting to see that higher rate of Fuzzy controller in the early stage does not always yield better early results (for thresholds > 0.85, the early QoE score drops as threshold value increases). b-FDQO, though does not have equivalent early performance as FDQO algorithm, it still has a relatively high result and outperforms all the Q-Learning based algorithms which doesn't get support from the Fuzzy Controllers.

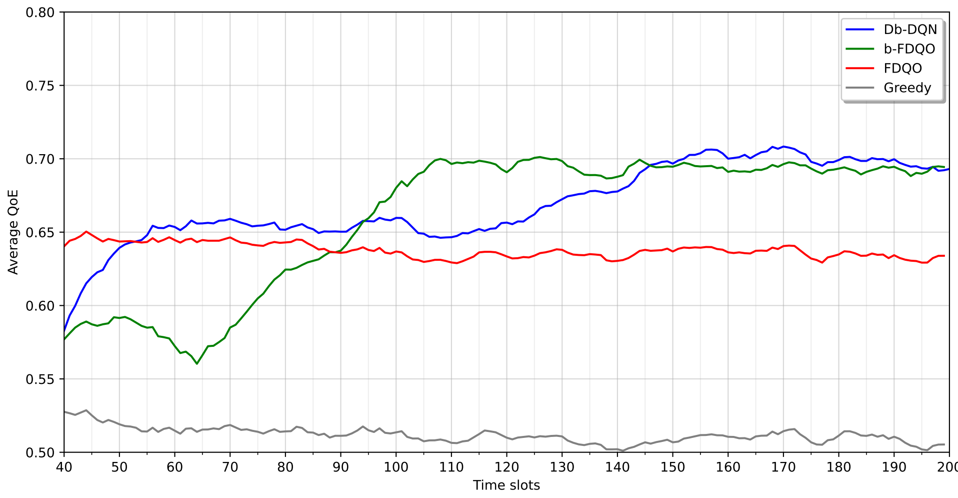


Figure 13: Moving average b-FDQO vs others.

Figure 13 is a moving average graph that takes the average of every 40 closest episodes at each time slot from the 40^{th} one. By looking at the later half of the experimental result, we can see that the FDQO model does not have a steady growth and even slightly decline in QoE score. On the other hand, the two models Db-DQN and b-FDQO both improve their performances consistently. Furthermore, the high results of b-FDQO and Db-DQN in the last 50 time slots shows that the reliance on the Fuzzy Controller is a factor that hinders the agents from learning and yielding better in the future. It's also worth noticing that b-FDQO seems to achieve optimal solution faster than Db-DQN by stabilizing at 0.7 QoE score from time slot 110 before Db-DQN at time slot 150.

Figure 14 represents the usage rate of Fuzzy controller when apply b-FDQO model. The graph only shows 2 spikes of Fuzzy Controller in the first 5 time slots and at 25 time slots, which means that including a baseline for FDQO can hinder the Fuzzy Controller from affecting the offloading decision in later time slots. This baseline method eliminates FDQO's issue of having Fuzzy Controller limiting the learning process, which can also explains why b-FDQO can still reach high QoE score that is both faster and comparable to Db-DQN in Figure 13.

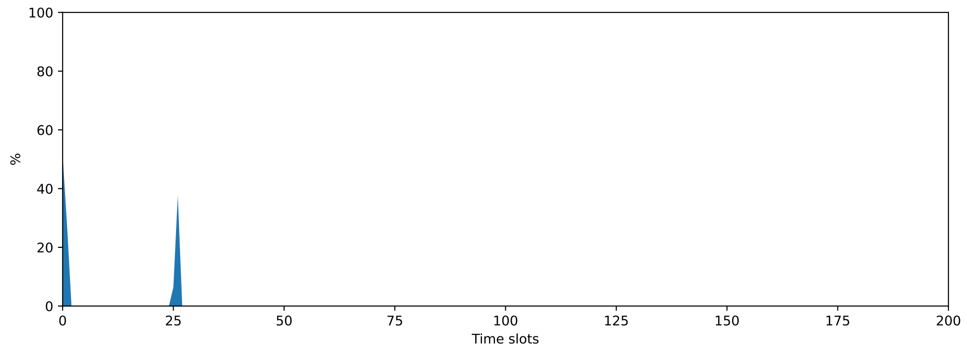
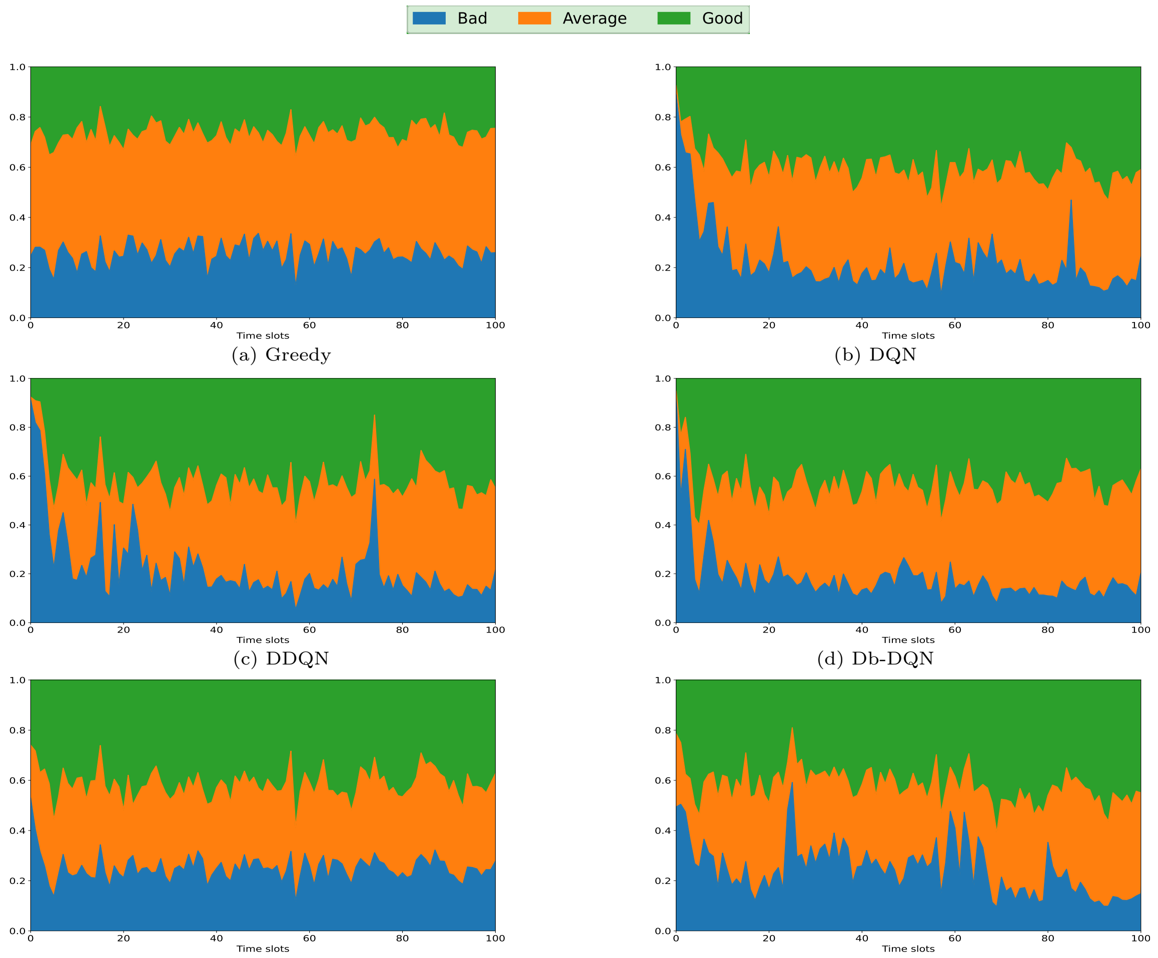


Figure 14: b-FDQO Percentage of using Fuzzy Controller.



(a) Greedy (b) DQN (c) DDQN (d) Db-DQN (e) FDQO (f) b-FDQO

Figure 15: Ratio of quality of tasks through time slots.

Figure 15 compares the quality of offloaded tasks by all models in three categories: Good, Average, and Bad. Figures 15b, c, and d show that all pure DQN models have much low quality offloaded tasks in early time slots with a high margin of Bad quality in the first five time slots (Blue area). Greedy model (Figure 15a), though it has a low level of Bad early tasks, has much fewer overall Good offloaded tasks than other Deep Q Network models. Furthermore, there can be potential inconsistency with DQN models as Figures b and c show sudden high level of Low quality tasks in late episodes 85 and 75 for models DQN and DDQN respectively.

Fuzzy-based models, on the other hand (Figure 15e, f), both have lower early Bad offloaded tasks (42%) compared to (82%) of DQN models - roughly (50%) improvement. Moreover, FDQO models have a high consistent rate of Good tasks in all episodes, about 40%, comparable to that of the best non-Fuzzy model b-DQN.

7. Conclusion

In this study, we proposed two approaches of Value-based reinforcement Learning: Baseline DQN and b-FDQO to tackle the task offloading problem in Delay Constraint Vehicular Edge Computing. The Sb-DQN and Db-DQN models does not require the construction of Fuzzy Controller. However, they suffer from slow early learning velocity and poor initial QoE scores. Then FDQO models with the support of Fuzzy Controllers can overcome that weakness. Nonetheless, this improvement comes with a price - the Fuzzy Controller itself hinders the agent from getting better results in the future. Therefore, the b-FDQO model is designed by the idea of combining the strength of the listed models. The experiment results have proved that the b-FDQO outperforms all non-Fuzzy-Logic algorithms in terms of early performance while retaining the high QoE score as the other Baseline DQN in the long run.

In near future, we will continue to research and build models that take more targets other than QoE into consideration. Furthermore, we will try more techniques that have potential to improve our proposed models. One of possible suggestions is to find a better way to surpass the hindrance of Fuzzy Logic when combining with reinforcement learning process.

======= ======= =======

References

[1] I. Markit, The internet of things: a movement, not a market, IHS Market (2017).

[2] H. T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture,

applications, and approaches, Wireless communications and mobile computing 13 (18) (2013) 1587–1611.

[3] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, et al., Mobile-edge computing introductory technical white paper, White paper, mobile-edge computing (MEC) industry initiative (2014) 1089–7801.

[4] B. Huang, Z. Li, P. Tang, S. Wang, J. Zhao, H. Hu, W. Li, V. Chang, Security modeling and efficient computation offloading for service workflow in mobile edge computing, Future Generation Computer Systems 97 (2019) 755–774.

[5] X. Xu, Y. Li, T. Huang, Y. Xue, K. Peng, L. Qi, W. Dou, An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks, Journal of Network and Computer Applications 133 (2019) 75–85.

[6] H. T. T. Binh, T. T. Anh, D. B. Son, P. A. Duc, B. M. Nguyen, An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment, in: Proceedings of the Ninth International Symposium on Information and Communication Technology, 2018, pp. 397–404.

[7] H. Guo, J. Liu, H. Qin, Collaborative mobile edge computation offloading for iot over fiber-wireless networks, IEEE Network 32 (1) (2018) 66–71.

[8] B. M. Nguyen, H. Thi Thanh Binh, B. Do Son, et al., Evolutionary algorithms to optimize task scheduling problem for the iot based bag-of-tasks application in cloud–fog computing environment, Applied Sciences 9 (9) (2019) 1730.

[9] L. Liu, C. Chen, Q. Pei, S. Maharjan, Y. Zhang, Vehicular edge computing and networking: A survey, Mobile Networks and Applications 26 (3) (2021) 1145–1168.

[10] J. Liu, Q. Zhang, Code-partitioning offloading schemes in mobile edge computing for augmented reality, Ieee Access 7 (2019) 11222–11236.

[11] S. Misra, S. Bera, Soft-van: Mobility-aware task offloading in software-defined vehicular network, IEEE Transactions on Vehicular Technology 69 (2) (2019) 2071–2078.

[12] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, G. Fortino, Autonomic computation offloading in mobile edge for iot applications, Future Generation Computer Systems 90 (2019) 149–157.

[13] Y. Sun, X. Guo, J. Song, S. Zhou, Z. Jiang, X. Liu, Z. Niu, Adaptive learning-based task offloading for vehicular edge computing systems, IEEE Transactions on Vehicular Technology 68 (4) (2019) 3061–3074.

[14] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, F. Xia, Deep reinforcement learning for vehicular edge computing: An intelligent offloading system, ACM Transactions on Intelligent Systems and Technology (TIST) 10 (6) (2019) 1–24.

[15] G. Wang, F. Xu, Regional intelligent resource allocation in mobile edge computing based vehicular network, IEEE Access 8 (2020) 7173–7182.

[16] K. Bylykbashi, E. Qafzezi, M. Ikeda, K. Matsuo, L. Barolli, Fuzzy-based driver monitoring system (fdms): Implementation of two intelligent fdmss and a testbed for safe driving in vanets, Future Generation Computer Systems 105 (2020) 665–674.

[17] V. Nguyen, T. T. Khanh, T. Z. Oo, N. H. Tran, E.-N. Huh, C. S. Hong, Latency minimization in a fuzzy-based mobile edge orchestrator for iot applications, IEEE Communications Letters 25 (1) (2020) 84–88.

[18] S. P. Singh, A. Sharma, R. Kumar, Design and exploration of load balancers for fog computing using fuzzy logic, Simulation Modelling Practice and Theory 101 (2020) 102017.

[19] D. B. Son, V. T. An, T. T. Hai, B. M. Nguyen, N. P. Le, H. T. T. Binh, Fuzzy deep q-learning task offloading in delay constrained vehicular fog computing, in: 2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1–8. doi:10.1109/IJCNN52387.2021.9533615.

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

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (7540) (2015) 529–533.

[22] G. Klir, B. Yuan, Fuzzy sets and fuzzy logic, Vol. 4, Prentice hall New Jersey, 1995.

[23] M. Johnson, Optical fibres, cables and systems, Switzerland Geneva (2009).

[24] X.-Q. Pham, T.-D. Nguyen, V. Nguyen, E.-N. Huh, Joint node selection and resource allocation for task offloading in scalable vehicle-assisted multi-access edge computing, Symmetry 11 (1) (2019) 58.

[25] Z. Yang, Y. Xie, Z. Wang, A theoretical analysis of deep q-learning, in: Learning for Dynamics and Control, PMLR, 2020, pp. 486–489.

[26] L. Huang, X. Feng, C. Zhang, L. Qian, Y. Wu, Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing, Digital Communications and Networks 5 (1) (2019) 10–17.

[27] J. G. Jetcheva, Y.-C. Hu, S. PalChaudhuri, A. Kumar, S. David, B. Johnson, Design and evaluation of a metropolitan area multitier wireless ad hoc network architecture (2003).