Expert Systems With Applications

Deadline-aware Task Offloading in Vehicular Networks using Deep Reinforcement Learning

--Manuscript Draft--

Manuscript Number:	ESWA-D-23-05847
Article Type:	Full length article
Keywords:	Computation Offloading; vehicular edge computing; Deep Reinforcement learning; deep Q-learning; Internet of Vehicles.
Corresponding Author:	Reza Entezari-Maleki, Ph.D. Iran University of Science and Technology Tehran, Tehran IRAN, ISLAMIC REPUBLIC OF
First Author:	Mina Khoshbazm Farimani, M.S.
Order of Authors:	Mina Khoshbazm Farimani, M.S.
	Soroush Karimian-Aliabadi, Ph.D.
	Reza Entezari-Maleki, Ph.D.
	Bernhard Egger, Ph.D.
	Leonel Sousa, Ph.D.
Abstract:	Smart vehicles have a rising demand for computation resources, and recently Vehicular Edge Computing (VEC) has been recognized as an effective solution. Edge servers deployed in roadside units are capable of accomplishing tasks beyond the capacity which is embedded inside the vehicles. However, the main challenge is to carefully select the tasks to be offloaded considering the deadlines, and in order to reduce energy consumption, while delivering a good performance. In this paper, we consider a VEC network in which multiple cars are moving at non-constant speed and produce tasks at each time slot. Then, we propose a task offloading algorithm, aware of the vehicle's direction, based on Rainbow, a deep Q-learning algorithm combining several independent improvements to the Deep Q-Network (DQN) algorithm. This is to overcome the conventional limits and to reach an optimal offloading policy, by effectively incorporating the computation resources of edge servers to jointly minimize average delay and energy consumption. Real-world traffic data is used to evaluate the performance of the proposed approach compared to other DQN algorithms namely DQN, Double DQN, and Deep Recurrent Q-Network (DRQN). Results of the experiments show an average reduction of 18% and 15% in energy consumption and delay, respectively, when using the the proposed Rainbow DQN-based algorithm compared to the state-of-the-art. Moreover, the stability and convergence of the learning process have significantly improved by adopting the Rainbow algorithm.
Suggested Reviewers:	Anfeng Liu, Dr. Professor, Central South University afengliu@mail.csu.edu.cn
	Paolo Montuschi, Ph.D. Professor, Polytechnic of Turin Department of Electronics and Telecommunications paolo.montuschi@polito.dot.it
	Xin Liu, Ph.D. Professor, University of California Davis XinLiu@ucdavis.edu
	Yuanzhu Chen, Ph.D. Professor, Queen's University yuanzhu.chen@queensu.ca
	C. Siva Ram Murthy, Ph.D. Professor, Indian Institute of Technology Madras murthy@iitm.ac.in
	Abbas Jamalipour, Ph.D.

	Professor, The University of Sydney abbas.jamalipour@sydney.edu.au
	Jiwei Hua, Ph.D. Professor, Tianjin Normal University huajiwei@tjnu.edu.cn

Cover Letter

Dear Editor-in-Chief,

Hereby, a paper entitled "Deadline-aware Task Offloading in Vehicular Networks using Deep Reinforcement Learning" is submitted to the *Expert Systems with Applications* to be reviewed for possible publication.

Thank you in advance for your attention.

–

Sincerely yours,

Reza Entezari-Maleki
Assistant Professor,
Department of Computer Engineering,
Iran University of Science and Technology (IUST),
Tehran, Iran.

Deadline-aware Task Offloading in Vehicular Networks using Deep Reinforcement Learning

Mina Khoshbazm Farimani[a], Soroush Karimian-Aliabadi[b], Reza Entezari-Maleki[a,d,*], Bernhard Egger[c], Leonel Sousa[d]

[a] *School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran*
[b] *Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*
[c] *Department of Computer Science and Engineering, Seoul National University, Seoul, South Korea*
[d] *INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal*

Abstract

Smart vehicles have a rising demand for computation resources, and recently Vehicular Edge Computing (VEC) has been recognized as an effective solution. Edge servers deployed in roadside units are capable of accomplishing tasks beyond the capacity which is embedded inside the vehicles. However, the main challenge is to carefully select the tasks to be offloaded considering the deadlines, and in order to reduce energy consumption, while delivering a good performance. In this paper, we consider a VEC network in which multiple cars are moving at non-constant speed and produce tasks at each time slot. Then, we propose a task offloading algorithm, aware of the vehicle's direction, based on Rainbow, a deep Q-learning algorithm combining several independent improvements to the Deep Q-Network (DQN) algorithm. This is to overcome the conventional limits and to reach an optimal offloading policy, by effectively incorporating the computation resources of edge servers to jointly minimize average delay and energy consumption. Real-world traffic data is used to evaluate the performance of the proposed approach compared to other DQN algorithms namely DQN, Double DQN, and Deep Recurrent Q-Network (DRQN). Results of the experiments show an average reduction of 18% and 15% in energy consumption and delay, respectively, when using the the proposed Rainbow DQN-based algorithm compared to the state-of-the-art. Moreover, the stability and convergence of the learning process have significantly improved by adopting the Rainbow algorithm.

Keywords: Computation Offloading, Vehicular Edge Computing, Deep Reinforcement Learning, Deep Q-Learning, Internet of Vehicles.

*Corresponding Author: Reza Entezari-Maleki, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.

Email addresses: `m_khoshbazmfarimani@comp.iust.ac.ir` (Mina Khoshbazm Farimani), `skarimian@ce.sharif.edu` (Soroush Karimian-Aliabadi), `entezari@iust.ac.ir` (Reza Entezari-Maleki), `bernhard@csap.snu.ac.kr` (Bernhard Egger), `las@inesc-id.pt` (Leonel Sousa)

1. Introduction

The future is always pictured with modern and fictional cars which are more like robots than classic transportation means. This image is today somehow realized with autonomous vehicles, and we are close to the futuristic transportation style more than ever. The market of the Internet of Vehicles (IoV) was estimated to correspond to \$96 billion in 2021, and is approximately predicted to grow to \$370 billion in 2028 [1]. This remarkable investment justifies the ever-growing research attention shifted toward vehicular networks. A significant number of advances are achieved every year in the fields of auto navigation, vehicular augmented reality, autonomous driving, and intelligent object recognition [2, 3, 4]. All these new applications are pushing the available infrastructures to their limits. They require Quality of Service (QoS), while at the same time executing complex tasks on massive data captured by sensors and communicating with the network.

Strict delay constraints are common in the above-mentioned applications, and this results in high energy consumption. However, autonomous vehicles are powered by batteries, and therefore, vehicles experience excessive pressure on the onboard resources. In order to relieve the conflict between application demands and resource-constrained vehicles, Vehicular Edge Computing (VEC) has been emerged as a promising computing paradigm [5, 6, 7]. With the integration of Mobile Edge Computing (MEC) and vehicular networks, VEC supports IoV by enabling the offloading of resource-intensive tasks onto MEC servers deployed on Road Side Units (RSU) via wireless networks [8]. MEC servers can alleviate computation pressure on the vehicle's onboard resources and reduce the processing delay of these applications by pushing the computation resources in proximity to vehicles. As a result, task offloading has recently received increasing attention in VEC [9, 10, 11]. However, due to the unique characteristics of vehicular networks, especially the high mobility of nodes and dynamic channel properties, designing an efficient edge-based task offloading scheme is quite challenging.

There have been many efforts devoted to the task offloading in VEC to seek solutions to the above issues and meeting the various performance requirements. Most of them have employed the Q-learning algorithm, a model-free Reinforcement Learning (RL) method, as a potential solution to solve offloading problems [12, 13]. However, they are slow in a huge state space. The state space for vehicular task offloading environment can be vast and high-dimensional due to the numerous factors involved. To address issues in Q-learning methods, DQN algorithms, which approximate the Q-function using deep neural networks, have emerged to prove their efficiency in solving offloading decision problems with huge dimensional state and action spaces, where the agent can adapt well to complex environments via continuous interaction [2, 14]. However, the major challenge in traditional DQN algorithms is the trend to overestimation which led to unstable training and low-quality policy. As an alternative, Double DQN was proposed [15], which has two identical neural network models, where one network is used to select the best action and the other is used to estimate the value of that action. It could reduce overestimation by separating the max operation in the target into action selection and action evaluation [16]. Deep Recurrent Q-Network (DRQN) is also practical with Partially-Observable Markov Decision Process (POMDP) [17]. In DRQN, a recurrent layer is added after the convolutional part of the original DQN to handle long term dependencies. Nevertheless, adding a recurrent layer to a DQN makes DRQN trickier and longer to train compared to classical DQN.

DQN and its derivations were an important milestone; however, several drawbacks have been identified, thanks to the extensive research. Consequently, many efforts have been made to propose improvements to DQNs. Yet it is unclear which of these advancements are indeed complementary, and can be integrated into a single agent to achieve state-of-the-art performance. In this paper, we focus on making efficient task-offloading decisions for multiple vehicles moving at a non-constant speed along the road. Vehicles generate computation tasks at each time slot and RSUs can provide computing services for the vehicles moving within the coverage radius. In order to reach an efficient solution to optimize joint delay and energy consumption, we design a task-offloading algorithm that exploits the state-of-the-art DQN variation designated Rainbow. Rainbow is an integration of six improvements to the DQN algorithm, namely Double DQN, prioritized experience replay, dueling networks, multi-step learning, distributional RL, and noisy nets [18].

The main contributions of this paper are summarized as follows.

- The speed of each vehicle is assumed to be non-constant while related work have mostly simplified the case to constant speed.
- The speed of the learning process is improved by considering penalties for wrong offloading decisions. In other words, when the vehicle is out of the coverage radius of the selected RSU, a penalty is imposed on the agent.
- The optimization problem for computation offloading to simultaneously consider the delay and energy consumption of vehicles is mathematically formulated.
- The target RSU is specified when the offloading action is suggested. In some related work, it is only decided whether to offload or process locally.
- The direction of vehicles is taken into account when selecting the RSU to offload tasks.
- The agent makes offloading decisions for all moving vehicles in the network, while most of the related work assume a single vehicle.
- Real traffic data, instead of synthetic data, is used to evaluate the performance of the proposed algorithm. Extensive simulations are carried out to illustrate the efficiency of the proposed algorithm compared to the state-of-the-art.

The remaining of this paper is organized as follows. Section 2 discuses the related work in the literature. Section 3 provides background information about deep Q-learning and Rainbow algorithm. Section 4 introduces the system model, in terms of communication and computational models, and formulates the optimization problem. In Section 5, a Rainbow DQN-based algorithm is proposed to solve task offloading problem in vehicular networks. Section 6 presents numerical results of the proposed approach, and finally, the paper is concluded in Section 7, with also some guidelines for future research.

2. Related Work

Vehicular edge computing is a promising computing paradigm that has attracted considerable attention in recent years. Considering mobile nodes and the time-varying dynamic networks, obtaining an efficient offloading strategy is challenging. In this context, task offloading algorithms play a critical role in optimizing resource utilization and improving system performance. Various approaches have been presented to solve task-offloading problems in VEC models [19, 20, 21, 22, 23, 24]. The high-dimensional state and action space of task offloading in vehicular networks, coupled with the dynamic topology of the network, can pose significant challenges in solving the problem. Due to the complex and dynamic nature of the problem, heuristic algorithms or conventional optimization techniques may fail to capture the full range of possible states and actions, leading to suboptimal performance. As a result, more advanced techniques such as Deep Reinforcement Learning (DRL), which can effectively handle high-dimensional state and action spaces, have become increasingly popular in this field [25, 26, 27]. Although DRL has been successful in handling high-dimensional state and action spaces, conventional DQN algorithms may still face challenges in solving task offloading problems in vehicular networks, due to the dynamic nature of the network. Several improvements to DQN algorithms have been proposed to enhance their performance in these scenarios. These advanced DQN algorithms have shown promising results in reducing the effects of high state and action space dimensionality, and effectively handling the dynamic topology of vehicular networks [28, 29, 30, 31, 32, 33].

Although DQN algorithms have shown desirable results in solving vehicular task offloading, there have been proposed some other approaches in recent years as well. Game theory has emerged as a promising solution for task offloading in VEC systems, leveraging its ability to model strategic interactions among multiple entities and optimizing system performance under uncertain conditions. For example, the authors in [19] proposed a two-stage game theory iterative algorithm to solve the problem of load imbalance in VEC systems with RSUs. Apart from game theory, other heuristic methods have succeeded as well in partially addressing this issue. The authors in [20] proposed an iterative algorithm based on reformulation linearization and generalized benders decomposition methods to obtain the optimal solution and a heuristic algorithm to provide a sub-optimal solution to minimize the total task processing delay through Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) communication modes. Partial task offloading algorithms, such as the one in [21], are common, as well. An urban scenario under V2V and V2I communication modes was considered in [21], and a partial task offloading algorithm was proposed to reduce the task execution delay. Semi-definite relaxation is known to be an effective technique specially when facing a high number of users and tasks. This technique was investigated in [22], where multi-hop vehicle computation resources were leveraged to minimize joint response delay and cost. Another attempt to bring heuristics into use is the online Lyapunov optimization-based multi-decision-making algorithm presented in [23], where the problem of minimizing the total network delay was formulated as a mixed integer nonlinear programming. In order to speed-up the solution, AI-based imitation learning was proposed with limited training samples. The problem of task offloading was formulated in [24] to minimize average delay in the multi-access edge computing system. An ant colony optimization algorithm was proposed to find the best solution for task offloading, with a pheromone matrix initialization based on the base station load and user-base station distance. The above-mentioned ad-hoc methods, may face limitations due to the huge state and action spaces, the dynamic topology, and uncertain environments, and therefore, experience difficulties in optimizing the decision making process.

Due to the high dimensional state-action spaces, and dynamic topology changes, most recent work has focused on DRL algorithms. The authors in [25] have investigated the resource allocation problem by considering cooperation between MEC and a central cloud to guarantee the required response time in a vehicular environment. They utilized a DRL approach to deal with the large state space and real-time network state transitions. In [26], a multi-agent DRL-based algorithm was proposed to solve computation offloading problems while minimizing latency, energy consumption, and cost. In another study, a shared offloading strategy was proposed, based on DRL, to solve the offloading problem considering energy consumption and delay [27]. However, traditional DQN suffers from limitations in solving vehicular task offloading problems due to overestimation and instability issues that can cause suboptimal action selection and slow convergence. Additionally, traditional DQN requires a large amount of training data, in dynamic vehicular environments, which can be quite challenging.

Recent advances in the context of DQN algorithms have shown good potential in solving task offloading problems in VEC systems. Double DQN is an extension of DQN that addresses overestimation of action values by using the target network to select actions, leading to more stable training and better performance [28]. Dueling DQN improves the accuracy of action value estimation by decomposing it into state value and advantage value, allowing for more effective learning of the Q-function [29]. Prioritized DQN assigns higher priority to more important experiences for replay, which can improve sample efficiency and reduce training time [30]. Distributed DQN extends the basic algorithm to distributed environments, allowing for faster and more efficient learning [31]. Noisy DQN introduces random noise into the network weights, encouraging exploration and improving learning [32]. Multi-step DQN improves the learning process by taking into account multiple consecutive observations and rewards, leading to more accurate value estimation and better performance [33]. Overall, these advances on DQN have shown promising results in improving the performance and efficiency of task offloading in VEC. However, they were typically designed and evaluated independently.

To the best of our knowledge, no prior studies have explored the full potential of integrating recent advancements in DQN to enhance the optimization and acceleration of task offloading decisions. In this regard, we propose a novel approach that leverages the Rainbow algorithm, which is a combination of six state-of-the-art DQN enhancements. Our approach aims to minimize the delay and energy consumption in task offloading decisions while also taking into account the vehicle's direction. Through this integration, we aim to demonstrate the significant benefits of using the Rainbow algorithm for improving the efficiency and effectiveness of solving task offloading problems in vehicular networks.

3. Background Information

This section provides an overview of deep Q-learning to establish a foundational understanding for the proposed approach presented in this paper. Afterward, we focus on the Rainbow algorithm which has demonstrated significant successes in solving problems in complex environments. We also emphasize its effectiveness in solving the problem of task offloading in the vehicular networks.

A. Deep Q-Learning

Q-learning is a well-known classical Reinforcement Learning (RL) algorithm that aims to find the optimal policy by estimating the Q-value of each state-action pair [34]. The Q-value function $Q_{(s_t,a_t)}$ estimates the expected discounted cumulative reward for each state-action pair. The Q-learning algorithm learns the optimal Q-value function by iteratively updating the Q-values based on the Bellman equation,

$$Q_{(s_t,a_t)} = Q_{(s_t,a_t)} + \alpha \left[r_t + \gamma \cdot \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}\right) - Q\left(s_t, a_t\right) \right], \tag{1}$$

where s_t is the current state in time step t, a_t is the action taken in time step t, r_t is the immediate reward received after taking action a_t in state s_t, α is the learning rate, γ is the discount factor, s_{t+1} is the next state, and finally, a_{t+1} is the action that maximizes the Q-value in the next state s_{t+1}.

The main limitation of Q-learning is that it can achieve the optimal policy when the state and action spaces are small. With regard to the continuous state space in our proposed approach, we have infinite possibilities, and therefore, a huge state space. Consequently, traditional Q-learning algorithms may not be able to find an optimal solution. To tackle this challenge, we utilize DQN, which integrates neural network techniques in Q-learning to replace ordinary neural networks and extract high-level features from raw input data [35]. Deep neural network is used to approximate the action values for a given state s_t, input of the neural network, which can support state and action spaces with larger dimensions. To avoid overestimation and stabilize the learning process, DQN uses two neural networks; one is the main network used to represent the Q function, denoted as $Q\left(s_t, a_t|\theta\right)$, where θ represents the weight parameter of the main network, and the other is the target network used to represent $Q^{'}$ function, denoted as the $Q^{'}\left(s_t, a_t|\theta^{'}\right)$ where $\theta^{'}$ represent the weight parameters of the target network.

In the following, we describe the key components of the DQN algorithm, including action selection, experience replay memory, and DQN network training. We discuss how these elements work together to enable an agent to learn effective policies in a wide range of environments.

- **Action selection.** The agent needs to balance between exploiting its current knowledge, i.e., taking the action with the highest Q-value, and exploring new actions that might lead to even higher rewards. To do this, the agent uses an exploration strategy called ϵ-greedy. At each step, the agent chooses an action with the highest Q-value with probability $1-\epsilon$, the exploitation part, and selects a random action with probability ϵ, the exploration part, based on the current state to avoid reaching the local optimal point, as shown in Eq. (2).

$$a_t = \begin{cases} \arg\max_{a_t} Q\left(s_t, a_t\right) & \textit{with probability } 1-\epsilon \\ \text{Randomly choose from action space} & \textit{with probability } \epsilon \end{cases} \tag{2}$$

- **Experience replay memory.** Experience replay memory refers to a technique used to improve the efficiency of the learning process by storing and reusing past experiences of an agent in its interactions with an environment. Each experience typically consists of the agent's current state, the action it took, the resulting reward, and the next state. During the learning process, the agent can randomly sample experiences from the replay buffer and use them to update its Q-values, which estimate the expected future rewards for taking different actions in different states. By randomly sampling experiences from the replay buffer, the agent can break the correlation between consecutive experiences and reduce the likelihood of getting stuck in local optima. Experience memory is a powerful technique that has been shown to improve the efficiency and stability of DQN learning, enabling agents to learn from past experiences and avoid overfitting to recent experiences.

- **DQN network training.** Finally, the DQN network is trained by minimizing the square temporal difference error between the target value of the target network and the estimated value of the main network. Therefore, the loss function is defined as Eq. (3)

$$L_t\left(\theta_t\right) = \left(y_t - Q\left(s_t, a_t | \theta_t\right)\right)^2, \tag{3}$$

where y_t is the target value of the target network for time step t, and $Q\left(s_t, a_t|\theta_t\right)$ is the estimated value of the main network with parameters θ_t. Also y_t can be defined by Eq. (4).

$$y_t = r_t + \gamma \max Q^{'}\left(s_{t+1}, a_{t+1} | \theta_t^{'}\right). \tag{4}$$

DQN is still facing major decision-making challenges in dynamic environments, hence since the introduction of DQN, several independent improvements have been made by the DRL community. However, it is not determined which of them can be integrated to provide the state-of-the-art DQN learner. In our case, we exploit Rainbow which is an extended DQN that combines several improvements into a single learner. DRL algorithms like Rainbow are particularly well-suited for this kind of task offloading problem, because they can learn to make decisions on complex and dynamic environments.

B. Rainbow

In this section, we briefly introduce the improvements of the Rainbow algorithm compared to the DQN. For more information about the Rainbow algorithm refer to [18, 36, 37].

- **Double DQN.** One challenge with traditional DQNs is their tendency to overestimate action values, leading to suboptimal values and a low-quality policy, particularly in complex environments such as vehicular networks. This overestimation bias can result in slow convergence and suboptimal task-offloading decisions, leading to larger time and energy consumption. To address this issue, Double DQN has been proposed as a means of reducing overestimation bias due to estimation errors. By separating the selection and evaluation processes, Double DQN allows one process to be used for selecting actions, while the other is used for estimating their values. This approach effectively decouples the action selection and value estimation processes, which has been shown to reduce overestimation bias and improve the overall performance of the agent [15, 18, 28].

- **Prioritized experience replay.** In a traditional DQN, the agent stores all of its experiences, i.e., the state, action, reward, and next state transitions, in a buffer, and then samples a random subset of experiences from the buffer to train its Q-network. However, each transition tuple saved in memory (s_t, a_t, r_t, s_{t+1}) has a different level of priority due to the different times of their occurrence. For instance, transition data during the daytime has a higher priority than at night due to higher traffic. Thus, the agent selects the training data according to their priority instead of selecting them uniformly at random from experience replay memory. Prioritized experience replay assigns a priority to each experience based on the amount of knowledge that the agent can learn from the experience and samples experiences from the buffer in a way that prioritizes experiences [30, 36]. The key idea behind prioritized experience replay is that some experiences may be more valuable for learning than others. For example, experiences that led to unexpected outcomes or large rewards may be more informative than experiences that did not. By assigning priorities to experiences, the agent can focus on the most valuable experiences and learn more efficiently from them. In the context of vehicular task offloading, prioritized experience replay could help the offloading system learn more efficiently from its experiences.

- **Dueling networks.** The Dueling DQN architecture represents an important advancement in value-based RL that addresses the limitations of traditional DQN. In traditional DQN, a neural network is used to estimate the Q-values of each possible action given a state. However, the Dueling DQN splits the network into two streams: one for estimating the value of the current state and another for estimating the advantage of each possible action [29, 37]. By combining these two streams, the Dueling DQN is able to more accurately estimate the Q-values of each possible action in a given state. This is particularly important in situations where many actions have similar values, as the advantage stream allows the agent to better differentiate between them.

Furthermore, it is unnecessary to know the value of each action at every time step. For instance, in a vehicular network, it is not necessary to know which action to take until vehicles have a task to process. In order to optimize computational efficiency, it is often preferable to neglect the exhaustive exploration of all possible actions in states of interest.

Moreover, the Dueling DQN architecture can be effectively used for real-time optimization of vehicular task offloading scenarios. By training the network with a dataset of such scenarios, the agent can learn to optimize the allocation of tasks based on the current state of the system. This can result in faster and more energy-efficient task completion. The use of Dueling DQN, in this context, allows the agent to more accurately estimate the value of each possible action, leading to better task offloading decisions.

- **Multi-step learning.** In traditional DQN algorithms, the agent updates its Q-values based solely on the immediate reward received after taking an action. However, in multi-step learning DQN, the agent receives a sequence of rewards and takes a sequence of actions before updating its Q-values. This approach estimates the expected reward for the next n steps, allowing the agent to incorporate information about future rewards beyond just the immediate reward [18, 33]. In vehicular networks, multi-step learning enables agents to look ahead and consider the future impact of their decisions on system performance. By predicting the likelihood of future task arrivals and offload decisions, the agents can make decisions that minimize energy consumption and latency over multiple steps. By considering multiple future rewards, the agents can learn to make more efficient and effective decisions in the long run. The resulting improvement in the learning process can reduce variance, increase efficiency, and improve robustness to noisy rewards and environmental stochasticity.

- **Distributional reinforcement learning (Distributional RL).** Traditional DQN algorithms predict the expected return reward as a scalar value to determine the optimal action in each state. However, this approach has limitations in highly dynamic environments, such as vehicular networks, where multiple possible outcomes exist for each action. To address this challenge, Distributional RL has been proposed to learn the entire distribution of the expected return, which can gain more insights and knowledge for the agent, leading to a much faster and more stable learning process [31, 37]. This approach enables the agent to consider multiple possibilities effectively, resulting in more informed and robust decision-making in the face of uncertainty.

- **Noisy nets.** One of the primary challenges in traditional DQN is the exploration limit imposed by the ϵ-greedy technique. This technique requires the execution of numerous actions to collect initial samples for training the network. In highly dynamic and complex vehicular networks, the agent must explore various offloading strategies to identify the optimal one. Additionally, the agent needs to learn how to predict and control unknown and often stochastic environments. To address this challenge, a variant of DQN called noisy nets has been proposed, which incorporates exploration by introducing noise to the network's parameters [32, 36]. The key idea is to encourage the agent to explore more by making the Q-values less deterministic, and hence less prone to overfitting. Noisy DQN can facilitate exploration and prevent the agent from overfitting to its past experiences, resulting in better performance and faster convergence to the optimal solution.

4. System Modeling and Problem Formulation

A detailed look into the target system and assumptions is essential before describing the proposed approach. In this section, each aspect of the system, namely network, communication, and computation, is modeled with parameters and variables that are directly or indirectly related to the time and energy consumption, to avoid complexity and at the same time increase the suitability of the solution. Finally, the problem is formulated to optimize the time and energy consumption of the task offloading scheme in a vehicle.

A. Network Model

We consider a vehicular network consisting a road where M RSUs, denoted as a set $R = \{1, 2, \cdots, M\}$, with specific communication coverage radius r_j^{rsu}, $1 \leq j \leq M$, are equidistantly distributed. Each RSU is equipped with an MEC server which provides computation capability for vehicles in adjacency via the wireless channel. It is assumed that all moving vehicles are always within the range of at least one RSU. Vehicles, denoted as a set $V = \{1, 2, \cdots, N\}$, where N is the total number of vehicles, are moving along the road at varying speeds generating indivisible compute-intensive tasks at each time slot. Each task of vehicle k, $1 \leq k \leq N$, is modeled as $T_{k,i} = \{c_{k,i}, a_{k,i}, d_{k,i}\}$, where $c_{k,i}$ stands for the number of CPU cycles required to complete the task i, $a_{k,i}$ denotes the input data size, and $d_{k,i}$ is the maximum acceptable time to fulfill the task from the moment task i is allocated. Produced tasks are either processed locally or offloaded to an RSU within the communication range via a wireless channel. Let $x_{k,i}^j \in \{0, 1\}$ denote the binary offloading decision variable, where $x_{k,i}^0 = 1$ means that the computation task i of vehicle k should be processed locally, and $x_{k,i}^j = 1$ means that vehicle k will offload task i to the RSU j.

B. Communication Model

When $x_{k,i}^j = 1$, the computation task will be offloaded to the RSU j because vehicle k cannot meet the low-latency demand of the task i, due to the lack of necessary computing resources. We assume that there is one mode of communication in the vehicular network, i.e., V2I, and each vehicle communicates with RSUs through a direct wireless link. According to the Shannon formula [38, 39], the data transmission rate between vehicle k and RSU j is calculated according to Eq. (5)

$$tr_{k,j} = B.\log_2\left(\frac{|h|^2.p_{tr,k}}{\sigma\left(distance_j^k\right)^{\varpi}}\right), \tag{5}$$

where B denotes channel bandwidth, h is the channel fading coefficient, $p_{tr,k}$ stands for the transmission power of vehicle k, σ represents the power of Gaussian white noise, $\bar{\omega}$ denotes path loss exponent, and $distance_j^k$ is the distance between vehicle k and RSU j which can be expressed by Eq. (6)

$$distance_j^k = \sqrt{(x_k - x_j)^2 + (y_k - y_j)^2}, \tag{6}$$

where x_k and y_k denote the coordinates of vehicle k while x_j and y_j denote the coordinates of RSU j. The time and energy consumed to transmit task i of vehicle k to RSU j are computed using Eq. (7) and Eq. (8), respectively.

$$Time_{k,i}^{comm,j} = \frac{a_{k,i}}{tr_{k,j}}, \tag{7}$$

$$Energy_{k,i}^{comm,j} = p_{tr,k}.Time_{k,j}^{comm,i}. \tag{8}$$

C. Computation Model

In our proposed computation offloading model, the task can be either accomplished by the vehicle locally or executed by one of the RSUs in proximity, so in the following, we discus both of these models.

C.1. Local Computing

For $x_{k,i}^0 = 1$, computation task will be processed locally on vehicle's own resource. The time consumed to process the task i by the vehicle k can be calculated by Eq. (9)

$$Time_{k,i}^{loc} = \frac{c_{k,i}}{f_k^{vehicle}}, \tag{9}$$

where $f_k^{vehicle}$ denotes the computation capability of the vehicle k (in CPU cycles per second). The corresponding amount of energy consumption for computing is expressed as Eq. (10)

$$Energy_{k,i}^{loc} = p_k.Time_{k,i}^{loc}, \tag{10}$$

where p_k represents the local computation power of the vehicle k and is formulated by Eq. (11)

$$p_k = L_k\big(f_k^{vehicle}\big)^3, \tag{11}$$

where L_k represents the effective switched capacitance coefficient depending on the chip architecture in the vehicle k [40].

C.2. Edge Computing

When the computation task is hard to be processed locally under the time constraints, the vehicle transmits the information to one of the adjacent RSUs. After fulfilling the task computation, the result is returned back to the vehicle. However, the result size is usually small and the RSU's downlink rate to the vehicle is often high enough. Therefore, the energy and time consumed for sending back the result are neglected [41, 42]. The total time for executing task i of the vehicle k on the RSU j is computed by Eq. (12)

$$Time_{k,i}^{edge,j} = \frac{c_{k,i}}{f_j^{rsu}}, \tag{12}$$

where f_j^{rsu} denotes the computation capability of the RSU j (in CPU cycles per second).

While the RSU is processing the task, the vehicle should wait until the result gets prepared and returned back. During this time period, we assume that the vehicle is in standby mode and the power consumption of this mode is denoted as $p_{idle,k}$. The corresponding energy consumption of the vehicle k is:

$$Energy_{k,i}^{edge,j} = p_{idle,k}.Time_{k,i}^{edge,j}. \tag{13}$$

D. Problem Formulation

We formulate the optimization problem, with the aim of jointly minimizing the total delay and energy for all vehicles moving within the network, by making offloading decisions for each vehicle. In this paper, we assume that all MEC servers have equal computation resources evenly shared between all vehicles in the network. Total time and energy consumed to process task i of vehicle k can be attained by Eq. (14) and Eq. (15), respectively.

$$Time_{k,i}^{comp} = Time_{k,i}^{loc}.x_{k,i}^{0} + \Sigma_{j=1}^{M} Time_{k,i}^{edge,j}.x_{k,i}^{j}, \tag{14}$$

$$Energy_{k,i}^{comp} = Energy_{k,i}^{loc}.x_{k,i}^{0} + \Sigma_{j=1}^{M} Energy_{k,i}^{edge,j}.x_{k,i}^{j}. \tag{15}$$

Total time and energy consumed to transmit task i of vehicle k can be attained using Eq. (16) and Eq. (17), respectively.

$$Time_{k,i}^{comm} = \Sigma_{j=1}^{M} Time_{k,i}^{comm,j}.x_{k,i}^{j}, \tag{16}$$

$$Energy_{k,i}^{comm} = \Sigma_{j=1}^{M} Energy_{k,i}^{comm,j}.x_{k,i}^{j}. \tag{17}$$

Considering the communication and computation models introduced in Section 4.B and Section 4.C, the total cost imposed on each vehicle is calculated by Eq. (18)

$$Cost_k = \Sigma_{i=1}^{I}\Big(\alpha_k.\left(Time_{k,i}^{comp} + Time_{k,i}^{comm}\right) + \beta_k.\left(Energy_{k,i}^{comp} + Energy_{k,i}^{comm}\right) + Penalty_{k,i}\Big), \tag{18}$$

where I is the total number of tasks generated by vehicle k, and α_k, $\beta_k \in [0,1]$, under constraint $\alpha_k + \beta_k = 1$, are weighting parameters expressing the importance of delay and energy consumption while making decision for each vehicle, respectively. It is worth noting that all operands in the cost formula of Eq. (18) have been normalized using the min-max method. Normalization is a crucial step in our proposed approach, as it ensures that all operands are on the same scale and have a similar impact on the final result. By using the min-max method, we can ensure that all operands are scaled to a range between 0 and 1, making them easier to compare and combine. We also consider a penalty for situations where a wrong offloading decision is made for a task, which accrued when the distance between the selected RSU and the vehicle is more than RSU's communication coverage radius. The penalty makes the agent learn which nodes to offload tasks to. Consequently, we define the penalty as the maximum cost that can be imposed to the vehicle by offloading decisions as follows.

$$Time_{max}^{comp} = \frac{\max_{k,i}\left(c_{k,i}\right)}{\min_{j}\left(f_{j}^{rsu}\right)}, \tag{19}$$

$$Energy_{max}^{comp} = p_{idle}.Time_{max}^{comp}, \tag{20}$$

$$Time_{max}^{comm} = \frac{\max_{k,i}\left(a_{k,i}\right)}{\min_{k,j}\left(tr_{k,j}\right)}, \tag{21}$$

$$Energy_{max}^{comm} = p_{tr}.Time_{max}^{comm}, \tag{22}$$

$$Penalty_{k,i} = \begin{cases} 0 & distance_{j}^{k} \leq r_{j}^{rsu} \\ \alpha_{k}.\left(Time_{max}^{comp} + Time_{max}^{comm}\right) + & \\ \beta_{k}.\left(Energy_{max}^{comp} + Energy_{max}^{comm}\right) & distance_{j}^{k} > r_{j}^{rsu} \end{cases} \tag{23}$$

For each task i generated by vehicle k, for which a wrong offloading decision is made, $Time_{max}^{comp}$ and $Energy_{max}^{comp}$ denote the maximum possible delay and energy consumption to process the task in remote processing mode, respectively. Furthermore, $Time_{max}^{comm}$ and $Energy_{max}^{comm}$ represent the maximum possible delay and energy consumption to transfer the task data to the RSUs, respectively. With the above cost model, we formulate the computation offloading as an optimization problem with two constraints that aims to minimize the weighted cost of the system in terms of delay and energy as Eq. (24)

$$\begin{aligned} &\min_{x} \Sigma_{k=1}^{N} Cost_{k} \\ &\Sigma_{j=1}^{M} x_{k,i}^{j} = 1 \qquad c1 \\ &x_{k,i}^{j} \in \{0,1\} \qquad c2, \end{aligned} \tag{24}$$

where $c1$ denotes constraint on offloading decisions and guarantees that each computation task can be processed only once while $c2$ denotes that offloading decision variable is binary.

In order to solve this optimization problem, it is necessary to find optimal values for decision variables $x_{k,i}^{j}$. Since offloading decision variables are binary, the optimization problem is not convex. Moreover, we assume a realistic environment where the state of the network dynamically changes. Consequently, the operator needs to collect a large amount of system state information and make the global decision on offloading operation based on the current network's state. Therefore, we apply Rainbow to solve this problem in the following section.

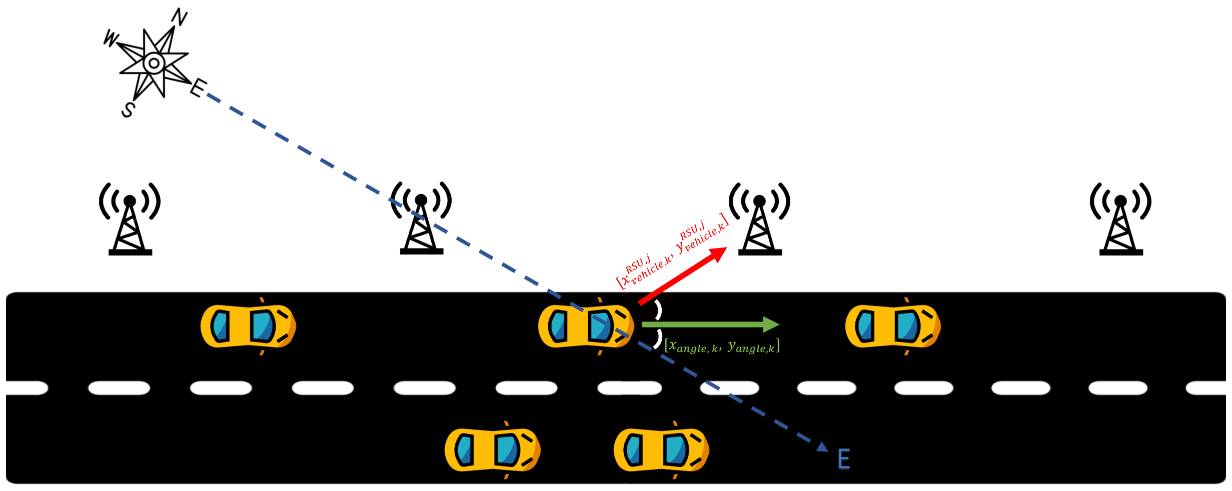
N
W
E
S
$[x^{RSU,j}_{vehicle,k}, y^{RSU,j}_{vehicle,k}]$
$[x_{angle,k}, y_{angle,k}]$
E

Figure 1: Calculus of the angle between the vehicle's movement vector and the distance vector.

5. Proposed Algorithm

A mapping from task offloading decisions in a vehicular network into a DRL formulation includes some key elements. A state space, an action space, and a reward function are necessary for complete mapping. This section starts with the characteristics of these elements and proposes a Rainbow DQN-based algorithm afterward to solve the task offloading problem in a dynamic vehicular network.

A. Definition of the State and Action Spaces, and Reward Function

At each time slot t, the agent reads important parameter assignments, namely, the state definition. One of the key steps of learning algorithms is to find an appropriate representation of the state space. The most intuitive parameter is the distance from the vehicle to each of the RSUs. Moreover, the angle with which the vehicle is deviating from each of the RSUs is also included here, in the state definition, as it is important for the agent to learn how likely is for the vehicle to get closer or further away from the RSU in the near future. We assume that the offloading decisions depend on the distance and angle of movement of each vehicles with all RSUs in the network. The state space is defined as $s_t = \{distance^k_{j,t}, \Omega^k_{j,t}, 1 \leq j \leq M, 1 \leq k \leq N\}$ in tth time slot, where $distance^k_j$ represents the distance between kth vehicle and jth RSU and $\Omega^k_{j,t}$ represents the angle of the kth vehicle's direction with regard to the jth RSU. In order to determine which roadside unit should be selected for offloading, we employed a vector-based approach that accounts for both the vehicle's movement direction relative to the east direction and the position of the RSUs. To do so, we first transform the angle of the vehicle's movement to a two-dimensional vector using Eq. (25), where $angle_{vehicle,k}$ shows the clockwise angle of vehicle k from the east direction, expressed in radians

$$[x_{angle,k}, y_{angle,k}] = [-\sin(angle_{vehicle,k}), \cos(angle_{vehicle,k})], \tag{25}$$

where $[x_{angle,k}, y_{angle,k}]$ is related to the direction vector of the kth vehicle, shown by the green vector in Fig. 1. In the next step, the distance vector from the vehicle to the RSU, shown by the red vector in Fig. 1, can be computed as Eq. (26)

$$\left[x_{vehicle,k}^{RSU,j}, y_{vehicle,k}^{RSU,j}\right] = \left[x_j - x_k, y_j - y_k\right], \tag{26}$$

where $\left[x_{vehicle,k}^{RSU,j}, y_{vehicle,k}^{RSU,j}\right]$ is the distance vector of the kth vehicle to the jth RSU. Finally, the angle between the vehicle direction vector and the distance vector can be computed by Eq. (27).

$$angle_{vehicle,k}^{RSU,j} = \cos^{-1}\left(\frac{x_{angle,k} \cdot x_{vehicle,k}^{RSU,j} + y_{angle,k} \cdot y_{vehicle,k}^{RSU,j}}{\sqrt{x_{angle,k}^2 + y_{angle,k}^2} \cdot \sqrt{x_{vehicle,k}^{RSU,j}{}^2 + y_{vehicle,k}^{RSU,j}{}^2}}\right). \tag{27}$$

In time slot t, and state space s_t, the agent takes an action in the action space. To address the high computational requirements of the Rainbow algorithm for vehicular task offloading, we propose utilizing an RSU among all, as the agent, due to its superior computation capabilities compared to vehicles. In the VEC network considered herein, the agent needs to decide whether to process a task locally or to offload it to an RSU. Hence, the action space is defined as $a_t = \{\left(x_{k,i}^j\right)_t | k \in N, j \in M, i \in I\}$.

The reward function is defined to address the objective of the optimization problem. Our objective is to minimize total delay and energy consumption for all vehicles in the network, while the goal of the agent is to achieve the maximum reward. Therefore, the reward function should negatively correlate with the objective function. We have also assumed that if the processing time of a task exceeds its deadline, it is considered as a failed task. The agent gains a reward r_t at the tth time slot, by taking the action a_t when in the state s_t, in order to maximize long-term cumulative reward. To this end, we define the reward function as Eq. (28).

$$r_t = \frac{(n_{tasks} - n_{failed})}{\Sigma_{k=1}^{N} Cost_k}, \tag{28}$$

where n_{tasks} denotes the total number of tasks generated in tth time slot, while n_{failed} denotes the total number of tasks failed due to the longer execution time than their deadlines.

Based on the discussion given in Sections 3 and 4, and the formulation above, we design the task offloading Algorithm 1 based on Rainbow. In the first step, we initialize the replay memory, the number of time slots and episodes, the main network, and the target network (line 1). Then, all generated tasks by vehicles are stored to process (line 4). The next part is to select the best action using the ϵ-greedy technique (line 5). After the action is accomplished, the agent observes a new state and reward, and the tuple (s_i, a_i, r_i, s_{i+1}) is stored as a transition in replay memory in order to train networks (line 6). Then, the agent samples some of the data as a batch from experience replay memory based on their computed complexity to train networks (line 7). The target value y_i is calculated for each transition (line 9). The algorithm minimizes the loss function $L_t(\theta_t)$ by updating the parameters of the main DQN network, θ_t (line 10). For a fixed number of time slots, the algorithm updates the parameters of the target network, $\theta_t^{'}$, to match the current parameters of the main DQN network, θ_t (line 11).

Algorithm 1: Rainbow DQN-based algorithm

```
Input:
      Initial parameters of the main DQN network and the target network,
      The replay memory D.
1.  for $episode = 1$ to $E$ do
2.    Initialize state $s_1$
3.    for $t = 1$ to $T$ do
4.      Store tasks resulted from the application decomposition, produced by vehicles
5.      Choose action $a_t$ based on the $\epsilon$-greedy policy
6.      Carry out action $a_t$, and observe the new state $s_{t+1}$ and $r_t$
7.      Calculate complexity of transitions and extract a batch of $i$ transitions from the
        memory $D < s_i, a_i, r_i, s_{i+1} >$
8.      for all $(s_i, a_i, r_i, s_{i+1})$ in $D$ do
9.          Calculate the target value $y_i$
10.     Minimize loss function $L_t(\theta_t)$, by updating $\theta_t$
11.     Update $\theta_t^{'}$ with $\theta_t$ in every fixed number of time-slots
```

6. Performance Evaluation

In this section, the performance of the proposed algorithm is evaluated through a simulation of a real traffic data. For comparison purposes, various DQN-based algorithms are also implemented and simulated to verify the suitability of the Rainbow algorithm.

A. Experimental Settings

We carry out our simulations on a real scenario using the vehicle trajectory data set of two hours (7:00 a.m. to 9:00 a.m.) on a circular road in Crete, France [43]. As shown in Fig. 2, it includes a roundabout with 4 entrances and exits, multiple two-lane or three-lane roads, one bus road, four-lane change points, and 15 traffic lights. We consider the trajectory of three vehicles, $N = 3$, driving at variable speed. The CPU computation capability of each vehicle is 20 GHz in our experiments. We assume $M = 5$ RSUs are distributed among the road at equal distances with communication range of 125 m, with a CPU computation capability of 100 GHz. Moreover, the bandwidth is set to $2MHz$. It is also assumed that the computation intensity of each task is 600 $cycles/bit$, and the input data size follows a uniform distribution within $[1, 8]$ MB. The service latency allowed for each task is 1.7 s. It is worth mentioning that all experiments were carried out on a computer system with an Intel(R) Core(TM) i7- 3610QM 2.30GHz processor and 4GB of main memory running an x64 Windows 10 operating system.

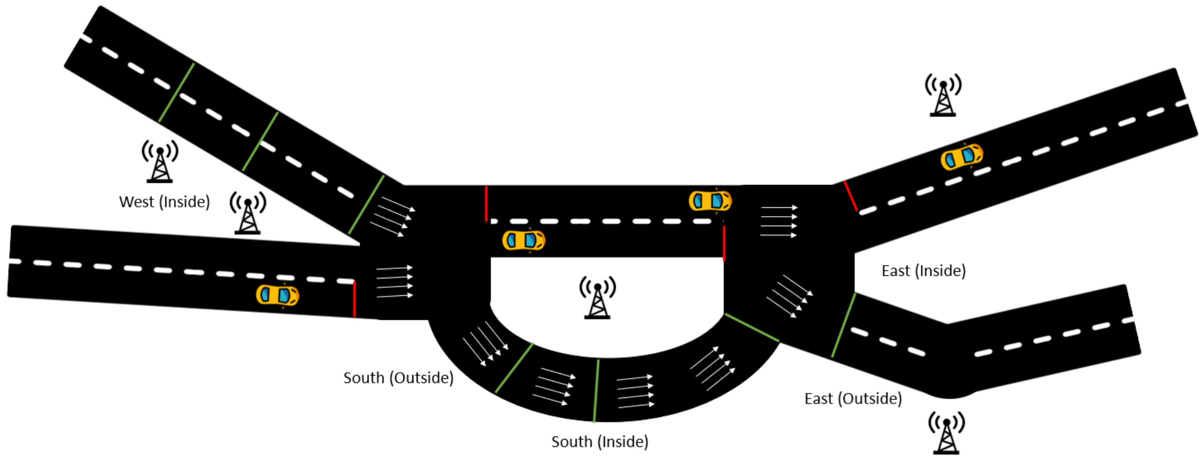

Figure 2: Vehicles route map

The proposed Rainbow DQN-based algorithm together with three state-of-the-art algorithms, namely Double DQN [44], DQN [2], and DRQN [45], are implemented by using PyTorch[1]; a machine learning framework based on the Torch library used for applications such as computer vision and natural language processing. The learning rate, discount factor, size of the mini-batch, and the replay buffer are set to 10^{-3}, 0.9, 64, and 2000, respectively. Herein a fixed number of time slots is considered and its value is set to 320. A detailed list of the parameters and their values are provided in Table 1.

B. *Experimental Results*

For comparison and analysis purposes, multiple criteria, namely, task failure, time, and energy consumption are considered with regards to changes in different parameters. Results are then depicted for the proposed algorithm and all of the state-of-the-arts. Fast convergence to high reward values is an indicator showing how effectively the iterations of the algorithm are proceeding. The convergence curve of the cumulative average reward is plotted for 150 episodes in Fig. 3. As shown in this figure, within 150 episodes, and in the first 10 iterations, the presented Rainbow DQN-based algorithm shows a faster convergence speed and gets the highest reward in comparison with the other benchmarks. DQN and Double DQN follow the same pattern approximately in the learning process while DRQN cannot converge well during this period due to its recurrent layer, which requires a longer time to train.

Next, the three main criteria are studied for the algorithms. Fig. 4 to Fig. 6, respectively, show the average task execution time, average energy consumption, and task failure rate versus the computation capability of the MEC servers deployed in the RSUs. According to the plots, Rainbow outperforms other algorithms by a significant margin and realizes better outcomes for the objective function. Meanwhile, Double DQN has a better performance

[1]https://github.com/pytorch/pytorch

Parameter	**Value**	**Description**
M	5	Number of RSUs
N	3	Number of vehicles
I	3	Number of tasks in each time slot
σ	-110 dbm	Gaussian white noise
$\bar{\omega}$	2	Path loss exponent
h	1	Channel fading coefficient
L_k	10^{-27}	Effective switched capacitance coefficient of vehicle k
$P_{tr,k}$	1.3 watts	Transmission power of vehicle k
$P_{idle,k}$	0.2 watts	Power consumption of standby mode of vehicle k
E	150	Episode size
T	185	Number of time slots
B	2 MHz	System bandwidth
$d_{k,i}$	1.7 s	Deadline of task i generated by vehicle k
$a_{k,i}$	$1-8$ MB	Data size of task i generated by vehicle k
$f_k^{vehicle}$	20 GHz	Computation capability of vehicle k
f_j^{rsu}	100 GHz	Computation capability of RSU j

Table 1: List of parameters and their values in the experiment

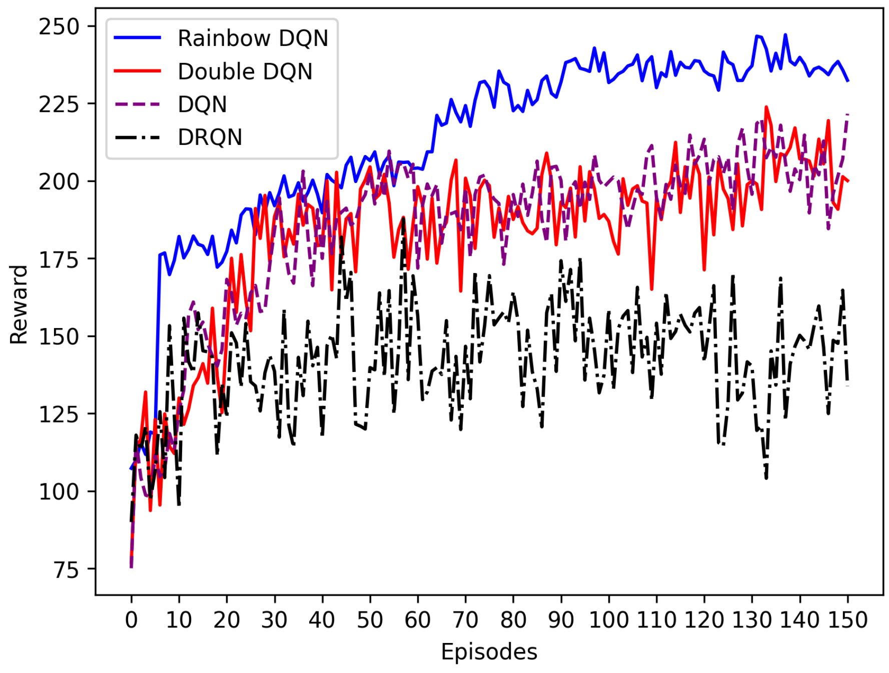

Figure 3: Convergence of the training stage for the proposed Rainbow DQN-based algorithm and three DQN-based algorithms

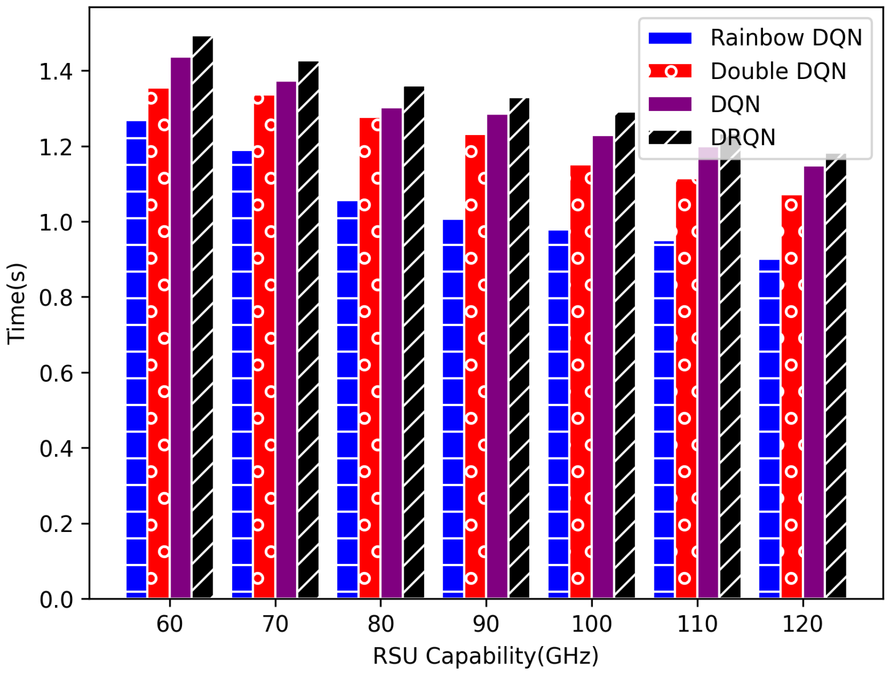

Figure 4: Total task processing delay against RSU capability

compared to DQN due to alleviating overestimation, and DRQN has the highest task execution time compared to the other three methods due to the lack of adequate time for training.

It is important to understand how different algorithms succeed in controlling the task failure rate when deadline intervals are too tight or too loose. This is illustrated in Fig. 7, where Rainbow is performing better than the other DQN-based algorithms, more specifically when the tasks are more time-sensitive. As can be seen in Fig. 7, when deadlines are tight, the superiority of the presented Rainbow DQN-based algorithm against the other algorithms is much clear.

The effect of increasing data size, on delay and energy consumption, is also investigated herein, and is depicted in Fig. 8 and Fig. 9. Expectedly, the energy consumption and execution time increase with the increase in the data size. For an input data size of 4 MB, the latency and energy of all algorithms are almost the same. The reason is that all schemes prefer to process tasks locally while ignoring capabilities of RSUs. At increased data sizes, Rainbow shows better performance and keeps the energy consumption and latency levels lower even when facing more complex tasks, since its gap with the other methods increases gradually.

Finally, the behavior of the two main factors, energy and time, is assessed when the bandwidth capacity is changing. In Fig. 10 and Fig. 11, with an increase of the network bandwidth, less time and energy will be spent on offloading tasks to the roadside units, and this ultimately leads to a reduction in energy consumption and total time in all four algorithms. According to the results, the proposed Rainbow DQN-based algorithm is keeping

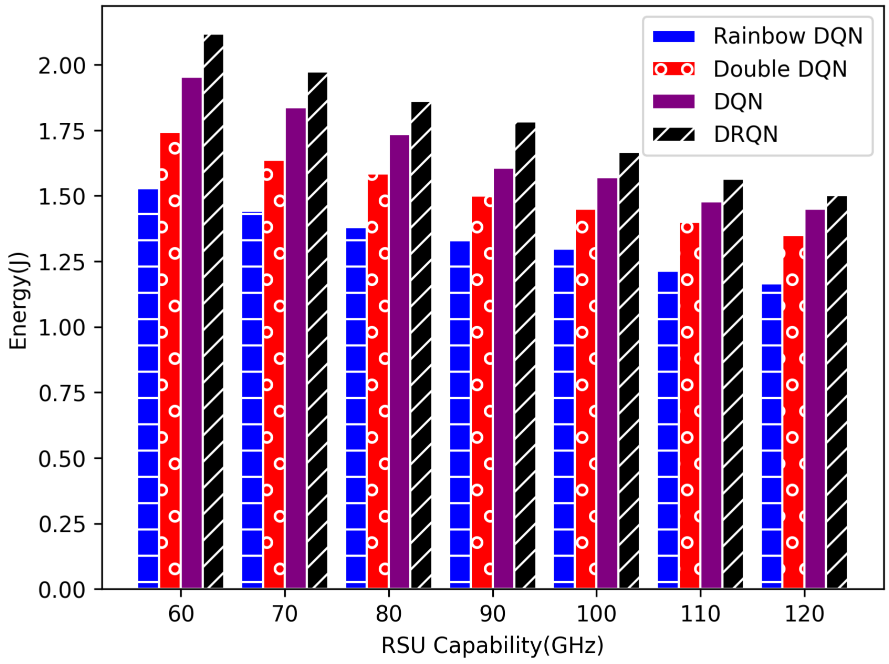

Figure 5: Total task energy consumption against RSU capability

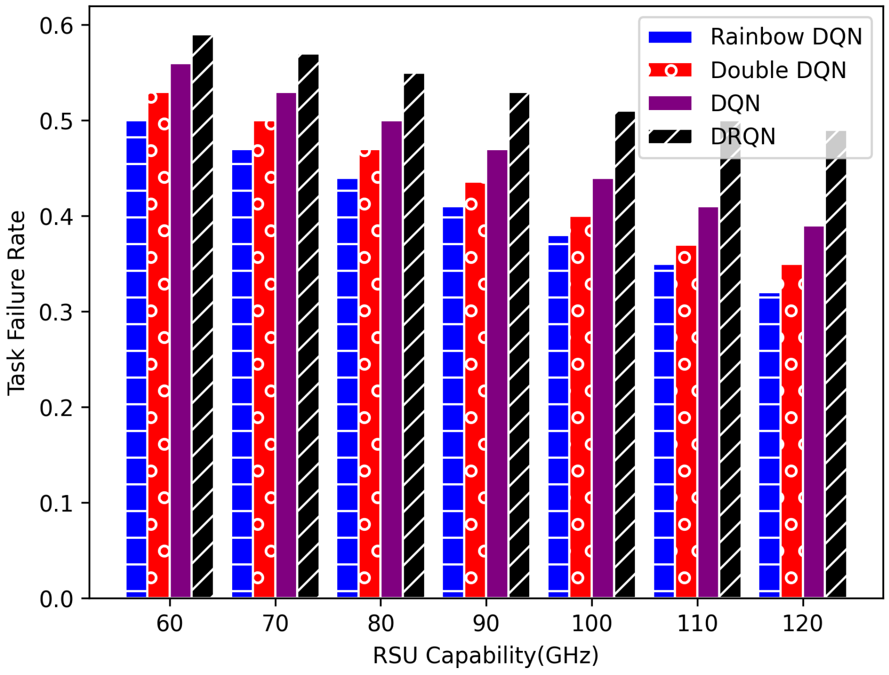

Figure 6: Task failure rate against RSU capability

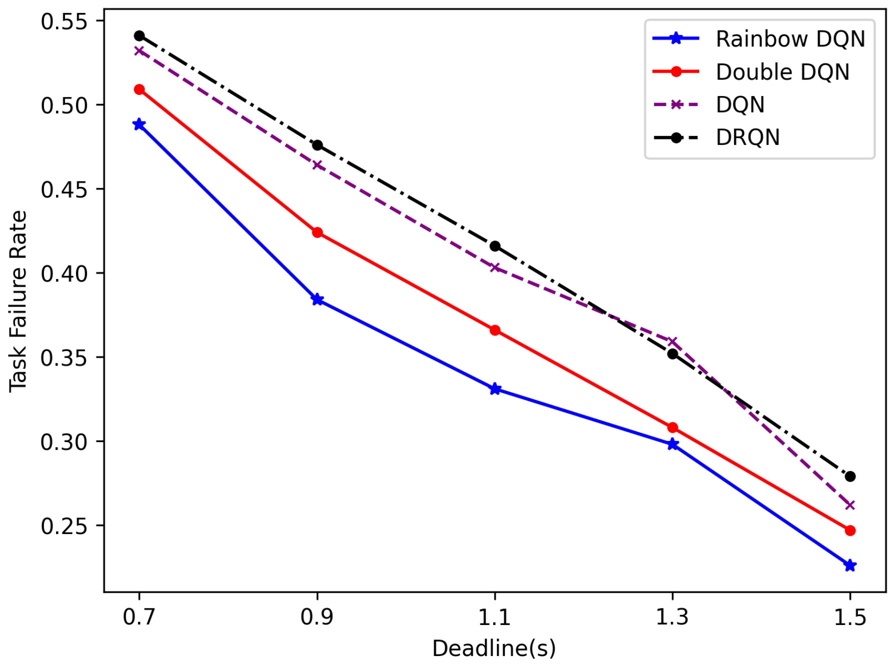

Figure 7: Task failure rate against the deadline

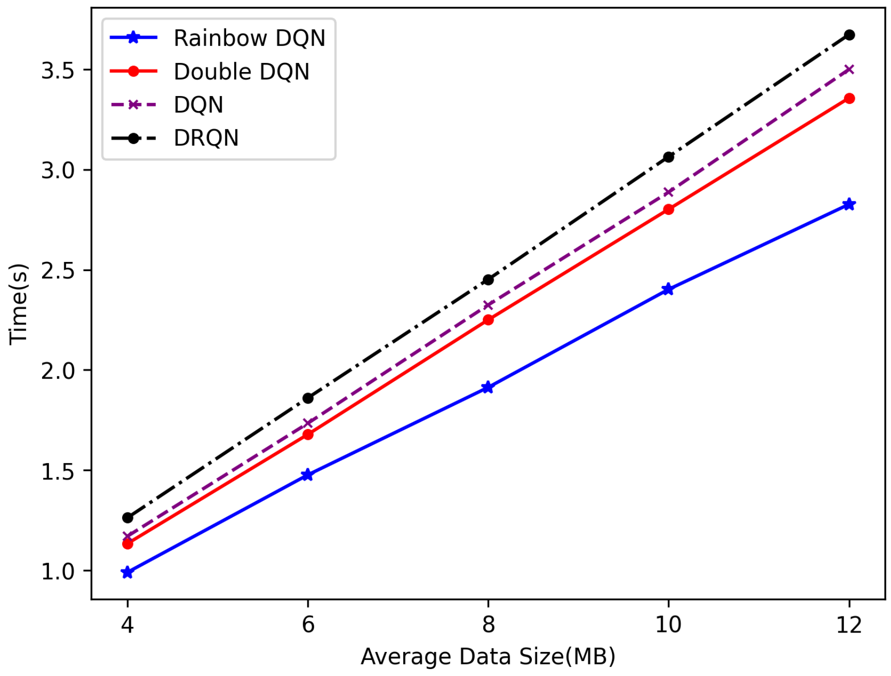

Figure 8: Total task processing delay against the average data size

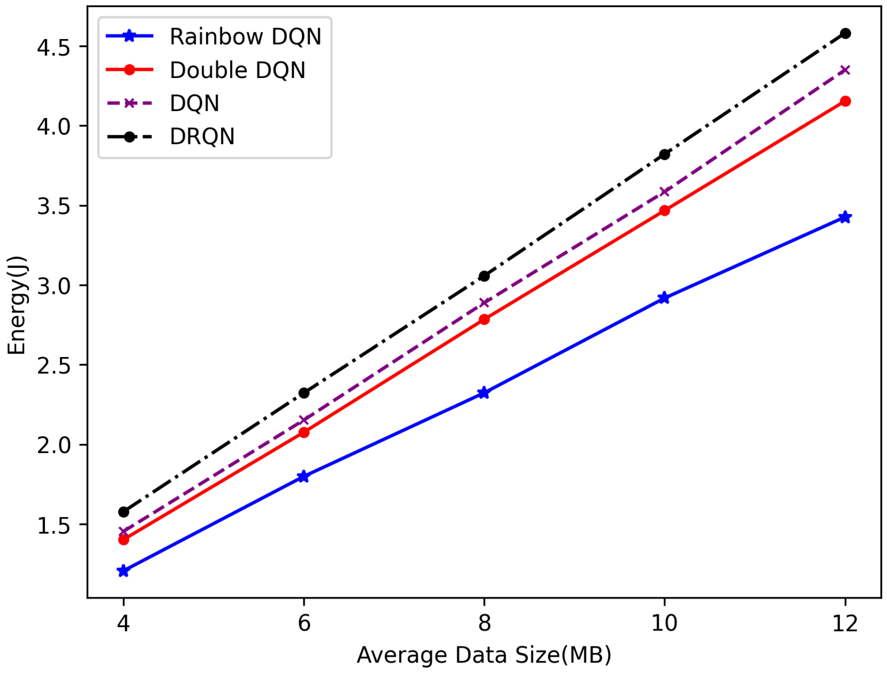

Figure 9: Energy consumption of task processing against the average data size

lower consumption values and more optimal latency levels, even facing the limited bandwidth of 1 MHz.

7. Conclusions and Future Work

In this paper, we focused on designing an efficient task offloading algorithm in vehicular edge computing networks in order to jointly minimize delay and energy consumption by taking advantage of MEC servers deployed on RSUs. In order to address the high mobility of vehicles and the complex dynamic vehicular environment, we leveraged Rainbow which is an integration of independent advances on DQN to obtain a fast learning process and efficient decisions. In order to evaluate the effectiveness of the proposed algorithm in comparison with state-of-the-art DQN algorithms, extensive experiments were conducted on a real-world traffic data. The proposed task offloading algorithm shows a better performance, 18% and 15% improvement in energy consumption and delay, respectively, compared to the state-of-the-arts.

While offloading the whole task to the RSU is the more usual case and is completely realistic, in some scenarios the vehicle divides a complex task into multiple sub-tasks and locally processes only a portion of them. In such a situation, it is important to keep track of the task dependency graphs while distributing tasks in order to successfully combine the results. Therefore, a possible future work is to develop a dependency aware task offloading algorithm. Adding load balancing and utilization concerns among the RSUs could be another field of the further study.

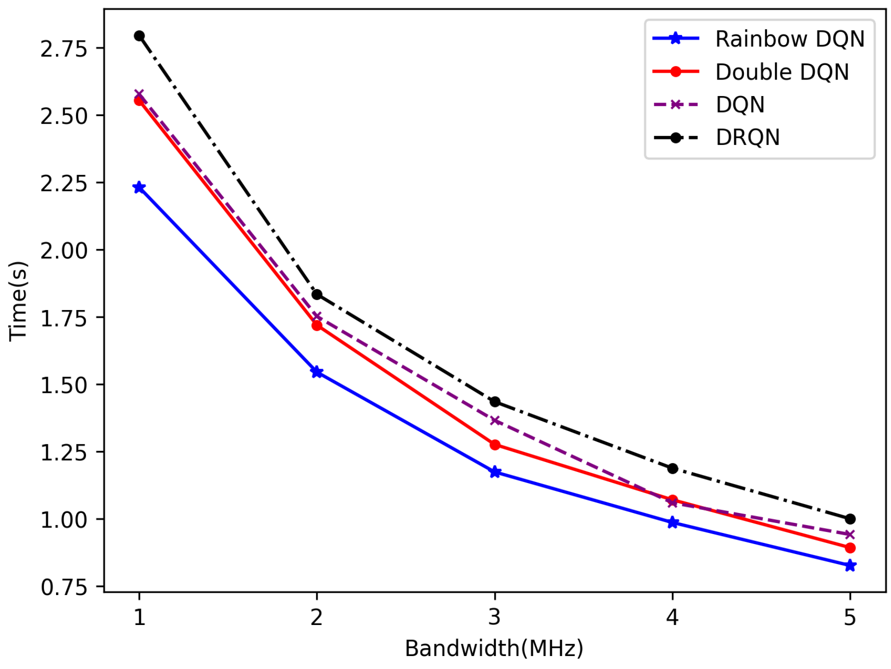

Figure 10: Total task processing delay against the bandwidth

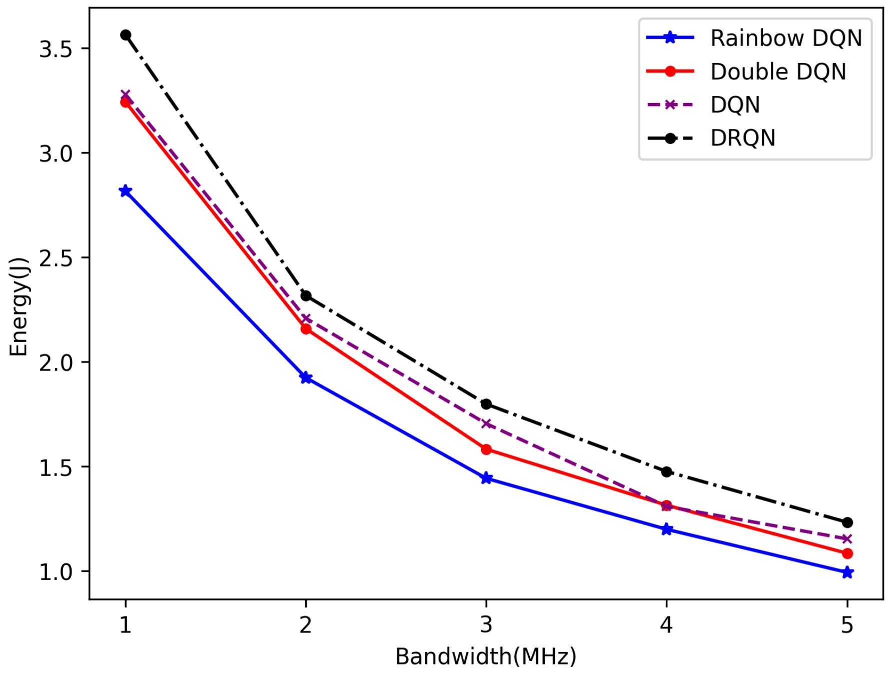

Figure 11: Energy consumption of task processing against the bandwidth

References

[1] Explosive trace detection market size, `https://fortunebusinessinsights.com/explosive-trace-detection-market-104050`, accessed: March 2023.

[2] X. Xu, B. Shen, S. Ding, G. Srivastava, M. Bilal, M. R. Khosravi, V. G. Menon, M. A. Jan, M. Wang, Service offloading with deep Q-network for digital twinning-empowered internet of vehicles in edge computing, IEEE Transactions on Industrial Informatics 18 (2) (2022) 1414–1423.

[3] M. Chen, M. Yi, M. Huang, G. Huang, Y. Ren, A. Liu, A novel deep policy gradient action quantization for trusted collaborative computation in intelligent vehicle networks, Expert Systems with Applications 221 (2023) 119743.

[4] L. Morra, F. Lamberti, F. G. Pratticò, S. L. Rosa, P. Montuschi, Building trust in autonomous vehicles: Role of virtual reality driving simulators in HMI design, IEEE Transactions on Vehicular Technology 68 (10) (2019) 9438–9450.

[5] 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.

[6] J. Zhang, H. Guo, J. Liu, Y. Zhang, Task offloading in vehicular edge computing networks: A load-balancing solution, IEEE Transactions on Vehicular Technology 69 (2) (2019) 2092–2104.

[7] L. Hu, Y. Tian, J. Yang, T. Taleb, L. Xiang, Y. Hao, Ready player one: UAV-clustering-based multi-task offloading for vehicular VR/AR gaming, IEEE Network 33 (3) (2019) 42–48.

[8] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, F. Yang, Dependency-aware task scheduling in vehicular edge computing, IEEE Internet of Things Journal 7 (6) (2020) 4961–4971.

[9] Y. Zhang, X. Lan, J. Ren, L. Cai, Efficient computing resource sharing for mobile edge-cloud computing networks, IEEE/ACM Transactions on Networking 28 (3) (2020) 1227–1240.

[10] L. Chen, J. Wu, J. Zhang, H.-N. Dai, X. Long, M. Yao, Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation, IEEE Transactions on Cloud Computing 10 (4) (2022) 2451–2468.

[11] X. Zhao, G. Huang, J. Jiang, L. Gao, M. Li, Task offloading of cooperative intrusion detection system based on deep Q-network in mobile edge computing, Expert Systems with Applications 206 (2022) 117860.

[12] A. E. Alchalabi, S. Shirmohammadi, S. Mohammed, S. Stoian, K. Vijayasuganthan, Fair server selection in edge computing with Q-value-normalized action-suppressed quadruple Q-Learning, IEEE Transactions on Artificial Intelligence 2 (6) (2021) 519–527.

[13] K. Jiang, H. Zhou, D. Li, X. Liu, S. Xu, A Q-learning based method for energy-efficient computation offloading in mobile edge computing, in: The 29th International Conference on Computer Communications and Networks, Honolulu, Hawaii, USA, 2020, pp. 1–7.

[14] Y. Liu, H. Yu, S. Xie, Y. Zhang, Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks, IEEE Transactions on Vehicular Technology 68 (11) (2019) 11158–11168.

[15] V. Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA, 2016, pp. 2094–2100.

[16] L. Tan, Z. Kuang, J. Gao, L. Zhao, Energy-efficient collaborative multi-access edge computing via deep reinforcement learning, IEEE Transactions on Industrial Informatics (2022) 1–10doi:10.1109/TII.2022.3213603.

[17] M. Hausknecht, P. Stone, Deep recurrent Q-learning for partially observable MDPs, in: AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents, Arlington, Virginia, USA, 2015.

[18] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: Combining improvements in deep reinforcement learning, in: Thirty-second AAAI conference on artificial intelligence, New Orleans, Louisiana, USA, 2018, pp. 3215–3222.

[19] Q. Jiang, X. Xu, Q. He, X. Zhang, F. Dai, L. Qi, W. Dou, Game theory-based task offloading and resource allocation for vehicular networks in edge-cloud computing, in: IEEE International Conference on Web Services, Chicago, Illinois, USA, 2021, pp. 341–346.

[20] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, Y. Liu, Joint task offloading and resource allocation for vehicular edge computing based on V2I and V2V modes, IEEE Transactions on Intelligent Transportation Systems (2022) 1–10doi:10.1109/TITS.2022.3230430.

[21] S. Raza, W. Liu, M. Ahmed, M. R. Anwar, M. A. Mirza, Q. Sun, S. Wang, An efficient task offloading scheme in vehicular edge computing, Journal of Cloud Computing: Advances, Systems and Applications 9 (1) (2020) 1–14.

[22] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, A. Taherkordi, Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks, IEEE Transactions on Intelligent Transportation Systems 24 (2) (2022) 2169–2182.

[23] Z. Ning, K. Zhang, X. Wang, L. Guo, X. Hu, J. Huang, B. Hu, R. Y. K. Kwok, Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution, IEEE Transactions on Intelligent Transportation Systems 22 (4) (2020) 2212–2225.

[24] S. Song, S. Ma, L. Yang, J. Zhao, F. Yang, L. Zhai, Delay-sensitive tasks offloading in multi-access edge computing, Expert Systems with Applications 198 (2022) 116730.

[25] E. Karimi, Y. Chen, B. Akbari, Task offloading in vehicular edge computing networks via deep reinforcement learning, Computer Communications 189 (2022) 193–204.

[26] M. Z. Alam, A. Jamalipour, Multi-agent DRL-based hungarian algorithm (MADRLHA) for task offloading in multi-access edge computing internet of vehicles (IoVs), IEEE Transactions on Wireless Communications 21 (9) (2022) 7641–7652.

[27] X. Peng, Z. Han, W. Xie, C. Yu, P. Zhu, J. Xiao, J. Yang, Deep reinforcement learning for shared offloading strategy in vehicle edge computing, IEEE Systems Journal (2022) 1–10doi:10.1109/JSYST.2022.3190926.

[28] H. Tang, H. Wu, G. Qu, R. Li, Double deep Q-network based dynamic framing offloading in vehicular edge computing, IEEE Transactions on Network Science and Engineering 10 (3) (2023) 1297–1310.

[29] M. Tang, V. W. S. Wong, Deep reinforcement learning for task offloading in mobile edge computing systems, IEEE Transactions on Mobile Computing 21 (6) (2022) 1985–1997.

[30] L. Liao, Y. Lai, F. Yang, W. Zeng, Online computation offloading with double reinforcement learning algorithm in mobile edge computing, Journal of Parallel and Distributed Computing 171 (2023) 28–39.

[31] Y. Lee, A. Masood, W. Noh, S. Cho, DQN based user association control in hierarchical mobile edge computing systems for mobile IoT services, Future Generation Computer Systems 137 (2022) 53–69.

[32] J. Yang, Y. Sun, Y. Lei, Z. Zhang, Y. Li, Y. Bao, Z. Lv, Reinforcement learning based edge computing in B5G, Digital Communications and Networks 8 (3) (2022) 469–476.

[33] F. Zhang, J. Ge, C. Wong, C. Li, X. Chen, S. Zhang, B. Luo, H. Zhang, V. Chang, Online learning offloading framework for heterogeneous mobile edge computing system, Journal of Parallel and Distributed Computing 128 (2019) 167–183.

[34] N. Kumar, S. N. Swain, C. S. R. Murthy, A novel distributed Q-learning based resource reservation framework for facilitating D2D content access requests in LTE-A networks, IEEE Transactions on Network and Service Management 15 (2) (2018) 718–731.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, in: Proceedings of the 27th International Conference on Machine Learning, Atlanta, GA, USA, 2013, pp. 1–9.

[36] J. S. Obando-Ceron, P. S. Castro, Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research (2021). arXiv:2011.14826.

[37] J. Jäger, F. Helfenstein, F. Scharf, Bring color to deep Q-networks: Limitations and improvements of DQN leading to Rainbow DQN, in: B. Belousov, H. Abdulsamad, P. Klink, S. Parisi, J. Peters (Eds.), Reinforcement Learning Algorithms: Analysis and Applications, Vol. 883 of Studies in Computational Intelligence SCI, Springer, 2021, pp. 135–149.

[38] L. Li, T. Q. Quek, J. Ren, H. H. Yang, Z. Chen, Y. Zhang, An incentive-aware job offloading control framework for multi-access edge computing, IEEE Transactions on Mobile Computing 20 (1) (2021) 63–75.

[39] Y. Mao, J. Zhang, S. Song, K. B. Letaief, Stochastic joint radio and computational resource management

for multi-user mobile-edge computing systems, IEEE Transactions on Wireless Communications 16 (9) (2017) 5994–6009.

[40] Y. Mao, J. Zhang, S. H. Song, K. B. Letaief, Power-delay tradeoff in multi-user mobile-edge computing systems, in: IEEE Global Communications Conference, Washington, DC, USA, 2016, pp. 1–6.

[41] Y. He, N. Zhao, H. Yin, Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach, IEEE Transactions on Vehicular Technology 67 (1) (2018) 44–55.

[42] P. Zhao, H. Tian, C. Qin, G. Nie, Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing, IEEE Access 5 (2017) 11255–11268.

[43] M. a. Lebre, F. L. Mouel, E. Menard, On the importance of real data for microscopic urban vehicular mobility trace, in: The 14th International Conference on ITS Telecommunications, Copenhagen, Denmark, 2015, pp. 22–26.

[44] C. A. Ardagna, H. Zhao, J. Hua, Z. Zhang, J. Zhu, Deep reinforcement learning-based task offloading for parked vehicle cooperation in vehicular edge computing, Mobile Information Systems-doi:10.1155/2022/9218266.

[45] X. Chen, T. Chen, Z. Zhao, H. Zhang, M. Bennis, Y. Ji, Resource awareness in unmanned aerial vehicle-assisted mobile-edge computing systems, in: IEEE 91st Vehicular Technology Conference, Antwerp, Belgium, 2020, pp. 1–6.

ORCID Information:

Mina Khoshbazm Farimani:
0009-0003-6732-9978

Soroush Karimian-Aliabadi:
0000-0003-1751-8343

Reza Entezari-Maleki:
0000-0003-3356-661X

Bernhard Egger:
0000-0002-6645-6161

Leonel Sousa:
0000-0002-8066-221X

HIGHLIGHTS

> We propose a Rainbow-based task offloading algorithm for vehicular networks.

> Rainbow is a deep Q-learning algorithm combining some improvements to Deep Q-Network.

> The proposed algorithm aims to jointly minimize average delay and energy consumption.

> The performance of the proposed approach is evaluated using real-world traffic data.

> Experiments demonstrate significant improvement achieved by the proposed algorithm.

Declaration of interests

☒The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: