




# Handover-Enabled Dynamic Computation Offloading for Vehicular Edge Computing Networks

Homa Maleki , *Member, IEEE*, Mehmet Başaran , *Member, IEEE*, and Lütifiye Durak-Ata , *Senior Member, IEEE*

**Abstract**—The computation offloading technique is a promising solution that empowers computationally limited resource devices to run delay-constrained applications efficiently. Vehicular edge computing incorporates the processing capabilities into the vehicles, and thus, provides computing services for other vehicles through computation offloading. Mobility affects the communication environment and leads to critical challenges for computation offloading. In this paper, we consider an intelligent task offloading scenario for vehicular environments including smart vehicles and roadside units, which can cooperate to perform resource sharing. Intending to minimize the average offloading cost which takes into account energy consumption together with delay in transmission and processing phases, we formulate the task offloading problem as an optimization problem and implement an algorithm based on deep reinforcement learning with Double Q-learning which allows user equipments to learn the offloading cost performance by observing the environment and make steady sequences of offloading decisions despite the uncertainties of the environment. Besides, concerning the high mobility of the environment, we propose a handover-enabled computation offloading strategy that leads to a better quality of service and experience for users in beyond 5G and 6G heterogeneous networks. Simulation results demonstrate that the proposed scheme achieves low-cost performance compared to the existing offloading decision strategies in the literature.

**Index Terms**—Computation offloading, handover, intelligent transportation, reinforcement learning, vehicular edge computing.

## I. INTRODUCTION

THE tremendous growth and rapid development of smart vehicles led to the use of numerous applications in vehicular environments [1]. This technology allows passengers to travel safely and comfortably with the help of network devices, cameras, and sensors. Furthermore, it provides the ability to store and process information, using driving assistance and autonomous vehicles. For instance, augmented reality (AR) can provide useful information for better visibility, especially in unfavorable weather conditions [2]. Most of these applications

need resources to perform massive computations, both in terms of energy and central processing unit (CPU) power, which are not available in the vehicles in most cases. Developing on-board computers in smart vehicles may be a solution; however, it may not be economical [3]. Therefore, vehicular edge computing (VEC) could be a suitable solution for task execution in vehicular environments beyond the 5G network, which incorporates the processing capabilities into the vehicles, and thus, provides computing services for other vehicles as well as pedestrians through computation offloading [4]. Offloading the computation could improve the quality of service (QoS) and quality of experience (QoE) for users and applications. Moreover, by offloading the computations, the users can increase battery lifetime even if they have the capability to run the application locally in order to pave the way for dense 6G applications. Additionally, in some cases, pedestrians with smart devices can act as edge servers by providing computation services and sharing their resources with other users [5].

Vehicular environments are highly dynamic due to the high mobility, where the topology of the network and wireless channel states change rapidly over time in beyond 5G and 6G communications. These circumstances cause vital problems for making a steady offloading decision [6]. Reinforcement Learning (RL) is a powerful solution for making decisions under the uncertainties of this scenario [7]. The idea behind RL is to learn by interacting with the environment and it is generally supposed that the agent has to act regardless of serious uncertainty about the environment. RL is a kind of learning technique that understands how to act in a manner that it maximizes a numerical reward. The learner is not informed beforehand about which actions that it has to take. In other words, the learner itself should determine which of the actions are more beneficial by learning them at that moment [8]. Q-Learning is one of the most prominent algorithms of RL and can be described as the quality  $Q$  of an action  $a$  in a state  $s$  under a policy  $\pi$ . During the training process, the agent updates  $Q$ -values for each state-action sequence and saves these  $Q$ -values. Due to overestimation of action values, Q-learning may have a weak performance in stochastic conditions. Therefore, double Q-learning (DQL) has been introduced to solve this problem by using two  $Q$  values instead of one  $Q$  value for each state-action, which prevents overestimation in large number of iterations through blocking higher action values [9]. Finally, in order to remove the time dependency and find a steady solution for computation offloading, DQL can be synchronized with deep neural network (DNN) [10] called double deep Q-network (DDQN) [11].

Manuscript received 23 April 2022; revised 22 August 2022 and 24 November 2022; accepted 7 February 2023. Date of publication 22 February 2023; date of current version 18 July 2023. This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Project 120E307. The review of this article was coordinated by Prof. Swades De. (*Corresponding author: Homa Maleki.*)

Homa Maleki and Lütifiye Durak-Ata are with the Information and Communications Research Group (ICRG), Informatics Institute, Istanbul Technical University, 34469 Istanbul, Turkey (e-mail: maleki18@itu.edu.tr; durakata@itu.edu.tr).

Mehmet Başaran is with the 6GEN Lab., 34854 Istanbul, Turkey, and also with the Information and Communications Research Group, Istanbul Technical University, 34469 Istanbul, Turkey (e-mail: mhmtbasaran@gmail.com).

Digital Object Identifier 10.1109/TVT.2023.3247889

## II. RELATED WORK AND MAIN CONTRIBUTIONS

The latest research in the literature attests to the importance and superiority of VEC. There are different algorithms and strategies in the literature proposed to solve the resource allocation problem in vehicular environments including stochastic methods, game theory, meta-heuristic algorithms, mathematical programming, and machine learning [2], [4], [12].

A commonly invoked strategy in this area is game theory (GT). Typically, GT in computation offloading [13] is an analytical approach to investigate the interaction between cooperating or competing users in respect of shared network resources to satisfy computation requirements. In [14], a multi-user non-cooperative computation offloading game scenario has been proposed, where each vehicle competes with other vehicles for the resources of an edge server, and the payoff function of this game is deliberated by considering the transmission model and the distance between the edge server and the vehicle.

Meta-heuristic algorithms have also been used in various research studies in this field. The authors of [15] have presented an event-triggered dynamic computation apportionment to the fog framework using both linear programming-based optimization and binary particle swarm optimization. To achieve a significant result, they have used a real-world taxi tracking simulation and have performed two illustrative tasks, including online video streaming and object identification. In [16], the authors have studied a method for optimizing execution delay and resource utilization in a multi-user and multi-server scenario based on the partheno-genetic algorithm that utilizes heuristic rules. The proposed algorithm decides where to offload the task and also illustrates the computation sequence in the server. [17] proposes an ant colony-based scheduling algorithm that manages the idle capacities of smart vehicles without the help of any other demanding infrastructures, intending to use them efficiently.

Another state-of-the-art method is machine learning, which generally intends to improve the performance of an algorithm by practice. To enhance the performance of the next-generation vehicular networks by reducing the computation delay, the authors of [18] have suggested the k-nearest neighbor algorithm which decides where to execute the task, either executing locally or offloading it to an edge or cloud.

Additionally, RL is commonly utilized to solve the resource allocation optimization problem in offloading situations. In this context, the authors of [6] have introduced an offloading method called adaptive learning-based task offloading (ALTO), in which they have used multi-armed bandit (MAB) theory for the purpose of minimizing the vehicle-to-vehicle (V2V) offloading delay. In [19], the authors have proposed a semi-Markov decision process (MDP) model for vehicular cloud computing (VCC), which considers heterogeneous vehicles and roadside units (RSUs), and have introduced a technique for determining the optimal strategy of VCC resource apportionment. The authors of [20] have developed a novel computation offloading framework for air-ground integrated VEC, which is called the learning-based Intent-aware Upper Confidence Bound (IUCB) algorithm. [21] has proposed a knowledge-driven computation offloading decision scenario that can adapt to different conditions including

environment changes, and then find the optimal solution directly from the environment via deep RL (DRL). A vehicle-assisted offloading system concerning the stochastic vehicle transactions, dynamic execution requests, and uncertainty of communication conditions has been suggested in [22]. Intending to maximize the permanent utility of the VEC system, the authors have presented an optimization problem as a semi-Markov process and introduced two RL approaches including Q-learning and DRL. [23] has proposed a DRL-based offloading scheme, which resembles the offloading policy with a deep neural network (DNN) and trains the DNN with the proximal policy optimization (PPO) algorithm without awareness of the environment dynamics. In order to save energy and provide efficient utilization of shared resources between user equipments (UEs), the authors of [24] have generated an equivalent RL model and proposed a distributed deep learning algorithm to find the near-optimal offloading decisions in which a set of DNNs have been used in parallel. In [25], the authors have proposed an intelligent offloading system for VEC by using DRL, where computation scheduling and resource allocation problems are formulated as a joint optimization problem to maximize QoE. In addition, a two-sided matching scheme and a DRL approach are developed to schedule offloading applications and designate network resources, respectively. The authors of [26] have designed an intelligent computation offloading system based on deep Q-network (DQN), where a software-defined network is introduced to achieve information collection. [27] investigates a two-layer unmanned aerial vehicle (UAV) maritime communication network with a centralized UAV on the top and a cluster of distributed bottom-UAV, with the purpose of satisfying the latency minimization problem for both communication and computation of mobile edge computing (MEC) network utilizing deep Q-network and deep deterministic policy gradient algorithms. Authors of [28] present a decentralized computation offloading solution based on the Attention-weighted Recurrent Multi-Agent Actor-Critic algorithm to tackle the challenges of large-scale mixed cooperative-competitive MEC environments. [29] presents a broad architecture for fast-adaptive resource allocation in dynamic vehicular situations. The dynamics of the vehicular environment are described as a sequence of connected MDPs, followed by hierarchical RL with meta-learning.

Motivated by these existing research studies explained in [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], which mainly ignore the collaboration between vehicles and RSUs for sharing their computational resources as edge servers, and also the handover issue due to mobility for computation offloading, the main contributions of this article are summarized as follows:

- 1) We propose an intelligent task offloading scenario for highly dynamic vehicular environments including smart vehicles and RSUs, which can cooperate to perform resource sharing. Furthermore, in order to increase the QoS and QoE, we apply a handover-enabled partial offloading strategy, which allows the vehicle to search for suitable servers at certain distances from its location and the state of the environment.



Fig. 1. An illustration of a vehicular edge computing scenario in an urban environment, where the target vehicle either offloads parts of the target task to different edge servers including neighboring vehicles and RSUs in each area, or executes it locally after making a series of decisions at certain distances through the proposed handover-enabled intelligent task offloading method.

- 2) To make an efficient execution decision, we define a cost function that calculates the execution cost of the target task, considering energy consumption, transmission delay, and processing delay. Then, we formulate the task offloading problem as an optimization problem and design an algorithm based on DDQN. The proposed algorithm aims to minimize the average weighted cost of computation of a target task with respect to the priorities of UEs (which could be computation delay or/and energy consumption); therefore, it allows UEs to learn the offloading cost performance by observing the environment and leads them in each step for making the best decision in the direction of choosing the best edge server (whether a vehicle or an RSU) for offloading.
- 3) The proposed algorithm consists of an evaluation neural network and a target neural network, where the former aims to produce the action value for each computation offloading step while the latter is used for producing the target Q-values for training the parameters of the proposed algorithm. To analyze the effectiveness and performance of the proposed algorithm, we perform a series of examinations and comparisons with existing offloading methods in the literature.

The remainder of the article is organized as follows: We introduce the system model and formulate the problem in Section III. We propose the solution for computation offloading problem

in Section IV. In Section V, simulation results are presented. Finally, the paper is concluded in Section VI.

### III. SYSTEM MODEL

In this section, we propose an intelligent task offloading framework for vehicular environments. The VEC assists resource-restricted smart devices to improve their performance through task offloading, where the user can send computationally-intensive applications to a remote edge server to be executed. As shown in Fig. 1, the considered vehicular environment consists of smart vehicles and fixed RSUs, which have the capability of sharing their idle resources with the neighboring UEs. The target vehicle generates a target task at the start point of its target route, then it tries to execute the task regarding its situation in the environment by offloading to an edge server, which could be a vehicle or an RSU, or executing locally. Due to its mobility, the target vehicle explores the environment again after a certain distance (area) and executes another part of the remaining target task in the new area. This procedure continues until the whole task has been executed. Key notations and corresponding definitions throughout the paper are listed in Table I. In this section, system architecture, movement characteristics of vehicles, communication model, and computation procedure are defined in detail.



TABLE I  
KEY NOTATIONS AND DEFINITIONS

Notation	Description
$V_i$	Vehicle edge servers
$B_j$	Roadside units
$V_T$	Target vehicle
$P_{V_T}, P_{V_i}, P_{B_j}$	Transmission power of target vehicle, edge vehicles, and edge RSUs
$R_{V_T, V_i}, R_{V_T, B_j}$	Data transmission rate between $V_T$ and edge servers
$R_{B_j, V_T}, R_{V_i, V_T}$	Data transmission rate between edge servers and $V_T$
$D_{loc}, E_{loc}$	Local computing delay and energy consumption
$D_{edge}$	Delay of processing the task in an edge server
$D_{UL, V_i}, D_{UL, B_j}$	Uplink transmission delay
$D_{DL, V_i}, D_{DL, B_j}$	Downlink transmission delay
$D_{off}, E_{off}$	Edge computing delay and energy consumption
$D_{tot}, E_{tot}$	Delay and energy of whole task completion
$f_{edge}$	CPU frequency of selected edge server ( $f_{V_i}, f_{B_j}$ )
$\alpha$	Uplink transmission overhead coefficient
$\beta$	Joint Downlink transmission overhead coefficient and output/input data ratio
$\rho$	CPU processing overhead
$L$	Input task size
$\mu_{V_i}, \mu_{B_j}$	Task processing size coefficients of each edge server
$\delta^D, \delta^E$	Weights of delay and energy consumption

### A. System Architecture

In this work, we consider a communication environment in which vehicles and RSUs operate as support infrastructures for computation. A set of presumptions are made for the communication environment. We define a set of vehicles  $V$  as

$$V = \{V_1, V_2, \dots, V_I\}, \quad (1)$$

where  $V$  contains  $I$  vehicles, aware of their features including speed, current position, moving direction, CPU frequency, and availability for sharing resources. These features can be shared with neighboring UEs, within dedicated short-range communication (DSRC) protocol, which allows sending periodic beaconing messages. Note that in this paper, we ignore data traffic and queuing issues; therefore, we assume that all the candidate servers are able to accept just one task at the moment, which we call availability.

Furthermore, we define a set  $B$  of RSUs, which are static nodes, as

$$B = \{B_1, B_2, \dots, B_J\}, \quad (2)$$

where  $J$  is the number of deployed RSUs.

The vehicular environment that we consider for simulation is designed similar to urban areas so that the system includes a varying number of vehicles with various speeds. For making the study more feasible and realistic, not all vehicles are assumed to be connected to the network during the entire process, and each of them enters into the network at a specific time and specific location, and exits from it at a later time.

### B. Movement Characteristics of Vehicles

The mobility model exceedingly contributes to the accurate simulation of VEC networks. Therefore, this paper considers a modified movement model which follows the principles of the Manhattan mobility model [30]. According to this model and as illustrated in Fig. 1, vehicles move in a vertical or horizontal direction on an urban map. We use a probabilistic strategy to choose movement direction at each intersection of vertical and horizontal roads. In contrast to the Manhattan model, which takes into account a fixed probability of 0.5 for continuing straight and 0.25 for taking a right or left side, we assume

precedences with higher probabilities in several points that can demonstrate the more appealing points of the environment, such as shopping centers or schools. Furthermore, we designate several intersections along the route as red traffic lights. When a vehicle arrives at one of these points, it stays there for a specified period of time before continuing on its way. Note that the advantage of assuming attractive points is that the target vehicle enters areas with a high density of vehicles, resulting in a large number of candidates to choose from. Consequently, the algorithm will learn to make more accurate offloading decisions.

### C. Communication Model

Communication between the vehicles and RSUs takes place over a wireless channel. Assuming block-fading Rayleigh channel, the data transmission rate between the target vehicle  $V_T$  and the selected edge server  $V_i$  or  $B_j$  is given by

$$R_{V_T, V_i} = R_{V_T, B_j} = W_0 \cdot \log_2 \left( 1 + \frac{P_{V_T} \cdot d^{-\xi} \cdot |h|^2}{N_0} \right), \quad (3)$$

where  $W_0$ ,  $P_{V_T}$ ,  $d$ ,  $\xi$ ,  $h$ , and  $N_0$  represent channel bandwidth, transmit power of the  $V_T$ , distance of the  $V_T$  from the selected edge server, path loss exponent, channel fading coefficient, and additive white Gaussian noise (AWGN) power, respectively [14]. Furthermore, the transmission rate from RSU and vehicle edge can be modeled respectively as

$$R_{B_j, V_T} = W_0 \cdot \log_2 \left( 1 + \frac{P_{B_j} \cdot d^{-\xi} \cdot |h|^2}{N_0} \right), \quad (4)$$

$$R_{V_i, V_T} = W_0 \cdot \log_2 \left( 1 + \frac{P_{V_i} \cdot d^{-\xi} \cdot |h|^2}{N_0} \right), \quad (5)$$

where  $P_{B_j}$  demonstrates the transmit power of each RSU and similarly  $P_{V_i}$  shows the transmit power of vehicle edge servers. Note that the wireless channel state is assumed to be static throughout the computation offloading during channel coherence time. Additionally, we consider and model the deployed RSUs in the communication area along with the route as properly placed whose distance is considerably higher such that their coverage area do not interfere much each other compared to the channel fading and noise power.

### D. Task Generation and Offloading Strategy

In this paper, we consider an application model in which each task generated by a vehicle or pedestrian is represented by two parameters  $L$  and  $\rho$ , where  $L$  and  $\rho$  denote the input data size of the task in bits, and processing overhead in CPU cycles per bit, respectively. These parameters influence energy consumption and execution time, whether in local computing or computation offloading. Furthermore, due to the large size of intelligent applications (e.g. image or video processing using deep learning methods), we assume the partial offloading method, where each generated target task can be processed in several parts with different sizes. Therefore, the target vehicle  $V_T$  will be able to decide to either offload each part to a suitable edge server or execute it locally according to the state of the vehicle in the

environment. Consequently, the offloading strategy for the target task can be characterized as:

- 1) Initially, a target vehicle of interest  $V_T$  or smart device of a pedestrian in a vehicle, generates an intelligent target task with a large size.
- 2)  $V_T$  begins to investigate the neighboring environment via sending beaconing messages. Afterward, regarding the obtained information such as CPU frequency and availability for accepting a computation, it prepares a list of suitable edge servers including the RSUs and smart vehicles in the communication range. By using this list, it will be able to make a decision among candidates and offloads the task for edge computing, and receives the result. Note that if there is no suitable candidate in the area, the target vehicle executes the task locally until the next server exploration.
- 3) Due to the high mobility and to prevent data loss, which may occur by changing the communication range, the target vehicle explores the environment at regular distances, called area (Fig. 1) along its target route and updates the candidate list periodically. After a certain assumed distance is exceeded, the environment transitions to a new state so that the list of possible candidate servers changes. In this case, the target vehicle offloads the remaining part of the task that is not executed during the passage of the previous area. In the case of a red traffic light, the target vehicle remains in that area temporarily, as if in motion, discovers the environment for new candidate servers at specific time intervals and offloads the task, then continues on the target route.
- 4) This process continues until the whole target task has been executed.

### E. Local Computing Model

As explained in the System Architecture subsection, the system model contains different types of UEs with various computation capacities. In situations where the target UE is not able to find a suitable edge server, it decides to execute the task locally until the next server exploration. For each UE, computation delay and energy consumption can be determined respectively as

$$D_{loc} = \frac{\rho \cdot L}{f_{loc}}, \quad (6)$$

$$E_{loc} = P^{CPU} \cdot \rho \cdot L, \quad (7)$$

where  $f_{loc}$  is the CPU cycle frequency or computation speed of  $V_T$  and  $P^{CPU}$  indicates the power consumption per CPU cycle that can be calculated as  $P^{CPU} = \kappa \cdot f_{loc}^2$ , with  $\kappa$  denoting a power consumption coefficient which relies on the chip architecture [31].

### F. Computation Offloading Model

In the proposed computation offloading scenario, first,  $V_T$  identifies neighboring vehicles and RSUs inside each area, then, selects one of the candidates and offloads the task for execution. Finally, it receives the result. The computation delay at the edge

server consists of three parts: uplink (UL) transmission delay, execution delay in the edge server  $D_{edge}$ , and downlink (DL) transmission delay. Execution delay in the edge server can be calculated as

$$D_{edge} = \frac{\rho \cdot L}{f_{edge}}, \quad (8)$$

where  $f_{edge}$  is the computation speed at the selected edge server. Similarly, delay in UL and DL transmissions can be expressed respectively as

$$D_{UL,V_i} = \alpha \cdot \frac{L}{R_{V_T,V_i}}, \quad D_{UL,B_j} = \alpha \cdot \frac{L}{R_{V_T,B_j}} \quad (9)$$

$$D_{DL,V_i} = \beta \cdot \frac{L}{R_{V_i,V_T}}, \quad D_{DL,B_j} = \beta \cdot \frac{L}{R_{B_j,V_T}} \quad (10)$$

where  $\alpha$  denotes UL transmission overhead coefficient while  $\beta$  is a coefficient modeled jointly by DL transmission overhead and the ratio of output to input data size [32]. Note that we assign these weights because generally the amount of processed data is smaller than the raw data. For instance, in an image processing task, the input data may be a high-quality image with a large size, but the feedback could be a yes or no which is very small and even negligible. Consequently, the total delay of task execution during computation offloading can be given by

$$D_{off} = D_{UL} + D_{edge} + D_{DL}. \quad (11)$$

Energy consumption during computation offloading to a vehicle edge server and RSU edge server can be expressed respectively as

$$E_{off,V} = \frac{P_{V_T} \cdot \rho \cdot L}{R_{V_T,V_i}}$$

$$E_{off,B} = \frac{P_{V_T} \cdot \rho \cdot L}{R_{V_T,B_j}}. \quad (12)$$

Since the proposed method is based on handover, the delay and energy consumption of each area are different and depend on the CPU frequency of the selected edge server and the size of transmitted data. Therefore, we can calculate the total delay for computing the whole task with size  $L$  as

$$D_{tot} = \sum_{i=1}^I \left( \alpha \cdot \frac{L}{R_{V_T,V_i}} + \frac{\rho \cdot \mu_{V_i} L}{f_{V_i}} + \beta \cdot \frac{\mu_{V_i} L}{R_{V_i,V_T}} \right) \\ + \sum_{j=1}^J \left( \alpha \cdot \frac{(1 - \mu_{V_i}) L}{R_{V_T,B_j}} + \frac{\rho \cdot \mu_{B_j} L}{f_{B_j}} + \beta \cdot \frac{\mu_{B_j} L}{R_{B_j,V_T}} \right) \\ + n \cdot \left( \frac{\rho \cdot \mu_{loc} L}{f_{loc}} \right), \quad (13)$$

where  $V_i$  for  $i \in 1, 2, \dots, I$  denotes the index of the selected vehicle,  $B_j$  for  $j \in 1, 2, \dots, J$  denotes the index of the selected RSU and  $n$  is the number of areas in which the vehicle executes the task locally. Moreover,  $\mu_{V_i} L$  is the part of the task which is executed in vehicle  $V_i$ ,  $\mu_{B_j} L$  is the part of the task which is executed in RSU  $B_j$  and  $\mu_{loc} L$  is the size of the task which is computed locally. Therefore we have  $0 \leq \mu_{V_i}, \mu_{B_j}, \mu_{loc} \leq 1$

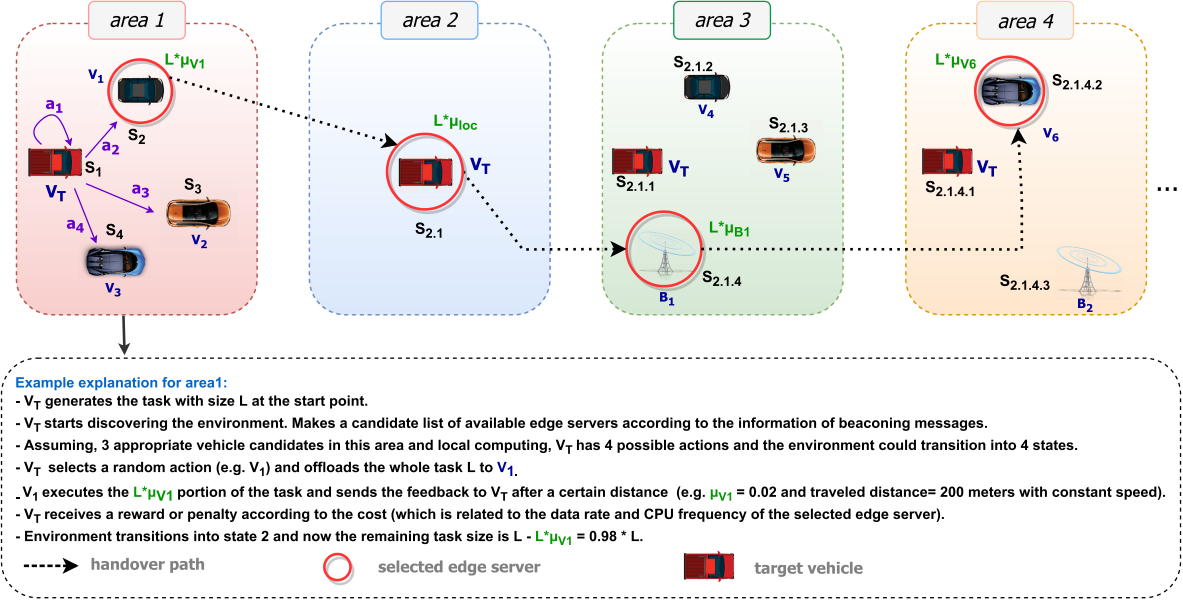


Fig. 2. An illustration of handover-enabled dynamic computation offloading algorithm.

and

$$\sum_{i=1}^I \mu_{V_i} + \sum_{j=1}^J \mu_{B_j} + n \cdot \mu_{loc} = 1. \quad (14)$$

Note that since the target vehicle is not aware of the conditions in the following steps and the amount of data that will be processed in the next area, it is not able to divide the task into small parts before offloading. So, the trick that we use in this situation is to offload only the remaining part of the target task in each step and decrease the delay and energy consumption caused by the large size of data. It means that in each area, after receiving the feedback from the edge server or local processing unit, the algorithm subtracts the amount of processed task size from the target task and offloads the entire remaining part in the following state. Fig. 2 illustrates this procedure. Similar to the logic of (13) and assuming (14), we can define the total consumed energy for computing a task as

$$E_{tot} = \sum_{i=1}^I \left( \frac{P_{V_T} \cdot \rho \cdot L}{R_{V_T, V_i}} \right) + \sum_{j=1}^J \left( \frac{P_{V_T} \cdot \rho \cdot (1 - \mu_{V_i}) \cdot L}{R_{V_T, B_j}} \right) + n \cdot (P^{CPU} \cdot \rho \cdot \mu_{loc} L), \quad 0 \leq \mu_{V_i}, \mu_{B_j}, \mu_{loc} \leq 1. \quad (15)$$

### G. Problem Formulation

In this article, we propose a method for making an intelligent computation offloading decision for dynamic vehicular environments with the aim of minimizing execution delay and energy consumption. Considering the preceding subsections, and taking into account  $D_{tot}$  and  $E_{tot}$  defined in (13) and (15), respectively, we can formulate the cost function as a function of the total delay for processing the whole task and the energy consumption for offloading in time period  $t$ . Accordingly, it can be defined as

$$\mathcal{C}(t) = \delta^D \cdot D_{tot}(t) + \delta^E \cdot E_{tot}(t), \quad 0 < \delta^D, \delta^E < 1, \quad (16)$$

where  $\delta^D$  and  $\delta^E$  are the weights of delay (1/Second) and energy consumption (1/Joule), respectively. These weights allow the users to make decisions with different priorities. As an illustration, if the user should process a highly delay-sensitive task, it may give priority to delay by assigning a higher value to  $\delta^D$  or if the task generator would be a smart device of a pedestrian in the vehicle, which has a limited source of energy, it may assign a higher value to  $\delta^E$ .

Assuming the total number of  $T$  consecutive time periods, the main purpose is to minimize the average cost of computation offloading by leading the user for making the best decision in the direction of choosing a suitable edge server. In this regard, the computational offloading problem is formulated as the following optimization problem:

$$\mathcal{C}_{min}(t) = \arg \min_c \left( \frac{1}{T} \sum_{t=1}^T \mathcal{C}(t) \right). \quad (17)$$

In order to find  $\mathcal{C}_{min}(t)$ , we need to minimize  $D_{tot}$  and  $E_{tot}$  at the same time as:

$$\mathcal{C}_{min}(t) = \arg \min_{D_{tot}, E_{tot}} \left( \frac{1}{T} \sum_{t=1}^T \delta^D \cdot D_{tot}(t) + \delta^E \cdot E_{tot}(t) \right).$$

$$\text{s.t. } C_1 : 0 \leq \delta^D \leq 1, 0 \leq \delta^E \leq 1, \delta^D + \delta^E = 1$$

$$C_2 : f_{B_j, min} (GHz) \leq f_{B_j} \leq f_{B_j, max} (GHz)$$

$$C_3 : f_{V_i, min} (GHz) \leq f_{V_i} \leq f_{V_i, max} (GHz)$$

$$C_4 : d_{min} (m) \leq d \leq d_{max} (m)$$

$$C_5 : D_{tot}(t) \leq D_{loc}(t) = D_{deadline}$$

$$C_6 : 0 \leq \mu_{V_i}, \mu_{B_j}, \mu_{loc} \leq 1$$

$$C_7 : \sum_{i=1}^I \mu_{V_i} + \sum_{j=1}^J \mu_{B_j} + n \cdot \mu_{loc} = 1 \quad (18)$$

By substituting  $D_{tot}$  and  $E_{tot}$  given in (13) and (15), respectively, in (18), the optimization problem of computation offloading with optimization variables  $f_{V_i}, f_{B_j}, R_{V_T, V_i}, R_{V_T, B_j}, R_{V_i, V_T}, R_{B_j, V_T}$  is defined in (19) shown at the bottom of this page.

Therefore, the problem depends on CPU frequencies of edge servers, data transmission rate, and consequently distances between selected servers and the target vehicle.

Note that we study the problem in discrete times; therefore, the target task executes at distributed time steps  $t = 1, 2, \dots, T$  and it must fully execute by the end of the time period  $t = T$ .

#### IV. PROBLEM SOLUTION USING DDQN ALGORITHM

##### A. Background on Markov Decision Processes

In order to solve the computation offloading problem illustrated in (19), which needs a steady decision making solution, first, we describe the problem as a discounted Markov Decision Process (MDP) [33] defined by  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, Pr, \mathcal{R}, \gamma)$  where  $\mathcal{S}$  is the set of states,  $\mathcal{A}$  is the set of actions,  $\mathcal{P}$  is the probability density over states,  $Pr : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$  is the Markov transition probability kernel,  $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{R})$  is the distribution of the reward, and  $\gamma$  is the discount factor ( $0 < \gamma \leq 1$ ) which determines how much importance is assigned to immediate and future rewards. In particular, for taking any action  $a \in \mathcal{A}$  at any state  $s \in \mathcal{S}$ ,  $Pr(\cdot | s, a)$  corresponds to probability distribution of the next state and  $\mathcal{R}(\cdot | s, a)$  to the distribution of the immediate reward. The policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  of the MDP maps each state  $s$  to a probability distribution over the action set  $\mathcal{A}$ . Furthermore, the expected action-value function  $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$  can be defined over expectation operator  $\mathbb{E}[\cdot]$  as

$$Q(s, a) = \mathbb{E}_{\pi} [\mathcal{R}_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) | \mathcal{S}_t = s, \mathcal{A}_t = a]. \quad (20)$$

For each given action-value function  $Q$  and each state  $s$ , the greedy policy  $\pi^*$  which chooses the action with the highest  $Q$  meets

$$\pi^*(a | s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s, a). \quad (21)$$

In order to find the optimal policy that achieves the largest reward, the optimal action-value function  $Q^*$  can be described as

$$Q^*(s, a) = \max_{\pi} Q(s, a). \quad (22)$$

Therefore, the deterministic optimal policy  $\pi^*$  can be obtained by maximizing over  $Q^*(s, a)$ . These optimal value functions are recursively connected by the Bellman optimality equations [22], and they need to satisfy the conditions of these equations

which define the highest reward an agent can earn if it makes the optimal decision at the current state and all the upcoming states.

##### B. Double Deep Q-Learning-Based Decision Making

In online interactive-based RL algorithms, estimation of the action-value function brings uncertainty. The DDQN algorithm has three main differences from other RL algorithms which provide a solution for these instabilities. The first difference is that the DDQN algorithm implements the optimal action estimation function via a DNN instead of using a Q-table. Secondly, the algorithm uses experience replay [34], [35] which as shown in Fig. 3, samples the trajectory of MDP consisting of states  $\mathcal{S}_t$ , actions  $\mathcal{A}_t$ , rewards  $\mathcal{R}_t$ , and the following state  $\mathcal{S}_{t+1}$  from the replay memory  $\mathcal{M}$  at each iteration as observations and then uses them to train the parameters of DNNs. Experience replay helps to achieve steadiness in uncertain environments by removing the time dependency of observations. Finally, the third difference is to use of two neural networks which could prevent overestimations of action values by separating the action selection and the strategy evaluation procedures [36].

According to the information given above, our proposed algorithm, which is shown in Algorithm 1 step by step, can be explained in the following. Computation offloading decisions follow a distributed procedure. Therefore, we concentrate on a single target vehicle of interest which has a target task with a large size that can break into different parts with different sizes where each part can be computed locally or on different servers. Following the generation of a task, the target vehicle starts to explore the neighboring vehicles and RSUs within the communication range and provides a candidate list accordingly. Note that in the proposed computation offloading scenario, we may not always have an RSU in each area because we deploy them randomly. This could enable the algorithm to decide more wisely in various circumstances. Therefore, we can define the states set,  $\mathcal{S}$ , which includes the number of available RSUs, the number of available vehicles, location, and CPU frequencies of each RSU and vehicle, speed and moving direction of each vehicle, and the remaining task size for execution. Also, we define the actions set  $\mathcal{A}$  as the vehicles and RSUs available for offloading. Since the main goal of this study is to achieve minimum cost during the computation of a task that brings us shorter delay and less energy consumption, we define the reward function based on the cost of computing a task in each area as

$$\mathcal{R}_t = \frac{1}{C_{min}^{current}(t)}, \quad (23)$$

$$C_{min}(t) = \underset{f_{V_i}, f_{B_j}, R_{V_T, V_i}, R_{B_j, V_T}}{\operatorname{argmin}} \left[ \frac{1}{T} \sum_{t=1}^T \delta^D \left( \sum_{i=1}^I \left( \alpha \frac{L}{R_{V_T, V_i}} + \frac{\rho \mu_{V_i} L}{f_{V_i}} + \beta \frac{\mu_{V_i} L}{R_{V_i, V_T}} \right) + \sum_{j=1}^J \left( \alpha \frac{(1 - \mu_{V_i}) L}{R_{V_T, B_j}} + \frac{\rho \mu_{B_j} L}{f_{B_j}} + \beta \frac{\mu_{B_j} L}{R_{B_j, V_T}} \right) \right. \right. \\ \left. \left. + n \left( \frac{\rho \mu_{loc} L}{f_{loc}} \right) \right) + \delta^E \left( \sum_{i=1}^I \left( \frac{P_{V_T} \rho L}{R_{V_T, V_i}} \right) + \sum_{j=1}^J \left( \frac{P_{V_T} \rho (1 - \mu_{V_i}) L}{R_{V_T, B_j}} \right) + n (P^{CPU} \rho \mu_{loc} L) \right) \right]. \quad (19)$$



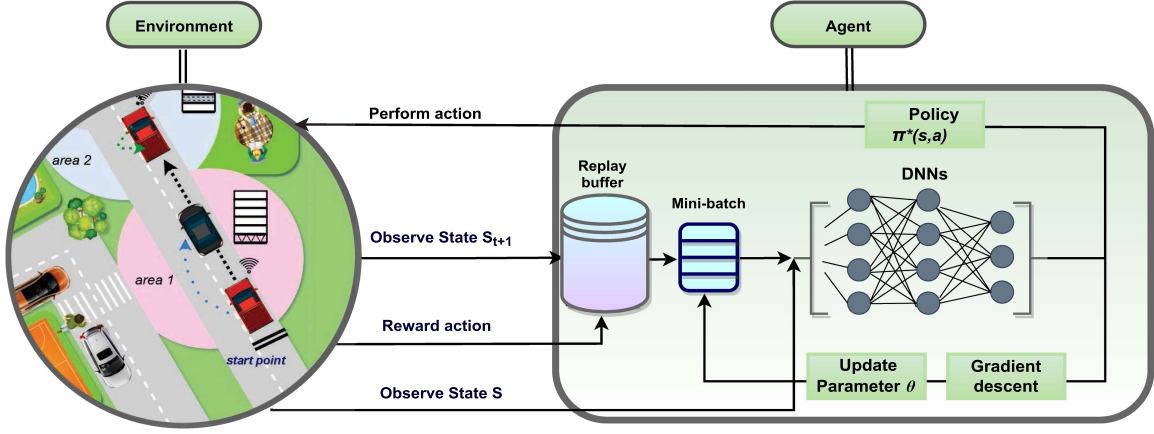


Fig. 3. Double Deep Q-Network application for the proposed urban scenario.

where  $C_{min}^{current}(t)$  is the minimum computation cost for the current state. Note that the aim of the algorithm is to find an optimal policy in order to maximize the cumulative reward during a long run over a time period  $t = 0, \dots, T$ . In addition, the proposed DDQN algorithm uses the  $\epsilon$ -greedy policy which balances the exploration and exploitation in the environment [37]. When the algorithm chooses exploration, it investigates the environment and collects information that can lead to making better offloading decisions by sending the tasks to different candidates. On the other hand, exploitation prefers to exploit the promising information found when the algorithm performed the exploration.

As mentioned above, the DDQN algorithm prevents selecting overestimated values resulting from using the same action value to evaluate and to make decision [11]. Therefore, the algorithm consists of two DNNs. One of these is the evaluation network  $Q_\theta$ , with parameter  $\theta$ , and the other is the target network  $Q_{\theta'}$  with parameter  $\theta'$ , where  $\theta$  is a parameter based on Q-learning, and both networks have the same structure. The objective of parameter  $\theta$  is to select the action with the maximum action value. On the other hand, the parameter  $\theta'$  is used to evaluate the action value of the optimal action. The target network parameter  $\theta'$  is updated every  $\mathcal{T}$  steps with the value of the evaluation network parameter as  $\theta' = \theta$  and remains fixed for the next  $\mathcal{T}$  steps. During each iteration of the DDQN, transitions of MDP  $(S_t, \mathcal{A}_t, \mathcal{R}_t, S_{t+1})$  are stored in the replay memory  $\mathcal{M}$ , then a random mini-batch, with size  $\mathcal{B}$ , of independent samples from  $\mathcal{M}$  is selected for training the parameters of neural networks.

Furthermore, with independent samples  $(s_\tau, a_\tau, r_\tau, s_{\tau+1})_{\tau \in [\mathcal{B}]}$  of  $\mathcal{M}$ , we can calculate the target  $\mathcal{Y}$  from the target network as

$$\mathcal{Y}_\tau \equiv r_{\tau+1} + \gamma Q \left( s_{\tau+1}, \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s_{\tau+1}, a; \theta_\tau), \theta'_\tau \right) \quad (24)$$

and calculate the evaluation network parameter  $\theta$  by performing a gradient step on the loss function  $\mathcal{L}(\theta)$  defined by

$$\mathcal{L}(\theta) = \frac{1}{\mathcal{B}} \sum_{\tau=1}^{\mathcal{B}} [\mathcal{Y}_\tau - Q_\theta(s_\tau, a_\tau)]^2 \quad (25)$$

---

**Algorithm 1: DDQN-Based Task Offloading Algorithm.**


---

**Input:** Initialize evaluation network  $Q_\theta$ , target network  $Q_{\theta'}$ , replay memory  $\mathcal{M}$ , mini-batch  $\mathcal{B}$ , exploration probability  $\epsilon$ , discount rate  $\gamma$ ,  $\mathcal{T}$

- 1: for learning episode  $x = 1, \dots, X$
- 2:   for  $L > 0$  do
- 3:     explore environment for candidates
- 4:   if no candidate found in the range
- 5:     execute the task locally,  $L = L - \text{executed task}$
- 6:   else
- 7:     for each evaluation step do
- 8:       Observe state  $s_t$  and select  $a_t \sim \pi(a_t, s_t)$
- 9:       Execute  $a_t$ ,  $L = L - \text{executed task}$ , observe
- 10:       next state  $s_{t+1}$  and reward  $r_t$
- 11:       Store  $(s_t, a_t, r_t, s_{t+1})$  in experience memory  $\mathcal{M}$
- 12:     end for
- 13:   end if
- 14:   for each target step do
- 15:     Sample random mini-batch
- 16:      $\mathcal{B}_\tau = (s_\tau, a_\tau, r_\tau, s_{\tau+1}) \sim \mathcal{M}$
- 17:     Compute target value for each  $\tau \in [\mathcal{B}]$
- 18:      $\mathcal{Y}_\tau = r_\tau + \gamma \cdot \max_{a \in \mathcal{A}} Q_{\theta'}(s_{\tau+1}, a)$
- 19:     Update the evaluation network by performing
- 20:     gradient descent step  $[\mathcal{Y}_\tau - Q_\theta(s_\tau, a_\tau)]^2$
- 21:     Update target network parameters every target
- 22:     step  $\mathcal{T} : \theta' \leftarrow \theta$
- 23:   end for
- 24:    $x = x + 1$
- 25: end for

---

as the mean-squared error between the target value and the evaluated Q-value. By assuming  $\mathcal{R}_\tau$  to be an immediate reward for taking action  $\mathcal{A}_\tau$  and  $S_{\tau+1}$  to be the next state, the Bellman



optimality equation can be presented as

$$(\vartheta Q_{\theta'}) (\mathcal{S}_\tau, \mathcal{A}_\tau) = \mathbb{E} [\mathcal{Y}_\tau | \mathcal{S}_\tau, \mathcal{A}_\tau], \quad (26)$$

where  $\vartheta$  denotes the Bellman optimality operator. Then, we consider  $\theta' = \theta$  and calculate the expected value of the loss function in terms of the mean-squared Bellman error and the variance of  $\mathcal{Y}$  as

$$\mathbb{E}[\mathcal{L}(\theta)] = \|Q_\theta - \vartheta Q_{\theta'}\|^2 + \mathbb{E} \left[ [\mathcal{Y}_\tau - (\vartheta Q_{\theta'}) (\mathcal{S}_\tau, \mathcal{A}_\tau)]^2 \right]. \quad (27)$$

Since the target  $\mathcal{Y}$  does not rely on  $\theta$ , the minimization of the loss function can be solved by

$$\min_{\theta} \|Q_\theta - \vartheta Q_{\theta'}\|^2, \quad (28)$$

where DDQN aims to solve this minimization problem in order to reach the optimal action-value function  $Q^*$  that leads to finding the optimal policy  $\pi^*$  and consequently making the best computation offloading decision  $\mathcal{A}^*$ .

### C. Time Complexity

In this section, we investigate the time complexity of the proposed DDQN-based algorithm. Generally, the complexity of RL algorithms relies on the size and properties of the state and the initial information of the agents [38]. Since we use a fixed exploration rate in our proposed algorithm, the required time for convergence with a state size of  $|S|$  and exploration rate  $\epsilon$  is bounded by  $\mathcal{O}(|S| \log |S| (\log(1/\epsilon))/\epsilon^2))$  which is linear in state size [39]. Additionally, the complexity of finding the optimal decision is  $\mathcal{O}(N^c)$  where  $N$  is the number of actions in each iteration and  $c$  denotes the number of available candidate edge servers including vehicles and RSUs [40].

## V. SIMULATIONS

In this section, we evaluate the performance of the proposed method for computation offloading in VEC. The simulations are performed in the Windows operating system with CPU Intel core i7-10750H, 16 GB of memory; GPU NVIDIA GeForce GTX1650, Python 3.7.6. Moreover, we use the TensorFlow library in order to create the deep learning model of the proposed algorithm and to speed up numerical computing. We implement a fully connected backpropagation neural network for both the evaluation network and the target network. Besides, the rectified linear units (ReLU) function is used as the activation function of each layer in the DDQN model to avoid vanishing gradients [41].

### A. Simulation Setup

We consider a computation offloading framework that consists of 80 vehicles and 30 randomly fixed RSUs located along a route for 10 km. The speed of vehicles is assumed to be different and distributed uniformly between [30,120] km/h. The communication range is considered as 200 meters. To increase the feasibility of the scenario, CPU frequencies of vehicles and RSUs are randomly distributed in the range [2,8] and [8,16] GHz, respectively. Note that once we set the random values for CPU frequencies and distances for each vehicle and RSU (at

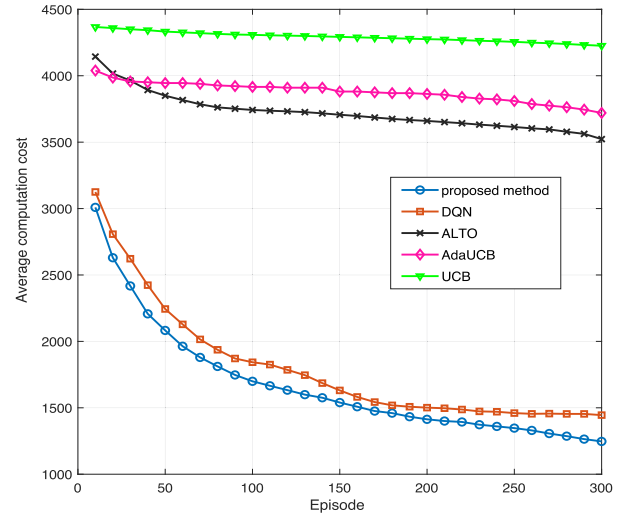


Fig. 4. Comparison of average cost during computation of one whole task.

the beginning of running the algorithm), we keep them constant until the end of the episode. Table II shows an example of the environment design parameters. The requested input data size for each task  $L$  is assumed to be 16 Mbits where the vehicle generates tasks continuously during the 10 km route (to measure the overall size of tasks that could be processed during the target route), processing complexity  $\rho$  is fixed as 1000 cycles/bit, and  $\kappa = 10^{-27}$  Watt  $\cdot$  s<sup>3</sup>/ cycles<sup>3</sup>. Communication parameters are considered as  $W = 10$  MHz,  $P_V = 0.1$  W,  $N_0 = 1$ ,  $\alpha = 1$ , and  $\beta = 0.01$ . Initial parameters of the proposed DDQN algorithm are considered as  $\mathcal{M} = 1024$ ,  $\mathcal{B} = 32$ ,  $\gamma = 0.9$ ,  $\epsilon = 0.9$ , and learning rate = 0.05.

### B. Simulation Results

In this part, we evaluate the performance and reliability of the proposed DDQN algorithm. To illustrate the performance of this algorithm, we adapt the DQN algorithm to our proposed scenario. Moreover, we compare the results with the ALTO algorithm introduced in [6] and two other RL algorithms, namely Upper-Confidence-Bound (UCB) and Adaptive Upper-Confidence-Bound (AdaUCB) defined in [42], [43], respectively. Note that these three algorithms are based on bandits and unlike the MDPs, actions have no influence on subsequent observations.

Regarding the learning time to achieve our performance, we examine the proposed algorithm with different numbers of episodes. Each episode takes a time of about 9-11 seconds. As illustrated in Figs. 4–8, there are no significant changes in the results after around 300 episodes and as we mentioned in the Time Complexity subsection, the required time for convergence is linear in the state size.

Fig. 4 indicates the average computation cost performance of the proposed DDQN algorithm during 300 learning episodes over the whole target route. The target task size is large and due to the high mobility of the vehicular environments, it is not feasible to consider the task execution without enabling the handover feature. Therefore, we adjust all the compared algorithms to

TABLE II  
SIMULATION STATE-SPACE PARAMETERS EXAMPLE

vehicle ID	entrance location	leaving location	moving direction	CPU frequency	availability	speed
Target vehicle	0	10000	→	2 GHz	available	60 km/h
1	220	350	→	2.5 GHz	available	100 km/h
2	480	3700	→	4 GHz	available	40 km/h
3	21	1200	←	3.6 GHz	not available	85 km/h
4	6700	10000	→	4.5 GHz	available	65 km/h
5	9200	9500	←	7 GHz	available	110 km/h

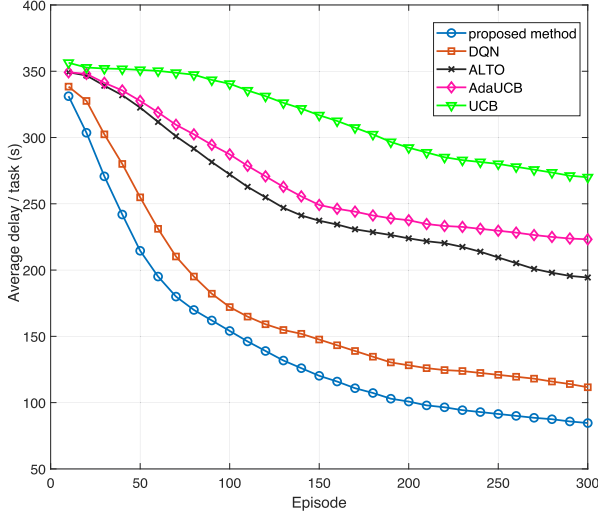


Fig. 5. Comparison of average delay for executing a whole target task.

resemble our proposed handover-enabled scenario. Following the generation of the task at the start point, the target vehicle interacts with the environment by offloading the task to different servers in different areas separated by a distance of 200 meters. It calculates the total reward during the computation of the whole task according to the feedback of each area. Then, the algorithm tries to find an optimal policy by carrying out this execution process over and over and achieves the highest cumulative reward. In the beginning, when the proposed algorithm starts to run, the evaluation network starts to learn and update the parameters; so, the loss value starts to increase, and accordingly, the cost increases. With the growth of learning episodes and due to the effect of training, the loss value drops down, and cost decreases steadily and tends to be stable. As presented in the figure, the DQN algorithm follows a trend similar to the proposed method and the results are close to our experiments. Besides, the other three algorithms, which are based on multi-armed bandits structure, achieve higher and a nearly steady amount of cost.

Fig. 5 shows the performance comparison of average delay per task during the 300 learning episodes. Simulation results demonstrate that, in terms of computation delay, the proposed method converges to the optimal faster than existing methods.

In Fig. 6, we compare the amount of executed tasks in a target vehicle with a fixed speed during the passage of a road for a distance of 10 km. We assume that after finishing the execution of one task, immediately another task generates in the target vehicle. Simulation results show that the proposed algorithm

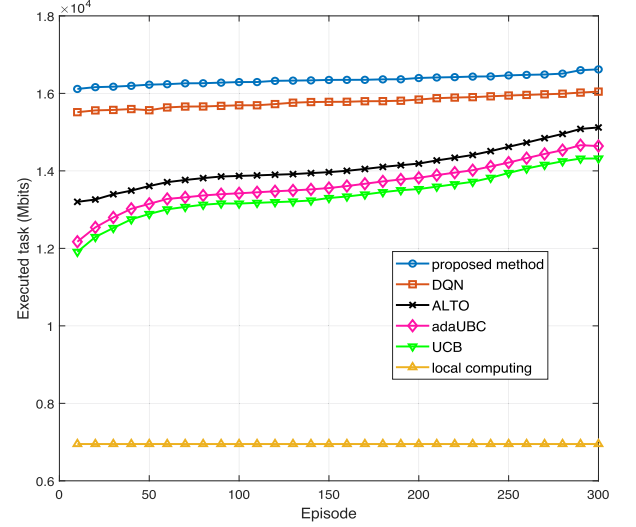


Fig. 6. Comparison of the amount of executed tasks along 10 km road during 300 learning episodes.

outperforms the other existing algorithms by executing larger-sized tasks by making the best offloading decisions regarding the computation performance of possible edge candidates in each area. Furthermore, as compared to the local computing approach, the proposed algorithm is capable of handling almost twice tasks (in Mbits).

We examine the amount of energy consumption during computation offloading in the first 5 areas of the target route in Fig. 7. The amount of energy consumption is related to the data transmission rate, which depends on the distance from the selected edge server, and the data size for offloading. Our proposed algorithm takes the location of the edge servers in the state space of the environment into account; therefore, it can learn the energy consumption performance by performing each action considering the distance and make an optimal decision with lower energy consumption. Additionally, the handover strategy helps the target vehicle to only offload the remaining part of the task in each step, which could eventually decrease the energy consumption.

We calculate the cost of computation defined in (14) by assigning the weights for energy consumption  $\delta^E$  and delay  $\delta^D$  to set priorities for each UE. Fig. 8 demonstrates the effect of preferences on the computation cost of one whole task. Since the total computation delay includes the UL and DL transmission delays, giving higher priority to delay by assuming  $\delta^D$  close to 1 causes a higher cost for UE. Therefore, delay-sensitive tasks are

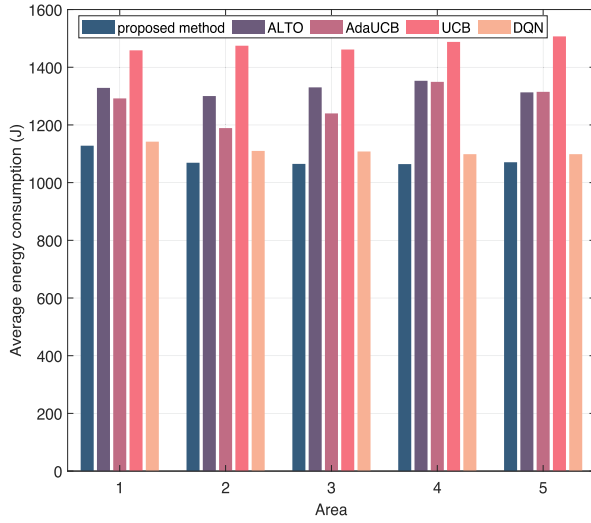


Fig. 7. Comparison of average energy consumption during task offloading in each area of environment.

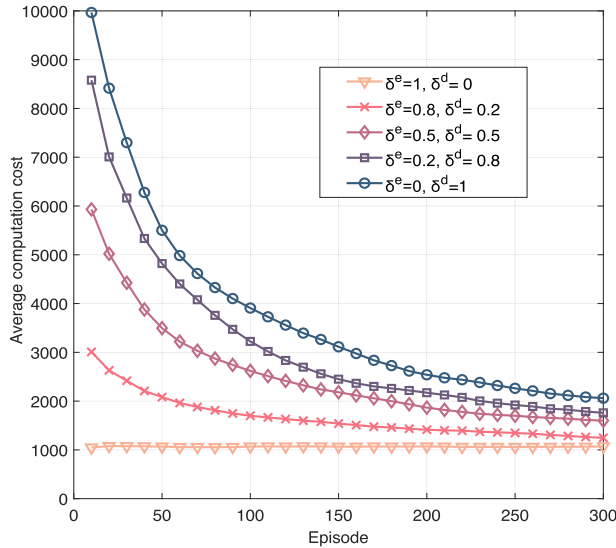


Fig. 8. Comparison of computation cost considering different priorities where higher  $\delta^E$  gives priority to less energy consumption while higher  $\delta^D$  favors low latency.

high-cost. Conversely, giving priority to energy consumption yields low-cost computation; therefore, offloading could be a good solution for battery saving of UEs.

## VI. CONCLUSION

In this article, we investigate the computation offloading problem in beyond 5G and 6G-enabled VEC networks. We define the offloading cost which takes into account energy consumption, transmission delay, and processing delay during computation offloading, as the optimization problem of task allocation. Accordingly, we design an intelligent algorithm based on DDQN to minimize the average offloading cost considering the high instability of vehicular environments that require an online solution. The proposed handover-enabled DDQN algorithm enables the

user to learn the offloading cost performance by interacting with the environment and trying to make a steady decision despite the uncertainty of the dynamic environment. The proposed algorithm has three important components: The first one is using experience memory which cooperates to obtain steadiness in unpredictable environments by eliminating the time dependency of observation, the second one is using two neural networks to prevent selecting overestimated values caused by using the same action value to evaluate and to make the decision, and the third one is using DNN for optimal action value estimating. Simulation results demonstrate that the proposed algorithm is able to achieve better performance in terms of average offloading cost under different conditions, which designates the feasibility and effectiveness of the proposed method. The DDQN algorithm is able to show the best performance in large-scale environments with a great number of states; therefore, as a future work, we plan to use a more realistic urban scenario, consisting of a high density of vehicles, pedestrians, and infrastructures, where they collaborate to share resources with a dynamic queue management system which controls the data traffic.

## REFERENCES

- [1] M. Keertikumar, M. Shubham, and R. Banakar, "Evolution of IoT in smart vehicles: An overview," in *Proc. IEEE Int. Conf. Green Comput. Internet Things*, 2015, pp. 804–809.
- [2] A. B. De et al., "Computation offloading for vehicular environments: A survey," *IEEE Access*, vol. 8, pp. 198214–198243, 2020.
- [3] S. Gyawali, S. Xu, Y. Qian, and R. Q. Hu, "Challenges and solutions for cellular based V2X communications," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 1, pp. 222–255, Jan.–Mar. 2021.
- [4] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Netw. Appl.*, vol. 26, no. 3, pp. 1145–1168, 2021.
- [5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
- [6] Y. Sun et al., "Adaptive learning-based task offloading for vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061–3074, Apr. 2019.
- [7] H. Maleki, M. Basaran, and L. Durak-Ata, "Reinforcement learning-based decision-making for vehicular edge computing," in *Proc. 29th IEEE Signal Process. Commun. Appl. Conf.*, 2021, pp. 1–4.
- [8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
- [9] H. Hasselt, "Double Q-learning," in *Proc. Adv. Neural Inform. Process. Syst.*, vol. 23, 2010, pp. 2613–2621.
- [10] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3826–3839, Sep. 2020.
- [11] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
- [12] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: Architecture, applications, technical issues, and future directions," *Wireless Commun. Mobile Comput.*, vol. 2019, pp. 1–19, 2019.
- [13] J. Moura and D. Hutchison, "Game theory for multi-access edge computing: Survey, use cases, and future trends," *IEEE Commun. Surv. Tuts.*, vol. 21, no. 1, pp. 260–288, Firstquarter 2019.
- [14] Y. Wang et al., "A game-based computation offloading method in vehicular multiaccess edge computing networks," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4987–4996, Jun. 2020.
- [15] C. Zhu et al., "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4150–4161, Jun. 2019.
- [16] J. Sun, Q. Gu, T. Zheng, P. Dong, A. Valera, and Y. Qin, "Joint optimization of computation offloading and task scheduling in vehicular edge computing networks," *IEEE Access*, vol. 8, pp. 10466–10477, 2020.



- [17] J. Feng, Z. Liu, C. Wu, and Y. Ji, "AVE: Autonomous vehicular edge computing framework with ACO-based scheduling," *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10660–10675, Dec. 2017.
- [18] Y. Cui, Y. Liang, and R. Wang, "Resource allocation algorithm with multi-platform intelligent offloading in D2D-enabled vehicular networks," *IEEE Access*, vol. 7, pp. 21246–21253, 2019.
- [19] C. Lin, D. Deng, and C. Yao, "Resource allocation in vehicular cloud computing systems with heterogeneous vehicles and roadside units," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3692–3700, Oct. 2018.
- [20] H. Liao et al., "Learning-based intent-aware task offloading for air-ground integrated vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5127–5139, Aug. 2021.
- [21] Q. Qi et al., "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, May 2019.
- [22] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.
- [23] W. Zhan, C. Luo, J. Wang, G. Min, and H. Duan, "Deep reinforcement learning-based computation offloading in vehicular edge computing," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.
- [24] M. Khayyat, I. A. Elgendy, A. Muthanna, A. S. Alshahrani, S. Alharbi, and A. Koucheryavy, "Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks," *IEEE Access*, vol. 8, pp. 137052–137062, 2020.
- [25] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, pp. 1–24, 2019.
- [26] H. Guo, J. Liu, J. Ren, and Y. Zhang, "Intelligent task offloading in vehicular edge computing networks," *IEEE Wireless Commun.*, vol. 27, no. 4, pp. 126–132, Aug. 2020.
- [27] Y. Liu, J. Yan, and X. Zhao, "Deep reinforcement learning based latency minimization for mobile edge computing with virtualization in maritime UAV communication network," *IEEE Trans. Veh. Technol.*, vol. 71, no. 4, pp. 4225–4236, Apr. 2022.
- [28] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3425–3443, Jun. 2023.
- [29] Y. He, Y. Wang, Q. Lin, and J. Li, "Meta-hierarchical reinforcement learning (MHRL)-based dynamic resource allocation for dynamic vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 4, pp. 3495–3506, Apr. 2022.
- [30] M. Alam, M. Sher, and S. A. Husain, "Integrated mobility model (IMM) for vanets simulation and its impact," in *Proc. Int. Conf. Emerg. Technol.*, 2009, pp. 452–456.
- [31] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," *HotCloud*, vol. 10, no. 4, p. 19, pp. 4–4, 2010, doi: [10.5555/1863103.1863107](https://doi.org/10.5555/1863103.1863107). [Online]. Available: <https://ieeexplore.ieee.org/document/9045579/references#references>
- [32] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
- [33] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.
- [34] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, no. 3/4, pp. 293–321, 1992.
- [35] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [36] L. Xi, L. Yu, Y. Xu, S. Wang, and X. Chen, "A novel multi-agent DDQN-AD method-based distributed strategy for automatic generation control of integrated energy systems," *IEEE Trans. Sustain. Energy*, vol. 11, no. 4, pp. 2417–2426, Oct. 2020.
- [37] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep Q-learning," in *Proc. 2nd Conf. Learn. Dyn. Control*, 2020, pp. 486–489.
- [38] S. D. Whitehead, "A complexity analysis of cooperative mechanisms in reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 1991, pp. 607–613.
- [39] M. Kearns and S. Singh, "Finite-sample convergence rates for q-learning and indirect algorithms," in *Proc. Adv. Neural Inform. Process. Syst.*, 1999, pp. 996–1002.
- [40] S. Goudarzi, M. H. Anisi, H. Ahmadi, and L. Musavian, "Dynamic resource allocation model for distribution operations using SDN," *IEEE Internet Things J.*, vol. 8, no. 2, pp. 976–988, Jan. 2021.
- [41] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A survey on deep learning for Big Data," *Inf. Fusion*, vol. 42, pp. 146–157, 2018.
- [42] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.
- [43] H. Wu, X. Guo, and X. Liu, "Adaptive exploration-exploitation tradeoff for opportunistic bandits," in *Proc. Int. 35th Conf. Mach. Learn.*, 2018, pp. 5306–5314.



**Homa Maleki** (Student Member, IEEE) received the B.Sc. degree in computer engineering from Tabriz University, Tabriz, Iran, and M.Sc. degree in information and communication engineering from Istanbul Technical University, Istanbul, Turkey, where she is currently working toward the Ph.D. degree in information and communication engineering with Informatics Institute. Between 2019 and 2022, she was a Research Assistant with ITU Vodafone Future Lab. Since June 2022, she has been a Data Scientist with NTT Data Business Solutions Turkey. Her research interests include machine learning in communication, optimization algorithms, cloud computing, and mobile edge computing. She is the Member with Information and Communications Research Group.



**Mehmet Başaran** (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical and electronics engineering from Istanbul University, Istanbul, Turkey, in 2008 and 2011, respectively, and the Ph.D. degree in electronics and communication engineering from Istanbul Technical University (ITU), Istanbul, Turkey, in 2018. He was a Research Assistant with ITU. In 2017, he was also a Visiting Researcher with RWTH Aachen University, Aachen, Germany, with the context of the FET EU Horizon 2020 Project. Between 2018 and 2021, he was an R&D Operations Manager with ITU Vodafone Future Lab. Between 2021 and 2023, he was a 5G Research Professional with Siemens Turkey. Since February 2023, he has been a 6G R&D Principal with Turkcell. His general research areas mainly include next-generation communication systems and signal processing for wireless communications.



**Lutfiye Durak-Ata** (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Bilkent University, Ankara, Turkey, in 1996, 1999, and 2003, respectively. From 2004 to 2005, she was with the Statistical Signal Processing Laboratory, Korean Advanced Institute of Science and Technology, Daejeon, South Korea, and with Electronics and Communications Engineering Department, Yildiz Technical University, Istanbul, Turkey, from 2005 to 2015. Since 2015, she has been with the Informatics Institute, Istanbul Technical University, Istanbul, Turkey, where she is currently a Full Professor. Her research interests include wireless communications and networking.