SQ-DDTO: Deep Reinforcement Learning-driven Task Offloading Decisions in Vehicular Edge Computing Networks

Ehzaz Mustafa[a,1,*], Junaid Shuja[b], Faisal Rehman[a], Abdallah Namoun[c], Muhammad Bilal[d], Kashif Bilal[a]

[a]*Department of Computer Science, Comsats University Islamabad, Abbottabad Campus.*
[b]*Department of Computer Science, Southeast Missouri State University, USA*
[c]*AI Centre, Faculty of Computer and Information Systems, Islamic University of Madinah, Madinah 42351, Saudi Arabia.*
[d] *School of Computing and Communications, Lancaster University, United Kingdom*

Abstract

Vehicular Edge Computing offers low latency and reduced energy consumption for innovative applications through computation offloading in vehicular networks. However, making optimal offloading decisions and resource allocation remains challenging due to varying speeds, locations, channel quality constraints, and characteristics of both vehicles and tasks. To address these challenges, we propose a three-layered architecture and introduce a two-level algorithm named Sequential Quadratic Programming-based Dueling Double Deep Q Networks (SQ-DDTO) for optimal offloading actions and resource allocation. The joint computation offloading decision and resource allocation is a mixed integer nonlinear programming problem. To solve it, we first decouple the computation offloading decision sub-problem from resource allocation and address it using Dueling DDQN, which incorporates separate state values and action advantages. This decomposition allows for more granular control of computation tasks, leading to significantly better results. To enhance sample efficiency and learning in such complex networks, we employ Prioritized Experience Replay (PER). By prioritizing experiences based on their importance, PER enhances learning efficiency, allowing the agent to adapt quickly to changing conditions and optimize task offloading decisions in real time. Following this decomposition, we use Sequential Quadratic Programming (SQP) to solve for optimal resource allocation. SQP is chosen due to its effectiveness in handling non-convexity and complex constraints. Moreover, it has strong local convergence properties and utilizes gradient information which is crucial where rapid decision-making is necessary. Experimental results demonstrate the effectiveness of the proposed algorithm in terms of average delay, energy consumption, and task loss rate. For example. the proposed algorithm reduces the system cost by 25.1% compared to DQN and 16.67% compared to both DDQN and DDPG. Similarly. our method reduces the task loss rate by 37.06% compared to DQN, 34.78% compared to DDPG and 10.2% compared to DDQN

Keywords: Mobile Edge Computing (MEC), Computation Offloading, Resource Allocation, Deep Reinforcement Learning (DRL), Deep Q Network(DQN)

1. Introduction

Mobile Edge Computing (MEC) and the Internet of Vehicles (IoV) are two rapidly evolving technologies that hold great potential for enhancing the performance and capabilities of autonomous vehicles [1]-[2]. MEC brings computational resources and storage closer to the network edge, enabling faster processing and reduced latency for various applications. On the other hand, IoV facilitates communication between vehicles and roadside infrastructure, enabling intelligent transportation systems and advanced vehicular services [3]. Integrating MEC with IoV forms Vehicular Edge Computing (VEC), a distributed computing paradigm designed to position computing resources and data storage near vehicles, to minimize latency and improve response times [4]-[5]. This setup allows vehicles to communicate with each other and with roadside infrastructure, enabling various applications, including autonomous vehicles, traffic management, collision avoidance, emergency services, and infotainment. By exchanging information such as position, speed, and road conditions, vehicles can improve safety, efficiency, and comfort on the roads [6]-[7].

However, this exchange of information poses several challenges. Firstly, the dynamic nature of IoV, with high mobility and varying channel conditions, requires careful design of computation offloading and resource allocation strategies to maintain optimal system performance [8]-[9]. In addition, ensuring reliable communication between vehicles and Road Side Units (RSUs) is crucial to maintaining network performance [10]-[11]. Recent studies face limitations when vehicles are anticipated to leave or remain within the connectivity zone of an RSU during task submission and receiving results. These studies often neglect the impact of high-speed mobility, leading to frequent changes in communication channels, potential loss of tasks, and disruptions [12]. Channel factors such as path loss and attenuation and vehicle factors such as speed significantly affect channel quality and are also explored in a limited way.

*Corresponding Author
Email address: aizazmustafa@cuiatd.edu.pk (Ehzaz Mustafa)

Deep Reinforcement Learning (DRL) is a viable option to optimize computation offloading and resource allocation decisions in dynamic vehicular networks [13]-[15]. Recent studies explored Deep Q Networks (DQN) [16] and Deep Deterministic Policy Gradients (DDPG) [17], which are widely used for making intelligent offloading and resource allocation decisions. However, DQN suffers from overestimation bias [18], while DDPG, which relies on noise to explore the action space and is specifically designed for continuous action spaces, might lead to inefficient exploration in discrete spaces [19]. Despite these, recent studies are also limited in anticipating whether vehicles will remain or leave the connectivity zone of Road Side Units (RSUs), considering channel quality factors such as path loss and attenuation, vehicle characteristics such as computational power, etc. These limitations lead to inefficient offloading and resource allocation decisions. Based on these limitations, we propose a two-level Sequential Quadratic Programming-based Dueling Double Deep Q Networks (SQ-DDTO) algorithm that provides an optimal mapping of offloading decisions and corresponding resource allocations. Our three-layered environment models the dynamic network scenario, allowing vehicles to make offloading decisions based on real-time states, such as speed and location within the RSU's zone. Second, we also carefully consider channel factors, vehicle characteristics, and task characteristics such as task time. **Finally, We use SQP instead of convex relaxation because it can effectively handle the non-convexity and complex constraints that usually occur in dynamic task offloading problems. SQP has strong local convergence properties and utilizes gradient information, which makes it optimal for dynamic environments where rapid decision-making is crucial. We provide the following novel contributions.**

- We consider a three-layered architecture for computation offloading where the lower layer encompasses vehicles traveling on a straight path with varying speeds and locations. In the middle layer, we deploy RSUs integrated with computing capabilities that receive tasks from vehicles and are responsible for task execution and result transfer. These RSUs are connected to an edge server with a high-fiber network. We consider a high-end edge server for DRL training in the upper layer. The idea is to relieve the computation burden from RSUs.
- **For optimal computation offloading decisions and resource allocation in highly complex networks while considering several factors such as vehicle speed, location, channel quality, and task characteristics, we propose a novel algorithm, SQ-DDTO, a two-level algorithm. We model the problem as a mixed integer nonlinear programming problem and solve it by first decomposing the original problem into subproblems. Then, we deploy Dueling DDQN for optimal offloading decisions. Due to the dueling architecture, the state values and action advantages are split, hence maximizing the learning ability and ensuring faster convergence. Finally, we use SQP to solve the subproblem of computing resource allocation because of its effectiveness in the non-convexity and complex constraints of dynamic offloading problems.**
- We compare the performance of the proposed algorithm with rule-based schemes and DRL-based schemes, including DQN, DDQN, and DDPG. The results demonstrate the effectiveness of the proposed algorithm in terms of delay, energy, and task loss rate.

2. Related Work

DRL, game theory numerical optimization, and heuristic schemes usually optimize computation offloading. In [20], a lightweight framework for mobile computation offloading is proposed to conserve energy and enhance smartphone performance by transferring compute-intensive tasks from the client to the server. This framework provides app-specific offloading services with customizable features and introduces a multi-task offloading strategy to handle high demand from multiple devices. The server's master node evaluates whether a task should remain on the device or should be offloaded, as well as which virtual machine to utilize. This approach efficiently lowers the energy consumption while enhancing performance for compute-intensive applications. This paper [21] investigates the issue of task offloading in VEC networks when demand is deterministic. It illustrates where tasks come in bursts, making resource scheduling and timely offloading even more difficult. A model for exploiting task characteristics to improve offloading strategies is proposed. The problem is formulated as a 0-1 programming task which has minimized the delays concerning tasks. This paper introduces an energy-efficient offloading algorithm for the interior point method (IPM) and has improved offload utility.

In [22] the authors present a new deep reinforcement learning framework focusing on low coupling and high scalability, enabling effective exploration. The foremost among these innovations is the separation of control dependencies from within an action, action masking, target attention, and a dual-clip PPO that encourages the training of an actor-critic network. The research results include target attention, action-making, control dependency decoupling, and a dual-clip PPO algorithm that enhances the training of the actor-critic network. In the paper [4], a blockchain-empowered VEC framework is proposed to boost the computing power of vehicles by using V2V task offloading. The challenges of vehicles with high mobility and allocating resources among unfamiliar vehicles have been addressed. A DRL-based computation offloading scheme combined with a smart contract running on the blockchain is proposed to enable secure and efficient task offloading. The practical Byzantine fault tolerance mechanism and a consensus algorithm for node selection are used that facilitate efficient consensus-making. It encourages the base stations to enhance reliability in the execution of task allocation.

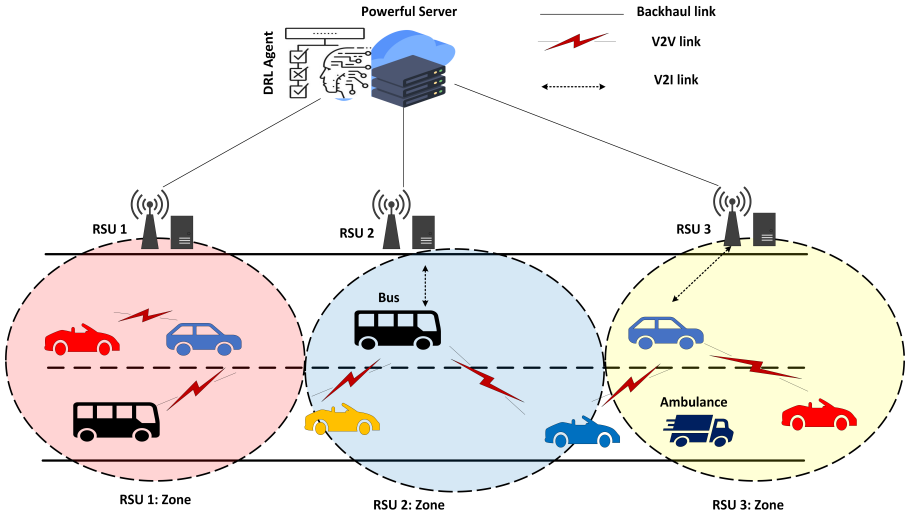

Figure 1: System Architecture

In [24] the authors introduce a new MEC platform through multiple UAVs that uses reinforcement learning to increase QoS and plan path-inefficient environments. This paper considers obstacle-prone environments, cooperation among multiple UAVs to maintain connectivity, and the challenges faced by mobile servers that provide services using UAVs. The main contributions of this research are threefold, i.e., formulation of a combined problem of QoS optimization and path planning in a joint RL framework, the path-inefficient function for modeling demand behavior patterns at terminal users following sigmoid-like behavior, and the introduction of risk into the reward matrix for reinforcement learning. The authors in [25] investigate vehicular edge computing, which can provide higher bandwidth and lower service latency over cloud computing. The paper deals with the challenges resulting from resource scattering and vehicle mobility, it concentrates on energy consumption minimization at edge computing servers. The approach to these problems is adaptive scheduling, which is based balance of an auction-bid scheme used to determine RSUs from the pool, reducing computing requests task dependency graph on a node-core level. It also proposes a deadline-aware queue jump algorithm for task assignment and group scheduling to improve application scheduling. In this paper author proposed a primitive version and experimental results show that it effectively reduces energy consumption with lower response time compared to existing methods.

The paper [26] highlights a key challenge regarding Wireless Body Area Networks is the sink node which is composed of a few energies and computationally restricted sensing devices. This paper proposes a solution for offloading computation tasks that enhances task management using mobile cloud and edge computing. As a result, the solution to an optimization problem targeted at minimizing energy consumption and delay is presented by the authors. The approach is used to reduce load and the offloading costs of computation tasks, as verified by simulations. In [27] discussed the pertinent issues of deploying DLR agents into power-saving constrained resource-sized embedded systems and presented an approach to significantly the Policy Network of RL agents, using Layer-wise Relevance Propagation (LRP) to score on convolution filters for pruning. It casts a compromise between pruning and fine-tuning using Policy Distillation for high efficiency. The proposed method significantly reduces model size and inference time while maintaining performance by comparing relative improvements in various video games of Atari. Overall, this paper outlines major progress toward making RL agents more friendly to embedded applications.

Similarly, the authors in [28] significantly the resource utilization efficiency and QoS issues over multiple time scales of computation offloading of collaboratively vehicle networks (CVNs). Therefore, an asynchronous federated deep Q-learning network-based task offloading algorithm (AF-DQN) is proposed for task management in a semi-distributed learning framework. It also introduces a heuristic queue backlog-aware algorithm for computation resource allocation. Simulations suggest that the proposed approaches dramatically reduce end-to-end queue times. Overall, the study attempts to optimize the throughput in CVNs while ensuring QoS constraints. This paper [29] deals with the problem of data analysis in vehicular networks, where today's vehicles generate huge sensing data. It proposes a new edge-computing-based technique to collaboratively and efficiently locate data in a distributed manner. They suggest strategies for balancing computational costs over the network with the latency to get to vehicles of interest. They achieve strong improvements over baseline methods, with up to a 40X performance gain and up to a 3X reduction in computational overhead compared to prior algorithms. The evolution is data-driven using real-world datasets to demonstrate the performance of their techniques in vehicular networks.

3. System Architecture

We consider a three-layered system, as depicted in Figure 1. The system includes three layers for vehicles, RSUs, and edge servers. In the lower layer, vehicles travel along a straight road. Here each vehicle has an On-Board Unit (OBU) that enables communication with RSUs and handles task processing. We divide the road into M zones. Each zone is covered by one RSU of equal length. $\mathcal{R} = \{R_1, R_2, \ldots, R_M\}$ represents the set of RSUs. It is crucial to note that the coverage of the RSUs does not overlap, so vehicles within the i-th zone are restricted to communicate with only one RSU R_i.

Table 1: Symbols and Their Meanings

Symbol	Meaning
$d_{v,j}(t)$	Distance between vehicle v and RSU j at time t
$w_{v,j}(t)$	Vehicle v's status with RSU j (1 if in range, else 0)
$t_v(t)$	Time to complete task locally on vehicle v at time t
$p_v(t)$	Computational power of vehicle v at time t
$\mathrm{sp}_v(t)$	Speed of vehicle v at time t
$g_{v,j}(t)$	Channel gain between vehicle v and RSU j at time t
$L_v(t)$	Number of tasks assigned to vehicle v at time t
$T_v(t)$	Total number of tasks assigned to vehicle v at time t
X_{ti}	Offloading decision for vehicle i at time t
B_{ti}	Resource allocation for vehicle i at time t
R_{local}	Reward for local computation
α	Weight for energy in the reward function
β	Weight for task loss in the reward function
E	Energy consumed during local computation
L_{off}	Latency of offloading task to RSU
CF	Congestion factor in offloading latency
E_{off}	Energy consumed during offloading
H	Channel gain
P	Proximity penalty based on distance from RSU
$R_{\text{out\_of\_range}}$	Negative reward for being out of range
$V(s)$	State value function in dueling DDQN
$A(s,a)$	Action advantage function in dueling DDQN
$Q(s,a)$	Action-value function in DDQN
γ	Discount factor
y_i'	Target Q-value for each branch in dueling DDQN
$\mathcal{L}_{oss}$	Loss function for updating the network
$\delta(i)$	TD error for tuple i
$p(i)$	Selection probability for tuple i
$w(i)$	Weight for tuple i
ϵ	Exploration rate in ϵ-greedy approach

In the middle layer, $\mathcal{R}$ RSUs are distributed along the road, each connected to an edge server $\mathcal{ES}$ through a fiber-wired connection. In each time interval t, the RSUs receive task requests from vehicles within their coverage area. Depending on the network conditions, traffic factors, computational load, vehicle's energy, task time, etc., RSUs can choose a binary offloading decision for all vehicles in their respective zones using vehicles' state information to reduce the computational load of vehicles and to minimize the latency, energy, and task loss rate. Each R_i has a coverage radius L_i and possesses computational capabilities defined by CPU processing frequency f_{rsu_i}. Due to the limited computational power, waiting tasks that require immediate execution are queued. The queue length at R_i is expressed as

$$q_{\text{rsu}_i} = \sum_{t_k \in Q_i} c_k \tag{1}$$

where c_k denotes the CPU cycles required for the task t_k in the queue [33].

In each time interval, the set of vehicles within the coverage of R_i is$\{v_{i,1}, v_{i,2}, \ldots, v_{i,K}\}$. The current task of each vehicle $v_{i,K}$ is depicted as

$$t_{i,k} = \{d_{i,k}, c_{i,k}, t_{\max_{i,k}}\} \tag{2}$$

Here, $d_{i,k}$ denotes the task data size generated by the vehicle v_i, $c_{i,k}$ represents the CPU cycles that are required to complete the task and $t_{\max_{i,k}}$ is the maximum tolerable latency of the given task. The tasks not offloaded and executed by the vehicle itself are processed on the OBU in local computing mode. The computational capacity of each vehicle $v_{i,K}$ is specified by its CPU processing frequency f_{v_i}, and length of the task queue by q_{v_i}. This means that the state of vehicle $v_{i,K}$ is described by its computational state $\{f_{v_i}, q_{v_i}\}$ and the information of the task $t_{i,k} = \{d_{i,k}, c_{i,k}, t_{\max_{i,k}}\}$, collectively denoted as

$$o_{v_i} = \{f_{v_i}, q_{v_i}, d_{i,k}, c_{i,k}, t_{\max_{i,k}}\} \tag{3}$$

Through this model, task offloading decisions of vehicles are depicted as at each time slot t, vehicles $\{v_{i,1}, v_{i,2}, \ldots, v_{i,K}\}$ within each R_i's range possess an offloading decision $\{x_{T_{i,1}}, x_{T_{i,2}}, \ldots, x_{T_{i,K}}\}$, where $x_{T_{i,K}} \in \{0, 1\}$, which is a binary decision. Here, 0 represents the local computation and 1 represents offloading. We aim to learn the optimal offloading decision, which ensures optimal computational resource management and minimizes overall costs in terms of delay, energy consumption, and task loss rate during task offloading modes.

3.1. Computation Model

In our proposed model, the set of vehicles in zone i at time t, denoted by $v_i(t)$, should complete their computing tasks before leaving the zone. We anticipate that vehicles will not exit the respective RSU zone during each time interval t. If some vehicle is expected to leave the zone and cannot complete the task in the time frame t, it should execute it locally. The task offloading decision for all vehicles in $v_i(t)$ is represented by $x_i(t)$, a binary variable defined as:

$$x_i(t) = \begin{cases} 1, & \text{if computation load } L_i(t) \text{ is offloaded to } R_i, \\ 0, & \text{if processed locally by the vehicles.} \end{cases} \tag{4}$$

3.1.1. Local Computation Model

We assume all vehicles have OBU with CPU processing frequency, denoted by f_v. The number of CPU cycles required to process one bit of data is denoted by c. In this case, the average local computing delay for the computation load $L_i(t)$ is given by:

$$T_{\text{local}_i}(t) = \frac{c \cdot L_i(t)}{f_v \cdot n_i(t)} \cdot (1 - x_i(t)), \quad i \in M, \tag{5}$$

where $n_i(t)$ is the number of vehicles in zone i at time t, and $x_i(t) = 0$ corresponds to local processing.

3.1.2. Remote Computation Model

In the edge computing model, each RSU R_i has to perform multiple tasks in parallel. The computational resources of the RSU R_i are limited and vary dynamically over time. The available computing resources of R_i at time t are denoted by $f_i(t)$.

If the computation load $L_i(t)$ is offloaded to R_i, the processing delay includes the vehicle to RSU communication delay and the RSU execution delay. The RSU R_i has to complete the task independently rather than sharing the computation load with adjacent RSUs. In this scenario, when the computation load $L_i(t)$ is completely executed at the respective RSU, the average computing delay is given by:

$$T_{\text{comp}_i}(t) = \frac{c \cdot L_i(t)}{f_i(t) \cdot n_i(t)}. \tag{6}$$

In this dynamic IoV scenario, vehicle-generated tasks must be processed while the vehicles are within the RSU's coverage area. It is assumed that the task must be processed within the same time slot. If not, it will be discarded by the RSU. Hence, RSUs must make the optimal decision to minimize the task loss rate.

3.2. Communication Model

For better communication, we assume each $\mathcal{R}$'s radio access utilizes Orthogonal Frequency Division Multiple Access (OFDMA). OFDMA is an efficient communication radio access method that divides the available spectrum into multiple sub-carriers. This enables parallel communication by assigning different sub-carriers to different vehicles.

For vehicle-to-RSU communication, the Signal to Interference plus Noise Ratio (SINR) from vehicle v_i to RSU $\mathcal{R}_j$ is expressed as:

$$\gamma_{v_{i,j}} = \frac{p_{v_{i,j}} H_{v_{i,j}}}{N_0 W + I_{v_{i,j}}}, \tag{7}$$

where $H_{v_{i,j}}$ denotes the channel gain. Here, $p_{v_{i,j}}$ denotes the vehicle's transmit power and is used for the vehicle to RSU communication. The channel capacity is W, and N_0 is the Gaussian white noise power spectral density. Hence, the Gaussian white noise power of the channel is depicted as $N_0 W$. Co-channel interference, denoted as $I_{v_{i,j}}$, arises from other vehicles using the same channels, while inter-channel interference is not present due to the orthogonality of OFDMA.

The maximum transmit power for vehicle i is $P_{v_{i,j}}$. Therefore, the transmit power $p_{v_{i,j}}$ must satisfy the constraint:

$$p_{v_{i,j}} \in [0, P_{v_{i,j}}], \quad i \in N_j, \ j \in M. \tag{8}$$

By using the Shannon capacity formula for a channel, the vehicle to RSU communication rate for the uplink task from vehicle i to RSU j is given by:

$$r_{v_{i,j}} = W \log_2 \left(1 + \gamma_{v_{i,j}}\right). \tag{9}$$

Hence, the vehicle to RSU task communication delay for vehicle i is computed as:

$$t_{v_{i,j}} = \frac{d_{in_{i,j}}}{r_{v_{i,j}}}, \tag{10}$$

where $d_{in_{i,j}}$ symbolizes the size of task data to be transmitted. The transmission power allocation for all vehicles into the set P is given by:

$$P = \{p_{v_{i,j}}\}_{i \in N_j, j \in M} \tag{11}$$

For a vehicle to RSU communication, we formulate the SINR from RSU $\mathcal{R}$ ($l \in L_{i,j}$) to vehicle i as:

$$\gamma_{r_{v_{i,j,l}}} = \frac{P_{r_{v_{i,j,l}}} H_{r_{v_{i,j,l}}}}{N_0 W + I_{r_{v_{i,j,l}}}}, \tag{12}$$

where $P_{r_{v_{i,j,l}}}$ is the transmit power of RSU $\mathcal{R}$ for sending the results of the task to vehicle i, the channel gain in the equation is $H_{r_{v_{i,j,l}}}$, $N_0 W$ is the Gaussian white noise power of the channel, and $I_{r_{v_{i,j,l}}}$ is the co-channel interference. By using the orthogonality of OFDMA, we ensure that there is no inter-channel interference.

The RSU to vehicle communication rate for the downlink from RSU $\mathcal{R}$ to vehicle i can be expressed as:

$$r_{r_{v_{i,j,l}}} = W \log_2 \left(1 + \gamma_{r_{v_{i,j,l}}}\right). \tag{13}$$

Thus, the RSU to vehicle task causes a communication delay from RSU $\mathcal{R}$ to vehicle i is given by:

$$t_{r_{v_{i,j,l}}} = \frac{d_{out_{i,j}}}{r_{r_{v_{i,j,l}}}}, \tag{14}$$

where $d_{out_{i,j}}$ denotes the size of the task result data to be transmitted [31].

3.2.1. Energy Consumption

The energy consumption for local computational processing for vehicle i is given by:

$$e_{v_{i,j}} = \kappa_{i,j} (f_{v_{i,j}})^2 \cdot c_{i,j}, \tag{15}$$

where the constant representing the effective switched capacitance is given by $\kappa_{i,j}$, which depends on the chip architecture.

In the case of task offloading of the vehicle i, we consider the energy consumption only during the vehicle to RSU task communication, which is given as:

$$e_{v_{r_{i,j}}} = p_{v_{i,j}} \cdot t_{v_{i,j}}. \tag{16}$$

3.2.2. Processing Cost

Our system model considers the processing cost of a vehicle's task as a weighted sum of the energy consumption and task execution delay. Let $w_{i,j}$ be the weight factor for vehicle i. In our proposed work, there are two cases of task processing. First, the task is processed locally, and second, the task is offloaded to the corresponding RSU. In case of local execution, we denote $\alpha_{i,j} = 0$, and $t_{v_{i,j}}$ is the task processing latency for vehicle i, and $E_{v_{i,j}}$ is the energy consumption. Hence, the total cost can be expressed as follows:

$$u_{i,j}^{(1)} = \alpha_{i,j} \left(t_{v_{i,j}} + w_{i,j} E_{v_{i,j}} \right). \tag{17}$$

In case of remote execution $\beta_{i,j,j} = 0$, the task processing delay for vehicle i encompasses the vehicle to RSU task communication delay from vehicle i to RSU j, denoted as $t_{v_{r_{i,j}}}$, the execution delay at the RSU j, denoted as $t_{r_{i,j,j}}$, and the task result migration delay from RSU j to vehicle i, denoted as $t_{r_{v_{i,j,j}}}$.

The energy consumption, in this case, is represented by the energy consumed during the vehicle to RSU transmission process, $E_{v_{r_{i,j}}}$. Furthermore, vehicle i must be within the coverage area of RSU j to receive the task result transmitted from RSU. Mathematically:

$$\beta_{i,j,j} \left(t_{v_{r_{i,j}}} + t_{r_{i,j,j}} + t_{r_{v_{i,j},j}} \right) \leq \chi_{i,j}, \quad i \in N_j, \ j \in M. \tag{18}$$

Hence, we can express the total task processing mathematically as:

$$u_{i,j}^{(2)} = \beta_{i,j,j} \left(t_{v_{r_{i,j}}} + t_{r_{i,j,j}} + t_{r_{v_{i,j},j}} + w_{i,j} E_{v_{r_{i,j}}} \right). \tag{19}$$

3.3. Vehicular Mobility Model

We assume that vehicles may enter or leave the road and zones at any time except during the time interval t. In this time frame, the set of vehicles within the coverage area of RSU r_i is denoted by $v_i(t)$. We consider that the duration of a time slot t is short (e.g., milliseconds), so the number of vehicles within the coverage area of RSU r_i during a one-time slot can be considered constant and is denoted by $n_i(t)$. However, $n_i(t)$ may vary over longer time scales due to changes in traffic flow dynamics. Thus, the average number of vehicles $v_i(t)$ is related to the average speed $s(t)$ on the road, which can be expressed as:

$$s(t) = s_{\text{lim}} \left(1 - \frac{n(t)}{n_{\text{max}}} \right) \tag{20}$$

Here, $s(t)$ represents the average speed of vehicles at time t, s_{lim} is the maximum speed limit for vehicles on the road, $n(t)$ represents the average number of vehicles in the zone at time t, and n_{max} is the maximum number of vehicles that can be accommodated in the zone.

4. Problem Formulation

We aim to optimize the offloading decisions and resource allocation to minimize the total cost associated with the energy, latency, and task loss rate. Simply, the objective function is to minimize the average total cost over time T, which includes energy consumption, latency, and task loss rate:

$$(P1) \quad \min_{x_{v,i}(t), a_{v,j}(t)} \frac{1}{T} \sum_{t=0}^{T-1} \left(\alpha \cdot E(t) + \beta \cdot L(t) + \gamma \cdot R(t) \right) \tag{21}$$

where, $x_{v,i}(t) \in \{0, 1\}$ is the offloading decision for vehicle v for task i at time t. Here, $x_{v,i}(t) = 1$ indicates that task i is offloaded to the RSU, and $x_{v,i}(t) = 0$ indicates that the task is computed locally. $a_{v,j}(t)$ represents the decision on the allocation of resources for vehicle v concerning RSU j at time t. $E(t)$ denotes the total energy consumption at time t. $L(t)$ denotes the total latency at time t. $R(t)$ denotes the total task loss rate at time t. α, β, γ are the weights associated with energy consumption, latency, and task loss rate, respectively.

The optimization problem is subject to the following constraints:

- **C1:** The offloading decision $x_{v,i}(t)$ must be binary:

$$x_{v,i}(t) \in \{0, 1\} \quad \forall v \in V, \forall i \in I, \forall t \in T \tag{22}$$

- **C2:** The resource allocation decision $a_{v,j}(t)$ must be between 0 and 1:

$$0 \leq a_{v,j}(t) \leq 1 \quad \forall v \in V, \forall j \in J, \forall t \in T \tag{23}$$

- **C3:** The total resource allocation across all RSUs for a vehicle must sum up to 1:

$$\sum_{j \in J} a_{v,j}(t) = 1 \quad \forall v \in V, \forall t \in T \tag{24}$$

- **C4:** The latency of processing a task must not exceed a predefined maximum delay:

$$L_{v,i}(t) \leq L_{\text{max}} \quad \forall v \in V, \forall i \in I, \forall t \in T \tag{25}$$

- **C5:** The total energy consumption must be within acceptable limits:

$$E_v(t) \leq E_{\text{max}} \quad \forall v \in V, \forall t \in T \tag{26}$$

- **C6:** The task loss rate must not exceed a predefined threshold:

$$R_v(t) \leq R_{\text{max}} \quad \forall v \in V, \forall t \in T \tag{27}$$

- **C7:** The vehicle should not leave the corresponding RSU zone before receiving results:

$$\beta_{i,j,j} \left(t_{v_{r_{i,j}}} + t_{r_{i,j,j}} + t_{r_{v_{i,j},j}} \right) \leq \chi_{i,j}, \quad i \in N_j, \ j \in M. \tag{28}$$

5. Proposed Solution

As our complete problem is a mixed integer non-linear programming (MINLP) problem due to the mixed discrete and continuous variables, the objective function, and non-convex constraints, it is difficult to solve by numerical methods directly. To solve this, we propose a sequential quadratic programming-based dueling DDQN (SQ-DDTO) for optimal task offloading decisions and resource allocation. To address the computing resource allocation problem, we can decouple the original problem (P1). Following this decoupling, the Dueling DDQN is used for binary offloading decisions, and computing resource allocation can be naturally decoupled. This allocation is indicated by the set $\mathcal{F}$, which includes the CPU resources allocated in the vehicle v ($\mathcal{F}_v$) and in the Roadside Unit (RSU) r ($\mathcal{F}_r$). We aim to minimize the total latency, energy consumption, and task loss rate across the vehicular edge network while satisfying the constraints of task offloading decisions and resource allocation. Hence, we can express the resource allocation problem by decoupling it from the offloading decisions as follows:

$$\text{P2:} \min_{\mathcal{F}_v, \mathcal{F}_r} \quad C_{\text{total}}(\mathcal{F}) = \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij}(\mathcal{F}_{v_i}, \mathcal{F}_{r_j}), \tag{29}$$

$$\text{s.t.} \quad C1, C2, C3, C4, C5, C6, C7. \tag{30}$$

The given problem, P2, is a highly dimensional and nonlinear optimization problem given by multiple constraints. We use the Sequential Quadratic Programming (SQP) algorithm, which is optimal for handling nonlinearity in multiple constraints. By using this, the Lagrangian function for the optimization problem can be defined as:

$$\mathcal{L}(\mathcal{F}, \lambda, \mu) = C_{\text{total}}(\mathcal{F}) + \lambda^T g(\mathcal{F}) + \mu^T h(\mathcal{F}) \tag{31}$$

Here, $g(\mathcal{F})$ and $h(\mathcal{F})$ denote equality and inequality constraints, respectively, and λ and μ are the associated Lagrange multipliers. At each iteration k, the SQP algorithm solves a quadratic programming (QP) subproblem as:

$$\min_{S_k} \quad \nabla \mathcal{L}(\mathcal{F}_k, \lambda_k, \mu_k)^T S_k + \frac{1}{2} S_k^T B_k S_k, \tag{32}$$

$$\text{s.t.} \quad g(\mathcal{F}_k) + \nabla g(\mathcal{F}_k)^T S_k = 0, \quad h(\mathcal{F}_k) + \nabla h(\mathcal{F}_k)^T S_k \leq 0, \tag{33}$$

where S_k denotes the search direction and B_k is the Hessian matrix. It is approximated using the Broyden Fletcher Goldfarb Shanno method. Hence, the update rule for the allocation is given by:

$$\mathcal{F}_{k+1} = \mathcal{F}_k + \alpha_k S_k, \tag{34}$$

where α_k denotes the step size obtained using a line search method such as the Wolfe conditions. This iterative process continues with the factors needed until convergence to the optimal solution $\mathcal{F}^*$, the optimal allocation of computing resources for each vehicle in the corresponding zone. Algorithm 1 provides the mathematical steps of the proposed SQP-based resource allocation strategy.

Algorithm 1 SQP-based Resource Allocation Strategy

Require: Initial resource allocation $\mathcal{F}_0$, convergence tolerance ϵ, offloading decision α_{ij}
Ensure: Optimal resource allocation $\mathcal{F}^*$

1: Initialize iteration counter $k = 0$ and symmetric positive definite matrix B_0
2: **while** true **do**
3: Formulate the Lagrangian $\mathcal{L}(\mathcal{F}_k, \lambda_k, \mu_k)$
4: Solve the QP subproblem:

$$S_k = \arg\min_{S_k} \nabla \mathcal{L}(\mathcal{F}_k, \lambda_k, \mu_k)^T S_k + \frac{1}{2} S_k^T B_k S_k$$

$$\text{s.t. } g(\mathcal{F}_k) + \nabla g(\mathcal{F}_k)^T S_k = 0, \tag{35}$$

$$h(\mathcal{F}_k) + \nabla h(\mathcal{F}_k)^T S_k \leq 0 \tag{36}$$

5: Perform line search to find step size α_k that satisfies Wolfe conditions
6: Update resource allocation:

$$\mathcal{F}_{k+1} = \mathcal{F}_k + \alpha_k S_k \tag{37}$$

7: **if** convergence criterion $\|\mathcal{F}_{k+1} - \mathcal{F}_k\| < \epsilon$ is met **then**
8: Set $\mathcal{F}^* = \mathcal{F}_{k+1}$ and break
9: **end if**
10: Update Hessian matrix B_{k+1} using BFGS method
11: Increment iteration counter $k = k + 1$
12: **end while**
13: **return** Optimal resource allocation $\mathcal{F}^*$

5.1. state space

The state space provides the system information observed by the agent at the beginning of each time interval t. Our state space encompasses various parameters that are crucial to the decision-making process. Specifically, our the state space $\mathcal{S}$ is defined as follows:

$$s_t = \left[d_{v,j}(t), w_{v,j}(t), t_v(t), p_v(t), \text{sp}_v(t), g_{v,j}(t), L_v(t), T_v(t) \right] \tag{38}$$

where: $d_{v,j}(t)$ denotes the distance between vehicle v and RSU j at time t. $w_{v,j}(t)$ indicates whether the vehicle is within the range of RSU j (1 if within range, otherwise 0). $t_v(t)$ denotes the time required to complete the task if computed locally at time t. $p_v(t)$ is the computational power of the vehicle v at time t. $\text{sp}_v(t)$ is the speed of vehicle v at time t. $h_{v,j}(t)$ is the channel gain between vehicle v and RSU j at time t. $L_v(t)$ is the number of tasks currently assigned to vehicle v at time t, and $T_v(t)$ denotes the total number of tasks assigned to vehicle v .

5.2. Action Space

The action space is the set of all possible actions an agent can take during decision-making. In our designed action space, the agent can choose between local execution or offloading based on specific parameters. Hence, the action space vector for offloading at time t is

$$X_t = \{x_{t1}, x_{t2}, \ldots, x_{tn}\},$$

where x_{ti} represents the offloading decision for vehicle i at time t.

5.3. Reward Functions

We aim to minimize the overall cost associated with the task offloading problem by assigning weights to distinct cost components. Hence, the reward for local computation is formulated as follows:

$$R_{\text{local}} = -L - \alpha E - \beta L_{\text{loss}} \tag{39}$$

Here, (L) represents the latency, which is the time t taken to complete the task when computation is performed locally on the vehicle itself. Lower latency results in a higher reward. Second, the energy consumed during local computation is calculated as the product of the vehicle's computational power P_{comp} and the task duration t. Hence, minimizing energy consumption results in improved reward. Finally, we also add a penalty function to the reward function to satisfy the constraints of the optimization problem, which is given as (L_{loss}). This penalty applies to tasks lost due to excessive latency, poor channel quality, etc. Fewer lost tasks lead to a higher reward.

$$E = P_{\text{comp}} \times t \tag{40}$$

On the other hand, the reward for offloading can be expressed as:

$$R_{\text{off}} = -\frac{L_{\text{off}}}{G \times S} - \alpha E_{\text{offload}} - \beta L_{\text{loss}} - \gamma P \tag{41}$$

$$L_{\text{off}} = \frac{t \times (1 + \text{CF})}{G \times S} \tag{42}$$

Here, L_{off} is the time required to complete the task when offloaded to an RSU. This is done by considering congestion factor (CF), channel gain (H), and vehicle speed (S). Thus, higher latency results in a lower reward. Next, (E_{offl}) is the energy required for offloading, which is calculated as the product of the energy consumption rate (ECR) and offloading latency (L_{off}). Hence, reducing energy consumption significantly improves the reward and can be expressed as:

$$E_{\text{off}} = \text{ECR} \times L_{\text{off}} \tag{43}$$

Similarly, the channel gain (H), which represents the quality of the communication channel and is derived from path loss and other factors, also contributed to the reward function. A higher channel gain improves the reward. For example:

$$H = 10^{-\text{PL}/10} \tag{44}$$

We also add a proximity penalty denoted as (P), which represents the penalty based on the distance from the RSU (d_{RSU}) relative to the maximum range ($d_{\max}$). Thus, a higher proximity penalty reduces the reward.

$$P = \frac{d_{\text{RSU}}}{d_{\max}} \tag{45}$$

Finally, we add a significant negative reward if the vehicle is out of range of any RSU, discouraging offloading decisions. Thus, the negative reward for being out of range is given by $R_{\text{out\_of\_range}} = -100$.

5.4. Proposed Dueling DDQN for enhanced Q-Value Estimation

After decoupling the resource allocation sub-problem, we model the rest of the problem as a Markov Decision Process (MDP). This mainly includes state space, action space, and reward function, which we discussed above. We implement a DDQN using a dueling architecture to make optimal binary offloading decisions while balancing the latency, energy consumption, and task loss rate. We consider a neural network which is a multi-layer perceptron, with an input layer with 8 neurons that accepts a state vector to approximate the Q function where each neuron corresponds to a parameter in the state space. Second, we consider three hidden layers with 64, 32, and 16 neurons with RelU activation function in each hidden layer. These layers are responsible for extracting relevant features for decision-making, and the output layer has the sigmoid activation function. Finally, we split the network into two streams. In the first stream, we estimate the value of being in a particular state, independent of the action being taken. This includes a fully connected layer and RelU activation function, followed by an output layer with a single neuron representing the state value. In the second stream, we estimate the advantage of each action, followed by an output layer generating possible actions.

To improve the stability and generalization of our proposed algorithm, the Q-value estimation is divided into two parts, including the state value function $V(s)$ and the action advantage function $A(s, a)$. This separation allows the model to independently evaluate the value of states and the advantage of actions, leading to more effective learning. The action-value function $Q(s, a)$ is computed as follows:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s,a') \right) \quad (12) \tag{46}$$

In our proposed algorithm, the combined Q-value function for each action in dimension d is given by:

$$Q_d(S, X_i) = V(S) + \left(A_d(S, X_i) - \frac{1}{n} \sum_{X' \in \mathcal{D}} A_d(S, X') \right) \quad (13) \tag{47}$$

Here, Q_d represents the Q-value of action X_i in dimension d, $V(S)$ is the state value, $A_d(S, X_i)$ is the advantage value for action X_i, and $\mathcal{D}$ denotes the set of actions in dimension d. The optimal action X_i in each dimension is selected as:

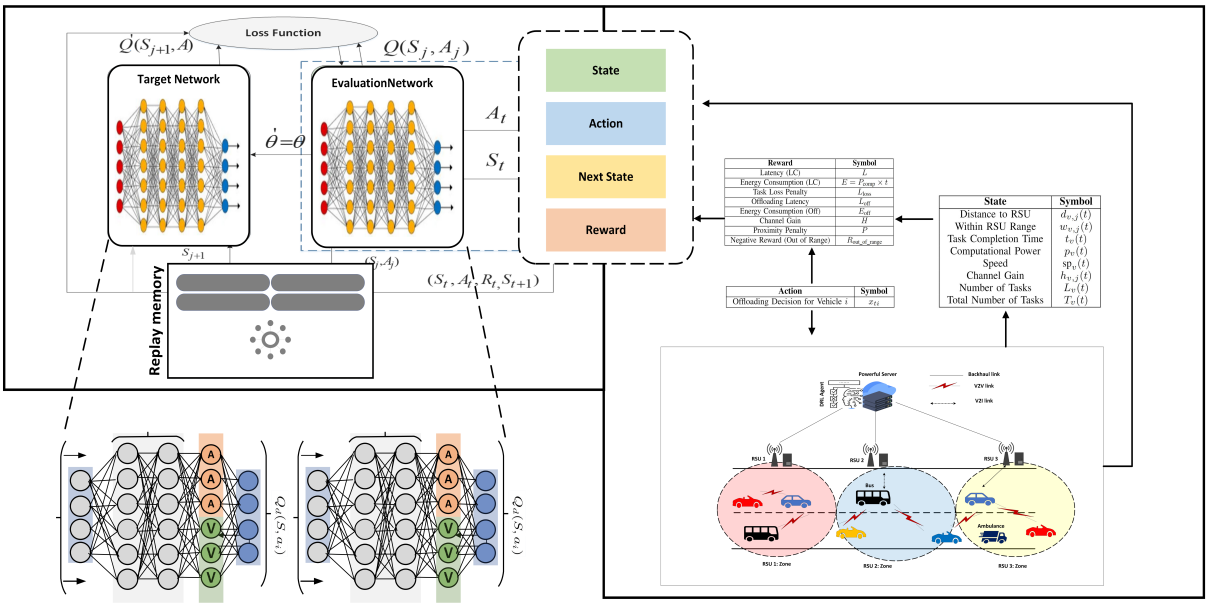

Figure 2: Proposed SQ-DDTO Algorithm

$$X_i = \arg\max_{X'_d \in \mathcal{A}_i} Q_i(S, X'_d) \quad (14) \tag{48}$$

The overall action vector, which includes both the offloading and local computation, is represented as:

$$X = [X_{x1}, X_{x2}, \ldots, X_{xN}] \quad (15) \tag{49}$$

Here, X_{xi} denotes the offloading decision for each vehicle i. As we aim to improve the generalization and stability of the learning process using dueling architecture, we separate the estimation of Q-values into two distinct components of the state value function $V(s)$ and the action advantage function $A(s, a)$. This separation improves the efficiency of action learning. In our proposed algorithm, the target Q-value y'_i for each branch is computed as:

$$y'_i = R_t + \gamma Q'_i(S_{t+1}, \arg\max_{a'_d \in \mathcal{D}_i} Q_i(S_{t+1}, a'_d)) \quad (16) \tag{50}$$

where R_t is the immediate reward, γ is the discount factor and $Q'_i(S_{t+1}, a'_d)$ represents the Q value for the next state S_{t+1} estimated by the target network. The loss function used to update the online network's parameters is:

$$\mathcal{L}_{oss} = \mathbb{E}\left[\frac{1}{2N}\sum_d (y'_d - Q_d(S_j, A_j, d))^2\right] \quad (17) \tag{51}$$

Here, $Q_d(S_j, A_j, d)$ is the Q-value for the action dimension d in the state S_j with action A_j, and y'_d is the target Q-value for branch d.

We use PER to improve sample selection efficiency. The probability $p(i)$ of selecting the i-th tuple is given by:

$$p(i) = \frac{(|\delta(i)| + \epsilon)^\alpha}{\sum_j (|\delta(j)| + \epsilon)^\alpha} \tag{52}$$

where $\delta(i)$ represents the TD error, corresponding to the difference between the expected and predicted Q-values. ϵ is a small constant to avoid zero probabilities. and α controls the level of prioritization.

The cumulative difference $\delta(i)$ is calculated as:

$$\delta(i) = |y_i - Q(s_i, a_i)| \tag{53}$$

To balance the impact of different losses and prevent overfitting, the weight $w(i)$ for each tuple is computed as:

$$w(i) = \left(\frac{1}{N \cdot p(i)}\right)^\beta \tag{54}$$

Moreover, we use ϵ-greedy approach as our exploration. In this setup, the agent explores the environment with high probability ($\epsilon = 1.0$) to encourage diverse experiences. After this, ϵ decays exponentially at a rate of 0.995 until it reaches a minimum threshold of 0.01. This strategy focuses on exploiting the learned policy. Finally, our agent uses a PER buffer to store past experiences and sample experiences based on their importance, as determined by the TD error. Hence, PER enhances learning efficiency by focusing on higher values to improve the policy, accelerating the learning process efficiently.

6. Experiment Results and Evaluations

Our proposed algorithm is developed in Python. We simulate the SQ-DDTO in a custom-built environment where vehicles are modeled with varying speeds and locations on a straight path which simulates real-world traffic settings.

Algorithm 2 Dueling DDQN-Based Offloading Decision Algorithm with Prioritized Experience Replay

Input: Initial parameters θ, θ', maximum episodes E, maximum steps per episode τ, exploration rate ε, prioritization factor α, and importance sampling factor β.
Output: Optimized offloading decisions for minimizing latency and energy consumption.
Initialize prioritized replay memory buffer with a maximum capacity.
Initialize online network parameters θ with random weights.
Initialize target network parameters θ' such that $\theta' = \theta$.
Set epsilon (ε) for the epsilon-greedy strategy.
for $episode = 1$ to E **do**
  Observe initial state S_0.
  for $t = 1$ to τ **do**
    With probability ε, select a random action A_t.
    Otherwise, select $A_t = \arg\max_a Q(S_t, a; \theta)$.
    Execute action A_t, receive reward R_t, and observe next state S_{t+1}.
    Compute the TD-error $\delta_t = |R_t + \gamma \max_{a'} Q(S_{t+1}, a'; \theta') - Q(S_t, A_t; \theta)|$.
    Use the TD-error to calculate the priority $p_t = (\delta_t + \epsilon)^\alpha$ and store the transition (S_t, A_t, R_t, S_{t+1}) in the replay memory buffer with priority p_t.
    Sample a minibatch of transitions (S_j, A_j, R_j, S_{j+1}) from the buffer based on their priorities.
    Calculate importance-sampling weights $w(i) = \left(\frac{1}{N \cdot p(i)}\right)^\beta$.
    Calculate the target Q-value using:

$$y_j = R_j + \gamma Q(S_{j+1}, \arg\max_{a'} Q(S_{j+1}, a'; \theta); \theta')$$

    Perform a gradient descent step on the weighted loss:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{j=1}^{N} w_j (y_j - Q(S_j, A_j; \theta))^2$$

    Update target network parameters θ' every t_{update} steps: $\theta' = \theta$.
    Update the current state S_t to the next state S_{t+1}.
  end for
  Decay epsilon ε for exploration-exploitation balance.
  Decay importance-sampling factor β towards 1 over time.
end for

Table 2: Hyperparameters of the SQ-DDTO Algorithm

Parameter	**Symbol**	**Value**
State Size	S	8
Action Size	A	2
Discount Factor	γ	0.9
Initial Epsilon	ϵ_0	1.0
Epsilon Decay Rate	ϵ_{decay}	0.995
Minimum Epsilon	ϵ_{min}	0.01
Batch Size	B	32-64
Max Episodes	E	500
Max Steps per Episode	T_{max}	500
Soft Update Weight	τ	0.995

With the high mobility models, vehicles change their positions and their locations are updated accordingly. RSUs are modeled at random intervals and equipped with computing capabilities. Moreover, channel quality between vehicles and RSUs is modeled using standard path loss and attenuation models to reflect realistic communication constraints. We evaluate our proposed algorithm SQ-DDTO with multiple experiments, performing different simulation parameters such as ranges, number of RSUs and vehicles, and setup of hyperparameters. We applied SQ-DDTO with different neurons, layers, discount factors, learning rates, memory sizes, weights, activation functions, etc., and selected the optimal one. We choose only those hyperparameter settings that maximize our reward. Our state space consists of 8 features, and the action space is binary, representing the offloading decisions. We use ReLU as an activation function for all hidden layers and set the learning rate to 0.01**. The discount factor is** 0.9**. We also manage the exploration-exploitation trade-off using an ϵ-greedy policy, where the initial value of ϵ is** 1.0**. In this setup, we use a decay rate of** 0.995 **and a minimum ϵ value of** 0.01**. [30]-[31]**

Table 3: Simulation Parameters

Definition	**Value**
Range of RSU	[100-300] m^2
Size of Task	[0.1-5] MB
Number of Vehicles	[10, 50]
Transmit Power	[0-24] dBm
Background Noise Variance	-100 dBm
System Bandwidth	20 MHz
Computing Capability of RSU	5 GHz
Computing Capability of Vehicles	[0.1-1] GHz
Energy Coefficient of Vehicles	5×10^{-27}
Weights of Time Cost	0.6
Weights of Energy Cost	0.4
Number of Base Stations	3-5

6.1. Convergence Evaluations

For better convergence and efficient and stable learning, we employ a dual architecture, focusing on the value of states and the advantage of actions independently. We also use PER, which

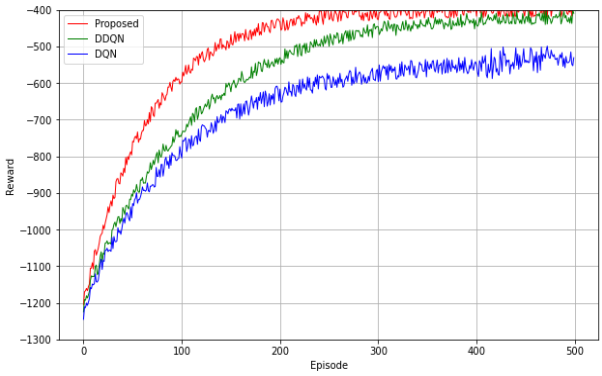

Figure 3: Convergence analysis of proposed algorithm in terms of reward over episodes

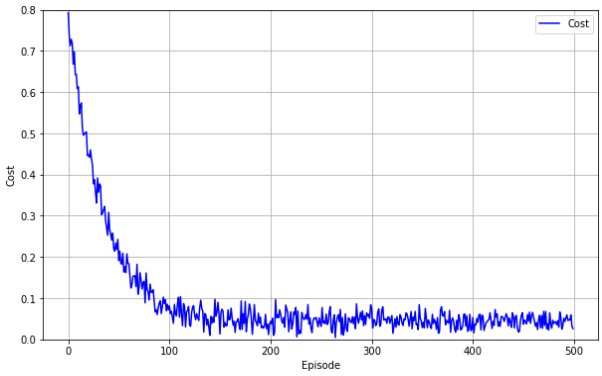

Figure 4: Training cost of the proposed algorithm over episodes

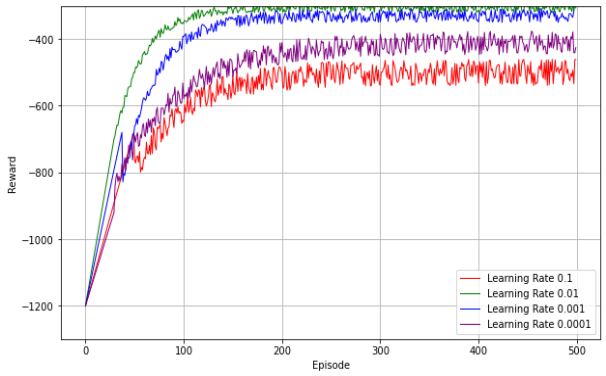

Figure 5: Performance of different learning rates over episodes

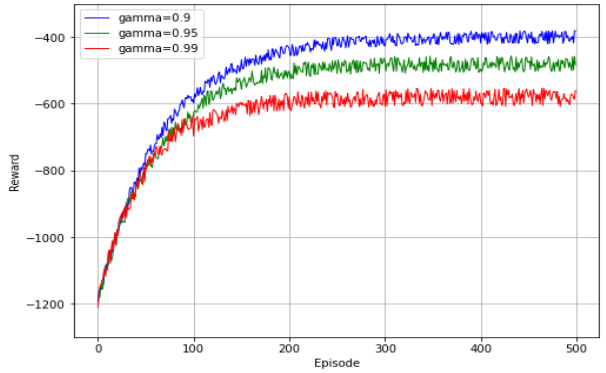

Figure 6: Performance of different discount values over episodes

can lead to faster convergence. In this setup, the agent learns more effectively by concentrating on experiences that are more likely to lead to significant policy or value function updates. Figure 3 shows the convergence analysis of the proposed algorithm in terms of rewards over training episodes. One can see that the proposed algorithm converges quickly with an increase in training episodes and becomes stable at around 250th episodes. We evaluated and compared our system with DQN and DDQN. It can be seen that DDQN slightly converges around the 380th episode, while DQN does not converge in the first 500 episodes. This is due to the ability of dueling architecture for better generalization and learning suitability, along with PER for faster convergence.

In figure 4, we provide a curve showing the training loss over episodes, which shows the superior convergence and learning ability of our proposed algorithm. As we utilize the dueling DDQN, enhanced by PER, we can infer that the training loss decreases over time as the agent's improved learning for optimal offloading decisions. It can also be seen that the loss decreased quickly in initial episodes due to the focus of the agent on high-priority experiences. After 250 episodes, the loss stabilizes, achieving minimal training cost values. This shows the stabilization of the agent's learning using dueling DDQN for optimal offloading actions with improved values and advantages estimation.

In figure 5, we illustrate the performance of the proposed algorithm with different learning rates, including 0.1, 0.01, 0.001, and 0.0001. The purpose is to validate the convergence of the proposed algorithm with optimal learning. With a minimal learning rate, the agent experiences significant fluctuations and unstable convergence. This reflects the highly volatile learning process. The learning rate of 0.01 and 0.001 provides substantial and stable convergence while achieving minimal loss and maximizing reward. However, these learning rates still face slighter fluctuations. Finally, the learning rate of 0.0001 also converges slowly, which leads to exceeding training times with gradual improvements. One can easily infer that the learning rate of 0.01 and 0.001 provides an optimal balance while achieving effective convergence with reduced fluctuations and stable performance. In figure 6, we compare different gamma values for the convergence of SQ-DDTO. We can see that 0.9 and 0.95 perform better than 0.99 because a lower value of gamma emphasizes more on immediate rewards. Moreover, the agent learns the optimal offloading decisions in our dynamic scenario faster. This gamma value allows the agent to learn more quickly about changes, reducing the impact of future uncertainties. At the same time, 0.99 may slow the learning performance due to more focus on future rewards.

6.2. *Performance Comparison with Rule-Based Schemes*

We compare our proposed algorithm regarding average delay, energy consumption, and task loss rate with the following rule-based benchmarks.

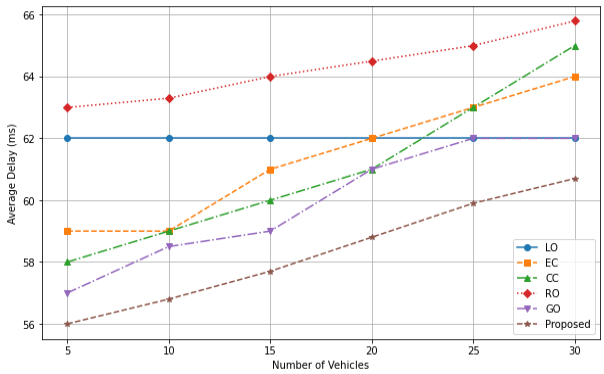

Figure 7: Performance of average delay with different number of vehicles

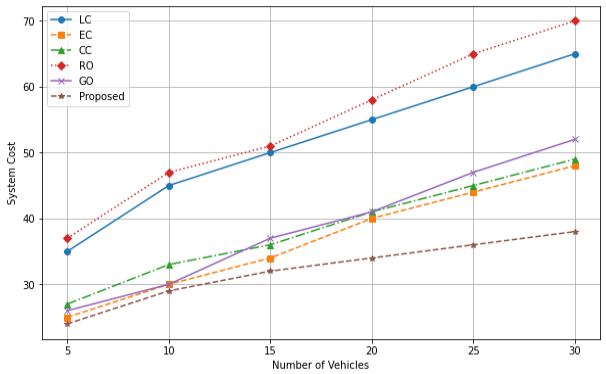

Figure 8: Performance of System cost with different number of vehicles

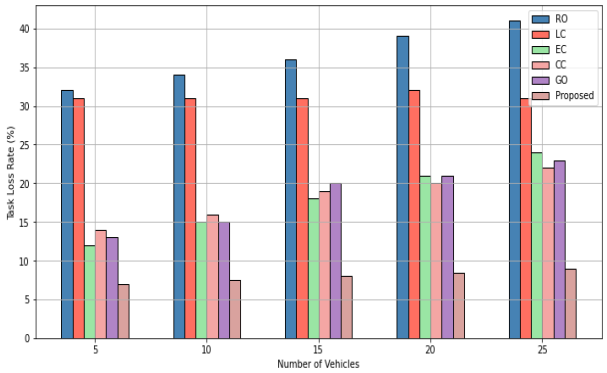

Figure 9: Performance of task loss rate with different number of vehicles

Local Computing (LC): All the tasks of vehicles are executed locally.
Edge Computing (EC): All the tasks of vehicles are offloaded to the RSU for remote execution.
Cloud Computing (CC): All the tasks of vehicles are offloaded to the cloud server for remote execution with powerful computing capabilities.
Random Offloading (RO): All the tasks are randomly distributed between local and remote computing without employing any informed decisions.
Greedy Binary Offloading (GO): In this approach, a greedy approach is used to make binary offloading decisions of executing either entirely locally or entirely on the edge server, without task partitioning. In each time frame t, if the total reward achieved through local execution exceeds the offloading mode for the tasks of each vehicle, then local execution will be selected for the next time frame $t + 1$. If not, the offloading mode is selected.

In Figure 7, we provide a comparative analysis of the proposed algorithm concerning the average delay. One can see that all algorithms face higher delays as the number of vehicles increases. This is due to an increase in the overall workload. More vehicles generate more tasks, which tends to increase delay significantly. We have validated that RO performs worse than all other schemes in experiments. This is because RO randomly selects the offloading decisions and resources without any informed and intelligent decisions based on specific parameters such as channel quality, speed, and vehicle direction. EC and CC perform better than local computing because they utilize high computing capabilities. Here, local computing provides almost static delay because local CPU cycles are used for each task without migrating to the RSUs. We can also see that the proposed algorithm performs better compared to all schemes with different numbers of vehicles and surpasses traditional schemes effectually due to its learning ability in dynamic and complex networks.

Figure 8 compares the average cost of the system in terms of energy consumption with varying numbers of vehicles. All algorithms suffer from increased system costs with a higher number of vehicles. RO performs worst due to its poor decision-making and resource allocation strategy. LC also performs worst because it consumes vehicle resources without sending resource-intensive tasks to RSUs, edge, or cloud servers. EC, CC, and GO perform better than RO and LC due to utilizing the computational resources of remote servers and relieving the burden of vehicles. As presented, the proposed algorithm excels in system cost by reducing the average energy consumption and optimal resource allocation using SQP. It can also be inferred that the proposed algorithm surpasses all rule-based methods with varying numbers of vehicles due to its optimized task offloading decisions and resource allocation strategies, irrespective of system dynamics, such as vehicle speeds, location, distance, and channel qualities. To evaluate the performance over task loss rate, we first define it as the ratio of lost tasks whose processing constraints, such as C4, C6, and C7, exceed their threshold, as presented in P1. Figure 9 compares the proposed algorithm with all rule-based methods. With an increase in the number of vehicles, the task loss rate of all algorithms increases. This is due to the extensive competition for computing resources and vehicle bandwidth allocations. This requires more processing time for all vehicle tasks, which leads to a higher task loss rate. Among all, the proposed algorithm performs better due to its optimal selection of computing modes and efficient allocation of resources for all vehicles.

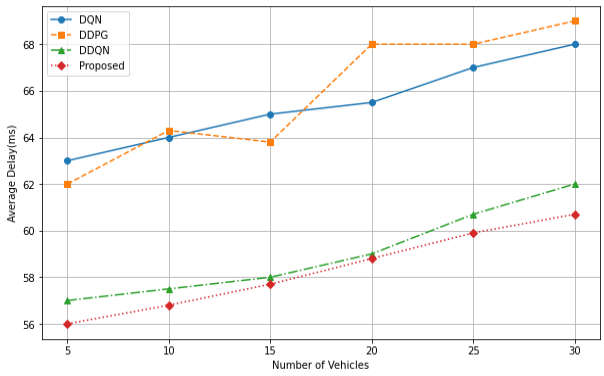

Figure 10: Performance of average delay with different number of vehicles

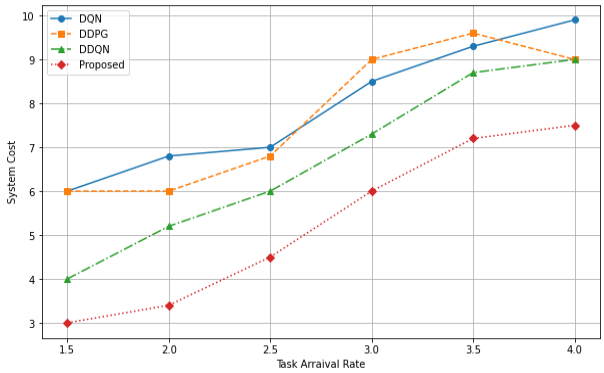

Figure 11: Performance of System cost with different number task arrival rates

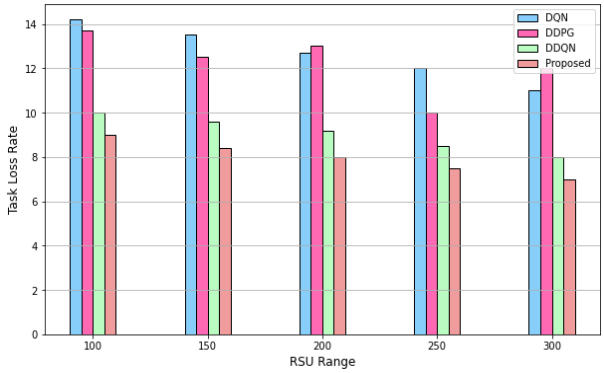

Figure 12: Performance of task loss rate with different number RSU ranges

6.3. Performance Comparison with DRL-based Schemes

We further evaluate the performance of our proposed algorithm with DRL-based schemes, including Deep Q Networks (DQN), Double Deep Q Networks (DDQN), and Deep Deterministic Policy Gradient (DDPG), which is suitable for continuous action space. In Figure 10, we compare the performance of the proposed algorithm in terms of average delay with different numbers of vehicles. When the number of vehicles increases in each zone, all the algorithms return a higher average delay. This is because when more vehicles enter the respective zones, they require more communication and communication resources, leading to a significant increase in delays. This also leads to more interference in vehicle-to-infrastructure channels and an increase in average delay. Here, we can infer that DDPG does not perform well due to the discrete action space; however, it prefers continuous action spaces. DQN also performs the worst due to overestimation bias, which is reduced by DDQN using different networks to select and evaluate actions. The proposed algorithm excels at minimizing average delay because it separates state values and advantages, which helps distinguish between valuable and less valuable states effectively. In Figure 11, we analyze the system cost in terms of energy consumption with variable task arrival rates. It is depicted that all algorithm results exceed system costs when the task arrival rate increases. However, with these variable task arrivals, the proposed algorithm performs better than DQN, DDQN, and DDPG.

We also compare the performance of the proposed algorithm in terms of task loss rate with a varying number of RSU ranges. The different range of RSUs significantly impacts the ratio of task loss rate. As presented in Figure 12, all algorithms have a high task loss ratio with smaller RSU coverage that decreases significantly with increasing RSU coverage. More substantial coverage of the RSU reduces the task loss rate because it increases the time that vehicles remain connected to the RSU, allowing more tasks to be successfully offloaded and processed at the RSUs. This extended connectivity reduces the chance that tasks are dropped because vehicles leave the coverage areas before completing their offloaded tasks. We can see that the proposed algorithm surpasses DQN, DDQN, and DDPG due to its stable learning for optimal offloading decisions and efficient resource allocation in such highly complex networks.

7. Conclusion

In this research, we provided a novel two-level SQ-DDTO algorithm for optimal offloading decisions and resource allocation. First, we decoupled both the problems of offloading decisions and resource allocation and then utilized dueling DDQN for better generalization and learning while making offloading decisions. Then, to improve the sample efficiency, we employed PER, which prioritizes each experience according to the temporal difference error instead of random selection. Following this decoupling, we use SQP to solve the problem of computing resource allocation. The experimental results show that the proposed algorithm surpasses rule-based and DRL-based schemes, including DQN, DDQN, and DDPG, in terms of average delay, energy consumption, and task loss rate. In the future, we aim to investigate the strength of DRL-based computation offloading and resource allocation algorithms in urban traffic scenarios with more realistic simulation tools such as Simulation of Urban Mobility (SUMO).

Declaration of Generative AI

The authors declare that they used Generative AI in the writing process solely to improve the readability and language of the introduction and conclusion.

References

[1] Gill, S.S., *A manifesto for modern fog and edge computing: Vision, new paradigms, opportunities, and future directions*, in Operationalizing Multi-Cloud Environments. 2022, Springer. p. 237-253.

[2] Panigrahy, S. & Emany, H. A survey and tutorial on network optimization for intelligent transport system using the internet of vehicles. *Sensors*. **23**, 555 (2023)

[3] Gong, T., Zhu, L., Yu, F. & Tang, T. Edge intelligence in intelligent transportation systems: A survey. *IEEE Transactions On Intelligent Transportation Systems*. **24**, 8919-8944 (2023)

[4] Zhang, X., Liu, J., Hu, T., Chang, Z., Zhang, Y. & Min, G. Federated learning-assisted vehicular edge computing: Architecture and research directions. *IEEE Vehicular Technology Magazine*. (2023)

[5] Georgiades, M. & Poullas, M. Emerging technologies for V2X communication and Vehicular Edge Computing in the 6G era: Challenges and Opportunities for Sustainable IoV. *2023 19th International Conference On Distributed Computing In Smart Systems And The Internet Of Things (DCOSS-IoT)*. pp. 684-693 (2023)

[6] Panigrahy, S. & Emany, H. A survey and tutorial on network optimization for intelligent transport system using the internet of vehicles. *Sensors*. **23**, 555 (2023)

[7] Marwein, P., Nath Sur, S., Gao, X. & Kandar, D. Recent Survey on Internet of Vehicles: Architecture, Applications, Challenges, and Its Solutions. *Journal Of Testing And Evaluation*. **52**, 731-753 (2024)

[8] Hasan, M., Jahan, N., Nazri, M., Islam, S., Khan, M., Alzahrani, A., Alalwan, N. & Nam, Y. Federated learning for computational offloading and resource management of vehicular edge computing in 6G-V2X network. *IEEE Transactions On Consumer Electronics*. (2024)

[9] Umar, A., Hassan, S., Jung, H., Garg, S., Hossain, M. & Guizani, M. Computation offloading in NOMA-MEC-enabled aerial-vehicular networks exploiting mmWave capabilities. *Computer Networks*. pp. 110335 (2024)

[10] Liao, Z., Xu, S., Huang, J. & Wang, J. Task Migration and Resource Allocation Scheme in IoV With Roadside Unit. *IEEE Transactions On Network And Service Management*. **20**, 4528-4541 (2023)

[11] Fan, X., Gu, W., Long, C., Gu, C. & He, S. Optimizing Task Offloading and Resource Allocation in Vehicular Edge Computing Based on Heterogeneous Cellular Networks. *IEEE Transactions On Vehicular Technology*. (2023)

[12] Shinde, S. & Tarchi, D. A markov decision process solution for energy-saving network selection and computation offloading in vehicular networks. *IEEE Transactions On Vehicular Technology*. **72**, 12031-12046 (2023)

[13] Zabihi, Z., Eftekhari Moghadam, A. & Rezvani, M. Reinforcement learning methods for computation offloading: a systematic review. *ACM Computing Surveys*. **56**, 1-41 (2023)

[14] Luo, Z. & Dai, X. Reinforcement learning-based computation offloading in edge computing: Principles, methods, challenges. *Alexandria Engineering Journal*. **108** pp. 89-107 (2024)

[15] Mustafa, E., Shuja, J., Rehman, F., Riaz, A., Maray, M., Bilal, M. & Khan, M. Deep Neural Networks meet computation offloading in mobile edge networks: Applications, taxonomy, and open issues. *Journal Of Network And Computer Applications*. pp. 103886 (2024)

[16] Li, C., Zhang, Y. & Luo, Y. DQN-enabled content caching and quantum ant colony-based computation offloading in MEC. *Applied Soft Computing*. **133** pp. 109900 (2023)

[17] Zhang, C., Peng, C., Lin, M., Du, Z. & Wu, C. Double DQN Reinforcement Learning-Based Computational Offloading and Resource Allocation for MEC. *International Conference On Mobile Networks And Management*. pp. 240-253 (2023)

[18] Nguyen, T., Nguyen, N. & Nahavandi, S. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions On Cybernetics*. **50**, 3826-3839 (2020)

[19] Du, W. & Ding, S. A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications. *Artificial Intelligence Review*. **54**, 3215-3238 (2021)

[20] J. Lu et al., âA Multi-Task Oriented Framework for Mobile Computation Offloading,â *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 187â201, Jan. 2022, doi: 10.1109/TCC.2019.2952346.

[21] H. Li, X. Li, and F. Shen, âTask offloading under deterministic demand for vehicular edge computing,â *Etri J.*, vol. 45, no. 4, pp. 627â635, Mar. 2023, doi: 10.4218/ETRIJ.2022-0115.

[22] D. Ye et al., âMastering Complex Control in MOBA Games with Deep Reinforcement Learning,â *AAAI 2020 - 34th AAAI Conf. Artif. Intell.*, vol. 34, no. 04, pp. 6672â6679, Apr. 2020, doi: 10.1609/AAAI.V34I04.6144.

[23] J. Shi, J. Du, Y. Shen, J. Wang, J. Yuan, and Z. Han, âDRL-Based V2V Computation Offloading for Blockchain-Enabled Vehicular Networks,â *IEEE Trans. Mob. Comput.*, vol. 22, no. 7, pp. 3882â3897, Jul. 2023, doi: 10.1109/TMC.2022.3153346.

[24] H. Chang, Y. Chen, B. Zhang, and D. Doermann, âMulti-UAV Mobile Edge Computing and Path Planning Platform Based on Reinforcement Learning,â *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 6, no. 3, pp. 489â498, Jun. 2022, doi: 10.1109/TETCI.2021.3083410.

[25] B. Hu, Y. Shi, and Z. Cao, âAdaptive Energy-Minimized Scheduling of Real-Time Applications in Vehicular Edge Computing,â *IEEE Trans. Ind. Informatics*, vol. 19, no. 5, pp. 6895â6906, May 2023, doi: 10.1109/TII.2022.3207754.

[26] R. Zhang, C. Zhou, and I. ICC, âA computation task offloading scheme based on mobile-cloud and edge computing for WBANs,â *IEEE Int. Conf. Commun. (ICC)*, 2022, Accessed: Oct. 01, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9838921/.

[27] R. Xu, S. Luan, Z. Gu, Q. Zhao, and G. Chen, âLRP-based policy pruning and distillation of reinforcement learning agents for embedded systems,â *IEEE Int. Symp. Real-Time Distrib. Comput.*, 2022, doi: 10.1109/ISORC52572.2022.9812837.

[28] G. Tian, Y. Ren, C. Pan, Z. Zhou, and X. Wang, âAsynchronous federated learning empowered computation offloading in collaborative vehicular networks,â *IEEE Wirel. Commun. Netw. Conf. (WCNC)*, 2022, Accessed: Oct. 01, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9771736/.

[29] R. Duvignau, B. Havers, V. Gulisano, and M. Papatriantafilou, âTime- and Computation-Efficient Data Localization at Vehicular Networksâ Edge,â *IEEE Access*, 2021, Accessed: Oct. 01, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9562541/.

[30] Sun, D., Chen, Y. & Li, H. Intelligent Vehicle Computation Offloading in Vehicular Ad Hoc Networks: A Multi-Agent LSTM Approach with Deep Reinforcement Learning. *Mathematics*. **12**, 424 (2024)

[31] Geng, L., Zhao, H., Wang, J., Kaushik, A., Yuan, S. & Feng, W. Deep-reinforcement-learning-based distributed computation offloading in vehicular edge computing networks. *IEEE Internet Of Things Journal*. **10**, 12416-12433 (2023)

[32] Chen, C., Zhang, Y., Wang, Z., Wan, S. & Pei, Q. Distributed computation offloading method based on deep reinforcement learning in ICV. *Applied Soft Computing*. **103** pp. 107108 (2021)

[33] Fan, W., Zhang, Y., Zhou, G. & Liu, Y. Deep Reinforcement Learning-Based Task Offloading for Vehicular Edge Computing With Flexible RSU-RSU Cooperation. *IEEE Transactions On Intelligent Transportation Systems*. (2024)