2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) | 978-1-6654-2263-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/WoWMoM51794.2021.00016

Context-Aware Fine-Grained Task Scheduling at Vehicular Edges: An Extreme Reinforcement Learning based Dynamic Approach

Shafkat Islam*, Shahriar Badsha†, Shamik Sengupta‡
University of Nevada, Reno, NV, USA
*shafkat@nevada.unr.edu,†sbadsha@unr.edu, ‡ssengupta@unr.edu

***Abstract*—Vehicular edge computing (VEC), being a novel computing paradigm, promises to provide divergent vehicular edge services, both functional (e.g., charging route prediction, emergency messages, etc.) and infotainment (e.g. video gaming applications, featured movie series, etc.), at the network edge while satisfying application-specific QoS requirements. Vehicles usually send these service requests to nearest roadside units (RSUs), which contain mobile edge servers, according to the functional requirements or the vehicle owner preferences. However, the VEC server's virtual resources may fall short compared to the unbounded amount of real-time service requests (infotainment/functional) during rush hours. This limitation entails VEC servers to fail to meet the stringent latency requirements which may create unwanted malfunction event during driving in the requested vehicles (if functional/critical service requests are delayed in processing). Moreover, the VEC environment's intrinsic properties, i.e. mobility, application-specific distinct latency requirements, traffic congestion, and uncertain task arrival rate, make the VEC task scheduling problem a non-trivial one. In this paper, we propose an extreme reinforcement learning (ERL) based context-aware VEC task scheduler that can make online adaptive scheduling decisions to meet the application-specific latency requirements for both types of tasks (i.e. functional and infotainment). The scheduler can make scheduling decisions directly from its experience without prior knowledge or the VEC environment model. Finally, we present extensive simulation results to confirm the efficacy of the proposed scheduler. Results show that the VEC server can achieve successful (by meeting QoS requirements) task completion rate of above 96% for different task arrival rates (ranging from 10 to 50 arrival/s) using the proposed scheduler. In the simulation, we also analyze the scheduling algorithm's scalability in response to the vertical expansion of the VEC server. Furthermore, we compare the performance of our proposed method with two baseline methods.**

***Index Terms*—Vehicular Edge Computing, Task Scheduling, Extreme Learning Machine, Reinforcement Learning, Markov Decision Process, QoS**

I. Introduction

With the advancement in the automotive industry, contemporary vehicular network technology is dependent on diversified delay-sensitive services, i.e. intelligent navigation [1], traffic safety [2], traffic management [3], collaborative data sharing [4], ride-hailing service [5], augmented vehicular reality [6], etc. These resource-intensive and real-time applications usually require a large amount of data and extensive computing power for appropriate functioning. In contrast, the resource-limited vehicular nodes fail to ensure the stringent delay requirements of these applications [7]. This limitation becomes a bottleneck in the growth of intelligent vehicular technology and hinders the progress towards autonomy level 5.0 [8].

To adapt to the explosive computation demands of vehicular applications, mobile cloud computing (MCC) [9] is envisioned as a potential solution. In MCC, vehicles can offload their computation-intensive workloads to traditional resource-intensive cloud environments via standard wireless technology [10]. Thus, the resource-constrained vehicular nodes can extend their computation capability and improve performance.

Though cloud computing can provide service to the extensive computation demands, it can not guarantee the fulfillment of stringent delay-sensitive requirements of vehicular applications since the traditional cloud infrastructures are usually located at remote locations that may cause unforeseeable delay and degrade the quality of service (QoS). To minimize service losses, mobile edge computing (MEC) [11] has become a promising paradigm that can bring additional computing power at the network edges. Hence, MEC can provide substantial computing services to the vehicles at their closest vicinity to minimize the transmission cost and reduce service latency, thus eliminating the requirements for massive onboard computing power for contemporary vehicular applications [12].

In recent years, Vehicular edge computing (VEC), a novel computing paradigm, is considered as a salient use case for mobile edge computing (MEC) to extend the cloud-like computing capability at the vehicular vicinity [13]. In vehicular edge computing architecture, MEC servers, located at the network edge (usually in base stations), provide service to the vehicles through RSUs. Vehicles usually offload computation-intensive tasks to edge servers for timely processing through the RSUs, and edge servers also return the response via RSUs. These types of computation-providing services have the potential to unleash unprecedented opportunities for vehicular nodes in adopting resource-intensive intelligent applications. Fig.1 illustrates the vehicular edge computing (VEC) architecture where $\mathbf{D}$ is the diameter of the coverage area for each server.

Most of the existing works [14]–[17] on VEC have focused on designing an optimal scheme to minimise service latency and enhance task offloading efficiency. However, the works do not consider the edge server's service load, thus may cause unwanted task congestion on individual edge servers. Since VEC servers have limited computation capability than the cloud, vehicular applications have distinct latency re-

978-1-6654-2263-5/21/$31.00 ©2021 IEEE
DOI 10.1109/WoWMoM51794.2021.00016

quirements and vehicle mobility; task scheduling in the VEC server is a non-trivial task. Moreover, the dynamic vehicular mobility pattern has also made the scheduling problem more complicated. Furthermore, few works [13] have focused on the VEC task scheduling problem though it does not consider dynamic vehicle mobility and task arrival rates.

Scheduling continuous task requests of safety-critical and delay-sensitive applications in a dynamic and unknown VEC environment is an unconventional scheduling problem. Scheduling decisions are made once a new task arrives. However, the VEC scheduler has information regarding its resource status and the new task request; it does not know future task arrival rates and resource demands. This work aims to develop an adaptive online scheduler that can learn the intrinsic pattern of VEC environment and thus make wise scheduling decisions by itself.

To overcome the above task scheduling problem and achieve the goal, we propose a task context and QoS-aware dynamic scheduler for prioritizing critical tasks (those have significantly low latency requirement) over non-critical ones (those have relaxed or higher latency requirement). Our goal is to develop an intelligent scheduler that can learn the intrinsic contextual pattern of tasks and make wise decisions to increase the overall task completion rate in the VEC environment. We state the main contributions of this work below.

- We formulate the task scheduling problem as a Markov decision process (MDP) based sequential decision making problem. Moreover, we distinguish the critical tasks from non-critical ones and prioritize the critical ones during scheduling.
- We use reinforcement learning (RL) for developing an optimal scheduling policy. To avoid the curse of dimensionality in RL, we use extreme learning machine (ELM) for function approximation. This work is the first attempt of using extreme reinforcement learning (ERL) in task scheduling problems to the best of our knowledge.
- We have conducted extensive simulations for confirming the efficacy of our proposed task. Simulation results show that our proposed scheduler can significantly improve the VEC server's efficiency in successfully executing requested tasks. Moreover, we analyse the performance of the proposed method in response to vertical expansion of VEC server and also compare the method with two different baseline methods (i.e., random scheduling, and round robin scheduling).

We organize the remainder of the paper according to the following order. Section II discusses the related works in task scheduling and vehicular edge computing, along with the research gaps. We describe the task scheduling problem, with distinct properties, for vehicular edge computing in section III. In section IV, we describe the system architecture of the ERL-based task scheduler for VEC server. We present the simulation results and the discussions on those findings in section V. Finally, we conclude the paper with future research directions in section VI.

II. Related Works

This section describes the related works in task scheduling and vehicular edge computing (VEC). We also discuss the research gap in existing works in building an efficient task scheduling framework for vehicular edge server.

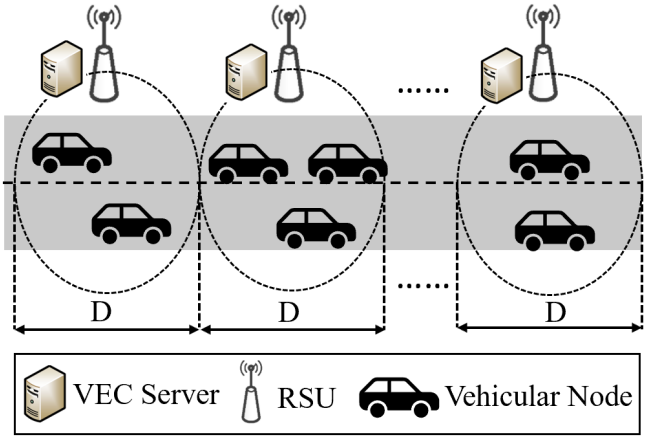

Fig. 1. Vehicular Edge Computing Architecture

A. Task Scheduling

Task scheduling, an indispensable but challenging part of distributed systems, has gained lots of attention from researchers for decades. However, most of the works aim to solve the scheduling problem for comparatively stable environments using theoretical analysis, i.e. fuzzy theory [18], nash equilibrium [19], and heuristic techniques, i.e. genetic algorithm [20], ant colony optimization [21], particle swarm optimization [22]. Moreover, these approaches are not suitable for dynamic and online task scheduling problems. Instead, the heuristic methods are more appropriate for solving offline task scheduling problems. Since the vehicular edge computing requires to deal with a dynamic vehicular environment with enormous distinctive task requests, it requires an online task scheduler that can adapt to this dynamic environment and makes scheduling decisions in an online manner. Hence, these techniques do not apply to our vehicular scenario.

In recent years, reinforcement learning (RL) [23] has been applied to solve complex decision making problems, i.e. resource allocation [24], network slicing [25], resource auto-scaling [26], etc. The authors of [27] propose an RL based task scheduling approach though they simplify the state space to overcome the curse of dimensionality problem in RL. In [28], the authors propose a deep reinforcement learning (DRL) based task scheduling scheme for clouds. However, the authors build the scheduling model while considering QoS requirements, it does not consider the mobility of the service users. Moreover, the scheduling scheme is built for cloud that has unbounded amount of computing capability whereas our vehicular edge network has limited computing capacity. Furthermore, the authors have used computation intensive deep neural networks [29], [30] for function approximation in DRL based task scheduling that will create enormous computation overhead in our resource constrained vehicular edge servers.

B. Vehicular Edge Computing

Mobile edge computing (MEC) provides additional computing capability to the resource constrained devices at the

network edge. In recent years, multiple research works have focused on designing optimized task offloading [31], scheduling [32] algorithms for MEC. Since vehicles utilize diverse resource intensive applications for autonomous navigation, vehicular network requires additional computing capability at its network edge as vehicles are considered as resource-constrained devices. To fulfil the gap of computing resources, researchers propose to integrate the vehicular network with MEC that creates a distinct computing paradigm of vehicular edge computing (VEC) [33].

In literature, there exist multiple works for task offloading at vehicular edge computing servers (alternatively, VEC). In [34], the authors propose a scheme for offloading vehicular applications at the VEC. Authors formulate the offloading problem as a mixed-integer non-linear problem and consider a constant vehicle speed. The authors of [16] propose a joint optimization technique for offloading and load balancing in VEC. They have also considered the optimization problem as a mixed-integer non-linear one. The authors of [35], and [36] propose a deep reinforcement learning-based dynamic service offloading scheme for VEC. The authors of [37] propose a software-defined network-based task offloading scheme with load balancing using fiber-wireless technology in VEC.

Though there exists multiple works on tasks offloading for the VEC, very few works have dealt with the task scheduling problem at the VEC servers. Since task offloading and scheduling have distinct properties, it requires distinctive solutions as well. The authors of [13] propose a dependency aware task scheduling scheme for VEC. However, the authors have considered a constant vehicle arrival rate and constant vehicle speed that simplifies the complex vehicular network into a simpler one and thus does not guarantee to reflect the dynamic nature of the vehicular network.

Therefore, we envision developing an online adaptive task scheduler for the vehicular edge computing server that can make scheduling decisions while considering the dynamic nature (i.e. mobility) of the vehicular network and the distinct QoS requirements for each service.

III. Problem Formulation

In this section we describe the vehicular task scheduling problem along with the underlying assumptions in brief.

A. Task Workloads

In VEC, edge servers host numerous vehicular applications that require extensive computing power or a large amount of storage capacity. Each vehicle submits task execution request to the closest edge server (as shown in fig.1), via RSUs, whereas the edge server schedules the tasks according to a scheduling policy. Each vehicle can declare the quality of service (QoS) requirement with the task request depending on the task type and the vehicle operational condition. Usually, vehicles and edge servers communicate with each other through RSUs. We assume each task is independent of others, and a single server hosts each job (or task) to eliminate inter-server communication overhead during task execution. Moreover, edge servers do not have any knowledge regarding task arrival rate in advance. We define each task request as a tuple of seven different entities.

$$Task_i^j = \{ID_i, location_j, speed_j, direction_j, type_i, QoS_i, timestamp\} \quad (1)$$

where $ID_i, type_i,$, QoS_i, $timestamp$ denote unique identification number, task type, QoS requirement, and generation time of the requested task (i); and $location_j, speed_j$ and $direction_j$ denote the current location, speed and destination direction of the requested vehicle (j).

B. VEC Server Resources

In this work, we assume that each edge server, located in different geographical locations, is identical in their resource capacity and type. Each edge server can offer multiple virtual machine (VM) instances for hosting vehicular tasks depending on the task type and requirements. For example, delay-sensitive applications, i.e., safety-critical applications, require high-CPU VMs for hosting, in contrast to it, sensor data storage applications require high storage VMs for hosting. We assume that each edge server has the absolute privilege of deciding the VM allocation for each task in a decentralized manner without any centralized governance. The resource state of m^{th} edge server (ES_m) can be described as following,

$$ES_m = \{VM_1,\ VM_2,\ VM_3\ , ...,\ VM_n\} \quad (2)$$

where $VM_1, VM_2, ..., VM_n$ denotes the processing queue of each VM at time (t). Moreover, we also assume that each server has an identical n number of VMs.

C. Task Execution

Vehicles offload task requests to the nearest edge servers and after receiving each request servers can take one of three following decisions (d_m^i).

$$d_m^i = \begin{cases} 0,\ reject \\ 1,\ accept \\ 2,\ forward \end{cases} \quad (3)$$

where $d_m^i = 0$ denotes that the i^{th} request is turned down by the m^{th} server due to the mismatch in available resource and QoS requirements, $d_m^i = 1$ denotes the request is scheduled in m^{th} server and $d_m^i = 2$ denotes the request is forwarded to the other nearest server. We assume that edge servers with heavy task loads can forward some of its traffic to other nearest edge servers to balance uneven load distribution. But it will eventually create overhead in task processing time. Moreover, we also assume that the edge servers follow standard protocols for engaging in the coalition.

We assume that each RSU location (or edge server clusters) possesses an independent task scheduler which can schedule task depending on its state and the task requirement. We also assume that every request is dispatched immediately after arrival in the edge server. As soon as a task is scheduled, it enters the waiting queue for that resource (or VM). In this case, we assume that the waiting queue is large enough for holding unexecuted tasks for a limited time. However, if the waiting time is too high, the QoS requirement for critical tasks will not be satisfied. In the queue, the tasks are served in a first-come,

first-serve basis. The execution process is interruption-free, which means a task can not be interrupted during execution by other tasks.

The task execution cost is defined as the total time required for processing a task in the edge server, and that we can define as following,

$$t_{total} = t_{exe} + t_{waiting} + t_{overhead} \tag{4}$$

where t_{exe} is the task execution time, $t_{waiting}$ is the waiting time in the queue, and $t_{overhead}$ is the inter-server communication overhead due to coalition (in the load balancing purpose). It is expected that the t_{total} is smaller than the latency QoS requirement (t_{max}). In this work, we assume that latency is the only QoS metric, and tasks are characterized into two distinct categories, i.e., critical and non-critical. The critical tasks usually have a stringent or small value for latency requirement whereas, the non-criticals have higher or relaxed values. This paper identifies critical tasks with latency requirements below a threshold value (t_{th}). The inter-server communication overhead $(t_{overhead})$ term is zero for tasks executed in the requested edge server and non-zero for tasks forwarded to other adjacent edge servers for processing.

D. Task Scheduling Problem

Each task execution request is regarded as successful if the task associated QoS requirement is fulfilled, otherwise the response is regarded as an unsuccessful attempt.

$$t_{total} = \begin{cases} \leq t_{max}, \ successful \\ otherwise, \ unsuccessful \end{cases} \tag{5}$$

The edge server only receives an incentive for executing successful turns; otherwise, it gets penalized. Therefore, each edge server's goal is to maximize the successful task completion rate (which depends on handling critical and non-critical task requests effectively) and receive maximum rewards with its limited capacity. A task scheduling decision is regarded as successful if the allocated resource can satisfy the QoS requirement for that task. Thus, increasing the successful task execution rate is a non-trivial problem due to the VEC environment's complex dynamics, i.e., mobility, limited edge resource, application-specific QoS, and uncertain arrival rates.

This work aims to design an adaptive online task scheduler that can assist edge servers in handling both the critical and non-critical tasks effectively (by learning the intrinsic features of the VEC environment) and thus improve the overall efficiency of the edge resource management.

IV. System Architecture

The vehicular edge computing system's efficacy depends on efficient handling of critical and non-critical tasks that can enhance successful completion rate. An intelligent online task scheduler-based edge computing system is presented in this section to create an appropriate policy for scheduling continuous task requests.

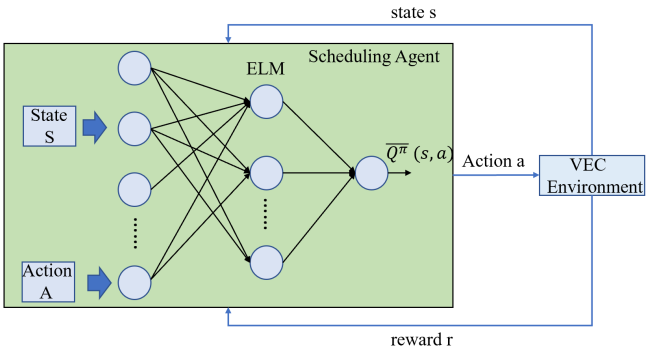

Fig. 2. EQL based VEC Task Scheduler

A. ELM-RL Architecture

Reinforcement learning (RL) [23], being a distinct type of machine learning algorithm, is widely used in the sequential decision (Markov Decision Process [38] based problems) making problems, i.e. in [39]. The distinguishable reinforcement learning features are feedback (or trial and error) based optimum decision search and delayed rewards. These features attract researchers in adopting reinforcement learning in widespread sectors, i.e. wireless communication [25], path planning [40], cybersecurity [41], etc. The RL model comprises of a learning agent, set of actions (A), set of states (S), an interacting environment, and a reward function (R) which we describe as $S \times A \xrightarrow{S'} R$ where S' denotes the new state of the agent after taking an action (A). In this model, the agent learns the optimal decision policy (π) in a trial and error manner by interacting with the environment. The goal for the agent is to learn a policy (π) which maximizes the value function $(V^{(\pi)})$ which we describe as following [42].

$$V^{(\pi)}(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s\right] \tag{6}$$

where γ denotes discount factor for future rewards, r_t denotes immediate reward, and s denotes state at time t. $V^{\pi}(s)$ is termed as state-value function for the policy π [23].

In order to determine appropriate actions in each state, action-value function $(Q^{\pi}(s,a))$ is used in [23] which is described as follows. The action-value $(Q^{\pi}(s,a))$ is used for evaluating the value of an action in each state, whereas, state-value function is used for assessing a policy.

$$Q^{\pi}(s,a) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s, a\right] \tag{7}$$

Hence, we can define the optimal action-value function as,

$$Q^*(s,a) = {}^{max}_{\pi}\, Q^{\pi}(s,a) \tag{8}$$

and the optimal policy as following.

$$\pi^*(s) = {}^{arg\ max}_{\pi}\, Q^*(s,a) \tag{9}$$

In naive RL based algorithms, the value of action-value $(Q^{\pi}(s,a))$ for each state-action pairs is stored in a tabular form. This approach of storing each state-action value limits the utility of reinforcement learning to only problems which

have discrete or limited state and action space. However, many real-world problems have continuous or extensive state and action space, requiring a method of function approximation instead of storing each value. Our task scheduling problem is also a sequential decision making problem with continuous state space. Hence, we need a hybrid solution to deal with such a curse of dimensionality issue.

Most of the existing works focus on deep neural network-based value function approximation method, which poses an enormous computational burden on the RL agent. Moreover, it requires lots of time and training samples for determining weights and bias term of the neural network. To avoid this issue, we use extreme learning machine (ELM) [43], [44] based value function approximator for the RL agent [45]. ELM is a single hidden layer feed-forward neural network in which the hidden neurons' weights are randomly assigned instead of learning. The advantage it provides over deep neural networks is that it does not require any backpropagation based optimization technique for learning weights.

B. Context-Aware Task Scheduling

Vehicular edge computing (VEC) facilitates the vehicular network (VANET) with additional computing power at the network edge. The workload amount in VEC depends on traffic mobility as well as vehicle owner preferences. Hence, the VEC server needs to utilize an online adaptive task scheduler to adapt to the changing workload pattern and update the scheduling policy over time. To adapt with this dynamic workload environment, reinforcement learning (RL) based models are well suited as RL uses an iterative policy improvement technique to learn a model-free environment without any supervision. We do not need to train the model with the workload scheduling data beforehand; on the contrary, the task scheduling agent will generate a large volume of data as it keeps making decisions, i.e., the task to VM (accept), forward, or reject. Thus, the agent will learn an optimal task scheduling policy online as the VEC service runs. Moreover, we have used extreme learning machine (ELM) for approximating the action-value function to ensure fast scheduling policy convergence.

Extreme Q-learning (EQL) is an upgraded version of traditional Q-learning [23] that can tackle an unbounded number of continuous state and action space. In VEC, the scheduling decision is a sequential decision-making problem that contains a continuous state space and a moderately large amount of action space (depending on the number of VMs in VEC server). ELM provides an edge (in terms of computation) over other function approximation methods (i.e. deep learning) as it does not require any time-consuming optimization techniques (gradient descend) for parameter updates. In this regard, we assume that the task scheduler makes a single scheduling decision for each requested task. To simplify the action space, we assume that an action can be any of three alternatives, i.e. accept (select a VM), reject, or forward. Each VM follows fast come fast serve (FCFS) policy in the respective waiting queue of that VM. We describe the detailed task scheduling model in the following.

1) State Space: In this model the scheduling agent (VEC scheduler) considers vehicular contextual information (i.e. vehicle location, speed, direction) in together with task information (i.e. type, QoS, timestamp) for making a decision. The information of vehicle status assists in taking a granular scheduling decision over static ones that considers only the task information. The state space (S_i) for the scheduler of m^{th} server (after arriving a task at time t) is the union of the state of i^{th} task, generated by the j^{th} vehicle, (T_i^s) and server status (V_m^s), i.e. $S_i = T_i^s \cup V_m^s$ where $T_i^s = \{location_j,\ speed_j\ direction_j\ type_i,\ QoS_i,\ timestamp\}$ and $V_m^s = \{VM_1^i, VM_2^i, ..., VM_n^i\}$.

2) Action Space: We assume that the scheduler makes a decision following an event-driven framework, thus, taking a scheduling decision once a new job arrives. The decision can be one of three alternatives, as mentioned in equation (3). We assume that the scheduler accepts a task when it can allocate a virtual machine for that task. Hence, we define the action space (A) as follows, $A = \{VM_1,\ VM_2,\ ....,\ VM_n,\ R,\ F\}$ where R denotes rejection and F denotes forwarding a task to other adjacent edge server. We assume that the engaging servers follow standard trading protocols to forward a task to other edge servers for load balancing purpose. We also assume that any server outsources its computing resources to its nearest servers if the adjacent server has sufficient idle resources for executing the task immediately. A scheduler rejects a task execution request if the scheduler can predict that the QoS requirement for that task can not be satisfied with the currently available resources.

3) Reward Function: In RL, reward motivates the agent to make wise decisions to fulfil the objective of the task. In this paper, we assume that tasks have different QoS requirements in which the objective is to schedule the tasks in such a manner that can successfully fulfil the distinct QoS requirements. To motivate the scheduling agent with this objective, we define the reward function as following,

$$\beta_{\mathbf{1}},\quad t_{total} \leq \eta \ \&\&\ L(t + t_{total}) \subset A(m) \tag{10}$$

$$\beta_{\mathbf{2}},\ \ t_{total} > \eta \ ||\ L(t + t_{total}) \not\subset A(m) \tag{11}$$

where β_1 is a positive reward, and β_2 is a negative reward, η is the latency requirements for fulfilling the QoS, $L(t+t_{total})$ denotes the location of the vehicle after executing the task and $A(m)$ represents the coverage area of the edge server (m^{th}). The second part of the reward function motivates the scheduler to consider the vehicle status (or contextual task information) in scheduling decisions.

4) State Transition: The ERL agent (scheduler) makes a scheduling decision (or takes an action) based on the present state status, and the estimated Q-values from the ELM network. In this regard we adopt the ϵ-greedy algorithm for choosing an action as following,

$$\pi(s) = \begin{cases} random\ action\ from\ A,\ with\ probability\ \epsilon \\ \underset{\pi}{arg\,max}\ Q_{ELM}(s,a),\ otherwise \end{cases} \tag{12}$$

where A denotes the action space, $Q_{ELM}(s,a)$ denotes the derived action values from the ELM network, and ϵ is design parameter whose value is in the range of 0 to 1, i.e. $0 < \epsilon \leq 1$. ϵ can be a variable in which the value can gradually decrease over time and set at a minimum

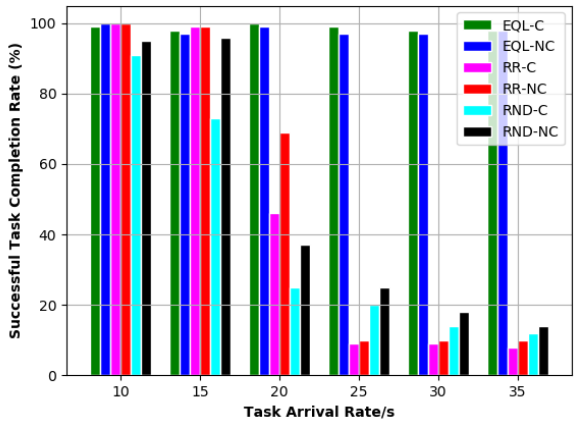

Fig. 3. Simulation results of three different task scheduling scheme (EQL, RR, RND) under different arrival rates. Probability of critical and non-critical task arrival rate is identical in this case.

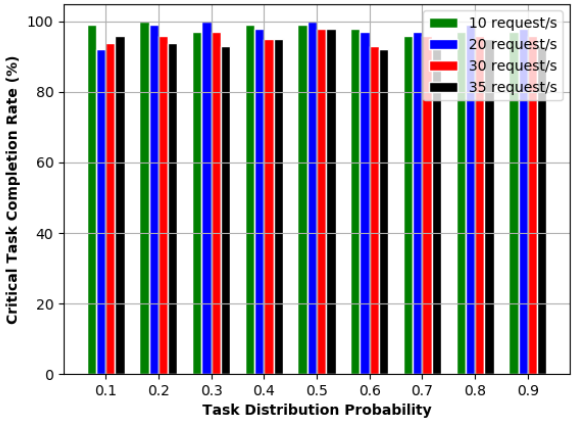

Fig. 4. Critical Task Completion rate (in percentage) in different task distribution probability using EQL Scheduling.

value. In this way, the scheduling agent explores random scheduling decisions at the beginning of the process, gather a large amount of scheduling experience. After a certain period, it exploits the experience to choose the action that returns maximum reward value. The scheduling agent transits to next state (S_{i+1}) (after the arrival of $(i+1)^{th}$ task from $(j+1)^{th}$ vehivle) from the current state (S_i) after executing an action (according to equation 12) over the arrival of i^{th} task from j^{th} vehicle where we can describe (S_{i+1}) as $S_{i+1} = \{location_{j+1},\ speed_{j+1},\ direction_{j+1},\ type_{i+1},\ QoS_{i+1},\ timestamp,\ VM_1^{i+1}, VM_2^{i+1}, ..., VM_n^{i+1}\}$.

5) Sample Experience Collection [45]: To learn the optimal scheduling policy, the ELM network needs a sample scheduling decision set (D_s) which we can describe as $D_s = \{S_i, r_i, a_i, S_{i+1}\}$ where S_i is the current state of the server, a_i is the action executed in the current state, r_i is the immediate reward for taking that action, and S_{i+1} is the next state the agent reaches after executing the current action. This sample set can be collected by executing random actions at the beginning or by a stabilizing controller that collects it online experience, whereas, in this work we assume that the server uses random action method for collecting this sample set.

In ELM, the approximator uses random hidden neurons for learning the state-action feature vector from the sample set (D_s). In every iteration the output weights $(\bar{w})$ of the ELM is updated to maximize the value of the approximate action-value function $(\bar{Q^{\pi}}(s,a))$ for all the actions $a \in A$, thus improving the scheduling policy. Fig.2 depicts the EQL-based VEC task scheduling architecture.

V. Simulation Results & Discussions

In this section, we evaluate the performance of EQL based task scheduling agent through a series of distinct experiments. We choose the successful task completion rate as a performance metric in different scenarios. In this regard, we define a task scheduling attempt is successful if the assigned resources (by the task scheduling agent) can successfully meet the QoS requirements and return the result to the respective vehicle without any connectivity loss. We consider the vehicle mobility to evaluate whether the vehicle is within the edge server range or not after completing the task. The mobility of vehicles makes the task scheduling approach distinctive in edge computing domain from other methods that do not consider high consumer mobility. We conduct the experiments in Intel core $i7$-8^{th} generation processor with 16 GB of RAM having a processor speed of 1.8 GHz and python 3 environment. In the following sections, we present the detailed performance evaluation approach along with numerical results.

A. Simulation Setup

In the experiments, we assume that an edge-based vehicular application provider rents n VMs (have identical processing capacity G) from infrastructure as a service (Iaas) vendor to provide vehicles with an enhanced computing experience when requires. We assume that vehicles request the edge-service provider to host different computing tasks that have distinctive QoS requirements. For the sake of simplicity, we assume that vehicles only have two different tasks, i.e. critical and non-critical, in which the critical tasks have stringent QoS requirements, and the non-critical have a relaxed one. We also assume that the latency requirement for the critical task is lower than the non-critical ones. We evaluate the performance of EQL-based scheduling agent for different task arrival rates, task distribution probabilities, and analyze the effect of n (number of VMs) and G (VM processing power) in task completion rates.

We assume that initially the EQL-agent takes arbitrary decisions (i.e., $\epsilon = 1$) regarding task scheduling and collects a sufficient amount of experience samples. Gradually, the value of ϵ (exploration-exploitation trade-off) reduces, and the agent begins to exploit the explored experiences. We also set the discount factor (γ) at 0.95 and learning rate (α) within $[0.1\ \ 0.01]$. The higher discount factor ensures that the agent gives appropriate consideration on future actions for taking the current action. The lower learning rate ensures that the agent does not confine itself to sub-optimal policy. We also use rectified linear unit (relu) as an activation function for the

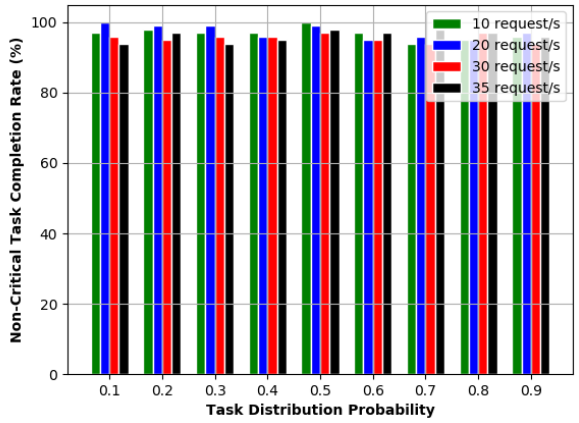

Fig. 5. Non-Critical Task Completion rate (in percentage) in different task distribution probability using EQL scheduling.

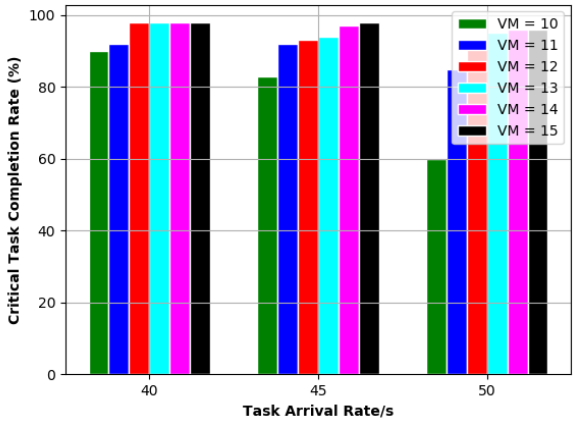

Fig. 6. Effect of number of VMs (n) in critical task completion rate using EQL scheduling.

random hidden neurons in the ELM-Q network.

We assume that the edge server provides service to vehicles in a unidirectional road of 100 km. The road has two lanes, i.e. one for incoming traffic and other for outgoing, and the average speed for each vehicle varies from 30-55 km/h. Hence, we consider the vehicle mobility to ensure faster processing of task requests from fast-moving vehicles since we assume that they have stringent QoS and low latency requirement than other slow-moving vehicles. This assumption makes the task scheduling approach more realistic in vehicular applications and distinctive from other methods.

To conform to the efficacy of our EQL-based scheduling agent we compare the results with the following two baseline solutions.

1) Random Scheduling: In a random scheduling method, the scheduler selects a random virtual machine (VM) to host a task. This method is considered as the most straightforward scheduling algorithm.

2) Round Robin Scheduling: In a round-robin based scheduling algorithm, each virtual machine (VM) is circularly assigned a task. In this technique, the load distribution for all the VMs remains relatively identical.

B. Performance Analysis Under Identical Task Generation Rate

In this section, we consider that the generation rate for critical and non-critical tasks is identical. We also assume that the latency requirement for critical tasks is half of the non-critical tasks. The server may forward the non-critical tasks to the nearest servers to balance load distributions, whereas, the critical tasks can not meet latency requirements if forwarded to other servers. However, there is no guarantee of meeting latency requirements if the non-critical tasks are forwarded since we assume that the load balancing causes addition latency causes overhead, i.e., inter-server communication overhead $(t_{overhead})$ for a task. We choose this overhead from a Gaussian distribution with mean (μ) and variance (σ). In this case, we assume that μ is one-tenth of QoS latency requirement, and σ is one-quarter of μ. In this part, we assume that the edge server has ten identical virtual machines $(n = 10)$ for providing service to the vehicular applications. We compare the EQL-based scheduling technique with two other baseline solutions, i.e., random scheduling and round-robin scheduling.

Fig. 3 presents the comparison of EQL based scheduling with round-robin (RR) and random scheduling (RND) in different task arrival rates. The figure illustrates that both the EQL and round-robin-based techniques work relatively identical at lower task arrival rates $(\leq 15/s)$, and the successful task (both critical (C) and non-critical (NC)) completion rate is approximately over 96% for both the cases. However, with the increase in arrival rate, the round-robin technique's performance drastically falls whereas, the EQL method still satisfactorily works till 35 arrivals/s while keeping the successful completion rate over 96%.

On the contrary, the random scheduling method can only perform better (over 95%) for non-critical tasks scheduling though at lower arrival rates $(\leq 15/s)$, however, the performance for critical tasks is poor for lower arrival rates as well. This result shows the resiliency of EQL-based dynamic scheduling policy over static policy (based on round-robin and random) at higher task arrival rates.

C. Performance of EQL-based Task Scheduling Under Non-identical Task Generation Rate

In this section, we analyze the performance of EQL-based scheduler under different task generation probability for critical (P_c) and non-critical (P_n) tasks where we assume that $P_c + P_n = 1$. We also assume that the number of VMs in the edge server is ten $(n = 10)$. Fig. 4 illustrate the critical completion rate for different task distribution probability, i.e. $P_c = 0.1, 0.2, ..., 0.9$. The results show that the performance of EQL-based scheduler is above 93% for different task distribution probability and task arrival rates.

Fig. 5 illustrates the performance of EQL-scheduler for non-critical tasks under different task generation probability. The successful non-critical task completion performance for the scheduler is above 94% in this case. These illustrations confirm

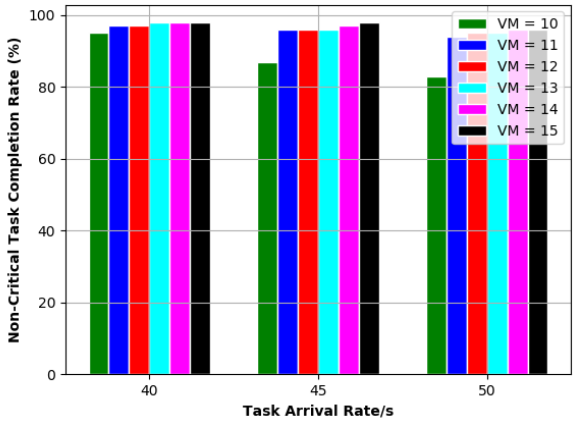

Fig. 7. Effect of number of VMs (n) in non-critical task completion rate using EQL scheduling.

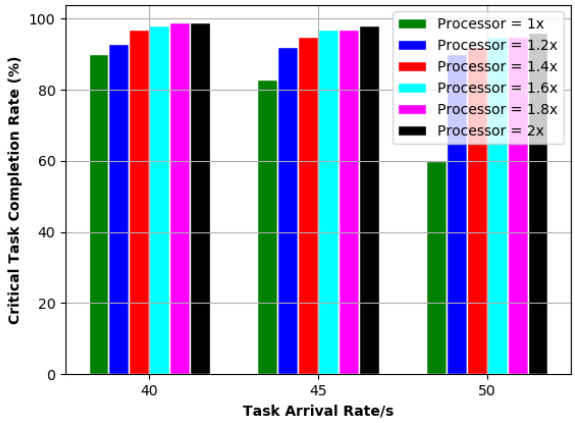

Fig. 8. Effect of Processing Power (G) in critical task completion rate using EQL scheduling.

our proposed EQL-based scheduler's efficacy and resiliency in different vehicular traffic scenarios for scheduling both critical and non-critical tasks.

D. Effect of Server Property in EQL-based Task Scheduling

In this section, we analyze the effect of server property on the EQL-based task scheduling performance. In the previous two sections (V-B and V-C), we observe that the successful completion rate for EQL-based scheduling starts reducing significantly over the arrival rate of $35/s$. Hence, we choose n (number of VMs) and G (processing power of each VM) for scaling up the server resources vertically. We analyze the task scheduler's performance in response to a vertical expansion to understand the scaling pattern of the proposed method.

1) Effect of $\mathbf{n}$ *in EQL-based Task Scheduling Performance:* Fig. 6 illustrates the impact of number of virtual machines (n) on critical task completion rate where we change the value of n from 10 to 15 linearly. We observe that with the increase in arrival rate from 40 to 50 arrival/s, the completion rate decreases from 90% to 60%. Since the critical tasks required to execute in the requested edge server for meeting the latency requirements, the impact of edge server VM shortage is significant on the critical task completion rate. As we increase the n (VM quantity) in the edge server, we observe that the successful completion rate for critical task also increases to 98% for 40 and 45 arrivals/s, and 96% for 50 arrival/s. We can see that the completion rate for 40 arrival/s reaches 98% when the number of VM is 12, and for higher VMs it does not increase. Hence, we can conclude that the computation resources with 12 VMs are sufficient for processing requests at this rate. However, for other arrival rates over 40 arrival/s, it is still in a gradual improvement phase.

Fig. 7 illustrates the impact of increasing VM quantity over non-critical task completion rate. We observe that VM number increment has a positive impact on the non-critical task completion rate as well. However, the improvement amount is less than the critical completion rate since the non-critical tasks do not solely depend on the edge server for execution. Instead, it has an alternative option for being forwarded to other edge servers for execution (though it does not guarantee a hundred per cent success rate).

2) Effect of $\mathbf{G}$ *in EQL-based Task Scheduling Performance:* In this part, we analyze the impact of increasing processing power of each VM on the EQL-based task scheduler and we also upkeep our assumption that the processing power of each VM is identical as well. Fig. 8 illustrates the effect of VM processing power over critical task completion rate. In this regard, we increment the processing power from $1x$ to $5x$ linearly where x denotes the base processing power. We can observe from the fig. 8 that the increment in processing power has positive impact on critical task processing rate where the large improvement occurs from 60% to 96% at task arrival rate of 50 tasks/s.

Fig. 9 illustrates the effect of increasing G (VM processing power) over non-critical task completion rate. This type of vertical expansion also has a beneficial impact on non-critical task completion rate, as shown in fig. 9. However, the impact margin is less compared to the critical task completion rate.

Hence, from the above experiments, we can conclude that the proposed EQL-based task scheduler is scalable with the edge server's vertical expansion. However, vertical expansion has a direct and enormous impact on the critical task completion rates. It also holds a significant impact on the non-critical task completion rate, though both the effects positively impact the overall task completion rate.

VI. Conclusion

Task scheduling in VEC server is challenging due to the diversified application-specific QoS requirements, vehicle mobility, limited VEC resource, and traffic congestion. To adapt to this complex and dynamic VEC environment, we propose an EQL based online adaptive task scheduler that can learn the complex VEC architecture and the uncertainties and arrival rate fluctuations. Simulation results show that the proposed technique can achieve 96% accuracy in successful task completion rate. Moreover, the EQL-based method outperforms the other two baseline methods (i.e. random scheduling and round-robin scheduling) in higher task arrival rates (over 15

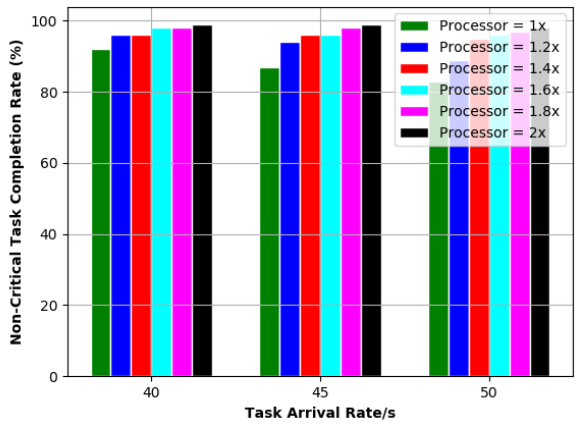

Fig. 9. Effect of Processing Power (G) in non-critical task completion rate using EQL scheduling.

arrival/s). We also observe that the proposed method is scalable in response to the VEC server's vertical expansion. In future works, we will analyze the privacy vulnerabilities of the proposed architecture. Firstly, we will use differentially private techniques to make the context-aware scheduling technique privacy-preserving. Secondly, we will adopt homomorphic computation techniques to ensure sensitive vehicular data privacy in server-side computations. Finally, we will compare our method with other learning-based scheduling approaches.

REFERENCES

[1] Y. Uehara, "Intelligent navigation system," Oct. 7 2014, uS Patent 8,855,847.

[2] Y. Zhang, Y. Zhang, and R. Su, "Pedestrian-safety-aware traffic light control strategy for urban traffic congestion alleviation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 1, pp. 178–193, 2021.

[3] P. Kasture and H. Nishimura, "Platoon definitions and analysis of correlation between number of platoons and jamming in traffic system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 1, pp. 319–328, 2021.

[4] S. Islam, S. Badsha, and S. Sengupta, "A light-weight blockchain architecture for v2v knowledge sharing at vehicular edges," in *2020 IEEE International Smart Cities Conference (ISC2)*. IEEE, 2020, pp. 1–8.

[5] S. Kudva, R. Norderhaug, S. Badsha, S. Sengupta, and A. Kayes, "Pebers: Practical ethereum blockchain based efficient ride hailing service," in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. IEEE, 2020, pp. 422–428.

[6] B. Dai, F. Xu, Y. Cao, and Y. Xu, "Hybrid sensing data fusion of cooperative perception for autonomous driving with augmented vehicular reality," *IEEE Systems Journal*, 2020.

[7] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 810–819, 2016.

[8] "Awaiting the realization of fully automated vehicles: Potential economic effects and the need for a new economic and social design," https://voxeu.org/article/potential-economic-and-social-effects-driverless-cars , accessed: 2020-11-08.

[9] M. Chen, B. Liang, and M. Dong, "Multi-user multi-task offloading and resource allocation in mobile cloud systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 10, pp. 6790–6805, 2018.

[10] A. u. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 393–413, 2014.

[11] L. Wan, L. Sun, X. Kong, Y. Yuan, K. Sun, and F. Xia, "Task-driven resource assignment in mobile edge computing exploiting evolutionary computation," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 94–101, 2019.

[12] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[13] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, 2020.

[14] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[15] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[16] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2019.

[17] Z. Zhou, P. Liu, Z. Chang, C. Xu, and Y. Zhang, "Energy-efficient workload offloading and power control in vehicular edge computing," in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2018, pp. 191–196.

[18] M. Shojafar, S. Javanmardi, S. Abolfazli, and N. Cordeschi, "Fuge: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method," *Cluster Computing*, vol. 18, no. 2, pp. 829–844, 2015.

[19] F. Palmieri, L. Buonanno, S. Venticinque, R. Aversa, and B. Di Martino, "A distributed scheduling framework based on selfish autonomous agents for federated cloud environments," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1461–1472, 2013.

[20] S. H. Jang, T. Y. Kim, J. K. Kim, and J. S. Lee, "The study of genetic algorithm-based task scheduling for cloud computing," *International Journal of Control and Automation*, vol. 5, no. 4, pp. 157–162, 2012.

[21] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *2013 8th international conference on computer engineering & systems (ICCES)*. IEEE, 2013, pp. 64–69.

[22] K. KRISHNASAMY *et al.*, "Task scheduling algorithm based on hybrid particle swarm optimization in cloud computing environment." *Journal of Theoretical & Applied Information Technology*, vol. 55, no. 1, 2013.

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

[24] M. Cheng, J. Li, and S. Nazarian, "Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 129–134.

[25] R. Li, Z. Zhao, Q. Sun, I. Chih-Lin, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74 429–74 441, 2018.

[26] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 64–73.

[27] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, and W. Lin, "Random task scheduling scheme based on reinforcement learning in cloud computing," *Cluster computing*, vol. 18, no. 4, pp. 1595–1607, 2015.

[28] Y. Wei, L. Pan, S. Liu, L. Wu, and X. Meng, "Drl-scheduling: An intelligent qos-aware job scheduling framework for applications in clouds," *IEEE Access*, vol. 6, pp. 55 112–55 125, 2018.

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

[30] M. Hosseini, H. Ren, H.-A. Rashid, A. N. Mazumder, B. Prakash, and T. Mohsenin, "Neural networks for pulmonary disease diagnosis using auditory and demographic information," *arXiv preprint arXiv:2011.13194*, 2020.

[31] Z. Xiao, X. Dai, H. Jiang, D. Wang, H. Chen, L. Yang, and F. Zeng, "Vehicular task offloading via heat-aware mec cooperation using game-theoretic method," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2038–2052, 2020.

[32] Z. Ning, J. Huang, X. Wang, J. J. Rodrigues, and L. Guo, "Mobile edge computing-enabled internet of vehicles: Toward energy-efficient scheduling," *IEEE Network*, vol. 33, no. 5, pp. 198–205, 2019.

[33] A. Hammoud, H. Sami, A. Mourad, H. Otrok, R. Mizouni, and J. Bentahar, "Ai, blockchain, and vehicular edge computing for smart and secure

iov: Challenges and directions," *IEEE Internet of Things Magazine*, vol. 3, no. 2, pp. 68–73, 2020.
[34] V. Nguyen, T. T. Khanh, N. H. Tran, E. Huh, and C. S. Hong, "Joint offloading and ieee 802.11p-based contention control in vehicular edge computing," *IEEE Wireless Communications Letters*, vol. 9, no. 7, pp. 1014–1018, 2020.
[35] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, and J. Liao, "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4192–4203, 2019.
[36] W. Zhan, C. Luo, J. Wang, G. Min, and H. Duan, "Deep reinforcement learning-based computation offloading in vehicular edge computing," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.
[37] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2092–2104, 2020.
[38] L. Xia, J. Xu, Y. Lan, J. Guo, W. Zeng, and X. Cheng, "Adapting markov decision process for search result diversification," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017, pp. 535–544.
[39] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.
[40] U. Challita, W. Saad, and C. Bettstetter, "Deep reinforcement learning for interference-aware path planning of cellular-connected uavs," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.
[41] X. Liu, J. Ospina, and C. Konstantinou, "Deep reinforcement learning for cybersecurity assessment of wind integrated power systems," *IEEE Access*, vol. 8, pp. 208 378–208 394, 2020.
[42] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE Transactions on Neural Networks*, vol. 18, no. 4, pp. 973–992, 2007.
[43] X. Rongjun, I. Khalil, S. Badsha, and M. Atiquzzaman, "Collaborative extreme learning machine with a confidence interval for p2p learning in healthcare," *Computer Networks*, vol. 149, pp. 127–143, 2019.
[44] R. Xie, I. Khalil, S. Badsha, and M. Atiquzzaman, "Fast and peer-to-peer vital signal learning system for cloud-based healthcare," *Future Generation Computer Systems*, vol. 88, pp. 220–233, 2018.
[45] J. Liu, L. Zuo, X. Xu, X. Zhang, J. Ren, Q. Fang, and X. Liu, "Efficient batch-mode reinforcement learning using extreme learning machines," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.
[46] M. Li and H. Li, "Application of deep neural network and deep reinforcement learning in wireless communication," *Plos one*, vol. 15, no. 7, p. e0235447, 2020.