Reinforcement Learning for Optimizing Delay-Sensitive Task Offloading in Vehicular Edge-Cloud Computing

Ta Huu Binh, Do Bao Son, Hiep Vo, Binh Minh Nguyen, Huynh Thi Thanh Binh, *Member, IEEE*

***Abstract*—With the appearance of more and more devices connected to the Internet, the world has witnessed an ever-growing number of data to be processed. Among those, many tasks require swift execution time, while the storage and computation capability of Internet of Things (IoT) devices are limited. To address the demands of delay-sensitive tasks, we present a Vehicular Edge-Cloud Computing (VECC) network that leverages powerful computation capabilities through the deployment of servers in proximity to task-generated devices, as well as the utilization of idle resources from smart vehicles to share the workload. Because these limited resources are vulnerable to sudden data arising, it is imperative to incorporate cloud servers to prevent system overload. The challenge now is to find a task offloading strategy that collaborates both edges and cloud resources to minimize the total time surpassing the quality baseline of each task (tolerance time) and make all tasks meet their soft deadlines of quality. To reach this goal, we first model the task offloading problem in VECC as a Markov Decision Process (MDP). Then, we propose Advantage-Oriented Task Offloading with a Dueling Actor-Insulator Network scheme to solve the problem. This value-based reinforcement learning method helps the agent find an effective policy when not knowing all the state attributes changes. The effectiveness of our method is demonstrated by performance evaluations based on real-world bus traces in Rio de Janeiro (Brazil). The experimental results show that our proposal reduces the tolerance time by at least 8.81% compared to other reinforcement learning algorithms and 75% compared to greedy approaches.**

***Index Terms*—Vehicular Edge Computing, Cloud Computing, Deep Q-learning, task offloading, delay-sensitive task.**

I. Introduction

There has been a rapid technological advance with evolving communication and computing devices in recent years. The evolution of Internet of Things (IoT) devices allow interconnection capability among billions of devices. Some innovative applicants include smart homes, self-driven vehicles, 5G internet, intelligent public transport systems, innovative personal equipment, and many more. These applications constantly demand high storage and computation capacity to process complicated tasks that are limited for mobile devices. Mobile Cloud Computing (MCC) [1] is then introduced to enable mobile devices to offload high-intensity tasks to the cloud for computation. However, MCC networks still have high risks of network overload since too many tasks are waiting to be processed by a single cloud system.

Ta Huu Binh, Binh Minh Nguyen, and Huynh Thi Thanh Binh are with the School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam. (Huynh Thi Thanh Binh is the corresponding author, Email: binh.th190094@sis.hust.edu.vn, minhnb@soict.hust.edu.vn, binhht@soict.hust.edu.vn)

Do Bao Son is with the Faculty of Information Technology, University of Transport Technology, Hanoi, Vietnam, and also with the School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam. (Email: sondb@utt.edu.vn)

Hiep Vo is with the School of Computer Science, University of Technology Sydney, Sydney, Australia. (Email: hiep.k.vo@student.uts.edu.au)

The European Telecommunications Standards Institute (ETSI) introduced Mobile Edge Computing (MEC) Network [2] (also known as Fog Computing) to spread the cloud's computational power to further edge nodes to reduce MCC's high network traffic. MEC technology can drastically reduce the task processing delay and computation power since mobile devices can send tasks to the nearby network edge [3–8]. With the recent advancements in-vehicle technology, it is possible to relieve some of the computational burdens for MEC networks by utilizing the vehicles' resources, including In-Vehicle Sensors, Communication Unit, On-Board Unit, etc [9]. The onboard units equipped with a vehicle can perform simple tasks such as collecting and computing local data and offloading data to nearby edge nodes. Utilizing these new vehicular technologies, Vehicular Edge Computing (VEC) [10] is introduced to expand the MEC's computational resources to innovative vehicles (both stationary and moving).

In general, edge servers are very suitable for short-deadline tasks to be offloaded. However, their resources are still limited, and the system is vulnerable when the total computing capacity required surpasses their computational ability. Without the help of cloud computing, the tolerance time of processing each task will get higher and higher over time. Therefore, our proposed framework consists of both cloud and edge servers (stationary or moving). It can be deployed in smart cities, where an enormous number of devices are connected to the Internet and smart vehicles, owing to the capability to work as an edge service. Among the VEC candidates, private ones, have uncertain mobility, while the trajectories of public ones, like buses, which follow the prescribed routes and timetables, are regular and predictable [11]. Therefore, we implement a real bus data scenario to simulate the network. From the growing popularity of the VEC network, the challenge now is how to find a task offloading strategy that utilizes both edge and cloud resources to minimize the total tolerance time or even make all tasks meet their soft deadlines of quality.

So far, Reinforcement Learning (RL) [12] has become a growing approach to solving the offloading problem in VEC by updating offloading policy while interacting with the environment. RL agents can observe the transition of the state to estimate the value of each action it can take. In general, each state influences the following one. However, inside each state, some attributes may not impact that of another state. For instance, the size of the current task cannot determine the size of the upcoming one. Our proposal, *Advantage-Oriented Task Offloading with Dueling Actor-Insulator Network scheme*, has a strategy that tackles that problem, reducing the tolerance time by nearly 10% compared to value-based reinforcement learning. We also introduce the strategy to apply the policy-based method in solving our problem.

The main contributions of this paper can be listed as follows:

- Present a Vehicular Edge-Cloud Computing framework for operative task offloading in smart cities and derive the formulas to calculate the delay of tasks.
- Formulate the offloading problem as a Markov Decision Process (MDP).
- Propose Advantage-Oriented Task Offloading with Dueling Actor-Insulator Network scheme (AODAI) - a value-based reinforcement learning method tackling attributes whose value is independent among all tasks to make the learning process more efficient.
- Introduce Actor-Critic based Task Offloading scheme (ACTO-n) - a policy-based reinforcement learning method that creates pseudo-episode to imitate Actor-Critic algorithms learning directly the strategy to offload tasks.
- Conduct intensive experiments to demonstrate the usefulness of the framework and the effectiveness of our proposed techniques.

The rest of this paper is organized as follows. Section II contains the reviews of the related work. In Section III, we describe the system model. Then, we formulate the problem in Section IV. Section V introduces our proposals. The simulation results and analysis are presented in Section VI. Finally, Section VII concludes our research and shows our future work.

II. Related Work

The main reason for applying edge computing in VEC is to solve the latency problem because the edge nodes are close to the required vehicles. However, their resources are still limited, and it is urgent when the total computing capacity required surpasses their computational ability.

In [13], the authors tried to minimize the execution delay experienced by mobile users in a MEC. The multi-armed-bandit problem is formulated with a reward equal to delay time. Wang et al. [14] aimed to minimize each task's delay while satisfying the tolerances of all task types. The reward is the inverse of the delay or set to negative if the processing time surpasses delay tolerance. They developed a deep reinforcement learning-based resource allocation method to adapt to the changing MEC environment and handle high-dimensional input. The author in [15] proposed a task offloading scheme to maximize the quality of Experience, depending on task processing delay. They use a DQN-based algorithm combined with fuzzy rules to provide a good offloading policy from the beginning of system deployment. Zhang et al. [16] solved the delay minimization problem in vehicular edge computing networks (VECNs). The reward is the proportion between the time saved and the time delay in the worst case. Then, they proposed an RL-based task offloading scheme to solve the problem. Saleem et al. [4] introduced a low-complexity online approach that implements dynamic computation offloading using Lyapunov optimization and reduces task execution time.

In [11], Liu et al. implemented a VEC framework with buses as moving servers to decrease the mobility uncertainty since they have fixed mobility trajectories and strong periodicity. To minimize the latency, a fluctuation-aware learning-based computation offloading algorithm based on multi-armed-bandit theory is given. Pham et al. [17] also adopted buses as candidate mobile volunteer nodes which provide their idle computing resources to help process tasks offloaded from the MEC provider thanks to their regular and predictable routes and timetables. In order to solve their given Mixed-Integer Nonlinear Programming problem, an alternative optimization technique is introduced to decompose the original problem into two sub-problems for solving. The authors in [18] proposed a new strategy to minimize latency in MEC both at the beginning time and in the long term. The real GPS data of buses in Rio de Janeiro, Brazil, was used for experiences.

Both [19] and [20] studied the collaboration between cloud and edge computing. The system in [19] contained multiple multi-antenna small base stations (empowered by edge clouds offering limited computing services for end users), and a MBS that provided via a restricted multiple-input multiple-output backhaul high-performance central cloud computing services to their small base stations. The target is to minimize the system energy consumption while satisfying processing latency constraints. Ren et al. [20] formulated a joint communication and computation resource allocation problem intending to minimize the weighted-sum latency of all mobile devices. A closed-form optimal task-splitting strategy was proposed to solve the problem. The authors in [21] considered the problem of task offloading for End-Edge-Cloud orchestrated computing in mobile networks. A low-complexity hierarchical heuristic algorithm is introduced to achieve server selection. A Cauchy-Schwards Inequality-based closed form is proposed to efficiently determine resource allocation to minimize the weighted sum of the average cost. In [22], Xiaolong Xu et al. proposed a computation offloading method over big data for IoT-enabled cloud-edge computing. Specifically, to address the multi-objective optimization problem of task offloading, NSGA-III is utilized.

In general, edge computing has recently been a hot topic since it is believed to be the key enabler in shaping future intelligent wireless networks. Many studies wanted to push the framework even further with the help of vehicles having plentiful idle resources. Some papers utilized buses as vehicular edge servers to tackle the drawbacks of high mobility and instability, we inherits this in our system. Edge computing, though, can still suffer from some issues, especially limited resources. Therefore, many researchers tried to find a strategy that effectively collaborates with edge and cloud servers. However, the environment is static (such as minimizing delay time when offloading a set of tasks simultaneously). Moreover, they were not concerned about how the present decision could affect future performance. In our dynamic scenario, we implied reinforcement learning to make the offloading strategy adaptive with the changing of highly unpredicted environment, especially our proposed algorithms, which are combined with some prior knowledge to get a better offloading decision.

III. System model

Our Vehicular Edge-Cloud Computing framework includes three layers, as illustrated in Fig. 1. Device layer is composed of all equipment that can generate and send data to the base station (BS) for offloading process. A cluster of stationary servers is located near the base station. Due to the fact that

among vehicular edge servers candidates, public ones, like buses, have much more stable routes and timetables compared to those of private ones, all vehicles chose to be edge servers in our framework are buses with idle computation resources ready to process offloaded tasks [17]. One of the challenges with utilizing vehicular edge computing systems is that there is high communication overhead between base stations and vehicular servers for wireless data transferring. To address this, we propose deploying access points near the bus traces to reduce the wireless transfer distance. Furthermore, these access points enable vehicular edge servers to periodically disseminate beacon messages that contain critical information such as their identity, system state, and position, making the information about the entire system more reliable and stable. Servers in the cloud layer are other reliable and available resources for offloading tasks.

The difference between our model and traditional MEC [2] is that we utilized computing vehicles for computation and communication in the edge computing network. In addition, access points are deployed across the prescribed routes of buses and act as messengers between the base station and vehicular edge servers to significantly reduce the wireless transmission rate. Intelligently transferring the workload to vehicles on the fly during task offloading tasks can improve the overall computation efficiency. Furthermore, our proposed framework reflects real-time decision-making for task-offloading via a realistic geo-distribution dataset.

The system time is divided into time slots of equal length. We represented N_i as i^{th} task being generated. Each task information is a tuple $(h_i, r_i, s_{i,out}, s_{i,in}, d_i)$, where h_i is the time when it is created; r_i is the amount of computational resources required for task i^{th}; $s_{i,in}$ and $s_{i,out}$ are the input and output data size for the task N_i, respectively; and d_i is the quality baseline of the task, which is soft deadline.

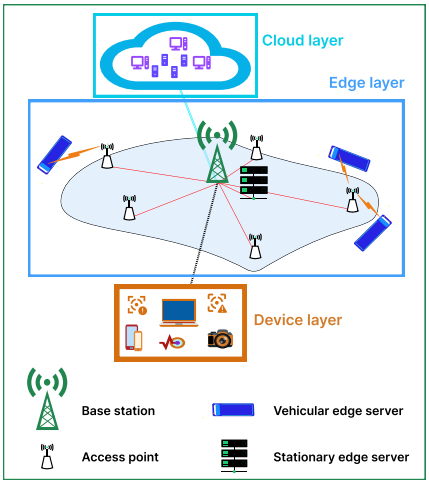


Fig. 1: Illustration of the system model

Throughout this paper, we denote the edge nodes as F_j ($j = 0, ..., n$), where F_0 is the server located at BS, and $F_1, ..., F_n$ is n vehicular edge server (VS) located at the vehicles. For each node F_j, we represent $\mathcal{C}(F_j)$ and $\mathcal{Q}(F_j)$ as the computational capacity and the queue time of node F_j (the time for node F_j to process all the remaining tasks in its queue), respectively. If $\mathcal{Q}(F_j) = 0$, there is no task in the queue.

A. Communication model

During each time slot, tasks are continually spawned from the device layer and sent to the BS, which has a role as the conductor of the offloading process, via wireless multiple-input multiple-output (MIMO) technology [23]. At both the source and the destination, multiple antennas are combined to optimize data speed and minimize errors, enabling data to travel over many signal paths at the same time. Each task requires a soft deadline of quality. When the base station receives a task along with its information, the deadline is recalculated based on the latency from the end user to BS so that the system knows the remaining time needed to meet the quality baseline. From now on, when referring to "deadline" or "quality baseline", we mean the delay time required for the offloading process to meet the origin quality baseline given by the task. All vehicular edge servers periodically disseminate beacon messages, which contain information about their identity, system state, and position. Based on information about the status of the servers in the system, the base station determines which one, edge nodes or cloud, should handle the task. For a task N_i, we denote the transmission time from BS to computation server, and vice versa is $\mathcal{D}_{b2s}(N_i)$ and $\mathcal{D}_{s2b}(N_i)$, respectively. There are three possible kinds of offloading tasks - offload to BS (type-1, the task is processed at the stationary server), offload to VS (type-2), or offload to cloud (type-3). Each of them has different characteristics; the total transmission delay is also different.

If the chosen edge server is the stationary one, the latency (transmission) is presumed to be 0 since they are located locally at the base station. It is more complicated with the type-2 task. In this case, the delay time depends on the transmission rate from the BS to the VS. VS communicates with BS through Access Point (AP) closest to VS. Therefore, the transmission time between VS and BS includes the transmission time between BS and AP (wired link), and the transmission time between the AP and VS (wireless link). Since the connection between AP and BS is a wired link with significantly higher bandwidth than the wireless link, the transmission time between AP and BS is set as 0 [24]. We represent the distance between AP and F_n as $d(AP, F_n)$. This distance depends on the location of VS, denoting $L(F_n)$. We can define the transmission rate between AP and F_n as follows:

$$v(AP, F_n) = W \cdot log_2 \left(1 + \frac{P \cdot d(AP, F_n)^{-\alpha}}{\sigma^2 + I}\right) \quad (1)$$

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

$$\mathcal{D}_{b2s}(N_i) = \frac{s_{i,in}}{v(AP, F_n)} = \frac{s_{i,in}}{W \cdot log_2 \left(1 + \frac{P \cdot d(AP, F_n)^{-\alpha}}{\sigma^2 + I}\right)} \quad (2)$$

After the task is performed, F_n sends the results to the BS via the nearest AP, denoted AP'. Because the vehicle is moving, the position when receiving the task is different from the position after the task is done, and the AP' may be different

from the AP. The time to transmit the result from F_n to BS is calculated as follows:

$$\mathcal{D}_{s2b}(N_i) = \frac{s_{i,out}}{v(F_n, AP')} = \frac{s_{i,out}}{W \cdot log_2\left(1 + \frac{P \cdot d(F_n, AP')^{-\alpha}}{\sigma^2 + I}\right)} \tag{3}$$

In terms of the communication between the BS and the central cloud with ultra-high computing capability, wired optical cables with high bandwidth are utilized. Similar to [21], the transmission time from BS to cloud servers and from cloud servers to BS, respectively is:

$$\mathcal{D}_{b2s}(N_i) = \frac{s_{i,in}}{v_c} \tag{4}$$

$$\mathcal{D}_{s2b}(N_i) = \frac{s_{i,out}}{v_c} \tag{5}$$

where v_c is the transmission rate between two ends of the communications circuit.

B. Computation model

The total delay of each task offloading to VS consists of four parts: (1) transmission time from BS to the computation server, (2) transmission time of results from the computation server back to BS, (3) waiting time in computation server's queue, and (4) processing time at computation server. Given the processing time at the computation node, we represent $\mathcal{D}_w(N_i)$ and $\mathcal{D}_e(N_i)$ as the time the task waits in the queue and the time to execute the task, respectively. Therefore, the total delay of N_i^t can be written as follows:

$$\mathcal{D}(N_i) = \mathcal{D}_{b2s}(N_i) + \mathcal{D}_w(N_i) + \mathcal{D}_e(N_i) + \mathcal{D}_{s2b}(N_i) \tag{6}$$

Latency of executing the task in the cloud is:

$$\mathcal{D}_e(N_i) = \frac{r_i}{\mathcal{F}^c} \tag{7}$$

The problem with the cloud is that it is too far from the task-generated devices, which makes the transmission time solely much bigger than the time transmitting to and processing at any edge node combined. All edge servers have the same processing time formula. With computational capacity $\mathcal{C}(F_n)$, latency of executing the task is determined as follows:

$$\mathcal{D}_e(N_i) = \frac{r_i}{\mathcal{C}(F_n)} \tag{8}$$

If a task is offloaded to BS, the time it must wait in the queue is the queue time of the edge node:

$$\mathcal{D}_w(N_i) = \mathcal{Q}(F_0) \tag{9}$$

On the other hand, if VS is where the task is offloaded to, because it takes time to transmit data from BS to VS, The time it has to wait in the queue is calculated as:

$$\mathcal{D}_w(N_i) = \max\left(0, \mathcal{Q}(F_n) - \frac{s_{i,in}}{W \cdot log_2\left(1 + \frac{P \cdot d(A, F_n)^{-\alpha}}{\sigma^2 + I}\right)}\right) \tag{10}$$

From (2), (3), (8), and (10), we derive the total delay of N_i as follows:

$$\mathcal{D}(N_i) = \max\left(\frac{s_{i,in}}{W \cdot log_2\left(1 + \frac{P \cdot d(AP, F_n)^{-\alpha}}{\sigma^2 + I}\right)}, \mathcal{Q}(F_n)\right) + \frac{s_{i,out}}{W \cdot log_2\left(1 + \frac{P \cdot d(F_n, AP')^{-\alpha}}{\sigma^2 + I}\right)} + \frac{r_i}{\mathcal{C}(F_n)} \tag{11}$$

IV. Problem formulation

Our work aims to find a task offloading policy that effectively reduces the overall tolerance time (amount of time surpassing the quality threshold). At each time, we can only know about current and past information on the tasks and the system status. The base station offloads each task to one and only one server, and the task coming first is processed first.

We have yet to know all about the future information of the environment. However, to facilitate comprehension, we add some unrealistic assumptions that the information in the future can be predicted precisely to formulate the task offloading problem as an integer programming problem. Supposed that we know the information of all tasks right at the beginning, our concerned problem is expressed as:

$$\begin{aligned} \min_{x_k^t,\ \forall t=\overline{1,\tau},\ \forall k=\overline{0,\kappa}} \quad & \sum_{t=1}^{\tau} max\Bigg(\sum_{k=0}^{\kappa}\Big(x_k^t \times \mathcal{D}_k(N_t)\Big) + \\ & \prod_{k=0}^{\kappa}\Big(1 - x_k^t\Big) \times \mathcal{D}_c(N_t) - \mathcal{T}(N_t),\ 0\Bigg) \\ \text{s.t.} \quad & \sum_{k=0}^{\kappa} x_k^t \leq 1 \quad \forall t = \overline{1,\tau} \\ & x_k^t \geq 0 \quad \forall t = \overline{1,\tau},\ \ \forall k = \overline{0,\kappa} \\ & x_k^t \in \mathbb{Z} \quad \forall t = \overline{1,\tau},\ \ \forall k = \overline{0,\kappa} \end{aligned} \tag{12}$$

where τ is the number of tasks, κ is the number of VS. Task t^{th} is offloaded to server F_k if and only if $x_k^t = 1$. If $x_k^t = 0\ \forall k = \overline{0,\kappa}$, task t^{th} is offloaded to cloud. Each task is inseparable and offloaded to exactly one server. $\mathcal{D}_k(N_t)$ is the delay time when offloading the task t^{th} to server F_k, while $\mathcal{D}_c(N_t)$ is the delay time when offloading that task to cloud. All tasks must be processed. If the task is finished before surpassing the quality baseline, the tolerance then is 0. Our objective is finding the policy that leads to the least tolerance time.

Nevertheless, we can not have information about future tasks as well as other dynamic changes in the environment. All past decisions, more or less, can affect the delay time of processing the current task. It will be suitable to model our problem as a Markov Decision Process, which has a 4-tuple (S, A, P, R). S is state space, A is action space, P is the probability that an action α at a state s at time t leads to state s' at time t', and R is reward function orienting the agent towards our objective. Note that, in practice, when an offloading decision is made, reward and information of the next state can not be observed immediately (the tasks must be completed to have the exact information). However, by storing past data, we can similate the environment that provides feedback right after an action has been made. Details of how we design each element in this process is provided in sections below.

A. States

For task N_i, we represent $S(N_i)$ as state system and define as follows: $S(N_i) = \{r_i, s_{i,in}, d_i, \mathcal{Q}(F_0), \{D(F_1), \mathcal{Q}(F_1)\}, ..., \{D(F_n), \mathcal{Q}(F_n)\}\}$ The state space stores information on current task offloading, the local server at the Base Station (BS), and all Vehicular Servers (VS). Three aspects of each task are considered,

including the computational resource required (r_i) to perform it, the size of the containing packet ($s_{i,in}$), and the task quality baseline (d_i), which is a soft deadline. The waiting time of queues ($\mathcal{Q}(F_0)$,..., $\mathcal{Q}(F_n)$) in each server (both BS and VS) also contributes to the offloading decision when a task arrives. The distance from each VS ($D(F_1)$,...,$D(F_n)$) to the nearest access point is also included in the state space because they determine the delay time to transfer the data from the access point to the VS.

B. Actions

Under system status, depicted in the state space, the BS chooses the server performing each data packet. Thus, an action is a solution to offload a task. Formally, suppose we have n VSs, the action vector for the task N_i^t is presented as an one-hot encoding vector $\alpha(N_i) = \{\alpha_0, \alpha_1, ..., \alpha_n, \alpha_{n+1}\}$. If the task is processed at the i^{th} vehicular edge server, $\alpha_i = 1$. Suppose it is processed locally at the base station, $\alpha_0 = 1$. Otherwise, the task is offloaded to the cloud, $\alpha_{n+1} = 1$.

C. Transition

After choosing an action in each state, we do not have the information of the next state right away (except for the simulated environment when past data is recorded). The reason is that the agent relies only on the current state when giving any offloading decision, which means the next state is only observed when a new task arrives. The reward is received after the corresponding task is finished. Moreover, the new task information does not depend on the last task.

D. Reward

Our goal is to minimize the tolerance time. Therefore, the agent's reward should be diminished by its tolerance time. Given delay time $\mathcal{D}(N_i)$, the reward is formulated as follows: $R(N_i) = max(0, \mathcal{D}(N_i))$. If the task is finished before the quality baseline, the agent will be neither rewarded nor punished. Otherwise, the reward is defined as minus tolerance time, which means the more tolerance time the agent causes, the more punishment it is given.

V. Proposal

Reinforcement Learning [12] is one of three main types of Machine Learning. It is a control-theoretic trial-and-error learning technique that uses rewards to direct the agent toward the expected goal. In the case of task offloading in a mobile edge computing network, it is too complex to write a program that could effectively manage every possible combination of circumstances, if not to say that the transition states are uncontrollable. That makes reinforcement learning techniques become a popular approach due to their high adaptiveness in an unstable environment. Moreover, deep reinforcement learning is preferred over tabular one because the state space is continuous. As a result, DQN-based algorithms are favored in many other works [25–28]. These approaches are categorized as value-based methods since they all need a function that estimates "the quality" of each action or state to support a predefined strategy to choose an action (a policy).

To further clarify, Reinforcement Learning (RL) methods are favored over traditional methods for our task offloading problem because RL algorithms can better cope with the uncertainties of the edge computing system since the model is trained on the dynamic environmental changes over time. RL belongs to the eager learning family – the model is pretrained with historical data – resulting in fast inference time. It is worth mentioning that finding the optimal solution to our problem is impossible since we can not comprehend all information about the environment. As a result, all approaches are heuristic. Other methods, non-machine-learning based or lazy-learning algorithms, such as genetic algorithms, are inappropriate for solving the optimization problem since they take too long to find task-offloading solutions. In other words, task offloading requires a real-time decision, and time-consuming algorithms like Genetic algorithms [29], Particle Swarm Optimization [30], etc., are unsuitable. The task-offloading research that implements these approaches tends to tackle non-continual offloading problems, or tasks are collected as bags of tasks before being offloaded to edge servers, which delays offloading decisions. Ruled-based approaches are also considered unsuitable because the environment is highly dynamic and complex [31]. Greedy algorithms can be fast, but the performance is dominant by machine-learning-based approaches, as presented in the experimental section.

Our first proposal, AODAI, is constructed on value-based approaches and presented in section V-A. The other one, ACTO-n, which is presented in section V-B, however, modifies the probability of choosing each action dynamically (policy-based method) though still relies on state-value estimation to prevent a high variance.

A. Advantage-Oriented Task Offloading with Dueling Actor-Insulator network scheme (AODAI)

We designed AODAI to work in Markov Decision Process in which our problem is formulated. To drive the agent toward the target, a metric called *return* is used, which is defined as follows:

$$\begin{aligned} G_t \doteq & \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...] \\ = & \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}] \end{aligned} \tag{13}$$

where γ is the *discount rate*, $\gamma \in [0; 1)$, R_t is the reward at time step t.

The objective is to maximize G_t. However, it can be easily seen that G_t can not be calculated directly since the future reward is unknown. Thus, AODAI uses the so-called Q-value that receives state and action information as inputs to estimate the *expected return*. The Q-value is updated on the fly as follows:

$$Q(s, a) = R(s, a) + \gamma \max_{a'}(Q(s', a')), \tag{14}$$

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

Note that rewards received by the agent depends on actions that the agent chooses. As a result, the ideal exact Q-value depends on the probability of choosing each action in a given state or *policy* of the agent.

1) Actor-Insulator network architecture: Two neural networks with the same architecture are utilized to approximate the Q-values. They take the environment's information as inputs and return the Q-values of all possible actions as outputs. Within each iteration, Actor is the network having parameters θ changing based on a loss function, while Insulator is the network that updates its parameters θ' toward θ:

$$\theta' := \theta' + \alpha \times (\theta - \theta'), \tag{15}$$

where $\alpha \in (0, 1)$ is the step-size that modifies the speed of the mimicking of Insulator toward Actor, also called *target update rate*. However, when the model is deployed, only Actor network contributes to the process of deciding which server to offload a task to (Algorithm 1).

Algorithm 1 Decision-making in deployment (test) phase

Input: $\mathcal{S} = \{d(F_1), \mathcal{Q}(F_1), ..., d(F_n), \mathcal{Q}(F_n), Q(F_0)\}$, $\mathcal{T}_i = \{r_i, s_{i,in}, d_i\}, i = \overline{1, \tau}$

Output: Servers that offload each task

1: **for** each task $\mathcal{T}_i$ in task queue **do**
2: $s_0 \leftarrow (S, \mathcal{T}_i.getTaskInfo())$ {S is status of system}
3: Save s_0 to update parameters of each network
4: Actor network observes s_0 and approximates Q-value of each action
5: Take action a_0 the highest Q-value
6: Agent executes a_0
7: **end for**

Insulator, with a more "stable" behavior, helps the training phase become more stable with the new Q-value estimation to teach the Actor defined as:

$$target \doteq R_t + \gamma Q(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta') \tag{16}$$

Actor gives the highest action based on what it learns, but the action is reevaluated by Insulator, preventing the Q-value estimation from being too optimistic.

Because the state space is huge and the agent needs to learn from the data collected from interacting with the environment, a replay memory mechanism is implemented to store the experience. This experience is presented as a 4-tuple of the current state, current action, the reward taken from performing the action at the current state, and the next state of the current training. In our schema, all 4-tuples can only be determined at the starting of a new decision cycle because the state vector contains some information about a newly arrived task. The approximation network samples a mini-batch from the memory mechanism to update its parameters θ by minimizing the loss function, which is defined as the mean squared error of the predicted Q-value and the target Q-value:

$$L(\theta) \doteq (target - Q(s_t, a_t; \theta))^2 \tag{17}$$

2) Advantage-oriented strategy: As can be seen in (14), the Q-value $Q(s, a)$ of a state and an action is gradually updated based on the biggest action-value of the next state s'. Generally speaking, s' is always affected by the last decision of the agent. However, among all attributes of s', some are independent of which action the agent takes as well as the corresponding attributes of the last state. For instance, the data size of one task cannot affect that of the next task. We call these properties *volatile*. The volatile attributes are important for the phase of taking action but should be handled in the training phase, especially when the experience replay buffer is limited; the learning process can be biased. Before updating the model, all volatile attributes in the *next state* (s_1) components of all experiences in the taken batch are changed to the estimated expected values.

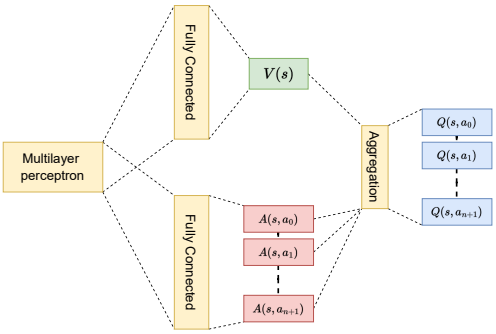


Fig. 2: Dueling DQN neural network architecture

For some timesteps, it is not too necessary to know the value of each action, such as when the waiting queues contain few tasks or the quality baseline is not too high. The decision does not impact much on the overall result. We can detect that by estimating how advantageous selecting an action relative to the others at the given state is. The Advantage is a quantity obtained by subtracting the Q-value from the V-value:

$$A_\pi(s_t, a_t) \doteq Q_\pi(s_t, a_t) - V_\pi(s_t) \tag{18}$$

Each network is split into two separate streams as in Fig. 2, one for estimating the state-value and the other for advantage-value. After these two streams, the last module of the network combines the state-value and advantage outputs.

B. Actor Critic based Task Offloading scheme (ACTO-n)

Value-based reinforcement learning needs to estimate action-value or state-value function and uses a predefined policy to make decisions. When the policy changes, the exact Q-value also changes. Therefore, AODAI requires a queue storing past experiences as shown in section V-B to train the model in a more stable way. As a result, more training iterations are required. On the other hand, policy-based reinforcement learning learns the policy directly without training on past experiences. However, we need rewards in a whole episode to train a policy. However, in our problem, there is no terminal state. Thus, we group each n-consecutive time steps to form a pseudo episode and design Actor Critic based Task Offloading scheme (ACTO-n) to solve our problem. We call θ as parameters a neural network named Actor. The target here is to maximize the value of policy score function $J_t(\theta)$, which is used to calculate the expected reward of the policy π estimated by Actor network:

$$\begin{aligned} J_t(\theta) \doteq \mathbb{E}_\pi[G_t] &= \sum d^\pi(s_t) * V(s_t) \\ &= \sum_s d^\pi(s_t) \sum_a \pi_\theta(a_t|s_t) Q_\pi(s_t, a_t), \end{aligned} \tag{19}$$

where $d(s_t)$ is the percentage that the t^{th} state is s_t. Based on policy gradient theorem [32]:

Algorithm 2 Networks' training in task offloading process

Result: Task offloading policy represented by θ

Initialize θ of the Actor network
Initialize $\theta' = \theta$ of the Insulator network
Initialize queue M as an experience replay buffer whose capacity is $\mathcal{M}$
Initialize policy $\mathcal{P}$
Initialize a set of estimated expected volatility values Ξ
Initialize $\eta \in [0, 1]$ $\{\eta$ is the update rate for each element in $\xi\}$
for each time slot t **do**
  for each arriving task $\mathcal{T}$ **do**
    $s_0 \leftarrow (S, \mathcal{T}.getTaskInfo())$ $\{S$ is status of system$\}$
    Actor network observes s_0 and approximates Q-value of each action.
    Take action a_0 following policy $\mathcal{P}$
    Agent executes a_0 and observe reward r_1 and new state s_1
    Push experience $\langle s_0, a_0, r_1, s_1 \rangle$ to M
    for each element $\xi \in \Xi$ **do**
      $\xi' \leftarrow$ value of corresponding volatility attribute of $\mathcal{T}$
      $\xi \leftarrow (1 - \eta) \times \xi + \eta \times \xi'$
    end for
    for every k execution step **do**
      Split M into discriminate replica random batches
      for each batch $\mathcal{E}$ **do**
        for each experience $e \in \mathcal{E}$ **do**
          Get s_1 from e
          Temporarily change value of volatility attributes in s_1 to corresponding element in Ξ
        end for
        Calculate *target* following equation (16)
        Update θ according to loss function in (17)
        Update θ' following equation (15)
      end for
    end for
  end for
end for

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \sum_s d^\pi(s_t) \sum_a Q_\pi(s_t, a_t) \nabla_\theta \pi_\theta(a_t|s_t) \\ &= \sum_s d^\pi(s_t) \sum_a \pi_\theta(a_t|s_t) Q_\pi(s_t, a_t) \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \\ &= \mathbb{E}[Q_\pi(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)]\end{aligned} \tag{20}$$

Based on Monte-Carlo method, the expectation of the sample gradient is equal to the actual gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}[G_t(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)] \tag{21}$$

We use a different neural network called Critic (with parameter θ^*) to calculate $V(s_t)$. Since $\mathbb{E}[\nabla_\theta \ln \pi_\theta(a_t|s_t)] = 0$, we have:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}[(G_t(s_t, a_t) - V(s_t)) \nabla_\theta \ln \pi_\theta(a_t|s_t)] \\ &= \mathbb{E}[A_t(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)]\end{aligned} \tag{22}$$

To break down the objective function to something tractable, we need a way to approximate that expectation accurately. However, its exact form involves an integral over a probability distribution policy we don't have access. Using Monte-Carlo approximation, we can rewrite:

$$J_t(\theta) \doteq \sum_s d^\pi(s_t) \sum_a \pi_\theta(a_t|s_t) R(s_t, a_t) \tag{23}$$

Equation (22) can now be rewrite as:

$$\nabla_\theta J(\theta) = \mathbb{E}[(\sum_{k=\iota}^{n} \gamma^{k-\iota} R_k - V(s_t)) \nabla_\theta \ln \pi_\theta(a_t|s_t)], \tag{24}$$

where $\sum_{k=\iota}^{n} \gamma^{k-\iota} R_k$ is the return value at ι^{th} time step in a pseudo episode. To train Critic network, we need to minimize this loss function:

$$L(\theta^*) \doteq (R_t + \gamma V(s_{t+1}; \theta^*) - V(s_t; \theta^*))^2 \tag{25}$$

Algorithm 3 Actor Critic based Task Offloading scheme

Result: Task offloading policy represented by θ

Initialize θ of the Actor network
Initialize θ^* of the Critic network
Initialize list $\mathcal{L}$ to store log probabilities of chosen actions
Initialize list $\mathcal{V}$ to store state values
Initialize list $\mathcal{R}$ to store given reward
for each time slot t **do**
  for every n consecutive tasks **do**
    Empty $\mathcal{L}$, $\mathcal{V}$, $\mathcal{R}$
    for each arriving task $\mathcal{T}$ **do**
      $s_0 \leftarrow (S, \mathcal{T}.getTaskInfo())$ $\{S$ is status of system$\}$
      Actor network observes s_0 and give policy $\mathcal{P}$.
      Take action a_0 following policy $\mathcal{P}$
      Agent executes a_0 and observe reward r_1, new state s_1 and $V(s_0)$ given by Critic network;
      Add $log(\mathcal{P}(a_0), V(s_0), r_1$ to $\mathcal{L}$, $\mathcal{V}$, $\mathcal{R}$ respectively
    end for
    Based on $\mathcal{L}$, $\mathcal{V}$, $\mathcal{R}$, update θ according to gradient calculated in (24)
    Based on $\mathcal{V}$, $\mathcal{R}$, update θ^* based on (25)
  end for
end for

VI. Experimental Results

A. Simulation Setting

The settings in our simulations are set as follows. We train each model in 120 time slots with 800 tasks per one, then test it in 30 time slots. For vehicle trajectories, we use real GPS data of buses in Rio de Janeiro, Brazil [1]. We take out the coordinates of the buses on route 371 on January 25, 2019, for experiments. Given the discrete time intervals observed in the data, estimating the position of a bus at an unobserved point in time involves determining its position at the two closest points in time (one before and one after), and computing the position estimate as a linear interpolation between these two corresponding positions. With the knowledge of each bus's location, we were able to calculate the distance from the bus to the nearest AP and determine the data transmission speed between the VS and AP. We set the channel noise and ignore inner interference same in [14, 17, 20]. The other parameter values in the simulation are configured as in the table I.

[1] https://www.kaggle.com/igorbalteiro/gps-data-from-rio-de-janeiro-buses

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3292591

TABLE I: Parameter values for simulation

Parameter	Value	Unit
Number of BS	1	
Number of VS	5	
Computational capacity of BS ($\mathcal{C}(F_0)$)	4	GHz
Computational capacity of VS ($\mathcal{C}(F_1), ..., \mathcal{C}(F_n)$)	2	GHz
Number of task in a time slot	800	
Required CPU cycles of task (r_i^t)	[500,600]	Megacycles
Input data size ($s_{i,in}^t$)	[1.5,2]	MB
Output data size ($s_{i,out}^t$)	[15,20]	KB
Deadline per task (d_i^t)	[1,1.5]	second (s)
Transmission power (P_r)	46	dBm
Channel bandwidth (W)	20	MHz
Path loss exponent (α)	4	
Background noise power (σ^2)	100	dBm

In the following, we will compare the effectiveness of different algorithms based on the main objective: To minimize the tolerance time. To illustrate the efficiency of AODAI, we compare it with other value-based approaches in subsection VI-B. The impact of target network update rate and discount factor on the performance of AODAI is also considered. The performance of policy-based approaches is illustrated in subsection VI-C. The performance through 30 time slots in test phase of different approaches, including greedy algorithms, is shown in this part. Besides, the last subsection VI-D will change some components in the settings to see the affections of the cloud, local edge server, and the number of vehicular edge servers on the tolerance time of offloading tasks in these systems.

B. Value-based approaches

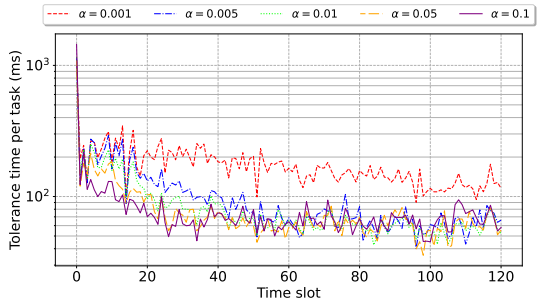


Fig. 3: Impact of target update rate on training performance of AODAI

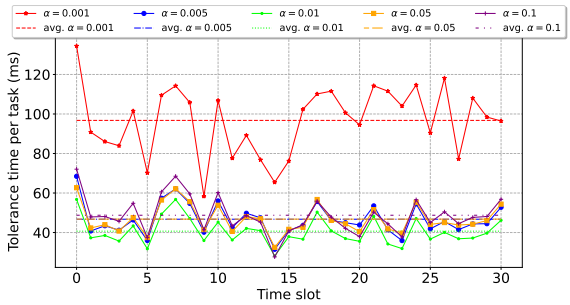


Fig. 4: Impact of target update rate on test performance of AODAI

Fig. 3 and Fig. 4 show the impact of the target update rate (α) (The rate that Insulator network parameters would track the Actor network parameters) on the test and training performance of AODAI. When training, the model tends to learn faster as alpha increases. In the test phase, the tolerance time per task decreases when adjusting alpha from 0.001 to 0.01 and increases when adjusted from 0.01 to 0.1. We observe $\alpha = 0.01$ as the best target update rate based on the testing phase's average tolerance time per task. We also deduct from Fig. 3 and Fig. 4 that the target update rate should not only be too low (leading to an extremely "conservative" behaviour of Insulator network) but also not too high (making the learning process of Insulator network unstable).

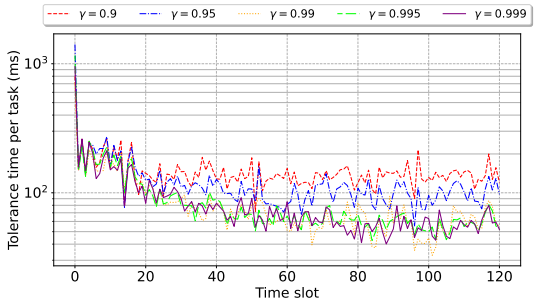


Fig. 5: Impact of discount factor on training performance of AODAI

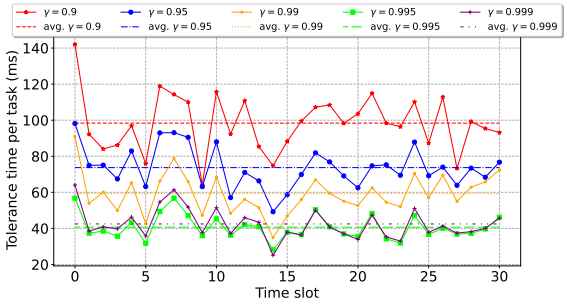


Fig. 6: Impact of discount factor on test performance of AODAI

Fig. 5 and 6 show the impact of the discount factor (γ) on the test and training performance of AODAI. During the training phase, the higher the discount factor, the higher the level of the agent's concentration on future rewards, and the tolerance time per task see a downward trend. We observe that the tolerance time per task tends to improve over time as we increase the γ value during the training phase. γ values of 0.995 and 0.999 perform relatively similarly and are superior to other gamma values. It is noteworthy that in the test phase, the performance of AODAI with $\gamma = 0.995$ is slightly better than that with $\gamma = 0.999$.

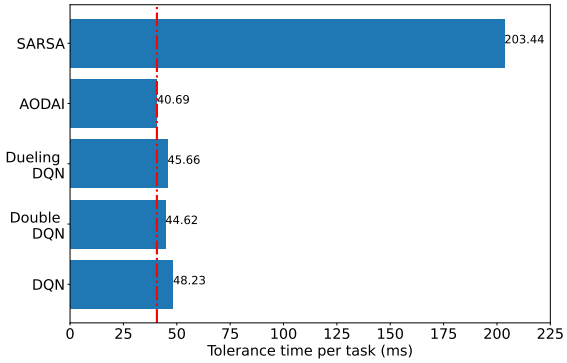
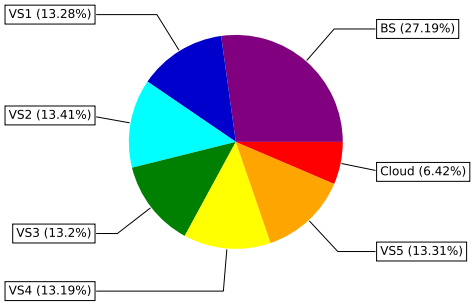
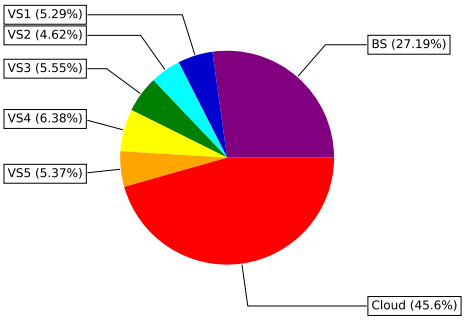


Fig. 7: Average tolerance time in 30 time slots using different value-based approaches in test phase

We compare the performances of five value-based approaches when deployed in Fig. 7. It can be seen clearly



(a) AODAI



(b) SARSA

Fig. 8: Percentage of tasks offloaded to each server.

that SARSA [33], an on-policy reinforcement learning, brings much higher tolerance time than other approaches, which are all off-policy. To update the Q-value estimator, SARSA relies on its current policy:

$$Q(s,a) = R(s,a) + \gamma(Q(s',a')) \tag{26}$$

Hence, SARSA cannot utilize the experience replay technique as in DQN-based algorithms, leading to unstable training as a sequence of autocorrelation problems. According to [12], the update method following equation (26) also makes SARSA more "conservative". The agent tends to avoid a dangerous optimal offloading strategy and only slowly learn to use it when the exploration parameters are reduced. This manner is illustrated in Fig. 8b as SARSA prefers to offload tasks to the Cloud rather than Edge servers since the computing resource of these servers is much lower and, thus, has a higher risk of task congestion. Meanwhile, Fig. 8a shows that AODAI leads to much fewer tasks offloaded to the Cloud. In both scenarios, tasks are offloaded evenly to each vehicular edge server. Ignoring SARSA, the simplest approach DQN, brings the worst performance, nearly 20% worse than AODAI in terms of average tolerance time over 30-time-slot-period. Dueling DQN [34] splits the Q-value into two different parts: the state value function that estimates how good it is for the agent to be in a given state and the advantages for each action in that state; and presents a change in the network structure compared to DQN has a slightly higher tolerance time compared to Double DQN. AODAI is the best approach, whose tolerance time is just 91.19% as high as that of the second best approach, Double DQN.

C. Policy-based approach

Table II shows the average tolerance time processing each task when applying Reinforce based approach (RF-n) [32] and ACTO-n with different n-value and discount factor of 0.9 or 0.99. Without the feedback of Critic-like network, the performance of RF-n is quite bad, regardless of how big the n-value and discount factor value is. We can see that because the agent might take a lot of actions over the course of an episode, it's hard to assign credit to the right action, which means that these updates have a high variance. As a result, RF-n does not have quite a good performance. On the other hand, ACTO-n, by taking state-value into consideration as can be seen in equation 24, has lower variance and thus, a better learning result, even in a long pseudo-episode as 20 or 30 time step per episode, but still does not handle the problem very well when the pseudo-episode becomes longer (such as when $n = 40$, the performance is even worse than RF-n in considering cases). By taking bigger discount factor value, the result seems better but only when the pseudo-episode is long enough, such as with $\gamma = 0.99$, utilizing ACTO-30 brings the lowest tolerance time.

TABLE II: Performance of RF-n and ACTO-n with different n-value in different discount factor value

Discount factor (γ)	Algorithm	Tolerance time (ms)
0.9	RF-10	146.72
	RF-20	191.26
	RF-30	163.09
	AC-10	89.26
	AC-20	80.96
	AC-30	87.13
0.99	RF-10	143.91
	RF-20	177.09
	RF-30	180.18
	AC-10	131.59
	AC-20	135.63
	AC-30	**57.22**
	AC-35	64.31
	AC-40	315.18

Performances through 30 time slots in the test phase between three reinforcement learning approaches and two greedy approaches, namely Shortest Latency algorithm (SL) and Shortest Tolerance Time (ST) algorithm, are illustrated in Fig. 9. Shortest Latency algorithm aims to prioritize actions bringing the lowest latency in each step, while the shortest Tolerance Time aims to take action bringing the lowest tolerance time in each step. The most conspicuous implication is that SL and ST bring much higher tolerance time than proposals using reinforcement learning techniques. In general, ST brings the worst outcome. On the opposite side, AODAI not only has the lowest average tolerance time but also own the most best-performance-in-a-time-slot times. In any time slot, the result brings by AODAI surpasses that of DQN. Only in time slot 7^{th} and 17^{th} does AODAI not have the best lowest result, both cases caused by ACTO-30. However, the performance of ACTO-30 fluctuates sharply compared to value-based approaches. This is because the policy in value-based is predefined while that of ACTO-30 is more dynamic (the agent learns the policy on the fly). Meanwhile, the performance of AODAI seems the most stable, thanks to the strategy that tackles volatility attributes (Algorithm 2).

D. Different environment settings

Fig. 10 demonstrates the need for a cloud layer in our framework. It shows that eliminating the cloud layer is much higher than removing some edge servers. With the "no cloud"

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3292591

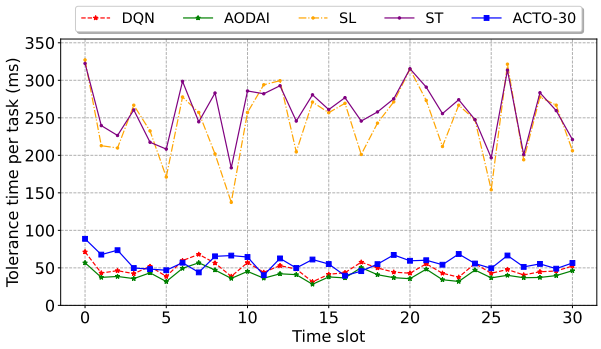


Fig. 9: Performance through 30 time slots in test phase of different approaches

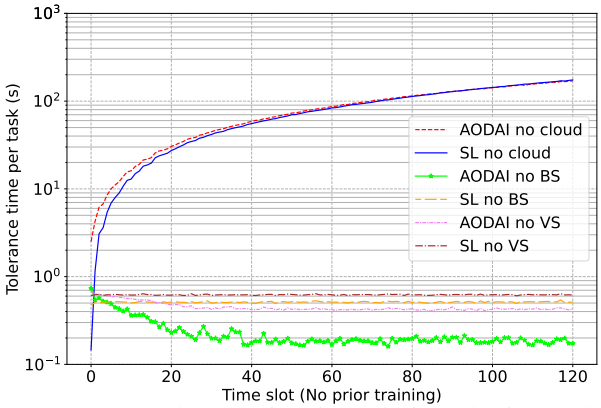


Fig. 10: Average tolerance time in each time slot in no-cloud, no-BS, and no-VS setting

scenario, it is clear to see that applying SL or AODAI cannot prevent the average tolerance time of processing each task that appeared in a time slot from getting higher and higher over time (We called this phenomenon *accumulated congestion*). Without prior training, the performance of AODAI is slightly worse than greedy approach (SL) at the beginning but gradually gets a lower average tolerance time of processing each task that appeared in a time slot compared with that performed by the greedy approach. If excluding BS or VS from the system model, *accumulated congestion* does not happen, and AODAI still holds better performance compared to that of SL. When no VS takes part in processing offloaded tasks, only two choices can be made: offloading to the cloud or offloading to BS. Thus, we can not see much difference between the results of the two approaches. However, when only BS is excluded, the action space is larger, leading to the observation that the superior of the learning-based algorithm is much more obvious.

Fig. 11 shows the average tolerance time per task, the rate of task offloading to each server, and the highest tolerance time in a time slot in different number-of-tasks settings using AODAI. Generally speaking, all metrics reduce when the number of vehicular edge servers increases. The only exception is the rising of the highest tolerance time in a time slot when the number of VS increases from 5 to 6 VS when the agent stops offloading to the cloud. In the case of 7 vehicular edge servers, the system's cloud's dependence is also zero, while the average tolerance time per task drops nearly 25 times compared to the case of 5 VS. This is because the agents prefer edge servers when waiting queues are not too "long." Even though the tolerance time is not thoroughly eliminated, offloading to edge

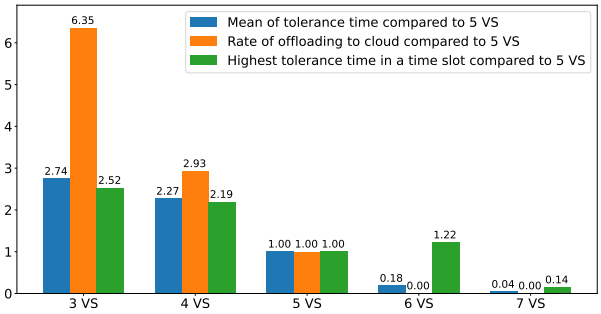


Fig. 11: Average tolerance time per task, rate of task offloading to each server, highest tolerance time in a time slot in different number-of-tasks settings using AODAI.

servers is still, most of the time, is still a better decision in these cases. When using only three vehicular edge servers, we can see that the mean tolerance time per task and the highest tolerance time in a time slot increase by 2.7 and 2.5 times, respectively. In addition, the number of offloading tasks to the cloud is 6.3 times higher because the number of vehicular edge servers cannot meet the computing demand and need to take advantage of more resources from the cloud.

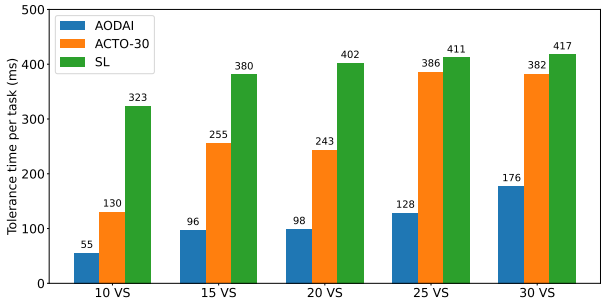


Fig. 12: Average tolerance time per task in different number-of-tasks-and-VSs settings using AODAI, AC-30, and SL.

If there is a substantial increase in the number of tasks, the service provider might opt to augment the vehicular edge server pool by hiring more smart vehicles. We configured the number of VSs as 10, 15, 20, 25, and 30 when the number of tasks per time slot increases to 1400, 2000, 2600, 3200, and 3800, respectively. Then, the performances of three different approaches in these scenarios are illustrated in Fig. 12. It is noteworthy that both reinforcement-learning approaches display a tendency towards an increased tolerance time in comparison to the SL algorithms. Among the five scenarios, AODAI always exhibits the highest performances, while SL always demonstrates the poorest ones. However, the gap between these three approaches gradually narrows over time, implying their scalabilities fall. In other words, reinforcement learning approaches do not scale well with large action spaces. ACTO-30 suffers from high variance since the policy is directly optimized rather than the value function. This challenges the policy-based to learn an optimal policy, especially when this environment is noisy or stochastic. AODAI is typically more stable and can provide more consistent performance since the pre-defined policy drives the agent gradually toward better strategies, and the volatility-handling techniques mitigate environmental noise. The open questions are, if in some task offloading problems, the reinforcement learning models when using must face a substantially large

actions space, what techniques will have the potential to exhibit satisfactory scaling performance, and is there a way to somehow modify the on-policy approaches to achieve the as stable or even better performance than the off-policy ones.

VII. Conclusion

To conclude, our framework can preserve the strength of vehicular edge computing while resorting to a powerful center cloud when needed. We proposed two algorithms; one follows value-based approach (*Advantage-Oriented Task Offloading with Dueling Actor-Insulator Network scheme*), and one follows policy-based approach (*Actor Critic based Task Offloading scheme*). Both can effectively reduce the latency, though they still face the problems of scalability when there are more VSs. AODAI, a reinforcement learning method that addresses attributes whose value is independent of all tasks to improve the learning process, reduces the tolerance time by at least 8.81% compared to other approaches. For future work, we will consider more targets, such as energy and cost, other than the time-quality baseline. Moreover, other reinforcement learning approaches will also be tested, along with other techniques, such as the Long Short-Term Memory network, to predict the density of task offloading for better offloading decisions. We also want to provide the intelligent agent with more human knowledge to cope with the problem.

Acknowledgements

This work was funded by Vingroup Joint Stock Company (Vingroup JSC), Viet Nam, Vingroup and supported by Vingroup Innovation Foundation (VINIF), Viet Nam under project code VINIF.2020.DA09.

References

[1] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, 2013.

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[3] J. Tan, W. Liu, T. Wang, M. Zhao, A. Liu, and S. Zhang, "A high-accurate content popularity prediction computational modeling for mobile edge computing using matrix completion technology," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 6, p. e3871, 2021.

[4] U. Saleem, Y. Liu, S. Jangsher, X. Tao, and Y. Li, "Latency minimization for d2d-enabled partial computation offloading in mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4472–4486, 2020.

[5] H. T. Thanh Binh, N. Phi Le, N. B. Minh, T. Thu Hai, N. Q. Minh, and D. Bao Son, "A reinforcement learning algorithm for resource provisioning in mobile edge computing network," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.

[6] 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.

[7] Z. Kuang, Z. Ma, Z. Li, and X. Deng, "Cooperative computation offloading and resource allocation for delay minimization in mobile edge computing," *Journal of Systems Architecture*, vol. 118, p. 102167, 2021.

[8] S. Chakraborty and K. Mazumdar, "Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing," *Journal of King Saud University-Computer and Information Sciences*, 2022.

[9] S. Abdelhamid, H. S. Hassanein, and G. Takahara, "Vehicle as a resource (vaar)," *IEEE Network*, vol. 29, no. 1, pp. 12–17, 2015.

[10] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile networks and applications*, vol. 26, no. 3, pp. 1145–1168, 2021.

[11] Z. Liu, X. Zhang, J. Zhang, D. Tang, and X. Tao, "Learning based fluctuation-aware computation offloading for vehicular edge computing system," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2020, pp. 1–7.

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

[13] B. Yılmaz, A. Ortiz, and A. Klein, "Delay minimization for edge computing with dynamic server computing capacity: A learning approach," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[14] G. Wang and F. Xu, "Regional intelligent resource allocation in mobile edge computing based vehicular network," *IEEE Access*, vol. 8, pp. 7173–7182, 2020.

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

[16] J. Zhang, H. Guo, and J. Liu, "A reinforcement learning based task offloading scheme for vehicular edge computing network," in *International conference on artificial intelligence for communications and networks*. Springer, 2019, pp. 438–449.

[17] X.-Q. Pham, T.-D. Nguyen, V. Nguyen, and E.-N. Huh, "Joint node selection and resource allocation for task offloading in scalable vehicle-assisted multi-access edge computing," *Symmetry*, vol. 11, no. 1, p. 58, 2019.

[18] D. B. Son, T. H. Binh, H. K. Vo, B. M. Nguyen, H. T. T. Binh, and S. Yu, "Value-based reinforcement learning approaches for task offloading in delay constrained vehicular edge computing," *Engineering Applications of Artificial Intelligence*, vol. 113, p. 104898, 2022.

[19] X. Hu, L. Wang, K.-K. Wong, M. Tao, Y. Zhang, and Z. Zheng, "Edge and central cloud computing: A perfect pairing for high energy efficiency and low-latency," *IEEE Transactions on Wireless Communications*, vol. 19, no. 2, pp. 1070–1083, 2019.

[20] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, 2019.

[21] C. Sun, L. Hui, X. Li, J. We, Q. Xiongl, X. Wang, and V. C. Leun, "Task offloading for end-edge-cloud orchestrated computing in mobile networks," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2020, pp. 1–6.

[22] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, S. Meng, and L. Qi, "A computation offloading method over big data for iot-enabled cloud-edge computing," *Future Generation Computer Systems*, vol. 95, pp. 522–533, 2019.

[23] R. Chataut and R. Akl, "Massive mimo systems for 5g and beyond networks—overview, recent trends, challenges, and future research direction," *Sensors*, vol. 20, no. 10, p. 2753, 2020.

[24] V. T. An, T. T. Hai, B. M. Nguyen, N. P. Le, H. T. T. Binh *et al.*, "Fuzzy deep q-learning task offloading in delay constrained vehicular fog computing," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[25] F. Dai, G. Liu, Q. Mo, W. Xu, and B. Huang, "Task offloading for vehicular edge computing with edge-cloud cooperation," *World Wide Web*, pp. 1–19, 2022.

[26] Y. Ouyang, "Task offloading algorithm of vehicle edge computing environment based on dueling-dqn," in *Journal of Physics: Conference Series*, vol. 1873, no. 1. IOP Publishing, 2021, p. 012046.

[27] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 3, pp. 881–892, 2021.

[28] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, 2019.

[29] R. O. Aburukba, M. AliKarrar, T. Landolsi, and K. El-Fakih, "Scheduling internet of things requests to minimize latency in hybrid fog-cloud computing," *Future Generation Computer Systems*, vol. 111, pp. 539–551, 2020.

[30] Y. Lu, L. Liu, J. Gu, J. Panneerselvam, and B. Yuan, "Ea-dfpso: An intelligent energy-efficient scheduling algorithm for mobile edge networks," *Digital Communications and Networks*, vol. 8, no. 3, pp. 237–246, 2022.

[31] G. Li, Y. Liu, J. Wu, D. Lin, and S. Zhao, "Methods of resource scheduling based on optimized fuzzy clustering in fog computing," *Sensors*, vol. 19, no. 9, p. 2122, 2019.

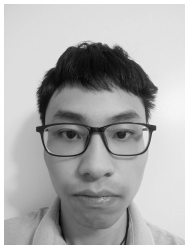
[32] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[33] S. Vemireddy and R. R. Rout, "Fuzzy reinforcement learning for energy efficient task offloading in vehicular fog computing," *Computer Networks*, vol. 199, p. 108463, 2021.

[34] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

TA HUU BINH (ORCID: 0000-0003-3553-5833) studies Computer Science (Talented Programme) at the School of Information and Communication Technology, Hanoi University of Science and Technology. He started to engage in academic research at the Modelling, Simulation, and Optimization Laboratory in 2020. His research interests include cloud/fog computing, computational intelligence, and reinforcement learning.

DO BAO SON (ORCID: 0000-0001-6441-9546) received his Bachelor's degree in Hanoi National University of Education (Vietnam) in 2013, and his Master's degree in Computer Science at the School of Information and Communication Technology (SoICT), Hanoi University of Science and Technology (Vietnam) in 2016. In 2013, he won the Fourth prize at the National Science Research Conference for Students, Ministry of Education and Training, Vietnam. Currently, he is a Ph.D. candidate in SoICT. His current research interests include cloud/fog computing, computational intelligence, and reinforcement learning.



HIEP VO (ORCID: 0000-0002-2709-2924) received his Bachelor's Degree in Data Science from Luther College (United States of America) in 2021. Currently, he is working on his Ph.D. degree in Computer Science from the School of Computer Science, University of Technology Sydney (UTS) in Australia. His research interests include security-preserving distributed systems and deep learning implementations for several application domains, including federated learning, generative adversarial networks, and cloud/fog computing.

BINH MINH NGUYEN (ORCID: 0000-0003-1328-3647) received his Dipl. Ing. degree in Computer-Aided Design from the Institute of Automation and Information Technologies, Tambov State Technical University (Russia) in 2008, and his Ph.D. degree in Applied Informatics from the Faculty of Informatics and Information Technology, Slovak University of Technology (STU) in Bratislava (Slovakia) in 2013. From 2008 to 2013, he worked as a researcher at the Institute of Informatics, Slovak Academy of Sciences (IISAS), Slovakia. Currently, he is an associate professor at the School of Information and Communication Technology (SoICT), Hanoi University of Science and Technology (Vietnam). His research interests include distributed systems and data analytics for several application domains, including cloud/fog computing and blockchain.

HUYNH THI THANH BINH (ORCID: 0000-0003-1976-6113) is an Associate Professor and the Vice Dean of the School of Information and Communication Technology, Hanoi University of Science and Technology where she is the Head of Modeling, Simulation and Optimization Lab. Her current research interests include – evolutionary computation, computational intelligence, evolutionary multitasking. Dr. Binh is an Associate Editor of the Engineering Applications of Artificial Intelligence Journal, IEEE Transactions on Emerging Topics in Computational Intelligence. She has served as a regular reviewer, a program committee member of numerous prestigious academic journals and conferences such as Applied Soft Computing, Memetic Computing, IEEE Access, IEEE Congress on Evolutionary Computation, Swarm and Evolutionary Computation. . . She is a member of IEEE Computational Intelligence Society – Women in Computational Intelligence Committee; the Chair of IEEE Computational Intelligence Society Vietnam Chapter; the IEEE Asia Pacific Executive Committee member, and the IEEE Asia Pacific Student Activities Committee Chair (2019, 2020).