AI Empowered Computing Resource Allocation in Vehicular Ad-hoc NETworks

Ayat Hama Saleh
Dept. of Electrical, Computer and Biomedical Engineering
Ryerson University
Toronto, Canada
ahamasaleh@ryerson.ca

Alagan Anpalagan
Dept. of Electrical, Computer and Biomedical Engineering
Ryerson University
Toronto, Canada
alagan@ee.ryerson.ca

2022 7th International Conference on Business and Industrial Research (ICBIR) | 978-1-6654-9475-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICBIR54589.2022.9786437

***Abstract*—The vehicular ad hoc network (VANET) has emerged as a heterogeneous network with no fixed infrastructure. This paper proposes an AI-empowered task offloading and computing resource allocation model which can manage the computing resources in VANET dynamically. The model is divided into two layers. First, the task offloading layer, where the Random Forest (RF) algorithm is used to determine offloading the vehicle's computing tasks whether to the Cloud Computing (CC) server or Mobile Edge Computing (MEC) server or to be processed locally (in-vehicle computing). Second, the resource allocation layer, where the Deep Deterministic Policy Gradient (DDPG) algorithm is used to determine the computing platform again when the task is determined to be offloaded to either MEC servers or the cloud servers. To evaluate the performance of the RF classifier, we applied the model to a real-world driving trajectory dataset, and then compared the results with a different set of Machine Learning (ML) algorithms namely, K-nearest neighbour (KNN), Multilayer Perceptron (MLP) and Support Vector Machine (SVM). The results show the RF model outperformed other models in classification accuracy score of 99.83% for task offloading decision, where the KNN, MLP and SVM achieved 98%, 94.81% and 90.94%, respectively. Moreover, the DDPG based resource allocation scheme converges within 150 episodes and reduced the latency cost by 85%.**

***Keywords*—Resource allocation, VANETs, machine learning, reinforcement learning, deep deterministic policy gradient**

I. INTRODUCTION

The modern vehicles are equipped with sensors and can communicate with one another as well as with the infrastructure around them, referred to as Vehicular Ad hoc Networks (VANET) or the Internet of Vehicles (IoV). The vehicles on the road are considered as nodes that communicate in-vehicle to- everything (V2X) paradigm in which vehicles can communicate with each other. This type of communication is called Vehicle-to-Vehicle communication (V2V) and the one that communicates with the infrastructure such as Road-Side Units (RSU) and Base Stations (BS), which is called vehicle-to-infrastructure communication (V2I). In such an environment, heterogeneous data are generated such as text, audio and video, which require robust, low latency, reliable and mobility-aware resource management solutions. Hence, efficient resource management in such high mobility, large scalability, and intermittent wireless connection environment is a challenge.

Artificial Intelligence (AI) can handle sparse and heterogeneous data which makes integrating AI techniques with the V2X paradigm a promising solution for unconventional applications such as real-time traffic flow prediction and resource management in VANETs. Recently, the application of AI for resource management in vehicular networks has drawn significant attention from the research community, many researchers have investigated AI-empowered resource management by implementing different approaches and techniques to provide robust and efficient optimization solutions that cannot be handled or even modelled in the traditional optimization framework, especially Reinforcement Learning (RL) models such as Q-learning [1], actor-critic [2], and other Deep Reinforcement Learning (DRL) algorithms have been vastly employed for resource management in wireless communication networks.

Inspired by the published works [1], [3] and taking into account the dynamic vehicular network characteristics such as high vehicle mobility and heterogeneous applications, we investigate how to exploit AI models to control the computing resources to support delay-sensitive applications in the distributed computing platform-based vehicular network.

The rest of the paper is organized as follows. In Section II, system model is described. In Section III AI-based computing resource allocation solutions are presented. Section IV presents the experiments conducted on all developed models and some results. Finally, section VI concludes the article.

II. SYSTEM MODEL

A. Vehicular Network Architecture

Inspired by the network model proposed in [1], we consider a heterogeneous vehicular computing architecture where three distributed computing platforms are utilized; Cloud Computing (CC), Mobile Edge Computing (MEC), and in-vehicle computing as shown in Figure 1. CC consists of data centers with dedicated servers that have high-capacity computing resources with spectrum resources and storage space to process large amounts of data for different in-vehicle applications such as video streaming, augmented reality, and automatic driving, all task computing, intelligence decision, and analysis happen at the central cloud. Furthermore, it is utilized as a backup for the roadside and the vehicular cloud layers. The central cloud's major drawback is the long-distance data transmission. A large amount of data processing and a long transmission distance cannot guarantee low latency in vehicular networks. Therefore, using cloud computing only for computation and data processing is not quite a feasible and cost-effective option. The MEC servers are installed at RSUs and BSs to provide resources storage and computation to vehicles via V2I communication protocol which improves the delay, computing power, and server utilization. While, in-vehicle computing, represented by the vehicles which are equipped with strong computation units for data processing and storage, the vehicles can directly process tasks and some long-term analysis and long-latency computing are transferred to the RSUs/BS or the central cloud [4].

B. Communication Model

In the architecture, mobile vehicles are generating computation tasks and requesting content with a certain speed and direction. They are deployed with a ML algorithm to determine whether tasks should be computed at the cloud

server, MEC server or processed locally based on the computation task size (i.e., image, video, control status, text, navigation system). The RSUs act as edge servers that manage vehicles resources and capable to provide communication, computing, and caching resources with low latency and low network traffic.

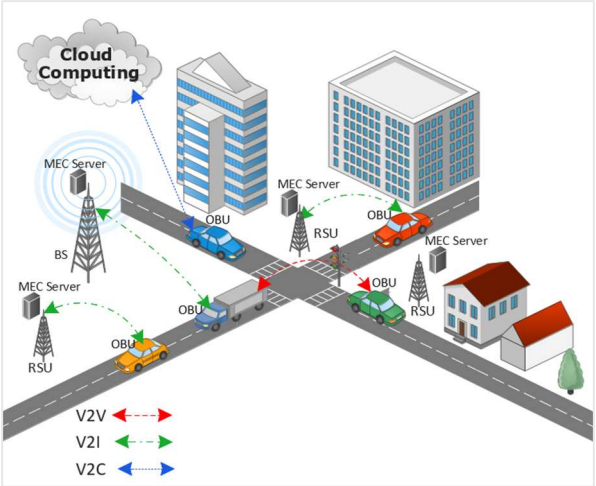

Fig. 1. Vehicular Ad hoc Networks (VANET) Architecture

Comparing with the RSUs, the BS is deployed with high-capacity computing servers and storage space to process large amounts of data for different in-vehicle applications such as navigation, video streaming, and smart traffic lights. RSUs and vehicles are located within the coverage area of an existing BS that serves as a backup for all RSUs. The cloud servers processing the tasks require high computational complexity and are not sensitive to latency. In the proposed model, the set of all RSUs and vehicles is defined as $K = \{1, \ldots . , k\}$ and $N = \{1, \ldots . n\}$. The set of MEC servers is denoted as $M = \{1, \ldots , m\}$. Considering vehicles heterogeneity, each vehicle requires different computing resources. We presume that each vehicle n has a computing task to process. Tasks can be offloaded to the cloud server, MEC server or to be executed locally. The MEC server has bounded capacity and may not be enough for all vehicles' offloaded tasks. We denote B the wireless channel spectral bandwidth. If multiple vehicles' tasks are offloaded concurrently, the wireless bandwidth is distributed equally to the vehicle to upload data. Considering the wireless channel between the vehicles and RSU/BS is a real time-differing channel, according to [5], the vehicle n obtainable upload data rate is

$$R_n = \frac{B}{N}\log\left(1 + \frac{P_n g_n}{\frac{B}{N}N_0}\right), \quad (1)$$

where N is the number of tasks, P_n is the transmission power of vehicle n uploading data, g_n is wireless channel gain of vehicle n, and the variance of complex white Gaussian channel noise is denoted as N_0 [6][1].

C. Computation Model

We consider that each vehicle n has a computing task denoted as $T_n = (S_n, U_n, \tau_n)$. The task T_n can be executed in the vehicle's central processing unit (CPU) or offloaded to either MEC server or CC servers for processing. S_n denotes the size of the input data required to calculate T_n. U_n denotes the total required CPU cycles to complete the calculation task T_n. The size of U_n is equal whether the task is computed in vehicle or in the MEC server or in the cloud. The acceptable delay denoted τ_n. Transmission time in a wired link is relatively small between RSU and BS due to the high and stable data rate acquired by the wired connection and is neglected in this paper.

1) In-Vehicle Computing

Based on the AI classifier embedded in-vehicle decision if vehicle n decides to process task T_n using the vehicle CPUsτ_n^v is defined as the in-vehicle task processing delay of vehicle n. It contains solely the local CPU processing delay. The computation capacity is denoted as f_n^v whereas the capacity is the number of CPU cycles per second. The vehicles can have different computational capacity due to the diversity present in the vehicular ad hoc environment. The local execution delay τ_n^v of task T_n is

$$\tau_n^v = \frac{U_n}{f_n^v}. \quad (2)$$

Taking into account the time consumption, the total cost of in-vehicle computing can be stated as

$$C_n^v = \tau_n^v. \quad (3)$$

2) Mobile Edge Computing

In the scenario when vehicle n decides to process task T_n at the MEC server, the vehicle n will upload the data of the task T_n to the nearest RSU/BS via the wireless access point distributed in the road, and the RSU/BS forwards the data to the MEC server. Then, the MEC server assigns part of the computing resources to process the computing task, lastly, the MEC server returns the execution result to the vehicle n. The computation capacity of the MEC server (i.e., number of CPU cycles per second) is defined as f_n^m. Based on the overhead steps, the transmission delay of vehicle n is defined as $\tau_{n,t}^m$

$$\tau_{n,t}^m = \frac{S_n}{R_n}, \quad (4)$$

here R_n is the upload data rate in the wireless channel for the proposed model, $\tau_{n,p}^m$ is the MEC server processing delay to execute the task T_n

$$\tau_{n,p}^m = \frac{U_n}{f_n^m}. \quad (5)$$

and $\tau_{n,d}^m$ is the processing outcomes download delay

$$\tau_{n,d}^m = \frac{S_n}{R_d}. \quad (6)$$

R_d is the download data rate of vehicle n, given the fact that the download data rate is very high and the resulting data size is considerably smaller in comparison to the input data size [7], thus, the delay of download data rate is neglected in the rest of the paper. Therefore, the MEC task processing total cost can be given as C_n^m

$$C_n^m \approx \tau_{n,t}^m + \tau_{n,p}^m. \quad (7)$$

3) Cloud Computing

In the scenario when vehicle n chooses to process task T_n at the cloud server, the vehicle n will upload the data of the task T_n to the nearest RSU/BS via the wireless access point, then the RSU/BS forwards the data to the cloud server, and the cloud server assigns the resources to process the computing task T_n, lastly, the cloud server forwards the execution result

to vehicle n. The computation capacity of the cloud server (i.e., number of CPU cycles per second) is defined as f_n^c. Based on the overhead steps, the transmission delay of vehicle n is defines as $\tau_{n,t}^c$

$$\tau_{n,t}^c = \frac{S_n}{R_n} \ , \tag{8}$$

and $\tau_{n,p}^c$ is the cloud server processing delay to execute the task T_n

$$\tau_{n,p}^c = \frac{U_n}{f_n^c} \ , \tag{9}$$

and $\tau_{n,d}^m$ is the processing results download delay, the download data rate R_d is very high, the data size of a task processed by the cloud computing server is typically very large, therefore the executed data will be also very large, thus, the delay of download data rate in cloud computing is considered. Therefore, the total cost of cloud server task processing can be given as C_n^c

$$C_n^c = \tau_{n,t}^c + \tau_{n,p}^c + \tau_{n,d}^c \ , \tag{10}$$

and the whole system total cost will be given as

$$C_T = \sum_{n=1}^{N} a_n^v C_n^v + a_n^m C_n^m + a_n^c C_n^c \ . \tag{11}$$

III. PROBLEM FORMULATION

Our goal is to reduce the vehicles total cost of delay and efficiently allocate the resources with satisfied QoS requirements to support multiple tasks offloading requests simultaneously. To achieve this goal, the problem is formulated as follows,

$$\min_{a,f} \sum_{n=1}^{N} a_n^v C_n^v + a_n^m C_n^m + a_n^c C_n^c$$

$$s.t. \begin{cases} H1: a_n^v, a_n^m, a_n^c \ , \ a_n^v + a_n^m + a_n^c = 1 & H1 \in (0,1) \\ H2: a_n^v \tau_n^v, a_n^m \tau_n^m, a_n^c \tau_n^c \ \leq \tau_n & \forall n \in N \\ H3: 0 \leq f_n^m \leq a_n^m f & \forall n \in N \\ H4: \sum_{n=1}^{N} a_n^m f_n^m \leq f & \forall n \in N \end{cases} \tag{12}$$

Here, A is the offloading decision vector expressed as $A = [a_1, a_2, a_3, \ldots\ldots, a_N]$. f is the computing resource allocation vector expressed as $f = [f_1, f_2, f_3, \ldots\ldots, f_N]$. $H1$ means that each vehicle n can offload tasks to the vehicle's local central processing unit (CPU) or MEC server or CC servers for computing. $H2$ implies that the time cost should be less than the maximum permitted delay. $H3$ implies that the computing resource allocated for the vehicle n cannot surpass the total resource f of the MEC server. $H4$ implies that the sum of the computing resources allocated to the vehicle n cannot overreach the total resources f of the MEC server.

To guarantee the QoS requirements of an offloaded task, determining the optimal offload decision vector A and the computational resource allocation f should be simultaneously satisfied, but since A is a binary vector, the feasible set of the problem (12) and the objective function is not convex. Moreover, if the number of offloaded tasks increments, the size of problem (12) will increment promptly, traditional optimization methods are infeasible to solve this non-convex problem. Hence, we exploit a DRL method to acquire an optimal resource allocation decision for delay-sensitive tasks in VANET dynamic environment.

IV. PROPOSED SOLUTION

A. Task Distribution Layer

Determining the computation platform for task offloading is a multi-label classification problem, taking into account the data size with the delay requirement of each task, the Random Forest (RF) algorithm is used to classify the tasks into three categories CC server, MEC server, and in-vehicle computing. The RF algorithm is utilized to decide the best platform for task offloading. RF is an ensemble learning technique that creates multiple decision trees in the data set training time before delivering the result. For classification tasks, it makes decisions based on the majority voting system which incorporates the output of multiple decision trees to deliver a single result.

For evaluating the performance of RF classifier, we applied the model and other ML algorithms KNN, MLP and SVM to a real-world driving trajectory dataset, then compared the classification results. The task offloading classification task is performed on the T-Drive trajectory dataset [8] available publicly on the Microsoft website. T-drive is a smart driving direction service based on GPS trajectories of many taxis. This dataset includes the GPS trajectories of 10,357 taxis during one week in Beijing. The total number of points in this dataset is about 15 million and the total distance of the trajectories reaches 9 million kilometres. Each text file of this dataset, which is named by the taxi ID, contains the trajectories of one taxi, the attributes in each file are the taxi, date time, longitude, and latitude. A total of 10,357 text files with the size of each file in Kbits are composed into a CSV file for the multi-class classification task. The "label" field refers to the true label of the predicted class. Class "0" means the computing task to be processed by the vehicle's CPUs, class "1" means to be offloaded to the MEC servers and class "2" is to be offloaded to the cloud servers. The ML models applied to this dataset make the offloading decision based on the values under the "Size" field of the dataset.

B. Deep RL Based Resource Allocation Layer

When the classifier determines to offload the task to either MEC or cloud servers, the resource allocation is considered a nonconvex optimization problem. To solve this problem, first we model the resource allocation decision making problem as Markov Decision Processes (MDPs), and then utilize a DRL method to select the computing platform again and perform the resource allocation problem of the computing resources. Thus, the data processing and computation are divided into two parts: the first part is processed by the RSU or BS, and the second part is processed by the cloud center when RSUs/BSs cannot support computation requirements of vehicles, such large computation tasks can be offloaded to the cloud server. RL is a semi-supervised learning algorithm where unlabelled data is given as input for the algorithm to learn the feature representation of the data. DRL refers to the contributions of Deep Neural Networks (DNNs) to RL algorithms where an agent learns by interacting with the environment to increase the cumulative reward value of system behavior. Using the DRL method can accommodate the vehicular networks environment and be the best resource allocation solution. In our proposed model, vehicles send their traveling state, location, and task information $T_n = (S_n, U_n, \tau_n)$ to the MEC server, the RSU/BS is responsible for collecting the MEC

server status, manage the spectrum, and computing resources among vehicles with task offloading requests, and combining the total information into an environment state. The RSU/BS then sends the combined environment state to the agent, the agent receive feedback on the optimal policy π for resources allocation decisions for a particular vehicle to maximize the total accumulated reward. The system state, action, and reward functions are defined as follows.

1) State

The state in DRL describes the current situation of the vehicular environment. Each vehicle periodically sends the moving state and task information to the RSU/BS. The state consists of three components $S = (K_n, C_j, B_j)$. K_n represents the state of vehicle n, that combines the computation task size, the required computation resources, acceptable time delay, popularity of the requested content, and locomotion parameters. C_j, B_j are the available computation resources, and bandwidth of MEC server j allocates to vehicle n respectively.

2) Action

The objective of an agent is to learn an optimal policy that maximizes the reward by mapping the environment space of states to the space of actions. In this system, the agent will make resource allocation decisions based on the resource allocation policy π. The action taken by the MEC server at RSU/BS can be described as the offloading decision vector $A = [a_1, a_2, a_3, \ldots\ldots, a_N]$ and the computing resource allocation vector $f = [f_1, f_2, f_3, \ldots\ldots, f_N]$ which includes the MEC server computation resources, and bandwidth allocates to vehicle n.

3) Reward

The agent receives a reward r_i from the environment according to the current state s_i and action a_i, then based on the received reward the agent updates the policy π for resource allocation till algorithm converge. The objective of DRL is to obtain the maximum reward, while the reward is inversely related to the system total cost; therefore, the reward element for offloaded T_n at time slot t defined as $r_{i(t)} = \frac{C_v - C_T}{C_v}$ [1]. Here C_v is the total cost for in-vehicle computation, and C_T is the total system cost.

C. DDPG BASED Resource Allocation Solution

In the non-local task computing stage, the environment state changes constantly because of the infinite channel states and dynamic task offloading requests, which make the agent unable to make resource allocation decisions in subsequent time slot based on the current observed environment state. Therefore, the Deep Deterministic Policy Gradient (DDPG) algorithm is utilized to design an efficient resource allocation scheme. In this stage, the agent is executed by the MEC server installed at each RSU/BS as well as by the cloud server to support the vehicular environment. Vehicles can offload computing tasks through wireless access points and obtain the computing resources located at the MEC server/cloud centre by cellular or Wi-Fi/DSRC technologies. DDPG is a model-free off-policy algorithm coheres the benefits of both the policy gradient and Deep Q-network (DQN) algorithms, it's mainly structured of two components, actor and critic networks. Both networks take on soft updated parameters target networks to attain durable convergence [9]. In DDPG, the objective of the agent is to find an optimal resource allocation policy π that attain maximum long-term cumulative reward. The DDPG is a powerful algorithm compared to other DRL algorithms, due to its capability to make decisions or allocations in continuous action space.

The DDPG consists of three functional components namely, main network, target network and replay memory. The main network parameters updated in real time, and it comprises of two DNNs, namely, actor-network $\mu(s_i|\theta^\mu)$ used to explore the policy π, and critic network $Q(s_i, a_i|\theta^Q)$ used to estimate the performance and provides the Q-value. Therefore, the main network input is the current state s of the vehicular environment, and the main network output of is the action a that is adopted by the RSU/BS. Where the target network with soft updated parameters is employed to produce the target value for training the main critic network, it includes a target actor network $\mu'(s_i|\theta^{\mu'})$ and a target critic network $Q'(s_i, a_i|\theta^{Q'})$, the input of the target network is the next state s' from replay memory and the output is a Q-value for training the main critic network $Q(s_i, a_i|\theta^Q)$. The reply memory stores experience tuples which include the current state s_i, the selected action a_i, reward $r(s_i, a_i)$, and next state s_{i+1}. Let $\theta^\mu, \theta^{\mu'}, \theta^Q, \theta^{Q'}$ be the parameter matrices of the main network and target network in the actor and the critic networks respectively. Then the parameters of the target networks are soft updated with that of the main networks as follows,

$$\theta^{\mu'} \leftarrow \beta\, \theta^{\mu} + (1-\beta)\, \theta^{\mu'} \ , \tag{13}$$

$$\theta^{Q'} \leftarrow \beta\, \theta^{Q} + (1-\beta)\, \theta^{Q'} \ , \tag{14}$$

with $\beta \ll 1$. This means that the target values are restrained to change gradually, significantly enhancing the solidity of learning [10].

The DDPG functionality can be explained in three steps. First, the state of the vehicular environment is sent to the main actor-network and replay memory. Second, based on the current state and experience tuples the agent uses the main network and target network to determine the next action a_{i+1}. In this step, target actor $\mu'(s_i|\theta^{\mu'})$ and target critic $Q'(s_i, a_i|\theta^{Q'})$, update target policy value and target Q-value based on experience tuples from replay memory. Then, target critic $Q'(s_i, a_i|\theta^{Q'})$, in target network sets y_i as follows:

$$y_i = r_i + \gamma\, Q'\left(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}\right), \tag{15}$$

with discounting factor $\gamma \in [0,1]$ and transmits it to the critic network $Q(s_i, a_i|\theta^Q)$ in the main network. After receiving y_i, the critic network $Q(s_i, a_i|\theta^Q)$ in the main network updates Q-value θ^Q by solving an optimization problem which aims to minimize the loss function:

$$Ls(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \ , \tag{16}$$

Based on θ^Q and experience tuples, the actor network μ in main network is updated using the sampled policy gradient π and then generates next action.

$$\nabla_{\theta\mu} J \approx \frac{1}{N}\sum \nabla_a \ Q(s, a|\theta^Q)|_{\,s=si,\ a=\mu(si)} \nabla_{\theta\mu}\, \mu(s|\theta^\mu)|_{si} \ . \tag{17}$$

In the last step, based on the resource allocation decision, the MEC/cloud server executes task computing. The agent will

receive a reward, then, the state of the environment and system reward are updated based on current resource allocation.

One of the major challenges of learning in continuous action space such as vehicular environment is exploration. In DDPG this problem can be solved separately from the learning algorithm. Following the original DDPG in [10], we modelled an exploration policy μ' by adding noise sampled from a noise process N_{noise} to the actor policy $\mu'(s_t) = \mu(s_t|\theta^{\mu}) + N_{noise}$. The entire process is shown in Figure 2. The detail of the DDPG process is shown in Algorithm 1.

Algorithm 1. The DDPG-based algorithm

/*Initialization phase */

Initialize the main actor $\mu(s|\theta^{\mu})$ and critic $Q(s,a|\theta^{Q})$ networks with random parameters

θ^{μ} and θ^{Q}.

Initialize the target actor μ' and critic Q' networks with random parameters

$\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$.

Initialize the replay memory buffer R.

/* Parameter updating phase */

For episode=1, M **do**

Initialize a random process N_{noise} for action exploration

Receive initial observation state s_0

For step t=1, T **do**

Make an action $a_t = \mu(s_t|\theta^{\mu}) + N_t$ according to the current policy and exploration noise

Receive the new state s_{0+1} and observe the reward r_t

Store the experience (s_t, a_t, r_t, s_{t+1}) in R

Randomly sample a mini batch of experiences (s_i, a_i, r_i, s_{i+1}) from R

Set $y_i = r_i + \gamma\, Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

Update the critic network by minimizing the loss:

$$Ls = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^{Q}))^2$$

Update the actor network using the sampled policy gradient:

$$\nabla_{\theta\mu} J \approx \frac{1}{N}\sum \nabla_a Q(s,a|\theta^{Q})|_{s=si,\ a=\mu(si)} \nabla_{\theta\mu}\, \mu(s|\theta^{\mu})|_{si}$$

Update the critic and actor target networks:

$$\theta^{Q'} \leftarrow \beta\, \theta^{Q} + (1-\beta)\, \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \beta\, \theta^{\mu} + (1-\beta)\, \theta^{\mu'}$$

end for
end for

V. SIMULATION RESULTS AND ANALYSIS

In this section, the performance evaluation of the proposed model is presented. First, the performance of the ML algorithms is evaluated by measuring the accuracy score. The dataset split into 70% training and 30% testing data to evaluate the performance of the algorithms. For training 7249 samples are used and 3108 samples for testing. The RF algorithm is the best performing with an accuracy of 99.83%. The SVM performs poorly as compared to all other algorithms with an accuracy of 90.94 %. The accuracy score of the ML algorithms is summarized in Table I.

TABLE I ACCURACY SCORE OF ML ALGORITHMS

Training data	Testing data	RF	KNN	MLP	SVM
70%	30%	99.83%	98%	94.81%	90.94%

The DDPG architecture implemented in two parts, the environment, and the agent. In the environment, there are one BS and three RSUs deployed with a MEC server located along a two-lane one-way country road. The computation tasks size ranged from 1 Kbits to 10 Gigabits. The computation resources of the vehicle's CPUs, MEC and cloud servers are assumed to be 1 GHz, 200 GHz, and 1000 GHz, respectively. The vehicles are randomly distributed within the coverage of the RSUs/BS. Python and TensorFlow are used to implement the agent. Figure 3 shows the performance of the DDPG algorithm using different learning rates of 0.1, 0.01, 0.001, and 0.0001 for the DNNs in both actor and critic networks. We can observe that the reward is very low at the beginning of the learning process. With the number of episodes, the algorithm converges and the reward increases. The reason is that, at the beginning of the learning process, because of lacking historical information, DDPG is easy to explore an inefficient resource allocation strategy. After training for a while, DDPG can explore a better resource allocation strategy faster and easier based on the learned information. The convergence of the model implies that the agent has learnt the best resource allocation strategy. To test the performance of the DDPG algorithm, we trained the model under different learning rates. When the learning rate is 0.1 the DDPG does not converge and only achieves a maximum average reward of -1227 over the 250 episodes. The DDPG converges when the learning rate is 0.01, 0.001 and 0.0001 with a maximum accumulative reward of -179, -224.7, and -229.5 respectively after running 150 episodes. However, the algorithm performance of learning rate 0.01 is better than the performance of learning rate 0.001 and 0.0001. Thus, 0.01 is the best learning rate for the proposed technique. The convergence of the DDPG algorithm implies that the agent has learnt the best resource allocation strategy that satisfies the QoS requirement and allocates resources quickly and efficiently. The DDPG training parameters are shown in Table II.

TABLE II PARAMETERS OF DDPG ALGORITHM

Parameter	Value
Data size of computing task	1 Kbits- 10 GB
Number of CPU cycles required to execute a computing task	[1- 100000] Mega cycles/sec
Computing resources at vehicle's CPU	1 GHz/sec
Computing resources at MEC	200 GHz/sec
Computing resources at the cloud server	1000 GHz/sec
Size of each mini batch of transitions	64
Discount factor on reward	0.99
Learning rate of the DNNs in actor/critic networks	[0.0001, 0.001, 0.01, 0.1]
Replay memory size R	1000000
N noise	0.1
B	0.002
Dense of DNNs in actor/critic networks	512

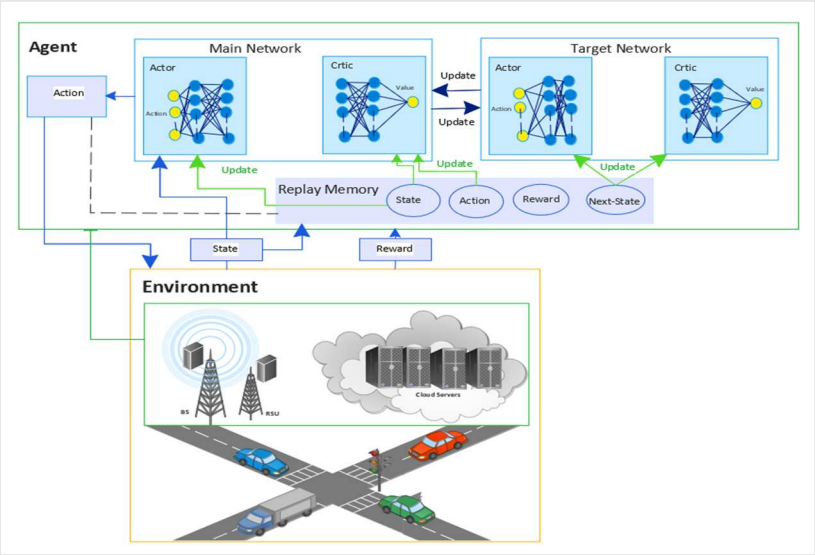

Fig. 2. The DDPG Process in MEC and CC Vehicular Resource Management

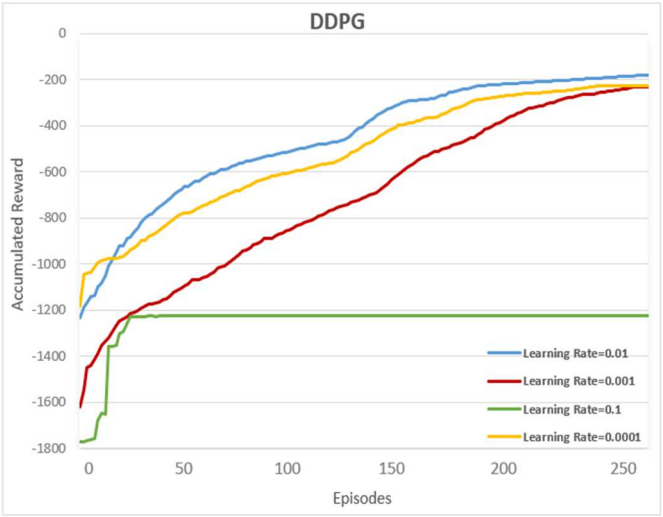

Fig. 3. DDPG Algorithm Total Reward of Each Episode

VI. CONCLUSION

In this paper, an AI-empowered offloading and resource allocation model is proposed that can dynamically manage the computing resources considering multiple computing platforms the CC, MEC and local computing to choose the best platform for task offloading and improve the performance of vehicular networks. Unlike some of the works, the ML models developed for this work are evaluated using real traffic data. The optimal offload decision and the computational resource allocation are simultaneously fulfilled to ensure the QoS requirements of an offloaded task using RF and DDPG algorithms. Numerical results are shown to support the effectiveness of the proposed technique. The results showed that using RF and DDPG algorithms for task offloading and resource allocation can reduce the latency of the system, the RF algorithm can accurately classify the distribution of task offloading and the DDPG converges within fewer training episodes to learn the best resource allocation strategy, reduce the latency cost and save system average cost by 85% compared to the published papers. For future work, different AI models can be exploited to optimize the spectrum, computing, and cashing resources jointly that support delay-sensitive applications in VANET.

REFERENCES

[1] Y. Cui, Y. Liang, and R. Wang, “Resource Allocation Algorithm with Multi-Platform Intelligent Offloading in D2D-Enabled Vehicular Networks,” *IEEE Access*, vol. 7, pp. 21246–21253, 2019.

[2] X. Jiang, et al, “Intelligent Resource Allocation for Video Analytics in Blockchain-Enabled Internet of Autonomous Vehicles with Edge Computing,” *IEEE Internet Things J.*, vol. 4662, no. c, pp. 1–1, 2020.

[3] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang, “Artificial Intelligence Empowered Edge Computing and Caching for Internet of Vehicles,” *IEEE Wirel. Commun.*, vol. 26, no. 3, pp. 12–18, 2019.

[4] Z. Lv, R. Lou, and A. K. Singh, “AI Empowered Communication Systems for Intelligent Transportation Systems,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4579–4587, 2021.

[5] P. Zhao, et al. “Energy-Saving Offloading by Jointly Allocating Radio and Computational Resources for Mobile Edge Computing,” *IEEE access*, vol. 5, pp. 11255–11268, 2017.

[6] R. Hussain, *et al.* “Rethinking Vehicular Communications: Merging VANET with cloud computing,” in *4th IEEE International Conference on Cloud Computing Technology and Science*, pp. 606–609, 2012.

[7] X. Chen, X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing,” *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2015.

[8] Y. Zheng, “T-Drive trajectory data sample.” Aug. 2011, [Online]. Available: https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/.

[9] Y. Ren, et al., “Vehicular Network Edge Intelligent Management : A Deep Deterministic Policy Gradient Approach for Service Offloading Decision,” *IWCMC 2020*, pp. 905–910, 2020.

[10] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, 2016.