DeepReserve: Dynamic Edge Server Reservation for Connected Vehicles with Deep Reinforcement Learning

Jiawei Zhang, Suhong Chen, Xudong Wang, and Yifei Zhu
UM-SJTU Joint Institute, Shanghai Jiao Tong University, Shanghai, China
Email:{jiaweizhang, sh-chen, wxudong, yifei.zhu}@sjtu.edu.cn

***Abstract*—Edge computing is promising to provide computational resources for connected vehicles. Resource demands for edge servers vary due to vehicle mobility. It is then challenging to reserve edge servers to meet variable demands. Existing schemes rely on statistical information of resource demands to determine edge server reservation. They are infeasible in practice, since the reservation based on statistics cannot adapt to time-varying demands. In this paper, a spatio-temporal reinforcement learning scheme called DeepReserve is developed to learn variable demands and then reserve edge servers accordingly. DeepReserve is adapted from the deep deterministic policy gradient algorithm with two major enhancements. First, by observing that the spatio-temporal correlation in vehicle traffic leads to the same property in resource demands of CVs, a convolutional LSTM network is employed to encode resource demands observed by edge servers for inference of future demands. Second, an action amender is designed to make sure an action does not violate spatio-temporal correlation. We also design a new training method, i.e., DR-Train, to stabilize the training procedure. DeepReserve is evaluated via experiments based on real-world datasets. Results show it achieves better performance than state-of-the-art approaches that require accurate demand information.**

I. Introduction

Recent years have witnessed a growing popularity of vehicles with the capabilities of 4G/5G connectivity. Top car companies including BMW, Audi, Mercedes Benz, Volkswagen, etc. have dominated the market of connected vehicles (CVs) valued at $63.03 billion in 2019 and projected to reach $225.16 billion by 2027 [1]. CVs enable a bunch of applications, e.g., traffic behavior analysis in the intelligent transportation system [2] and vehicular crowdsensing [3]. To support these intelligent applications, CVs need to offload data (or structured data to save communication bandwidth) to application servers for analysis.

Apart from cloud computing adopted in existing solutions [4], the emerging edge computing attracts extensive attention for hosting application servers [2]. Edge servers, especially multiple-access edge computing (MEC) servers that are collocated with base stations (BSes), are envisioned to be widely deployed [5]. Due to the short distance between CVs and edge servers, the following benefits can be achieved: 1) The communication bandwidth to cloud servers can be saved; 2) The latency of transmitting data to the edge-supported application servers is low. For deploying an application server, an edge server usually adopts containers [6] to dynamically allocate computational resource according to demands, and various middleware to manage metadata, task lifecycles, input/output data, etc [7]. To guarantee a quick response time for latency-sensitive vehicular applications, edge servers need to be reserved to maintain warm pools of containers before CVs can offload data to it for process [6], [7].

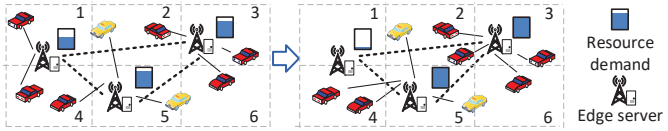


Fig. 1. Resource-demand change of edge servers due to CVs' mobility.

However, compared with de facto operations to maintain warm pools and middleware in the cloud, the high mobility of vehicles and the resulting frequent edge server hand-off, make the edge-server reservation problem extremely difficult. For instance, the CVs connected with the edge server in area 1 in Fig. 1 move to the edge servers in areas 3 and 5, so that the edge server in area 1 is idle and can be released. Ignoring such mobility could introduce severe resource wastage problem, making the limited edge resource even more scarce. Therefore, the problem becomes how to dynamically reserve edge servers to guarantee CVs to finish their computation tasks in nearby edge servers, without incurring resource too much wastage.

Existing solutions are mainly focused on determining the placement of edge servers to better match the demand [8]–[14]. These solutions usually assume that the statistical information on user demands is given. Based on such information, they apply optimization approaches to select edge servers' locations that satisfy statistical demands. However, the statistical information cannot be perfectly aligned with real-time demands from dynamically moving CVs, resulting in severe resource under-provisioning or over-provisioning problems.

In this paper, a deep reinforcement learning (DRL) based scheme called DeepReserve is developed to learn variable demands for a policy to reserve edge servers accordingly. DRL based methods do not rely on an accurate system model or complete system information [15]–[19]. It gradually improves the policy based on the effects of previous reservations. Specifically, the DRL agent takes the demands observed by edge servers, e.g., their resource utilization as input states. Based on the states, a proper action, i.e., the edge servers

to reserve, is selected for future demands. In particular, due to the large number of edge servers to be selected, the deep deterministic policy gradient (DDPG) algorithm [20] is selected as the basic building block in DeepReserve, since it is known for handling problems with high-dimensional state space and action space.

Nevertheless, there still remain two major problems that prevent classical DDPG from reserving the optimal edge servers for various demands. First, the deep neural network (DNN) in DDPG lacks both representative ability for accurate state inference and sufficient high-reward experiences for training. To solve this problem, ConvLSTM [21] is first adopted in DeepReserve, which is inspired by the observation that the spatio-temporal correlation of vehicle traffic [22] reflects on the resource demands. ConvLSTM extracts spatio-temporal features from the input states, which are mapped into actions via a fully-connected layer and the addition of random noise [20]. The actions are then checked and amended to enhance the probability of achieving high rewards. The second problem lies in that the existing training method cannot stably exert DRL models' power to reserve proper edge servers in real-life deployment, to solve which a training method named DR-Train is designed. To prevent non-convergence due to blind explorations when the agent is inexperienced, the experience pool for training is first initialized by a greedy algorithm. Afterwards, considering that vehicle traffic patterns vary between weekdays and weekends [23], the model is divided into two branches and trained respectively to generate the policies accommodated to two possible patterns. DeepReserve is evaluated utilizing real-world vehicle traffic and BS-location datasets. The results show that DeepReserve performs better than state-of-the-art approaches that require accurate information of demands.

In summary, our contributions are

- The system model of edge computing based CV system is built and the edge-server reservation problem is formulated, which is proved to be NP-hard.
- A DRL based scheme called DeepReserve is developed, which is adapted from DDPG with two improvements, i.e., adopting ConvLSTM and the action amender. DeepReserve can efficiently learn to dynamically reserve edge servers without accurate demand information.
- A training method called DR-Train is designed. Featured with two techniques, i.e., experience-pool initialization and model branches, DR-Train can stably train models for different vehicle traffic patterns.

The rest of the paper is organized as follows. In Section II, the system model of edge-computing based CV system is built and the edge-server reservation problem is formulated. Afterwards, DeepReserve is developed in Section III and the training method DR-Train is presented in Section IV. In Section V, comparative evaluation results are presented, followed by the review of related work in Section VI. Finally, the paper is concluded in Section VII.

TABLE I
NOTATIONS IN THE SYSTEM MODEL

Notations	Descriptions
$x_{i,t}$	The indicator of whether MEC server i is reserved as slot t
$y_{i,j,t}$	The indicator of whether CV j is connected with MEC server i as slot t
$u_{i,t}$	The resource utilization of MEC server i as slot t
$d_{i,j,t}$	The latency between CV j and MEC server i as slot t
$q_{i,t}$	The number of denied connections recorded by MEC server i at slot t
$\mathcal{E}, E$	The set of MEC servers and the size of the set
$\mathcal{V}_t, V_t$	The set of CVs at slot t and the size of the set
U	The total resources of an MEC server
D	The maximum latency that can be tolerated
α,β,γ	The cost to reserve an MEC server, the profit for a CV, and the punishment for a denied connection in a slot
$\mathbf{s}_t$,$\mathbf{a}_t$,r_t	The states, actions, and reward at slot t

II. EDGE SERVER RESERVATION IN THE CV SYSTEM

A. System Model

In the CV system, MEC servers are utilized to process data from CVs. Each MEC server is assumed to be collocated with a BS and a BS can access to every MEC server. MEC servers can dynamically allocate computational resources to an application server hosted in it according to the demand. In each slot of the CV system, an MEC server can be either reserved or released. A centralized server is utilized to gather observations of MEC servers at the end of each slot and adjust the reservations of MEC servers for the next slot.

To successfully process data, a CV needs to connect with a reserved MEC server and transmit data to it. The connection is set up according to the status of the CV and MEC servers as follows: 1) A CV first sets up the radio link with a BS following the access procedure in cellular networks, e.g., selecting the BS with the largest signal-noise-ratio (SNR). 2) The user-plane function (UPF) [24] of the BS then finds the unsaturated MEC server with the smallest latency according to real-time system information, which includes but is not limited to network status, UPF's workload, and UPF's location [24]. If such an MEC server is found, the connection is built and UPF steers the data traffic to the MEC server. Otherwise, the connection requests are denied and such failure cases are recorded by the saturated MEC server with the smallest latency. These failed CVs can request for MEC servers in the next slot. Established connections can be changed when reserved MEC servers are released. In this case, the data traffic of the CVs that connect to these MEC servers is steered to another unsaturated MEC servers with the smallest access latency, while the radio links of these CVs are still maintained. If no such MEC server is found, the connections break. The broken connections are also recorded as failure cases.

The notations to describe the CV system are listed in Tab. I. The set of MEC servers that can be reserved for the CV system is denoted as $\mathcal{E}$ with a total number E. In each time slot t, MEC server $i \in \mathcal{E}$ can be either reserved or released, which is denoted as a binary indicator $x_{i,t}$. The maximum resources of an MEC server that can be allocated to the CV system are

U units and each CV requires a unit. The connection and the latency between CV j and MEC server i at slot t are denoted as $y_{i,j,t}$ and $d_{i,j,t}$, respectively. The maximum tolerable latency is D. The resource utilization of an MEC server depends on the number of served CVs, i.e., $u_{i,t} = \sum_{j \in \mathcal{V}_t} y_{i,j,t}$, where $\mathcal{V}_t$ is the set of CVs in slot t and the number of CVs is denoted as V_t. The recorded denied requests of each MEC server is denoted as $q_{i,t}$. The total number of failed connections recorded by all MEC servers equals to the number of CVs that cannot connect to MEC servers, i.e., $\sum_{i \in \mathcal{E}} q_{i,t} = \sum_{j \in \mathcal{V}_t} (1 - \sum_{i \in \mathcal{E}} y_{i,j,t})$.

The system utility is composed of three parts. The cost paid to the MEC server operator for reserving an MEC server in a slot is α. For each connected CV, the system gains a profit of β, which equals the value of obtained data subtracted with the cost for a unit of resource in MEC server. Meanwhile, the system is punished with γ for a denied connection.

B. Problem Formulation

The problem of edge-server reservation (Ω) in the CV system is to determine the proper MEC servers to reserve, so that the system utility can be maximized in each slot.

Problem Ω:

$$\max \quad \sum_{i \in \mathcal{E}} (-\alpha x_{i,t} + \beta u_{i,t} - \gamma q_{i,t}) \tag{1}$$

$$\textit{s.t.} \quad d_{i,j,t} y_{i,j,t} \leq D x_{i,t}, \tag{2}$$

$$\sum_{i \in \mathcal{E}} y_{i,j,t} \leq 1, \tag{3}$$

$$\sum_{j \in \mathcal{V}_t} y_{i,j,t} \leq U x_{i,t}, \tag{4}$$

$$y_{i,j,t} \leq x_{i,t}, \tag{5}$$

$$x_{i,t}, y_{i,j,t} \in \{0, 1\}, \tag{6}$$

where E.q. (2) - (6) must be valid for $\forall i \in \mathcal{E}$ and $\forall j \in \mathcal{V}_t$. E.q. (2) limits that the latency between a CV and its connected MEC server cannot be larger than the latency constraint. E.q. (3) indicates that a CV can connect with at most one MEC server. E.q. (4) ensures that the resources utilization will not exceed the capacity of an MEC server. E.q. (5) illustrates that a CV can only connect with a reserved MEC server. E.q. (6) shows that the reservations of MEC servers and CV-server connections are binary variables.

Theorem 1. *Problem Ω is NP-hard.*

Proof. The maximum coverage problem can be reduced to Ω. Given a set of sets $\mathcal{S}' = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_E\}$, each set $\mathcal{S}_i$ covers some elements from $\mathcal{V}'$. The problem is to find no more than k sets from $\mathcal{S}'$, such that the number of covered elements is maximized. In slot t, there are E ($E = k$) MEC servers in total and each MEC server is able to serve certain CVs from $\mathcal{V}_t$, which satisfy the latency constraint, i.e., E.q. (2). Without limiting the connections, i.e., E.q. (3), and the capacity of MEC server, i.e., E.q. (4), the solution that maximizes the system utility ($\alpha = 0$, $\beta = 1$, and $\gamma = 0$) is also the optimal solution to the maximum coverage problem. Since the maximum coverage problem is NP-hard [25], the problem Ω is also NP-hard. □

In realistic scenarios, the statistical information of demands, reflected by the statistics of CV amounts ($\mathcal{V}_t$) and CVs' latency ($d_{i,j,t}$) to edge servers, cannot accurately represent the variable demands in each slot. In addition, collecting demand information of each slot in a real-time manner is also infeasible due to the dynamical mobility of CVs. Such factors make the NP-hard problem Ω even more difficult. The model-free DRL method is then promising to solve the problem. It can learn to reserve from the demands observed by previously-reserved edge servers ($u_{i,t}$ and $q_{i,t}$) rather than relying on accurate demand information. However, without the information ($\mathcal{V}_t$ and $d_{i,j,t}$) that directly represents spatio-temporal features of demands as input, existing DRL algorithms fail to exploit such features for the inference of future demands, which is further explained and resolved in the section below.

III. DeepReserve for Edge-Server Reservation

In this section, the preliminaries of DDPG are first introduced, followed by its limits in the edge-server reservation problem. Afterwards, the basic ideas and design details of DeepReserve are presented.

A. Preliminaries of DDPG

In the framework of RL, there is an agent interacting with an environment. At each slot t, the agent observes the environment and obtains the system state $\mathbf{s}_t$. The agent then takes an action $\mathbf{a}_t$ according to a policy that gives the probability to take each action given a state, and receives a reward r_t from the environment. Afterwards, the system state transits to $\mathbf{s}_{t+1}$, based on which the agent tasks another action $\mathbf{a}_{t+1}$ and gets the corresponding reward r_{t+1}. The target is then to determine the policy that maximizes the system utility.

The policy is learned by the agent from the interaction with the environment. A simple way is to record the map between states and actions in a matrix, e.g., the Q-table in Q-learning [26]. Nevertheless, the number of states in complex systems can be large, so that the cost of storing a Q-table is unacceptable. To solve this problem, a DNN is leveraged to replace the matrix, i.e., DQN in [27]. Two techniques are also designed to guarantee stable training, i.e., experience replay and the target network.

DDPG inherits the framework of DQN. Moreover, the actor-critic architecture that contains an actor network and a critic network [28] is adopted in DDPG to tackle the continuous actions. Hence, DDPG consists of four DNNs in total, i.e., each of the actor network and the critic network is further composed of an online network and a target network. In DDPG, the online network in the actor network can directly determine an action according to the probability distribution of actions given by the policy gradient method, while the critic network returns an approximated Q value to assist the training of the actor network.

B. The Limits of DDPG

The dimensions of both the action space and the state space in the edge-server reservation problem equal the number E of

MEC servers in a city, which can be extremely large. DDPG is suitable because the states can be recorded by the neural network, and the actor-critic architecture can quickly select a high-dimension action without searching.

However, DDPG does not necessarily result in high rewards, i.e., appropriate reservations, which can be caused by the lack of both representative ability of DNN adopted in DDPG and high-reward experiences for training as explained below. First, due to the fact that demands show spatio-temporal correlation, while the fully-connected layers in DNN adopted by DDPG do not encode any spatial information [21] and temporal features [22], the output of DNN is not an accurate prediction of future states. According to such output, the map to actions cannot achieve high rewards. Second, an agent usually randomly samples actions from the huge action space [20] in order to gain sufficient high-reward experiences for the neural network to learn the policy. Nevertheless, random exploration from the huge action space is unlikely to gather enough high-reward experiences within limited times of exploration [29], which also leads to low system utility during exploration.

C. Design of DeepReserve

1) DRL Model Design: To leverage DRL, the state space, action space, and reward of the MEC server reservation problem are first designed as follows.

States. In practical CV systems, the demands that are observable to MEC servers are their resource utilization and the recorded failed connections. The addiction of them is then reported as the state to the centralized server in each slot, i.e., $\mathbf{s}_t = [u_{1,t} + q_{1,t}, \cdots, u_{E,t} + q_{E,t}]$. The released MEC servers are not required to report any information, thus, their states are set as zero.

Actions. The actions to take are the MEC servers to be reserved in each slot, i.e., $\mathbf{a}_t = [x_{1,t}, \cdots, x_{E,t}]$.

Reward. The reward received via applying $\mathbf{a}_t$ to $\mathbf{s}_t$ is the system utility of the CV system gained from CVs minus the cost of reserving MEC servers and the punishment of failed connections. Formally, $r_t = \sum_{i=1}^{E}(-\alpha x_{i,t} + \beta u_{i,t} - \gamma q_{i,t})$.

2) Basic Ideas Behind DeepReserve: To enable DDPG to exploit spatio-temporal features buried in states, two improvements are made in DeepReserve.

Replacing fully-connected DNN with ConvLSTM. As explained in Section I, the spatio-temporal correlated demands result in the same property in states. In each slot, the states can be represented in a map (or image), where the value of each grid (or pixel) indicates the state corresponding to the MEC server in the gird (let each grid contains at most an MEC server). The convolution operation is then efficient to capture the relations between adjacent grids (or pixels) [22], i.e., the states of adjacent MEC servers. Meanwhile, the state maps (images) in continuous slots form a time series, the relation among which can be captured by the concatenated memory cells in LSTM [21]. By exploiting the correlation among previous states, the prediction accuracy of future states increases [21], [22], so that the actions taken correspondingly can gain high rewards. ConvLSTM is a neural network composed of both CNN and LSTM that are designed for capturing features in images and time series, respectively. Thus, ConvLSTM is adopted in DeepReserve.

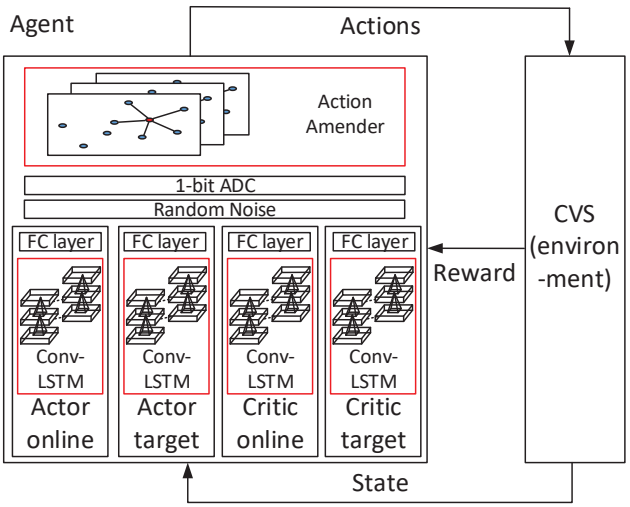
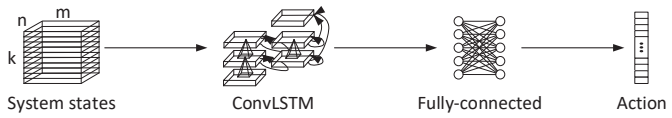


Fig. 2. The framework of DeepReserve.

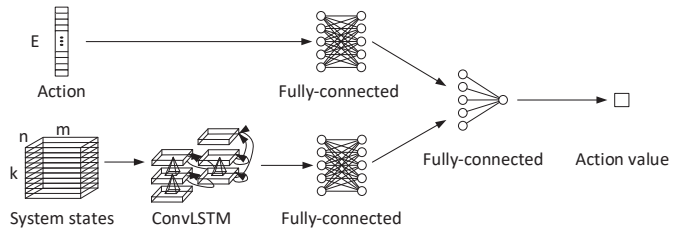
Amending the actions to satisfy the spatio-temporal correlation. To boost the procedure of gaining "good" experiences, supervising the selection of actions via external knowledge has been proved to be efficient [30], since the agent does not need to learn from scratch. In the context of edge-server reservation, considering the spatio-temporal correlation of demands, the external knowledge is that the demands in a specific area will not surge without an accumulative demand increment in previous slots (temporal) or nearby areas (spatial). To leverage such knowledge, an action amender is designed to first check the actions selected according to the policy learned or random exploration. If they violate the knowledge, the actions are reversed, i.e., the decision to reserve an MEC server is changed to release.

3) Design Details: The framework of DeepReserve is first shown in Fig. 2. Based on the framework, the procedure of DeepReserve is stated as follows. Based on the state observed from the CV system, the actor network selects an action, meanwhile, all the networks are trained for an iteration (the training method is introduced in the next section). The action selected is added with a random noise and converted into binary values, which are further checked and amended by the action amender before applied to the CV system. In the framework, two major improvements aforementioned are made compared with classical DDPG (marked by red boxes): 1) The ConvLSTM [21] is adopted in DDPG; 2) An action amender is adopted to check and modify the output of the actor network. The design details of these improvements are illustrated below.

ConvLSTM is a combination of CNN and LSTM, i.e., the convolution operation is adopted in each gate of the LSTM cell. It takes a sequence of 2-D tensors (equivalent to a 3-D tensor) of states at continuous slots as input, and outputs a 2-D tensor as the prediction of the state in the next slot. ConvLSTM is employed in both the actor network and the critic network, as shown in Fig. 3(a) and Fig. 3(b), respectively.



(a) The actor network



(b) The critic network

Fig. 3. The structures of the actor network and the critic network in DeepReserve.

In the actor network, the input data is the states of E MEC servers in previous k slots. To accommodate the input data to ConvLSTM, the state in each slot is structured as a 2-D $m \times n$ tensor. Each element indicates the state of MEC server in the corresponding grid on the map. For example, the input data of the CV system in Fig. 1 is a sequence of 2×3 tensors with elements 1, 3, and 5 indicating states while the remaining elements as zero. With such a sequence of 2-D tensors as input, ConvLSTM returns a 2-D tensor, which implies the state in the next slot. Afterwards, the 2-D tensor is reshaped into a 1-D tensor, which is activated with LeakyReLu. A fully-connected layer is then applied to further map the 1-D tensor to the action of E MEC servers with Sigmoid as the activation function. The critic network adopts the same top three structures as the actor network, but the output of the fully-connected layer is added with the output of another fully-connected layer that takes the action selected by the actor network as input. The result of addition is mapped into an action value via a fully-connected layer and the LeakyReLu activation function for the judgment of the action selected by the actor network.

Since the output data from the fully-connected layer of the actor network (added with a random noise [20]) is continuous, a 1-bit ADC is utilized to covert the output data into binary values [31]. The binary output is then regarded as the action chosen by the actor network, denoted as $\mathbf{a}_{i,t}^{(\lambda)} = [x_{1,t}^{(\lambda)}, \cdots, x_{E,t}^{(\lambda)}]$.

The action amender is designed as follows. It first records the states (the addition of resource utilization and failed connections $u_{i,t} + q_{i,t}$) observed in previous l slots, i.e., $\{\mathbf{s}_{t-l}, ..., \mathbf{s}_t\}$. Given an action selected by the actor network on an MEC server, if the action is to release an MEC server, i.e., $x_{i,t}^{(\lambda)} = 0$, the action amender takes the emendation that directly adopts $x_{i,t}^{(\lambda)}$ as the action $x_{i,t}$ to apply in the environment. If the action selected by the actor network is to reserve an MEC server, i.e., $x_{i,t}^{(\lambda)} = 1$, the amender conducts the following checking: 1) the states of the MEC server in previous l slots, and 2) states of the reserved MEC servers among the nearest g MEC servers of the MEC server at slot t (the set is denoted as $\mathcal{E}_g$). If none of the states show there are demands, i.e., the summation of the states equals zero, the amender reverses the action with a probability p. Formally, for an action $x_{i,t}^{(\lambda)}$ selected by the actor network, the amender takes the following operation to map $x_{i,t}^{(\lambda)}$ into $x_{i,t}$, denoted as X_a, with probability p:

$$x_{i,t} = x_{i,t}^{(\lambda)} H[\sum_{j=0}^{l}(u_{i,t-j} + q_{i,t-j}) + \sum_{\delta \in \mathcal{E}_g}(u_{\delta,t} + q_{\delta,t})], \quad (7)$$

where H is the unit step function and $P(X_a) = p$. Meanwhile, with probability $1 - p$, the agent directly adopts the action of the actor network, i.e., $x_{i,t} = x_{i,t}^{(\lambda)}$. Such an operation is denoted as X_b and $P(X_b) = 1 - p$.

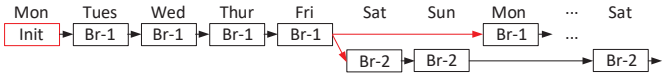


Fig. 4. The procedure of DR-Train.

IV. Training Method of DeepReserve

In order to guarantee a stable reservation performance in real-life CV systems, a training method, i.e., DR-Train, is designed in this section.

The training method for DDPG in [20] cannot exert DeepReserve's capability to reserve proper MEC servers due to two reasons. The first reason is when an agent is inexperienced, i.e., ConvLSTM is not trained or the action amender has not obtained enough states for the checking, the random action exploration inevitably results in actions with low rewards. Consequently, with a pool of "bad" experiences, a model easily converges to a suboptimal policy or even does not converge. The second reason lies in that the vehicle traffic patterns are different on weekdays and weekends [23], which means the environments faced by the agent are different. Hence, with the classical training method, the policy learned indiscriminately from data of weekdays and weekends cannot gain high rewards for both patterns simulatively.

To stably converge models and achieve high rewards under different vehicle traffic patterns, two techniques are designed in DR-Train, the procedure of which is shown in Fig. 4. First, in order to avoid the agent learns from randomly-taken actions, an algorithm that can choose actions with higher rewards than the randomly-taken actions is utilized to initialize the experience pool. As shown in Fig. 4, a greedy algorithm (reserve MEC servers nearby saturated ones and release idle ones, which is modified from [14]) is adopted in place of the DRL agent to take actions in the first day. Second, for enabling the model to work well on both weekdays and weekends, two model branches are divided to learn the policies for weekdays and weekends, respectively. As shown in Fig. 4, the training procedure begins on a weekday. When the first weekend arrives, a new branch of model is forked with parameters copied from the model trained on weekdays. Afterwards, these two branches of models are applied iteratively on weekdays and weekends.

The details of DR-Train are shown in Alg. 1. First, the parameters of the critic online network $Q_w(\cdot)$ and the actor network $\mu_w(\cdot)$ for weekdays, are randomly set as θ_w^Q and θ_w^μ,

Algorithm 1 DR-Train

1: Randomly initialize critic online network $Q_w(\cdot)$ and actor online network $\mu_w(\cdot)$ for weekdays with parameters θ_w^Q and θ_w^μ, respectively;
2: Initialize target networks $Q'_w(\cdot)$ and $\mu'_w(\cdot)$ with parameters $\theta_w^{Q'} \leftarrow \theta_w^Q$ and $\theta_w^{\mu'} \leftarrow \theta_w^\mu$, respectively;
3: Initialize a random process $\mathcal{N}$ and experience pools R_w and R_h for weekdays and weekends, respectively;
4: Receive initial observation state $\mathbf{s}_1$;
5: **for** z in $\mathcal{Z}$ **do**
6: **if** z is weekday **then**
7: **for** $t = 1$ **to** T **do**
8: **if** z is the first weekday **then**
9: Select $\mathbf{a}_t$ according to the greedy algorithm;
10: Execute $\mathbf{a}_t$ and observe r_t and $\mathbf{s}_{t+1}$;
11: Store transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in R_w;
12: **else**
13: Select action $\mathbf{a}_t^{(\lambda)}$ based on binarized $\mu_w(\mathbf{s}_{t,t-k}|\theta_w^Q) + \mathcal{N}$;
14: Amend $\mathbf{a}_t^{(\lambda)}$ to obtain $\mathbf{a}_t$ according to E.q. (7) with a probability p;
15: Execute $\mathbf{a}_t$ and observe r_t and $\mathbf{s}_{t+1}$;
16: Store transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in R_w;
17: Sample a random sequence of k transitions $(\mathbf{s}_{i,i-k}, \mathbf{a}_{i,i-k}, \mathbf{r}_{i,i-k}, \mathbf{s}_{i+1})$ from R_w;
18: Update the critic network by minimizing the loss: $L = (\phi_i - Q_w(\mathbf{s}_{i,i-k}, \mathbf{a}_{i,i-k}|\theta_w^Q))^2$, where $\phi_i = r_i + \xi Q'_w(\mathbf{s}_{i+1}, \mu'_w(\mathbf{s}_{i+1}|\theta_w^{\mu'})|\theta_w^{Q'})$;
19: Update the actor policy using the sampled policy gradient: $\nabla_{\theta_w^\mu} \approx \nabla_{\mathbf{a}} Q_w(\mathbf{s}, \mathbf{a}|\theta_w^Q)|_{\mathbf{s}=\mathbf{s}_{i,i-k}, \mathbf{a}=\mu_w(\mathbf{s}_{i,i-k})} \nabla_{\theta_w^\mu} \mu_w(\mathbf{s}|\theta_w^\mu)|_{\mathbf{s}_{i,i-k}}$;
20: Update the target networks: $\theta_w^{Q'} \leftarrow \psi\theta_w^Q + (1-\psi)\theta_w^{Q'}$, $\theta_w^{\mu'} \leftarrow \psi\theta_w^\mu + (1-\psi)\theta_w^{\mu'}$;
21: **else**
22: **if** z is the first weekend **then**
23: Initiate the critic networks $Q_h(\cdot)$ and $Q'_h(\cdot)$ and actor networks $\mu_h(\cdot)$ and $\mu'_h(\cdot)$ for weekends with parameters cloned from the networks for weekdays: $\theta_h^Q \leftarrow \theta_w^Q$, $\theta_h^\mu \leftarrow \theta_w^\mu$, $\theta_h^{Q'} \leftarrow \theta_w^{Q'}$, and $\theta_h^{\mu'} \leftarrow \theta_w^{\mu'}$;
24: **for** $t = 1$ **to** T **do**
25: Execute actions and update networks following the procedure in L13-20 with modifications: $Q_w(\cdot) \leftarrow Q_h(\cdot)$, $\mu_w(\cdot) \leftarrow \mu_h(\cdot)$, $Q'_w(\cdot) \leftarrow Q'_h(\cdot)$, $\mu'_w(\cdot) \leftarrow \mu'_h(\cdot)$, $\theta_w^Q \leftarrow \theta_h^Q$,$\theta_w^\mu \leftarrow \theta_h^\mu$, $\theta_w^{Q'} \leftarrow \theta_h^{Q'}$, $\theta_w^{\mu'} \leftarrow \theta_h^{\mu'}$, and $R_w \leftarrow R_h$;

respectively (L1). The parameters of the target networks, i.e., $Q'_w(\cdot)$ and $\mu'_w(\cdot)$, are copied from the online networks (L2) and gradually updated from the online networks with an update parameter ψ (L20). An Ornstein-Uhlenbeck process [20] and the experience pools, i.e., R_w and R_h, are also initialized (L3). Starting from the first state observed (L4), the agent adopts the greedy algorithm to take actions and record the experiences

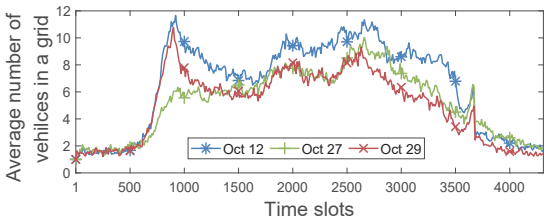


Fig. 5. The average numbers of vehicles in each grid with vehicles.

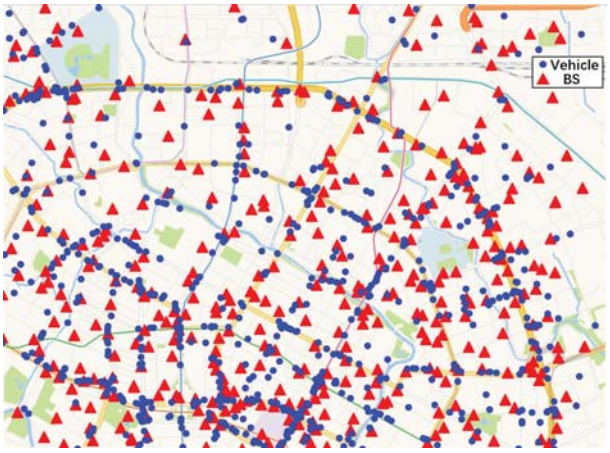


Fig. 6. The geographical distributions of vehicles and BSes.

in the first day of the CV system (L9-11). Afterwards, for slot t in weekday z of the CV-system operation duration $\mathcal{Z}$ (L5-7), the action $\mathbf{a}_t$ is chosen by the ConvLSTM and the amender according to the states in previous k slots (L13-15). Such action is recorded in R_w (L16). For training the critic online network, a sequence of k samples are taken from R_w and the parameters are updated via minimizing the squared error loss, where ϕ_i is the target value obtained according to the Bellman equation and the discount factor is ξ (L17-18). The actor network is then trained by sampled policy gradient (L19). When the first weekend arrives, the networks for weekends, i.e., $Q_h(\cdot)$, $\mu_h(\cdot)$, $Q'_h(\cdot)$ and $\mu'_h(\cdot)$, are initiated with parameters copied from the networks for weekdays (L22-23). After initialization, the procedure to take actions and train is the same as that on weekdays shown in L13-L20.

V. Performance Evaluation

The performance of DeepReserve is evaluated comprehensively based on real-world datasets. In this section, experimental settings are first illustrated, followed by experimental results and corresponding analysis.

A. Experimental Settings

Dataset Description. The performance of DeepReserve is evaluated based on two real-world datasets: locations of BSes and vehicle trajectories, as detailed as follows: 1) BSCD contains the locations of 46050 BSes run by CMCC and CUCC in Chendu City collected in Dec. 2019, among which 1910 BSes in the northeast of the city (30.653°N - 30.705°N and 104.042°E - 104.122°E) are utilized in the performance evaluation. 2) DECD18 includes the trajectories of Didi expresses in Chendu City during Oct 8, 2018 - Oct 31, 2018. The location of each express is updated every 20 seconds. The

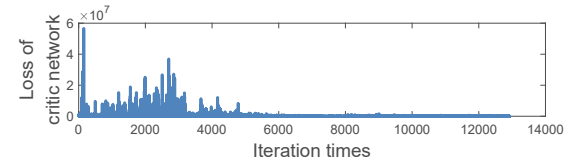
trajectories of vehicles in the same area as the selected BSes are utilized in the performance evaluation. To show the vehicle traffic patterns, the variance of the average numbers of vehicles in each grid with vehicles in it (the whole area is divided into 100 grids) on Oct 12 (Friday), Oct 27 (Saturday), and Oct 29 (Monday) is shown in Fig. 5, from which it can be observed that vehicle traffic patterns of weekdays and weekends are disparate, and the pattern of a weekday is also distinct from the pattern of another weekday. Based on these two datasets, a possible reservation scheme is shown in Fig. 6, where blue dots and red triangles indicate vehicles and BSes, respectively.

System Setup. In the CV system, a time slot is 20 seconds. The latency between a CV and an MEC server $d_{i,j,t}$ consists of two parts: 1) the latency between a CV and an MEC server, which is proportional to their geo-distance with a maximum value of 20 ms [32]; 2) the network status, which causes a random latency [33] with a maximum value of 10 ms. The latency constraint D is set to 10 ms and the maximum resources U of an MEC server for the CV system is 5 units. In the reward function, the parameters are set as $\alpha = 15$, $\beta = 6$, and $\gamma = 5$.

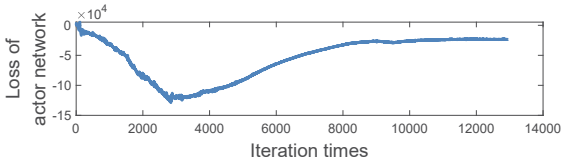
Training Setup. The python library PyTorch (version 1.1.0) is used to build the model. All experiments are tested on a Linux workstation (CPU: Intel i7-6850K @3.6GHz, RAM: 32 GB DDR4, GPU: NVIDIA GeForce GTX 1080). A 5-layer ConvLSTM with convolution kernel size equal to 5 [21] is adopted in both of the actor network and the critic network, the learning rates of which are 0.01 and 0.005, respectively. If not stated otherwise, the parameters are set as follows: 1) in the action amender, $l = 1$ and $p = 0.99$; 2) in the actor network and the critic network, $m = 70$, $n = 70$, and $k = 10$; 3) in the training procedure, $\psi = 0.01$ and $\xi = 0.9$.

Metrics. Four metrics are utilized to evaluate DeepReserve: 1) system utility; 2) average resource utilization of reserved MEC servers; 3) the probability of successful connections among all CVs; and 4) training loss of both the critic network and actor network from initialization to convergence.

Benchmark Approaches. The following benchmark approaches are selected to compare with DeepReserve: 1) DDPG with ConvLSTM (DC) is the classical DDPG [20], while replacing the embedded DNN with ConvLSTM. 2) DDPG with the action amender (DA) is the classical DDPG [20], while added with the action amender. 3) The classical DDPG algorithm in [20]. 4) User Clustering (UC) [8] is applied by clustering user demands and the MEC servers near the clustering centers are reserved. 5) Heaviest-AP First (HAF) [13] first gathers user demands by letting CVs to request MEC servers freely. The MEC server with the most requests is reserved to serve CVs. The procedure iterates for the remaining MEC servers until all CVs can be connected. 6) Greedy Service Placement (GSP) [14] is modified for the CV system that the MEC server closest to the saturated MEC server is reserved in each slot, and the idle MEC servers in previous slots are released. 7) The optimal reservation scheme is obtained by searching all feasible schemes. 8) Reserved MEC servers are randomly selected in each slot. In addition, to validate the effectiveness of DR-Train, the classical training method for DDPG in [8] is adopted as comparison.

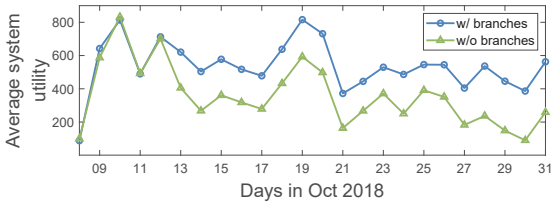


(a) Convergence of the critic network

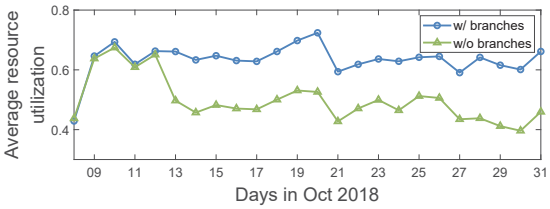


(b) Convergence of the actor network

Fig. 7. The training loss with experience-pool initialization.



(a) Average system utility

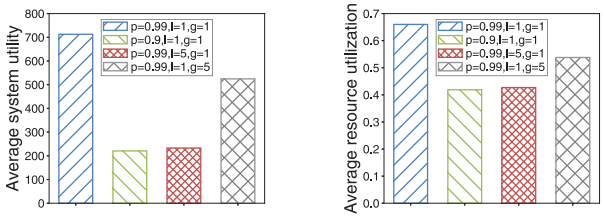


(b) Average resource utilization

Fig. 8. The effectiveness of dividing model branches.

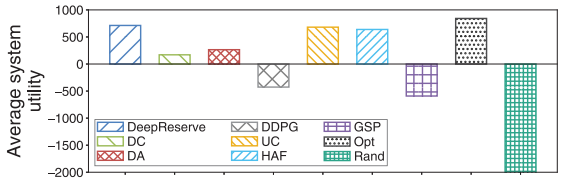
B. Experimental Results

Experiments are first conducted to evaluate the convergence performance of DeepReserve trained by DR-Train. The training loss of both the critic network and the actor network is shown in Fig. 7. It can be observed that both networks can converge quickly within 8000 times of iteration, which requires the experiences gathered within two days (8640 slots). Fast convergence enables DeepReserve to be quickly deployed in real-life CV systems. In contrast, models can hardly converge without the experience-pool initialization (delete L8-12 in Alg. 1). Moreover, to justify the effectiveness of dividing model branches, the system performance with two model branches over the 24 days in dataset DECD18 is compared with that without branches (delete L6 and L21-25 in Alg. 1). As shown in Fig. 8(a) and Fig. 8(b), the average system utilities and average resource utilization of two cases are the same until Oct 13 (the first weekend), after when they show a 25% gap and a 20% gap, respectively. These results demonstrate the necessity to maintain policies for weekdays and weekends, respectively.

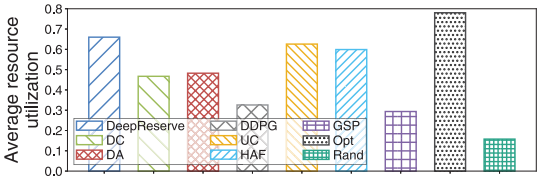


(a) Average system utility (b) Average resource utilization

Fig. 9. The performance under different parameters in the action amender.



(a) Average system utility

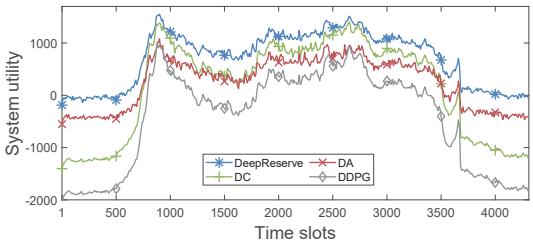


(b) Average resource utilization

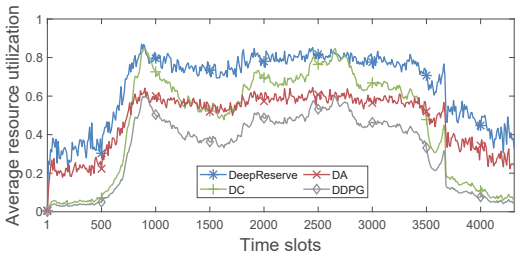
Fig. 10. The performance compared with benchmark approaches.

The performance of DeepReserve with different parameters in the action amender is then evaluated. The average system utility and the average resource utilization of MEC servers on Oct 12 (when the model is well-converged and is not impacted by the data of weekends) are shown in Fig. 9(a) and Fig. 9(b), respectively. It can be observed that both system utility and average resource utilization decrease for around 70% and 27% respectively, as the emendation probability p decreases from 0.99 to 0.9, which can be viewed as the results of decreasing the intensity to amend actions. This result validates the effectiveness of the action amender in DeepReserve. However, p cannot be set to 1 (the system utility is too low to be shown in the figure). This is because the MEC server released by the amender easily satisfies the condition of amending (being idle in previous slots triggers X_a according to E.q. (7)) again in the following slots, thus, it will always be released. The system then converges to the state that nearly all MEC servers are released. In addition, as the number of slots l with zero states or the number of nearby MEC servers g to be checked increases from 1 to 5, the probability that an action chosen by the neural network is modified decreases. Hence, similar to reducing p, the performance decreases.

DeepReserve is also compared with benchmark approaches, among which UC and HAF are conducted under the assumption that accurate information of demands in each slot is available. The average performance on Oct 12 is shown in Fig. 10. From Fig. 10(a) and Fig. 10(b), it can be observed that DeepReserve achieves higher system utility



(a) System utility



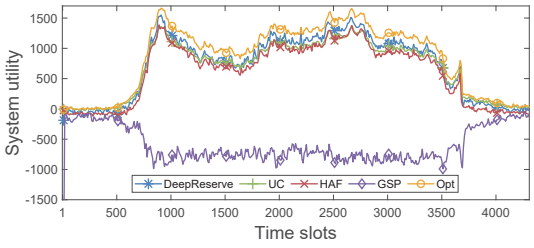
(b) Average resource utilization

Fig. 11. The performance compared with different variants of DeepReserve.

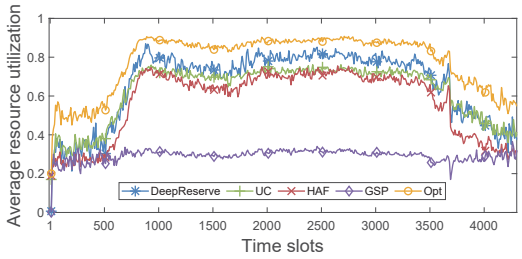
and resource utilization than all the benchmark approaches except for a slight worse than the optimal solution. These results demonstrate that DeepReserve can reserve proper edge servers for enhancing the system utility.

To further validate the effectiveness of adopting ConvLSTM and the action amender in DeepReserve, a deep dive into the detailed performance in each slot of Oct 12 is shown in Fig. 11(a) and Fig. 11(b). According to the results, both of ConvLSTM and the action amender bring significant performance enhancement. Interestingly, when the CV density is high, i.e., when slots are around 900 and 2700 according to Fig. 5, the performance of DC is close to that of DeepReserve. Such a result is owing to that the relations between nearby MEC servers are significant, i.e., the values in nearby grids of the input state map are closely related, which match with the size of convolution kernel. Thus, ConvLSTM can efficiently capture spatio-temporal features when CVs are densely distributed. Meanwhile, when CV density is too low for ConvLSTM to learn a good policy, i.e., at the slots less than 900 or larger than 3500, the performance of DA is better than DDPG and DC thanks to the action amender. Armed with ConvLSTM and the amender that complementarily contribute performance gain when CVs are densely and sparsely distributed, respectively, DeepReserve consistently achieves much better performance than DDPG.

Finally, the detailed performance comparison between DeepReserve and state-of-the-art approaches is shown in Fig. 12(a) and Fig. 12(b) (the results of the random approach are too low to be included in the figures). By comparing DeepReserve with the approaches with accurate demand information in each slot, DeepReserve achieves slightly better performance than UC and HAF and around 10% worse than the optimal solution constantly in all slots, which confirms the DeepReserve's power of quickly adapting the reservation to real-time demands. In contrast, as an algorithm without the



(a) System utility



(b) Average resource utilization

Fig. 12. The performance compared with state-of-the-art approaches.

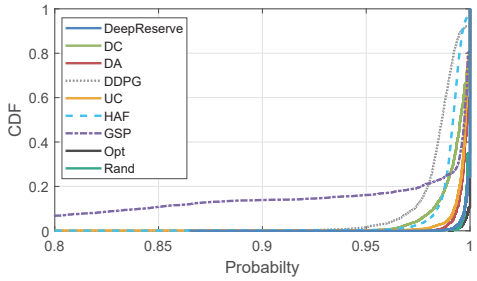


Fig. 13. The CDF plot of the probability of successful connection.

requirement of accurate demand information, GSP performs much worse than DeepReserve, especially when the density of CVs is high. This is because the adjustment of reservations cannot catch up with the variance of demands, which results in severe punishments due to massive connection failures. This reason is further verified by the CDF of the probability that a CV successfully connects to an MEC server as shown in Fig. 13. From the results, it is obvious that GSP has much more failed connections than the other approaches. These results also illustrate that DeepReserve can ensure the connections of over 99% CVs to MEC severs in 99% slots, which is the same as the optimal solution. The high success probability of connections guarantees the high availability of the service provided by MEC servers.

VI. Related Work

A. Edge Server Reservation

Existing literature focuses on determining the placement of edge servers or service from candidate locations. The solutions in [8], [9] aim to determine the positions of edge servers that are geographically close to users. The locations of edge servers in [10] are chosen based on the number of user requests aggregated in nearby BSs, such that the latency to transmit requests to edge servers can be minimized. The authors of [11] extend the server placement problem to choose suitable edge servers to hold multiple interrelated services. The work in [12] further considers the placement of multiple services into edge servers with heterogeneous capacities to maximize the system utility. This literature usually determines edge-server placement based on given statistical demands in an offline manner. In our study, we make reservation decisions in an online manner adaptive to the real-time demands, without relying on accurate demand information. The solution in [34] does not assume user demands are directly given, but leverages the collected contexts of connected users (e.g., equipment types and external environment factors) to predict the demands, so as to guide the placement decision. However, these contexts are not available in CV systems.

B. DRL in Edge Computing for CVs

DRL has been widely applied to solve problems in the edge-computing systems for CVs. The authors of [19] leverage DRL to determine the task-offloading scheme for CVs, so that the tradeoff between QoE of CVs and profit of edge servers can be achieved. In [15], the task offloading problem is extended to the scenario where CVs act as supplementary edge servers. Furthermore, the authors in [16] take the dependencies between tasks into consideration for making offloading decisions. In addition, there are also papers that utilize DRL to jointly determine the data caching schemes and computational-resource allocation schemes [17], or jointly consider cooperative content placement and delivery [18]. All the aforementioned papers explicitly rely on CV information, like locations and latency to construct the state information in DRL. In contrast, we study the edge server reservation problem without this direct information. Modifications are made allowing DRL agent to exploit the demand information from other obtainable system information.

VII. Conclusion

In this paper, the edge server reservation problem is studied to support the emerging intelligent edge applications. We first formally formulated the edge-server reservation problem, revealed its complexity, and discussed the difficulties to solve when the demand information is unavailable. We then developed a novel DRL based scheme, DeepReserve, to make decisions simply relying on the partial demand information. DeepReserve enhanced the classical DDPG algorithm by adding an extra ConvLSTM module and an action amender to capture the spatio-temporal correlation of demands. The training process was further enhanced with a newly proposed DR-Train algorithm to improve training stabilization. Extensive trace-driven experiments validated the effectiveness of the DeepReserve in exploiting the underlying spatio-temporal correlations and the capability to handle dynamic traffic patterns. Consequently, DeepReserve demonstrated superior performances when compared with state-of-the-art approaches.

Acknowledgment

This work was supported by National Natural Science Foundation of China (NSFC) under grant 61771312.

REFERENCES

[1] S. Abhay and K. Lalit, "Connected car market: Global opportunity analysis and industry forecast, 2020-2027," *Allied Market Research*, 2020.

[2] A. Ferdowsi, U. Challita, and W. Saad, "Deep learning for reliable mobile edge analytics in intelligent transportation systems: An overview," *IEEE Vehicular Tech. Mag.*, vol. 14, no. 1, pp. 62–70, 2019.

[3] J. Ni, A. Zhang, X. Lin, and X. S. Shen, "Security, privacy, and fairness in fog-based vehicular crowdsensing," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 146–152, 2017.

[4] M. Whaiduzzaman, M. Sookhak, A. Gani, and R. Buyya, "A survey on vehicular cloud computing," *J. Network and Computer Applications*, vol. 40, pp. 325–344, 2014.

[5] 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.

[6] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proc. Int. Middleware Conf.*, 2016, pp. 1–13.

[7] A. Carrega, M. Repetto, P. Gouvas, and A. Zafeiropoulos, "A middleware for mobile edge computing," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 26–37, 2017.

[8] H. Yin, X. Zhang, H. H. Liu, Y. Luo, C. Tian, S. Zhao, and F. Li, "Edge provisioning with flexible server placement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1031–1045, 2016.

[9] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, 2015.

[10] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Computing*, vol. 127, pp. 160–168, 2019.

[11] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things J.*, vol. 6, no. 2, pp. 3641–3651, 2018.

[12] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. Int. Conf. Computer Commun.(INFOCOM)*. IEEE, 2019, pp. 514–522.

[13] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2015.

[14] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *Proc. Int. Conf. Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 365–375.

[15] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Vehicular Tech.*, vol. 68, no. 11, pp. 11 158–11 168, 2019.

[16] 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 Trans. Vehicular Tech.*, vol. 68, no. 5, pp. 4192–4203, 2019.

[17] R. Q. Hu *et al.*, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Trans. Vehicular Tech.*, vol. 67, no. 11, pp. 10 190–10 203, 2018.

[18] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet of Things J.*, vol. 7, no. 1, pp. 247–257, 2019.

[19] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intelligent Syst. and Tech. (TIST)*, vol. 10, no. 6, pp. 1–24, 2019.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2016.

[21] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional LSTM network: a machine learning approach for precipitation nowcasting," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 802–810.

[22] B. Yu, H. Yin, and Z. Zhanxing, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting," in *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, 2018, pp. 3634–3640.

[23] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Proc. AAAI Conf. Artificial Intelligence (AAAI)*, 2017.

[24] 3GPP TS23.501, "System architecture for the 5G System (5GS)," Rel. 16, V16.4.0, 2020.

[25] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[28] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2000, pp. 1008–1014.

[29] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. Int. Conf. Computer Commun.(INFOCOM)*. IEEE, 2018, pp. 1871–1879.

[30] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild, "Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression," in *Proc. AAAI Conf. Artificial Intelligence (AAAI)*, 2005, pp. 819–824.

[31] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[32] O. Krajsa and L. Fojtova, "RTT measurement and its dependence on the real geographical distance," in *Proc. Int. Conf. Telecommunications and Signal Processing (TSP)*. IEEE, 2011, pp. 231–234.

[33] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, L. Sun *et al.*, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *Proc. Int. Conf. Computer Commun.(INFOCOM)*. IEEE, 2019, pp. 910–918.

[34] L. Chen, J. Xu, S. Ren, and P. Zhou, "Spatio–temporal edge service placement: A bandit learning approach," *IEEE Trans. Wireless Commun.*, vol. 17, no. 12, pp. 8388–8401, 2018.