Service Migration Optimization for System Overhead Minimization in VECNs via Deep Reinforcement Learning

Yuan Yuan, *Graduate Student Member, IEEE*, Bin Yang, Wei Su, Jie Ma, *Graduate Student Member, IEEE*, Yihua Peng, *Graduate Student Member, IEEE*, Qi Liu, and Tarik Taleb, *Senior Member, IEEE*

***Abstract*— In vehicular edge computing networks (VECNs), service migration among edge servers is critical to addressing the challenge of service interruption caused by high mobility of vehicles and limited coverage of each edge server. In this article, we tackle this challenge by optimizing service migration among edge servers through a joint management of resource scheduling and dynamic server selection. Specifically, we aim to minimize system overhead consisting of system time and energy consumption taking account for resource scheduling and dynamic server selection, which is formulated as a constrained optimization problem. To solve this optimization problem, we propose a learning-driven joint resource scheduling and dynamic server selection strategy (LD-JRS3) based on deep reinforcement learning. Under the LD-JRS3 strategy, we first model joint resource scheduling and dynamic server selection as a Markov decision process (MDP). Then, we adopt a recurrent neural network (RNN)-empowered feedback mechanism based on historical information to achieve the optimal system performance. We fully consider the advantages of the soft actor–critic (SAC) algorithm to obtain the optimal decision (i.e., computational resources allocation and servers selection). Notably, we employ an improved SAC algorithm, which takes into account prioritized experience replay and automatic tuning of temperature parameters. Extensive simulation results are presented to verify the effectiveness of our proposed LD-JRS3 algorithm, and also to illustrate the advantage of our algorithm on improving the time consumption and energy consumption compared with the baseline schemes. LD-JRS3 has 19%, 24%, and 11% higher utility values than DDRN, DQN-based, and multiarmed bandit-based systems, respectively.**

***Index Terms*—Deep reinforcement learning, multiaccess edge computing (MEC), recurrent neural network (RNN), resource scheduling, service migration, vehicular networks.**

Received 15 August 2024; revised 20 September 2024; accepted 9 October 2024. Date of publication 16 October 2024; date of current version 6 February 2025. This work was supported in part by the National Key Research and Development Project of China under Grant 2022YFB2901603; in part by the Ministry of Education Innovation Group Joint Fund under Grant 8091B042222; in part by the National Natural Science Foundation of China under Grant 62372076; in part by the Natural Science Foundation of Anhui Province under Grant KJ2021ZD0128 and Grant KJ2021B01; in part by the Anhui Talent Project under Grant DTR2023051; in part by the ICTFICIAL Oy, Finland; and in part by the European Union's HE Research and Innovation Program HORIZON-JUSNS-2023 through the 6G-Path Project under Grant 101139172. *(Corresponding authors: Wei Su; Bin Yang.)*

Yuan Yuan, Wei Su, Jie Ma, and Yihua Peng are with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: yuan.yuan@bjtu.edu.cn; wsu@bjtu.edu.cn; 20111028@bjtu.edu.cn; 22110024@bjtu.edu.cn).

Bin Yang is with the School of Computer and Information Engineering, Chuzhou University, Chuzhou 239000, Anhui, China (e-mail: yangbinchi@gmail.com).

Qi Liu is with the Technology Innovation Center, Smart City Research Institute of China Unicom, Beijing 100033, China (e-mail: liuqi49@chinaunicom.cn).

Tarik Taleb is with the Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, 44801 Bochum, Germany (e-mail: tarik.taleb@rub.de).

Digital Object Identifier 10.1109/JIOT.2024.3481232

I. Introduction

THE VEHICULAR edge computing networks (VECNs) are a promising network paradigm by integrating multiaccess edge computing (MEC) into vehicular networks, where edge computing servers are deployed in close proximity to vehicles to provide computing and caching services [1], [2], [3]. Such networks can effectively reduce transmission latency and enhance real-time data processing capacity, making it particularly suitable for vehicular applications that require fast response times, such as augmented reality (AR), virtual reality (VR), extended reality (XR), intelligent transportation systems, and the Internet of Vehicles (IoV) [4]. The mobility of vehicles and the limited coverage of each edge server equipped at a base station can result in service interruption and degrade the Quality of Service (QoS) for vehicles. Service migration has great potential to address the negative effects through migrating services from an edge server to another according to the mobility of vehicles and dynamically environmental characteristics [5], [6], [7]. Thus, it is critical to explore the dynamic service migration scheme for an efficient support of IoV applications in VECN [9].

Existing works on service migration works mainly employ traditional mathematical methods [10], [11], [12], [13] and learning algorithms [14], [15], [16], [17], [18], [19], [20], [21] to optimize system performance (see Section II for details).

Note that these works mainly consider that the computational tasks have the same priority, and explore the minimization of time and energy consumption separately. In practice scenarios, the tasks usually have different priorities. For example, advanced smart driving vehicles should be allocated more resources to meet their requirements of computation, storage, and communication. It is essential to balance the time and energy consumption by a joint optimization on these two metrics. Furthermore, the dynamic information of vehicles and the environment cannot be entirely observed, and

thus it is necessary to utilize historical time-series data to predict future vehicle states using a mathematical framework like the recurrent neural network (RNN).

To address the above issues, this article explores the service migration in the VECN to guarantee the tradeoff between time consumption and energy consumption under the scenario with different task priorities. Service migration is essentially a dynamic problem, and server selection is an important subproblem of service migration. Thus, the existing static strategies are not applicable to the dynamic problem. We then propose a learning-driven joint resource scheduling and dynamic server selecting (LD-JRS3) strategy to achieve the minimum long-term system overhead. This scheme employs an RNN-based feedback strategy and a Markov decision process (MDP)-based reinforcement learning strategy. First, the optimal actions (i.e., the number of allocated blocks of computational resources and the index of selected service servers) are obtained through reinforcement learning. Further, the feedback strategy is used to determine different priorities so that network resources (i.e., channel bandwidth) and edge server computational resources are provided proportionally according to the priority coefficients. Notably, we employ an improved soft actor–critic (SAC) algorithm as a reinforcement learning solution, which takes into account prioritized experience replay (PER) and automatic tuning of temperature parameters. It should be noted that we validate the proposed solution through convergence and performance comparisons.

The main contributions of this article can be summarized as follows.

1) We define a system overhead function consisting of system time and energy consumption. We formulate the dynamic service migration process as an optimization problem with the aim of minimizing the long-term cumulative system overhead.
2) To solve this optimization problem, we model joint resource scheduling and dynamic server selection as an MDP involving the constraints of transmission link rate and priority matching. We then propose an RNN-empowered feedback mechanism based on historical information.
3) We further propose an LD-JRS3 strategy to solve this optimization problem, which fully considers the advantages of the SAC algorithm to obtain the optimal decision for achieving the minimization of the long-term system overhead.
4) Extensive simulation results are presented to validate our proposed LD-JRS3 strategy, and also to illustrate that our LD-JRS3 strategy outperforms the baseline scheme in system performance improvements. The simulation results show that LD-JRS3 is able to obtain a high average system utility, which is 19%, 24% and 11% higher than DDRN, DQN-based and multiarmed bandit (MAB)-based, respectively.

The remainder of this article is organized as follows. Section II summarizes the related works. Section III introduces the system model of the VECN and formulates the problem. Section IV constructs the optimization problem as an MDP and propose the RNN enabled LD-JRS3 strategy to solve it. Section V provides extensive simulation results. Finally, Section VI concludes this article.

II. Related Work

Available service migration works mainly employ traditional mathematical methods [10], [11], [12], [13] and learning algorithms [14], [15], [16], [17], [18], [19], [20], [21] to optimize system performance.

For the traditional mathematical methods, Chen et al. [10] presented a Lyapunov optimization-based algorithm that dynamically optimizes service migration and request routing to improve network performance and reduce operational costs in multicell mobile-edge environments. Liang et al. [11] developed a framework to optimize MEC by integrating service migration and resource allocation across multiple cells, enhancing network efficiency and user service continuity in dynamic environments. Xu et al. [12] developed an algorithm to optimize network path selection, balancing service quality and cost-efficiency to facilitate seamless service migration in vehicular networks. Zhou et al. [13] proposed a novel energy-efficient online algorithm for service migration in dense cellular networks, reducing average energy consumption and enhancing service latency without requiring user trajectory prediction.

As for learning algorithms, Yuan et al. [14] explored an innovative strategy to enhance the efficiency of intelligent and connected vehicles by optimizing service migration and mobility planning to better manage resource demands in vehicular networks. Xu et al. [15] introduced a novel network in box (NIB) task migration method optimized through the strength Pareto evolutionary algorithm (SPEA), significantly enhancing the balance between energy consumption and time cost during service migration for the IoV in a 6G context. Ning et al. [16] introduced a bi-objective optimization approach using lightweight imitation learning to facilitate efficient, real-time service migration in edge networks, optimizing both execution latency, and cost. Dalgkitsis et al. [17] proposed a dual approach combining Convolutional neural networks (NNs) and genetic algorithms to predict vehicular movements and optimize service migration in vehicular networks, aiming to minimize latency and enhance network performance by adaptively relocating services closer to users. Liu et al. [18] proposed an asynchronous deep-reinforcement-learning framework that optimizes collaborative computing and resource distribution for vehicular services, showcasing improvements in system utility and service performance. Wang et al. [19] reformulated the microservice coordination problem using the MDP framework, and then propose a reinforcement learning-based online microservice coordination algorithm to learn the optimal policy that reduces the overall service latency at a lower cost. Ouyang et al. [20], in order to overcome the unavailability of future information and the unknown nature of system dynamics, formulate the dynamic service placement problem as a contextual MAB problem, and then propose an online learning algorithm based on Thompson sampling to explore dynamic MEC environments, which further assists the user in making adaptive service placement decisions.

TABLE I
COMPREHENSIVE COMPARATIVE ANALYSIS OF THE DYNAMIC SERVICE MIGRATION

Comparison Metrics	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	The proposed scheme
Optimization Method	(M)	(M)	(M)	(M)	(L)	(L)	(L)	(L)	(L)	(L)	(L)	(L)	(L)
Realistic Dataset	–	–	✓	–	–	–	✓	–	–	✓	–	✓	✓
Resource Scheduling	–	✓	✓	–	✓	–	–	✓	✓	–	–	–	✓
Objective of Time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Objective of Energy	–	–	–	✓	–	✓	–	–	–	–	✓	✓	✓
Priority of Tasks	–	–	–	–	–	–	–	✓	✓	–	–	–	✓

[1] Optimization Method: (M) represents the Optimization scheme based on mathematical algorithm. (L) represents the Optimization scheme based on learning algorithm.
[2] Symbol ✓ indicates a high relevance. Symbol – indicates a low relevance.

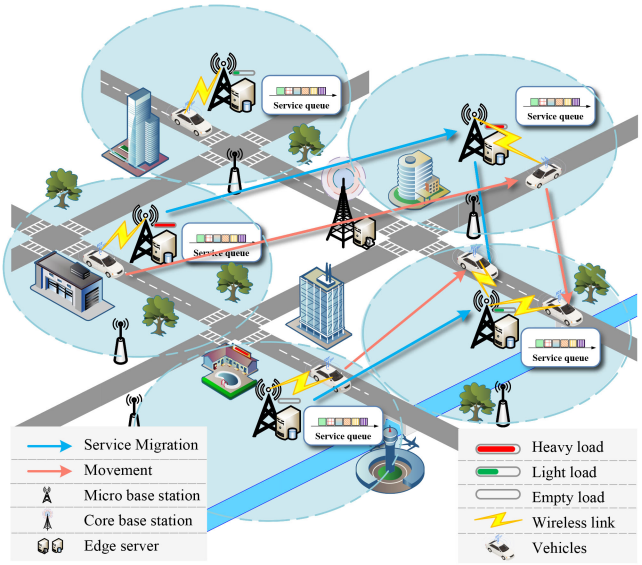


Fig. 1. VECN scenario.

Wang et al. [21] proposed a deep recurrent actor–critic model for managing service migrations in MEC environments, effectively handling the challenges of incomplete system information and dynamic conditions through a novel learning-driven approach.

The above studies on service migration, which mainly concentrate on optimizing time and energy consumption through centralized or distributed management, are summarized in Table I.

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. Network Model

As illustrated in Fig. 1, we consider a VECN for service migration consisting of a core base station (CBS) equipped with a regional center server, M micro base stations (MBSs) each of which is equipped with an edge server, and N intelligent connected vehicles (ICVs).

The CBS plays a pivotal role in aggregating both the configuration specifics of the network and the dynamic data of its surrounding environment, subsequently sharing this intelligence with the MBS. The edge server, facilitated by the CBS, equips ICVs with service migration policies utilizing learning algorithms. An MBS equipped with an edge server is called as a service node (SN). We consider that each edge server is capable of hosting an application instance to cater to the service demands from the ICV. The ICV sends a request to the SN through the cellular network, which responds to the request and forwards it to the application instance deployed by the edge server, and then returns the computational result to the ICV. The orthogonal frequency division multiple access (OFDMA) technique is adopted to support multiple access, where cellular links use different subchannels and thus there is no interference among them. In addition, all wireless channels encounter additive white Gaussian noise with variance σ^2. We assume that the size of each time-critical task is F. Note that ICV services have different priorities denoted by $V_n(t)$. Here, we propose a feedback mechanism, i.e., the priority of ICV services depends on the residence time in the area covered by a single MBS. We consider fast-moving or frequent cross-domain switching users (i.e., users with short residence time) as higher intelligence level users. We give more bandwidth resources and computing resources to the users with higher intelligence. The identifier of ICV task priority is given by

$$V_n(t) = \begin{cases} 1, & \text{if } T_n^{\text{res}}(t) \geq T_{\text{th}} \\ 0.5, & \text{otherwise} \end{cases} \tag{1}$$

where $T_n^{\text{res}}(t)$ represents the residence time in the area covered by a single MBS, and T_{th} represents the time threshold of QoS. Among them, different priorities correspond to different computational capabilities and channel bandwidth levels.

In the VECN, we consider a time-slotted system, where the vehicular moving and task requests change only once in a time slot. We then utilize service migration strategies to meet the requirements of time-critical applications, such as smart driving. Edge servers are considered as core network entities that provide powerful compute and storage capabilities to users. We need to propose a service migration scheme to tradeoff the energy consumption and time consumption of the system. The VECN can sense the time and system

TABLE II
DEFINITION OF MATHEMATICAL SYMBOLS

Symbol	Definition
N	The number of intelligent connected vehicles.
M	The number of micro base stations.
F	The size of each time-critical task.
$T_n^{\text{res}}(t)$	The residence time in the area covered by micro base stations.
$T_n^{\text{c}}(t)$	System time consumption.
$T_n^{\text{mig}}(t)$	The time consumption of migration.
$T_n^{\text{com}}(t)$	The time consumption of computation.
$T_n^{\text{tra}}(t)$	The sum of transmission latency among wireless networks and wires networks.
$T_n^{\text{a}}(t)$	The access latency of wireless networks.
$T_n^{\text{b}}(t)$	The backhaul latency of wires networks.
$F_n^{\text{app}}(t)$	The size of the service instance to be migrated.
$F_n^{\text{off}}(t)$	The size of the offloading task.
F_{load}	The workload of edge server.
$\mathcal{S}_n^{\text{loc}}(t)$	The local server of user n.
$\mathcal{S}_n^{\text{ser}}(t)$	The serving server of user n.
$I_m(t)$	Migration identifier for service node m.
$V_n(t)$	Service priority identifier for user n.
$\Delta_n^{\text{mig}}(t)$	The number of hops on the migration path.
$\Delta_n^{\text{back}}(t)$	The number of hops on the backhaul path.
h_{nm}	The Rayleigh channel gain between user n and service node m.
$R_n(t)$	The downlink transmission rate for user n.
$p_n^{\text{sup}}(t)$	The power supplied to the user n by the antenna.
$Z_n(t)$	The number of resource blocks assigned to user n.
$E_n^{\text{mig}}(t)$	The energy consumption for migration action.
$E_n^{\text{pro}}(t)$	The energy consumption of migration process.
$E_n^{\text{tot}}(t)$	The total energy consumption for each user n.
$K_n^{\text{tot}}(t)$	The overhead function of user n in time slot t.
B	The channel bandwidth of the base station.
B_0	The size of a unit of channel bandwidth.
B_{w}	The network bandwidth on the migration path.
f_0	The computational resources contained in a unit resource block.
$f_n^{\text{sup}}(t)$	The computational capability of the server for user n.
p	The transmit power coefficient of the edge server that initiated the service migration.
ρ	The processing density in CPU.
β_n	The ratio of resource allocation.
f_{e}	The capability of edge server.
R_T	The threshold of transmission rate.
T_{th}	The time threshold of QoS.
ω_T	The unit price of time related overhead.
ω_E	The unit price of energy related overhead.
ψ_n	The decision on server selecting.
$I(t)$	The identifier of service migration.
K_{ave}	The average system overhead.
U_{ave}	The average system utility.

energy consumption, and then jointly considers task priority and resource allocation to achieve service migration.

The symbols used in this article are described in Table II.

B. Performance Model

1) Time Consumption Model: System time consumption $T_n^{\text{c}}(t)$ is defined as

$$T_n^{\text{c}}(t) = T_n^{\text{mig}}(t) + T_n^{\text{com}}(t) + T_n^{\text{tra}}(t) \tag{2}$$

where $T_n^{\text{mig}}(t)$, $T_n^{\text{com}}(t)$, and $T_n^{\text{tra}}(t)$ represent time consumption of migration, computation, and communication, respectively.

For the time consumption of migration, $T_n^{\text{mig}}(t)$ is the latency that a running application instance is migrated from the current edge server to another edge server. Generally, migration latency is caused by the service interruption during the data transmission process. Migration latency changes with transmission distance, network performance, data volume, and system load. In this article, we focus on the data transmission latency and network latency of backhaul links in wired networks. Therefore, $T_n^{\text{mig}}(t)$ is given by

$$T_n^{\text{mig}}(t) = \begin{cases} 0, & \Delta_n^{\text{mig}}(t) = 0 \\ F_n^{\text{app}}(t)/B_{\text{w}} + \mu_A \Delta_n^{\text{mig}}(t), & \Delta_n^{\text{mig}}(t) \neq 0 \end{cases} \tag{3}$$

where $\Delta_n^{\text{mig}}(t)$represents the number of hops on the migration path, μ_A is the coefficient of migration latency, B_{W} is the network bandwidth on the migration path, and $F_n^{\text{app}}(t)$ is the size of the service instance to be migrated. In our proposed scenario, the user's local server is not necessarily the serving server. Relay hops refer to the number of node hops that need to be traversed from the serving server to reach the local server [21].

As for the time consumption of computation, $T_n^{\text{com}}(t)$ is defined as the latency required by the offloaded computing-intensive task on the server, which is affected by the number of cycles required for computational task execution and the computational resources provided by the server. Here, the time consumption $T_n^{\text{com}}(t)$ can be expressed as

$$T_n^{\text{com}}(t) = \frac{(F + F_{\text{load}})\rho}{\beta_n f_e} \tag{4}$$

where F is the size of computational task, F_{load} is the workload of edge server, ρ is the processing density in CPU, β_n is the ratio of resource allocation, and f_e is the capability of edge server.

As for the time consumption of communication, $T_n^{\text{tra}}(t)$ is defined as the sum of transmission latency among wireless networks and wired networks, which is expressed as

$$T_n^{\text{tra}}(t) = T_n^{a}(t) + T_n^{\text{b}}(t) \tag{5}$$

where $T_n^{a}(t)$ is the access latency of wireless networks, and $T_n^{\text{b}}(t)$ is the backhaul latency of wired networks.

The access latency of wireless networks includes uplink and downlink cases. Note that the uplink case focuses on analyzing the impact of offloading data. The downlink is used to transmit the results of the computation, which can be ignored due to the fact that the amount of data is particularly small. Among them, access latency $T_n^{a}(t)$ can be expressed as

$$T_n^{a}(t) = \frac{F_n^{\text{off}}(t)}{R_n(t)} \tag{6}$$

where $F_n^{\text{off}}(t)$ is the file size of the offloading task, and $R_n(t)$ is the uplink transmission rate.

We consider the Rayleigh channel model in small-scale fading, i.e., we assume that the received signal is a sum of a large number of independent and identically distributed multipath components that can be approximated to obey

independent Gaussian distributions. In our OFDMA system, h_{nm} denotes the Rayleigh channel coefficients with Gaussian distribution. The uplink transmission rate $R_n(t)$ is determined by transmit power, channel gain, channel bandwidth, and noise power, which can be expressed as

$$R_n(t) = B\log_2\left(1 + \frac{p_n^{\sup}(t)h_{nm}^2}{\sigma^2}\right) \tag{7}$$

where B is the channel bandwidth of the base station, $p_n^{\sup}(t)$ is the power provided by the antenna to the user n, h_{nm} is the channel gain between user n and SN m, and σ^2 is noise power. B is given by

$$B = \begin{cases} B_0, & \text{if } V_n(t) = 1 \\ \frac{1}{2}B_0, & \text{otherwise.} \end{cases} \tag{8}$$

Here, B_0 is the size of a unit of channel bandwidth. Since we consider an OFDMA system, there is no channel interference. A signal received at each user can be successfully decoded if and only if the transmission rate from BS to the user is greater than some threshold value R_T, i.e.,

$$R_n(t) \geq R_{\mathrm{T}} \tag{9}$$

where $R_n(t)$ represents the downlink transmission rate for user n.

The backhaul latency of wired networks is generated by the multihop transmission between edge servers due to the fact that the server providing service is not the local server. The backhaul latency depends on the distance of transmission, the amount of data of the offloading task, and the transmission latency of the computational results. Moreover, the amount of data used to computational results is usually small, so this part of the latency can be ignored. Among them, access latency $T_n^{\mathrm{b}}(t)$ can be expressed as

$$T_n^{\mathrm{b}}(t) = \begin{cases} 0, & \Delta_n^{\mathrm{back}}(t) = 0 \\ F_n^{\mathrm{off}}(t)/B_{\mathrm{w}} + \mu_B\Delta_n^{\mathrm{back}}(t), & \Delta_n^{\mathrm{back}}(t) \neq 0 \end{cases} \tag{10}$$

where μ_B is the coefficient of migration latency, B_{W} is the network bandwidth on the migration path, and $F_n^{\mathrm{off}}(t)$ is the size of the service instance to be migrated.

In our network model, the channel bandwidth and server computational capability can be flexibly controlled through dynamic adjusting the task priority and the number of resource blocks. The computational capability of the server for user n is given by

$$f_n^{\sup}(t) = V_n(t)Z_n(t)f_0 \tag{11}$$

where $V_n(t)$ is the task prioritization of user n, $Z_n(t)$ is the number of resource blocks assigned to user n, and f_0 is the computational resources contained in a unit resource block. The number of resource blocks allocated to all N users does not exceed G, and then

$$\sum_{n=1}^{N} Z_n(t) \leq G. \tag{12}$$

Then, the ratio of resource allocation can be expressed as

$$\beta_n = \frac{V_n(t)Z_n(t)}{G}. \tag{13}$$

2) Energy Consumption Model: The process of service migration likewise incurs a certain amount of energy overhead due to compute, network, storage, transmission, etc. These energy expenses can be divided into data transmission energy, service computing energy, and system storage energy. Among them, data transmission energy consumption is closely related to data volume, transmission distance, network topology, and transmission protocol. Service computing energy consumption refers to the task that after service migration, the new service server needs to start and configure new computational resources. The migration process may also need to perform some extra computational tasks, such as data decompression, configuration update, etc., which will also generate energy overhead. The energy consumption of system storage means that service migration involves the migration or replication of data storage, and the storage system often generates corresponding energy consumption. Typically, the percentage of this energy consumption is relatively low.

Due to the large number and wide range of categories involved in the energy overhead, it is often difficult to give precise expressions for representation. In this article, we simplify the energy consumption generated during service migration into an approximate expression consisting of migration action energy consumption $E_n^{\mathrm{act}}(t)$ and migration process energy consumption $E_n^{\mathrm{pro}}(t)$, which is expressed as

$$E_n^{\mathrm{tot}}(t) = E_n^{\mathrm{act}}(t) + E_n^{\mathrm{pro}}(t). \tag{14}$$

Here, $E_n^{\mathrm{act}}(t)$ is given by

$$E_n^{\mathrm{act}}(t) = I(t)E_n^{\mathrm{mig}}(t) \tag{15}$$

where $E_n^{\mathrm{mig}}(t)$ represents the energy consumption to complete one migration action, and $I(t)$ represents the identifier of service migration. When $\Delta_n^{\mathrm{mig}}(t) \neq 0$, $I(t) = 1$, and $I(t) = 0$, otherwise. $E_n^{\mathrm{pro}}(t)$ is given by

$$E_n^{\mathrm{pro}}(t) = pT_n^{\mathrm{mig}}(t) \tag{16}$$

where p is the transmit power coefficient of the edge server that initiated the service migration, and $T_n^{\mathrm{mig}}(t)$ represents the time consumption of migration.

C. Problem Formulation

Our objective is to minimize the time consumption and also to reduce the energy consumption caused by service migration. Note that we are able to dynamically adjust the tradeoff between latency and energy consumption by flexibly setting weight parameters of these two metrics according to different application requirements. For example, for time-sensitive applications, we set a larger value of latency weight to improve response time and QoS. For energy-sensitive applications, we set a larger value of energy consumption weight to reduce the energy consumption of the device.

We then construct the overhead function of user n in time slot t, which is given by

$$K_n^{\mathrm{tot}}(t) = \omega_T T_n^{\mathrm{c}}(t) + \omega_E E_n^{\mathrm{tot}}(t) \tag{17}$$

where ω_T and ω_E are used to nondimensionalize the function and can realize a tradeoff between the time consumption and the energy consumption in each time slot.

We use $\mathcal{N} = \{1, 2, \ldots, N\}$, $\mathcal{M} = \{1, 2, \ldots, M\}$ and $\mathcal{T} = \{1, 2, \ldots, T\}$ to represent the sets of the mobile users, SNs, and time slots, respectively. Based on (17), the optimization problem can be formulated as

$$\mathbf{P1:} \quad \min \quad \sum_{t=1}^{T} K_n^{\text{tot}}(t) \tag{18}$$

$$\text{s.t.} \quad n \in \mathcal{N}, \ \ t \in \mathcal{T} \tag{18a}$$

$$\psi_n(t) \in \mathcal{M} \tag{18b}$$

$$R_n(t) \geq R_{\text{T}} \tag{18c}$$

$$\sum_{n=1}^{N} Z_n(t) \leq G \tag{18d}$$

$$\beta_n \in [0, 1] \tag{18e}$$

where (18) represents the set of ICVs and time slots, (18a) represents that the user n can only migrate its applications instances to SNs in the region $m \in \{1, \ldots, M\}$, (18b) represents that a signal received at each user can be successfully decoded if and only if the transmission rate from BS to the user is greater than some threshold value R_{T}, (18c) represents that the number of resource blocks allocated to all N users does not exceed G, and (18d) represents that the ratio of computational resources allocated to application instances should be between 0 and 1.

Obviously, it is very difficult to obtain the optimal solution for the above objective, which requires the user's movement trajectory and complete system-level information over the entire time horizon T. However, in real-world scenarios, it is impractical to predict all the relevant information in advance. To address this challenge, we propose a long-short-term-memory (LSTM)-based vehicle trajectory prediction algorithm that can make effective migration decisions based on the observed information.

Meanwhile, this is a nonlinear and nonconvex optimization problem, which is generally difficult to solve. In the following section, we propose an LD-JRS3 algorithm to solve above problem.

Based on the overhead of each user defined in (18), we further define average system overhead denoted by K_{ave} as the average value of all users' overheads. Then, we have

$$K_{\text{ave}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} K_n^{\text{tot}}(t). \tag{19}$$

IV. Deep-Reinforcement-Learning-Based Algorithm

A. LSTM-Based Vehicular Trajectory Prediction

To obtain more adequate known information to assist in making optimal migration decisions, we need to predict vehicle trajectories based on historical information. In general, the historical information of vehicle trajectories in a region is similar. Meanwhile, the vehicular driving route is temporal information, which is related to the behavior in a time period. Therefore, recurrent neural can be used to predict the future trajectories of ICVs in the form of a sliding window. Based on future trajectory information, we combine it with QoS boundaries to predict the residence time of ICVs in the coverage area of a single SN. LSTM is a widely used RNN model. Our proposed algorithm predicts the future trajectories of ICVs with the residence time within the region by LSTM.

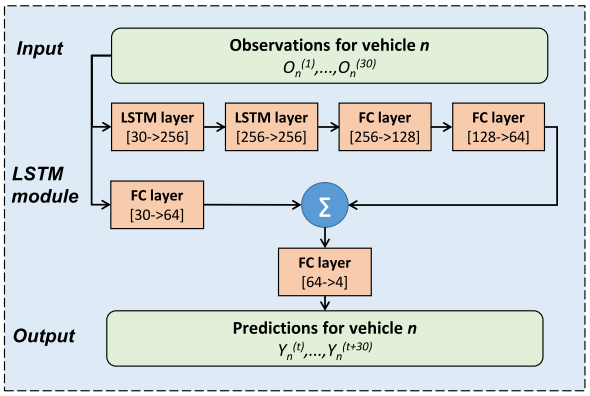


Fig. 2. Architecture of LSTM-based network.

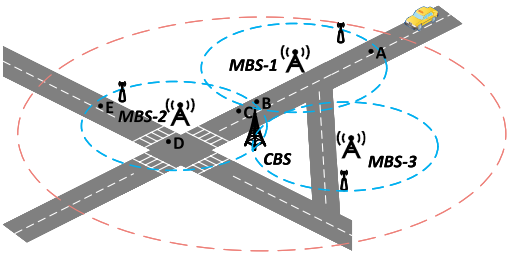


Fig. 3. Schematic of the traffic scenario.

Furthermore, we construct a five-layer DNN model, which includes three normal fully connected layers and two fully connected layer consisting of LSTM units. In the described network architecture, N vehicles are selected as observation units, and their movement trajectories, speeds, directions, and accelerations are recorded over 30 time intervals. This data are used as input for network training.

The architecture of LSTM-based network is illustrated as Fig. 2. Initially, the 30-D data from the vehicles are input into a fully connected layer consisting of 256 LSTM units. Subsequently, the 256-D data output from the fully connected layer is then fed into the fully connected layer, which outputs 256-D feature data. To facilitate machine learning of pertinent information, two fully connected layers (256→128 and 128→64) are employed for feature extraction and transformation, aiding the network in learning advanced representations and features of the input data. Furthermore, to capitalize on the relevance during vehicle operation, the observed data are also input into an additional fully connected layer (30→64), and the results are combined with those from the previously mentioned layers. This approach established a robust correlation between the input and prediction sequences, enabling the machine to more effectively learn the variations in vehicle movement over time and improve training speed. After completing these steps, the aggregated feature data are passed through another fully connected layer to produce the

final output sequence, which is then converted into actual coordinates and saved for future LSTM network capture.

The input information of the model can be expressed as

$$O_n(t) = \left\{x_n^t, y_n^t, \phi_n^t, v_n^t, \kappa_n^t\right\}, t \in [0, T_{\mathrm{H}}] \tag{20}$$

where $\{x_n^t, y_n^t\}$ represents the latitude and longitude coordinate, ϕ_n^t represents the traveling angle, v_n^t represents the velocity, and κ_n^t represents the acceleration. Among them, we define T_{H} as the length of the historical input time series.

The output information of the model can be expressed as

$$\hat{Y}_n(t) = \left\{\hat{v}_{x,n}^t, \hat{v}_{y,n}^t, \hat{x}_n^t, \hat{y}_n^t\right\} \tag{21}$$

where $\hat{Y}_n(t)$ denotes the predicted velocities and coordinates for user n at time slot t. It is worth noting that all the above coordinate information is converted from the Cartesian coordinate system to the Frenet coordinate system.

In the prioritized feedback mechanism of this article, $T_n^{\mathrm{res}}(t)$ is calculated based on the moments corresponding to the predicted coordinates. Based on each candidate MBS boundary coverage area and the proposed vehicle trajectory prediction model, the residence time of the vehicle in each candidate MBS coverage area can be obtained. The steps for obtaining the time are described here. As shown in Fig. 3, the red dotted circle indicates the coverage area of the CBS, assuming that $\{\overline{AB}, \overline{BC}, \overline{CD}, \overline{DE}\}$ is the future traveling trajectory obtained by the vehicle trajectory prediction model, and points C and E are the two predicted locations closest to the boundary within the coverage area of MBS-2. Therefore, the residence time $T_n^{\mathrm{res}}(t)$ of vehicle n in MBS-2 can be determined to be $t_{n,E} - t_{n,C}$, where $t_{n,C}$ and $t_{n,E}$ are the moments when vehicle n reaches positions C and E. Similarly, the residence time $T_n^{\mathrm{res}}(t)$ of vehicle n in MBS-1 can be determined to be $t_{n,A} - t_{n,B}$, where $t_{n,A}$ and $t_{n,B}$ are the moments when vehicle n reaches positions A and B. Similar to previous work, we also use an LSTM-based scheme to predict future information [22]. It is worth noting that the work in this article is based on LSTM to complete trajectory prediction.

B. MDP Model

We construct the resource scheduling and server selecting process as an MDP process defined by the quaternion tuple $\langle S, A, P, U\rangle$, where S, A, P, and U denote the state space, action space, state transition probability, and utility function, respectively. The specific process is described as follows.

1) *State Space S:* $s(t) = (F_1^{\mathrm{off}}(t), \ldots, F_N^{\mathrm{off}}(t), F_1^{\mathrm{app}}(t), \ldots, F_N^{\mathrm{app}}(t), \mathcal{S}_1^{\mathrm{ser}}(t-1), \ldots, \mathcal{S}_N^{\mathrm{ser}}(t-1), V_1(t), \ldots, V_N(t))$ is defined as the system state at time slot t, which is composed of the size of the offloading task $F_n^{\mathrm{off}}(t)$ for each user, the size of the service instance $F_n^{\mathrm{app}}(t)$ for each user, the service server tuple $\mathcal{S}_n^{\mathrm{ser}}(t)$ for each user, the task prioritization $V_n(t)$ for each user, and the calibration parameter ϵ for users.
2) *Action Space A:* $a(t) = (\psi_1(t), \psi_2(t), \ldots, \psi_N(t), Z_1(t), Z_2(t), \ldots, Z_N(t))$ is defined as the system action set A in time slot t, which represents the decisions with both resource scheduling and servers selecting for each user, i.e., $\psi_n(t)$ and $Z_n(t)$).
3) *State Transition Probability P:* $P = S \times A \times S \rightarrow [0, 1]$ represents the distribution of the transition probability $P(s' \mid s, a)$ from the system state s to a new system state s' $(s, s' \in S)$ when an action $a \in A$ is chosen, which is mainly affected by environmental changes, such as the user's request arrival rate, the priority of vehicles, the threshold for rate, and the transmission failure probability.
4) *Utility Function U:* $S \times A \rightarrow U$ maps a state–action pair to a value $U(s(t), a(t))$. Our objective in this article is to minimize the overhead $K_n^{\mathrm{tot}}(t)$ given in (17) under the constrained conditions, and then we can define the utility function as $U(s(t), a(t)) = -\sum_{n=1}^{N} K_n^{\mathrm{tot}}(t)$. Meanwhile, we define the average system utility denoted by U_{ave} as the average value of total utility of N users, and then we have

$$U_{\mathrm{ave}} = \frac{1}{N}\sum_{t=1}^{T} U(s(t), a(t)). \tag{22}$$

Note that a larger average system utility (i.e., lower system overhead) implies higher system performance.

The state space of MDP consists of the size of the user's offloading task, the size of the user's application instance, the user's last time slot's serving server index and the user's task priority coefficient, which is an 4N-D space. To cope with the problem of high-dimensional catastrophes, we consider the following aspects. First, NNs are used as function approximators to approximate Q-value functions and policies. NNs can efficiently process high-dimensional input data and extract useful features, thus reducing the dependence on the original high-dimensional state space. Second, we use a PER buffer to store the experience of interacting with the environment, which can be used to train the Q-network and the policy network multiple times. Then, we use two Q networks to mitigate excessive bias in Q-value estimation to prevent the problem of Q-value overestimation in high-dimensional state spaces. Finally, we consider the maximum entropy strategy to encourage exploration. In a high-dimensional state space, the maximum entropy strategy can explore the state space more comprehensively and avoid falling into local optimal solutions.

Here, we could obtain the strategy π based on the system state $s(t) \in S$ and action $a(t) \in A$. For ease of writing, we let $a(t) = a_t$, and $s(t) = s_t$. Then, the optimal strategy π^* with state $s(t')$ is given by

$$\arg\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t)\sim\xi_j}\left[U(s_t, a_t) + \alpha\mathcal{H}\left(\pi\left(\cdot \mid s_t'\right)\right)\right]. \tag{23}$$

C. LD-JRS3 Algorithm

Based on the above MDP model, we can well characterize the relationship between different system states and individual behaviors in smart driving scenarios. However, it is difficult to optimize long-term system rewards under the case of unknown environments and vehicle movements. Moreover, the decision-making processes for service migration among different vehicles influence each other, rendering the optimization of service migration strategies a highly challenging endeavor.

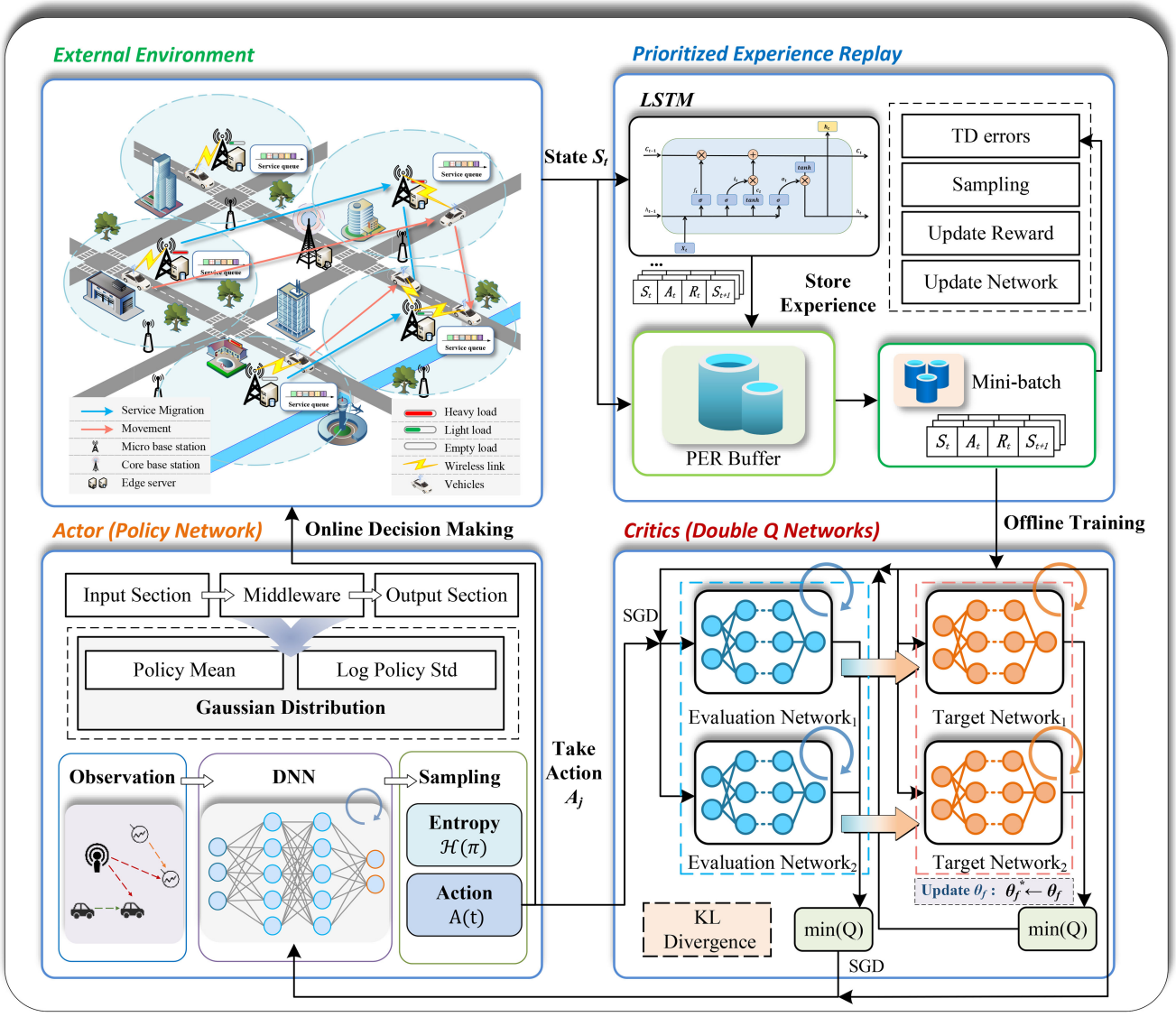


Fig. 4. Architecture of LD-JRS3 algorithm.

Here, we can design an LD-JRS3, which aims to be able to make the system execute the optimal joint resource allocation and service migration strategy in each state, thus maximizing the expectation of the cumulative utility function and making the strategy more stochastic. The architecture of our proposed LD-JRS3 algorithm is shown in Fig. 4.

In our scenario, we consider combining a PER approach with the SAC algorithm to solve the optimization problem posed in the previous section. The SAC algorithm is an off-policy deep-reinforcement-learning method that maintains the benefits of entropy maximization and stability while providing efficient learning over sampling [24]. SAC builds upon an actor–critic framework. The actor module of SAC contains a policy network responsible for maximizing the expected utility and the entropy. The critic module of SAC contains two Q-function networks responsible for providing parameter updates and guaranteeing with the effectiveness of the policy. It is worth noting that our strategies are modeled as Gaussian distributions, given in combination with the mean and covariance values of the output values of the fully connected network. Moreover, the Q-function is also approximated using a fully connected NN. The exact updating process of the algorithm is described as follows.

The goal of the SAC algorithmic architecture is to maximize the expected cumulative utility and the expected entropy of the strategy. To this end, we define the objective function and the optimal strategy as

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[U(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot \mid s_t))] \quad (24)$$

and

$$\pi^* = \arg\max_{\pi} J(\pi). \quad (25)$$

Here, α is the temperature control coefficient, which is used to adjust the importance of entropy to the utility function and adjust the randomness strategy. It is noted that a greater entropy will encourage the intelligent to explore more actions.

And $\mathcal{H}(\pi(\cdot \mid s_t))$ is the entropy function of the strategy under the action, which can be expressed as

$$\mathcal{H}(\pi(\cdot \mid s_t)) = \mathbb{E}_{a_t}\big[-\log \pi(a_t \mid s_t)\big]. \tag{26}$$

To spread out the probability distribution of the actions, we use the NN to approximate the Q-function, which can be expressed as

$$Q_\theta(s_t, a_t) = U(s_t, a_t) + \gamma \mathbb{E}\big[V_\theta(s_{t+1})\big]. \tag{27}$$

Here, the parameter of the Q-function is updated by soft Bellman residuals. Thus, the objective function of the Q-function can be expressed as

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\xi_j}\left[\frac{1}{2}\Big(Q_\theta(s_t, a_t) - \hat{Q}_\theta(s_t, a_t)\Big)^2\right] \tag{28}$$

where ξ_j is a batch of sampling states and actions in the experience replay. $\hat{Q}_\theta(s_t, a_t)$ is the estimated value of the Q-function, which can be expressed as

$$\hat{Q}_\theta(s_t, a_t) = U(s_t, a_t) + \gamma \mathbb{E}\big[V_{\bar{\theta}}(s_{t+1})\big] \tag{29}$$

where $\bar{\theta}$ is a parameter obtained as a moving average of the weight exponent of the soft Q-function, and $V_{\bar{\theta}}(s_{t+1})$ is the soft state value function, which is expressed as

$$V_{\bar{\theta}}(s_t) = \mathbb{E}_{a_t\sim\pi}\big[Q_{\bar{\theta}}(s_t, a_t) - \alpha \log \pi(a_t \mid s_t)\big]. \tag{30}$$

The parameter can be optimized with stochastic gradients (SGDs) by

$$\begin{aligned}\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t)(Q_\theta(s_t, a_t) - (U(s_t, a_t) \\ + \gamma\big(Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha \log\big(\pi_\phi(a_{t+1} \mid s_{t+1})\big)\big))).\end{aligned} \tag{31}$$

Then, we can update the parameter $\theta_i \leftarrow \theta_i - \lambda\nabla_{\theta_i} J_Q(\theta_i)$, for $i \in \{1, 2\}$. The output of the policy network is the mean and standard deviation of a Gaussian distribution, which is sampled to obtain the decision action of the policy. Since the policy is distributed, direct sampling results cannot be obtained, and gradient propagation is not feasible. Therefore, the reparameterization method is required, such as

$$a_t = f_\phi(\epsilon_t; s_t) \tag{32}$$

where ϵ is a normally distributed variable, and $\epsilon \sim \mathcal{N}(0, 1)$. Here, the policy can be learned by minimizing the expected KL-divergence, and the objective function is given by

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\xi_j}\big[\mathbb{E}_{a_t\sim\pi_\phi}\big[\alpha \log\big(\pi_\phi(a_t \mid s_t)\big) - Q_\theta(s_t, a_t)\big]\big]. \tag{33}$$

The parameter can be optimized with SGD by

$$\begin{aligned}\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \alpha \log\big(\pi_\phi(a_t \mid s_t)\big) + \big(\nabla_{a_t} \alpha \log\big(\pi_\phi(a_t \mid s_t)\big) \\ - \nabla_{a_t} Q(s_t, a_t)\big)\nabla_\phi f_\phi(\epsilon_t; s_t).\end{aligned} \tag{34}$$

Then, we can update the parameter $\phi_i \leftarrow \phi_i - \lambda\nabla_{\phi_i} J_Q(\phi_i)$. To achieve the balance between exploration and decision making through automatic entropy adjustment, we have to require that the entropy of the obtained strategy does not fall below a certain predefined threshold, which is expressed as

$$\mathcal{H}(\pi_t) \geq \mathcal{H}_0 \tag{35}$$

where $\mathcal{H}_0$ denotes the threshold of minimum strategy entropy, and the objective function of α can be expressed as

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi}\big[-\alpha \log \pi(a_t \mid s_t) - \alpha\mathcal{H}_0\big]. \tag{36}$$

The parameter (i.e., α) can be optimized with SGD via $\alpha_i \leftarrow \alpha_i - \lambda\nabla_{\alpha_i} J_Q(\alpha_i)$.

During NN training, we integrate the method of PER to divide the samples into batches of experience tuples $\mathbb{M} = \{\xi_1, \xi_2, \ldots, \xi_J\}$, where $\xi_j = (s_j, a_j, U(s_j, a_j), s_j')$, $j \in \{1, 2, \ldots, J\}$. And we let the predefined batch size be W_m. Higher priority samples have a higher probability of being sampled during the training process, due to the fact that these samples contribute to the improvement of the strategy. Particularly, the temporal difference (TD) error δ_j is used to measure the priority of the empirical tuples, which can be formulated as

$$\begin{aligned}\delta_j = |U(s_t, a_t) + \gamma\big(Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha \log\big(\pi_\phi(a_{t+1}|s_{t+1})\big)\big) \\ - Q_{\bar{\theta}}(s_t, a_t)|.\end{aligned} \tag{37}$$

Meanwhile, the sampling priority of each cached experience tuple ξ_j can be determined as $p_t = \delta_j$.

The pseudo code in Algorithm 1 shows the details of the proposed LD-JRS3 algorithm.

In the preparation phase of the algorithm, we initialize the replay buffer $\mathbb{M}$ to a constant size and initialize the policy network parameters. In our pseudocode proposed above, each episode represents the complete process of user movement in the service migration scenario, and the deployment location of the service is generated completely randomly when resetting the environment. Next, we input the state $s(t)$ of time slot t to the policy network. The policy network uses Softmax to perform the action $a(t)$ with the highest probability and obtains the utility $U(s(t), a(t))$ according to the formula derived in this section. After performing action $a(t)$, the state space $s(t)$ of the next time slot is acquired. We store the tuple $\langle s(t), a(t), U(s(t), a(t)), s(t+1)\rangle$ in the replay buffer $\mathbb{M}$. Then, the Q-network, the policy network, and the target Q-network are sequentially updated by sampling from $\mathbb{M}$. Finally, the algorithm will repeat the above steps until convergence.

The SAC algorithm is a policy-based reinforcement learning algorithm which is suitable for solving problems in continuous action space. And DQN is a value-based reinforcement learning algorithm which is suitable for solving problems in discrete action space. The problem studied in this article is the service migration problem, which is a continuous action space problem. Therefore, the SAC algorithm is more in line with the requirements of the scenario presented in this article. This article improves the sac algorithm. On the one hand, this scheme combines the PER, which speeds up the learning process and can improve the sample efficiency. On the other hand, this scheme can automatically adjust the temperature parameter, and this method can dynamically adjust the degree of exploration according to the exploration–utilization balance of the current policy, which is useful for the user to adaptively adjust the degree of exploration of the policy in the high-dimensional space. However, this scheme requires powerful computility resources to complete the training, and it is not

Algorithm 1: LD-JRS3: A Learning-Driven Joint Resource Scheduling and Dynamic Server Selecting Strategy

Input: PER buffer size $|\mathbb{M}|$, state S, pretraining step T_{pre}, training episode T_{tran}, learning rates λ and ξ.
Output: Optimal θ_f, β_n and action a_t.

▷ **Initialization**
Initialize model parameters θ and θ^*, $\theta^* = \theta$;
Initialize average utility $\overline{U} = 0$;
Initialize $t = 0$;
▷ **Implementation**
for $n \leq T_{tran}$ **do**
  while *True* **do**
    ▷ **Observing & Acting**
    Obtain the state $s(t)$ from VECN;
    Output the corresponding $\min Q_\theta(s(t), a(t))$ of actions from double Q networks;
    Sample action $a(t) \sim \pi$ from policy network;
    ▷ **Replaying**
    Refresh replay buffer;
    Execute $a(t)$, obtain the utility $U(s(t), a(t))$ and the following state $s(t+1)$, append the new experience tuple $\langle s(t), a(t), U(s(t), a(t)), s(t+1)\rangle$ to PER buffer $\mathbb{M}$;
    if $t \geq T_{pre}$ **then**
      break;
    end
  end
  $t \leftarrow t + 1$;
  Select a batch from PER buffer with *SumTree* method and calculate the target values;
  ▷ **Updating**
  Update Q-function of Critic parameters with $\beta_i \leftarrow \beta_i - \lambda \nabla_{\beta_i} J_Q(\beta_i)$, for $i \in \{1, 2\}$;
  Soft update target network of Critic parameters with $\overline{\theta}_f \leftarrow \psi\theta_f + (1-\psi)\overline{\theta}_f$, for $i \in \{1, 2\}$;
  Update policy function of Actor parameter with $\phi_i \leftarrow \phi_i - \lambda \nabla_{\phi_i} J_Q(\phi_i)$, for $i \in \{1, 2\}$;
end

easy to go for deployment in embedded platforms. The training of our proposed algorithm can be completed offline using historical data. Thus, the offline training time consumption of our proposed algorithm does not degrade system performance for online prediction.

D. Complexity Analysis

The LD-JRS3 algorithm consists of the LSTM algorithm for trajectory prediction and the improved SAC algorithm for generating strategies. Thus, the time complexity of the LD-JRS3 algorithm is the sum of the complexity of the LSTM algorithm and the complexity of the improved SAC algorithm. We first analyze the complexity of the LSTM algorithm. Its complexity depends on its input dimension X and output dimension Y of the LSTM, which can be expressed as $O(XY + Y^2)$ [25]. We then analyze the complexity of the improved SAC algorithm for strategy generation. This complexity is mainly affected by the time required to train the DNNs utilized by the actor and critic. According to [26], the complexity of the actor and the critic can be expressed as $O(W_m \times (\sum_{i=0}^{I-1} h_i \times h_{i+1} + \sum_{j=0}^{J-1} h_j \times h_{j+1}))$, where W_m denotes the batch size, and I and J are the number of fully connected layers in the actor and critic networks, respectively. In addition, h_i and h_j represent the number of neurons in the ith layer of actor and jth layer of critic, respectively. Therefore, the time complexity of the improved SAC algorithm can be estimated as $O(W_m \times g \times T \times (\sum_{i=0}^{I-1} h_i \times h_{i+1} + \sum_{j=0}^{J-1} h_j \times h_{j+1}))$, where g represents the number of episodes in the training process and T represents the duration of one slot. Meanwhile, the time complexity of PER method is $O(log_2(W_m))$ [27]. Therefore, the complexity of the LD-JRS3 algorithm can be expressed as $O(XY + Y^2 + log_2(W_m) + W_m \times g \times T \times (\sum_{i=0}^{I-1} h_i \times h_{i+1} + \sum_{j=0}^{J-1} h_j \times h_{j+1}))$.

V. Simulation Results and Analysis

We simulate our proposed LD-JRS3 algorithm to verify the convergence performance and also to illustrate the impact of critical parameter on system performance. We then show the efficiency analysis compared with other baseline algorithms. The whole experiment is implemented by the TensorFlow frame and runs on a PC with Intel Core i9-10980XE CPU @3.00 GHz, Memory 32G, and GPU for NVIDIA GeForce RTX 3090.

A. Simulation Settings

The experimental data for the simulation experiments used vehicle trajectory data collected by the Federal Highway Administration for the next generation simulation (NGSIM) project located near Lankershim Boulevard. Within this experimental field, we deployed 16 base stations equipped with edge servers with the same computation and storage capacity, each covering an area of a square region centered on the server with a side length of 1 km, which can completely cover our study roadway, and the tasks reaching the mobile users and edge servers satisfy the Poisson distribution within each time slot. Moreover, in this setting, we consider that each base station has a bandwidth of 20 MHz and a computational capability of 100 GHz. In the initial state of the system ($t = 0$), all vehicles in the system access the server closest to the vehicle location. The detailed parameter setting is listed in Table III.

B. Simulation Results

1) Convergence Performance: To ensure the reliability of our proposed LD-JRS3 algorithm, we first verify its convergence performance.

Fig. 5 demonstrates the trajectory prediction results for a portion of the vehicles, which illustrates the change in longitude information and latitude information of the vehicles, and the corresponding horizontal and vertical coordinate values indicate the relative position of the vehicle with respect to the reference point. Note that the red line represents the trajectory of the corresponding moment obtained by the algorithm based on the prediction of the known information of the vehicle,

TABLE III
SIMULATION PARAMETERS

Parameters	Values
Mobile users	96
Service nodes	16
Server computational capability	100 GHz
Service instance size	[1,10] MB
Offloading service size	[0.5,1] MB
Processing density	3000 CPU cycle/bit
Backhaul network bandwidth	500 Mb/s
Positive coefficient of migration latency	1 s/hop
Positive coefficient of backhaul latency	0.05 s/hop
Offloading task generation rate	1 unit/slot
Energy consumption for migration	[50,100] mW
Noise power	-114 dBm
Optimizer	Adam [23]
Activation function	ReLU/GELU
Learning rate of utility	0.001, 0.0001, 0.00001
Learning rate of neural network	0.0001
Batch size	400, 500, 600
Cached experience replay buffer size	1×10^5
Iteration epoch	100
Training step	2000

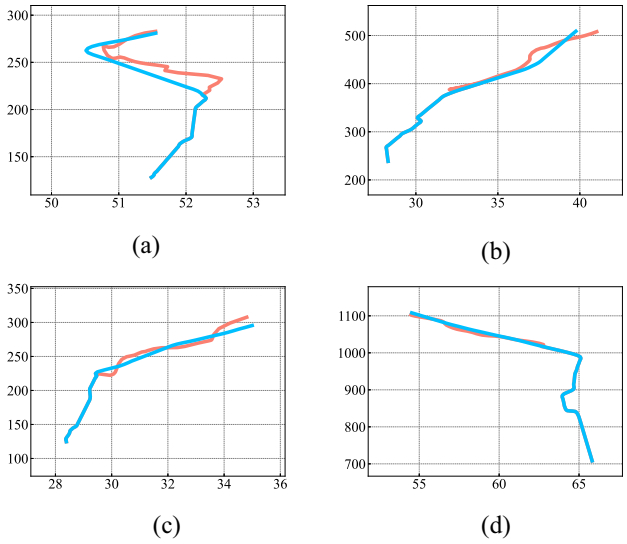


Fig. 5. Comparison of predicted and actual trajectories. (a) User A. (b) User B. (c) User C. (d) User D.

and the blue line represents the trajectory coordinates of the vehicle that actually traveled at the corresponding moment. Based on the predicted trajectories obtained, we calculated the average absolute value of the distance deviation between the predicted trajectories and the actual trajectories of the algorithm in the horizontal and vertical directions at different prediction time slot lengths, and the calculation results are shown in Table IV. We can see that the prediction accuracy of our proposed LSTM-based trajectory prediction algorithm can reach an error of less than 0.8 m in a 2-s prediction time slot, and the algorithm can still keep the distance error averaged within 1.5 m in a 7-s prediction time slot, which is basically negligible compared to the edge server coverage ($R \leq 500$ m) in the service migration scenario.

TABLE IV
TRAJECTORY PREDICTION MODEL ERROR (M)

Time slot	Longitude error	Latitude error	Distance error
$t = 1$	0.07	0.35	0.35
$t = 2$	0.07	0.75	0.75
$t = 3$	0.08	1.05	1.06
$t = 4$	0.08	1.16	1.17
$t = 5$	0.09	1.21	1.22
$t = 6$	0.10	1.34	1.35
$t = 7$	0.11	1.43	1.44
$t = 8$	0.15	1.69	1.70
$t = 9$	0.22	2.53	2.53

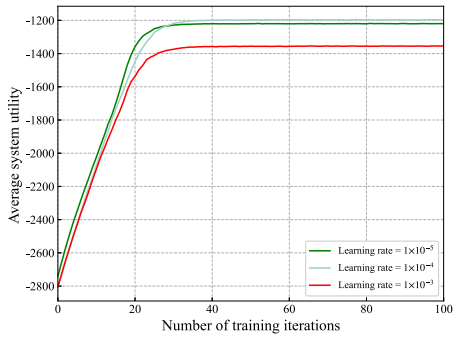


Fig. 6. System utility under the LD-JRS3 algorithm for different learning rates.

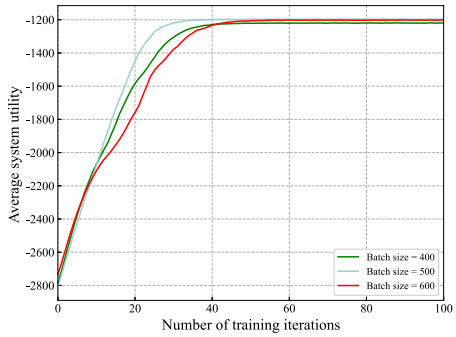


Fig. 7. System utility under the LD-JRS3 algorithm for different batch sizes.

Fig. 6 illustrates the average system utility under the LD-JRS3 algorithm at various learning rates. It is clear that the setting of the learning rate has an effect on the utility curve of the LD-JRS3 algorithm, and choosing the appropriate learning rate allows the algorithm to converge to a higher average system utility. It can be explained as follows. A learning rate that is too high can cause the agent to take excessively large update steps, leading to oscillations around the optimal solution. Conversely, a learning rate that is too low results in slow updates, which may prevent the agent from adapting to environmental changes in a timely manner. Based on these observations, we decided to set the learning rate at 0.0001 for subsequent experiments.

Fig. 7 displays the average system utility under the LD-JRS3 algorithm for various batch sizes. It is clear that batch size has an impact on the utility of the LD-JRS3 algorithm implementation, and the appropriate batch size helps the model accelerate convergence. It can be explained as follows.

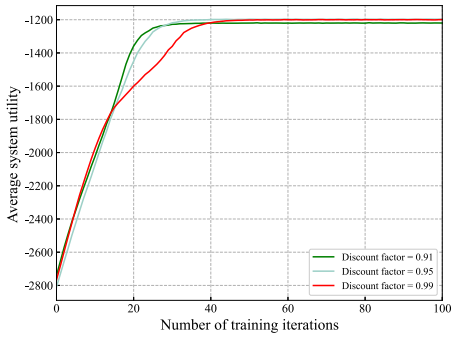


Fig. 8. System utility under the LD-JRS3 algorithm for different discount factors.

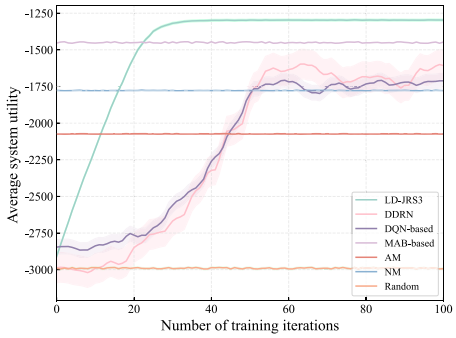


Fig. 9. System utility under the LD-JRS3 algorithm compared to these under the baseline algorithms.

First, larger batch sizes enrich the empirical samples necessary for model training, thereby accelerating model convergence. However, excessively large batch sizes may cause the model to settle into local optimum, thus affecting the convergence ability of the algorithm. Based on these insights, we have chosen to set the batch size to 500 for subsequent experiments.

Fig. 8 illustrates the average system utility under the LD-JRS3 algorithm with varying discount factors, i.e., $\gamma \in \{0.91, 0.95, 0.99\}$. It is evident that the discount factor has an effect on both the utility performance and convergence behavior. A lower discount factor of $\gamma = 0.91$ results in faster convergence but at the expense of lower utility. In contrast, a higher discount factor of $\gamma = 0.99$ yields greater utility but with less stable convergence performance. This can be explained as follows. A smaller γ prioritizes immediate rewards, simplifying the training process. On the other hand, a larger γ emphasizes long-term benefits, which can complicate convergence. Based on these findings, we have decided to set the discount factor to $\gamma = 0.95$ in our subsequent experiments.

2) Efficiency Analysis: We compare our proposed LD-JRS3 algorithm with the following benchmark algorithms.

R-Learning Empowered Double Deep Q-Learning Algorithm (DDRN): Existing work [6] solves the optimization problem based on the algorithm of *R*-learning empowered double deep *Q*-learning, which is more concerned with the long-term average revenue of the users. The vehicle will select the best action based on the maximum system utility. Notably, the algorithm uses a centralized training and distributed execution model.

Deep-Q-Learning-Based Algorithm (DQN-Based): Existing work [19] addresses the problem of service migration with an algorithm based on a two-stream DQN, an algorithm that efficiently learns and optimizes action strategies in complex environments. The vehicle will select the optimal server for service migration based on the maximum *Q*-value. Notably, the algorithm uses a centralized training and distributed execution model.

MAB-Based Algorithm (MAB-Based): Existing work [20] formulates the dynamic service migration problem as a contextual multiarm bandit (MAB) problem, and then proposes an online learning algorithm based on Thompson sampling to explore the dynamic MEC environment.

Always Migrate Algorithm (AM): The running instance will AM to the server closest to the vehicle according to the vehicle's movement trajectory.

Always Non-Migration Algorithm (NM): The running instance will always maintain execution on the original server until the task is completed, and the vehicles covered by the original server will receive the service content by means of multihop data transmission between servers.

Random Migration Algorithm (Random): The running instance will be randomly migrated to an optional server when the vehicles are far away from the coverage of the current server.

Fig. 9 shows the convergence of our proposed scheme compared to other algorithms. We set the iteration period to 100 epochs, and one epoch is set to be 2000 steps. We can observe that the LD-JRS3 algorithm and the MAB-based algorithm obtain higher system utilities. Our algorithm obtains higher system returns. In addition, the algorithm proposed in this article also slightly outperforms the DQN-based algorithm in terms of the stability of the total gain of the system. The reason for this can be explained as follows. Our algorithm and DQN-based algorithm use randomized actions to explore the environment in the early stages of training, where it has lower system returns before 30 rounds. Our algorithm converges faster because of the prioritized sampling scheme. It also combines the advantages of SAC, i.e., overestimation bias is reduced by using a Gaussian strategy to output continuous action values and employing a dual-*Q*-learning architecture. Further, incorporating the prioritization feedback mechanism empowered by RNNs, computational and bandwidth resources are rationally allocated. The model can be trained offline and predicted online. In this article, both time and energy costs together constitute the system overhead. We supplement the experiment with time and energy related costs as shown in Figs. 10 and 11. The experimental results demonstrate that our proposed scheme can minimize unnecessary time consumption and energy consumption while maintaining QoS. Fig. 12 shows the convergence of entropy loss, which also illustrates the convergence performance of our proposed algorithm.

Fig. 13 shows the impact of different task generation rates on the average system overhead. In our simulation, we incrementally increased the offloading task generation rate

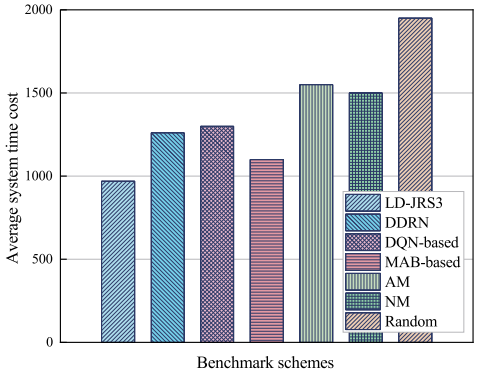


Fig. 10. Average system time cost under the LD-JRS3 algorithm compared to these under the baseline algorithms.

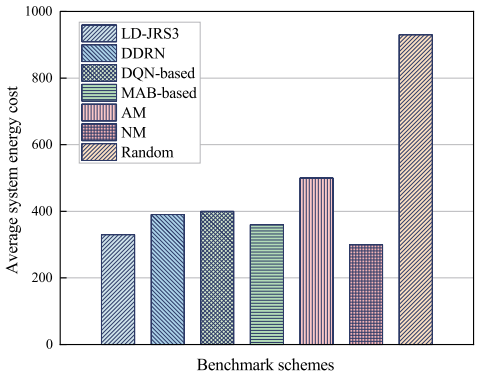


Fig. 11. Average system energy cost under the LD-JRS3 algorithm compared to these under the baseline algorithms.

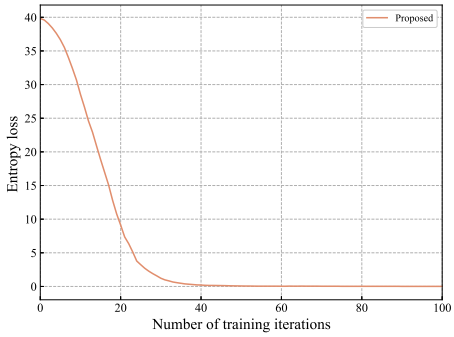


Fig. 12. Convergence performance of entropy loss for the LD-JRS3 Algorithm.

from 1 unit to 6 units in each time slot, while setting the migration cost at 2 s/hop. The figure clearly shows that as the generation rate of the offloading tasks increases, the average system overhead also increases. This increase is due to the fact that processing more offloading tasks requires more CPU cycles, which leads to an increase in computational latency. In addition, the figure shows that our proposed algorithm significantly outperforms benchmark algorithms, such as DDRN, DQN-based, and MAB-based algorithms under different task generation rates. Notably, the random algorithm incurs the highest total overhead. This is likely due to the fact that the algorithm migrates service instances randomly, which introduces unnecessary additional overheads.

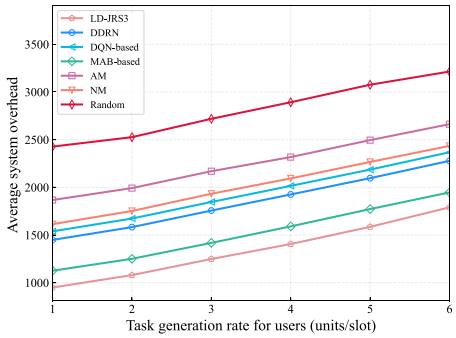


Fig. 13. Impact of different task generation rates on the average system overhead.

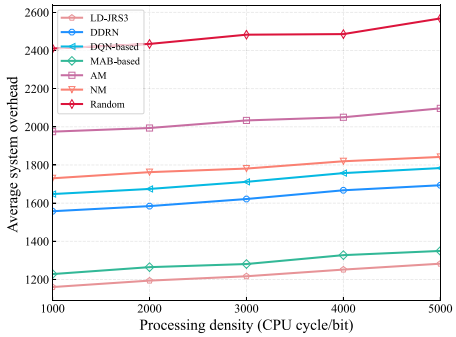


Fig. 14. Impact of different processing densities on the average system overhead.

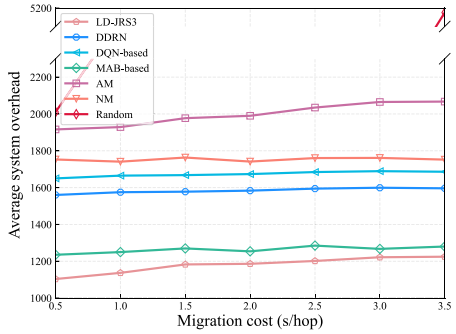


Fig. 15. Impact of different migration costs on the average system overhead.

Fig. 14 shows the impact of different processing densities on the average system overhead. Our findings show that the LD-JRS3 algorithm adapts well to changes in processing density and consistently outperforms all baseline algorithms. The reasons for this superior performance are explained as follows. First, changes in processing density primarily affect computational latency. LD-JRS3, a learning-based algorithm, is adept at avoiding server overload, which in turn reduces computational latency. This adaptability is key in maintaining high system efficiency under varying load conditions. Second, the LD-JRS3 algorithm integrates a resource scheduling method that optimizes the migration process. By doing so, it minimizes unnecessary migrations, further enhancing the overall efficiency and effectiveness of the system. These features collectively contribute to the LD-JRS3 algorithm's ability to manage processing density changes more effectively than traditional algorithms, thereby ensuring better performance across various scenarios.

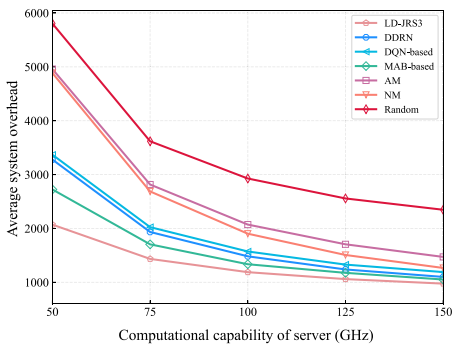


Fig. 16. Impact of different computational capabilities on the average system overhead.

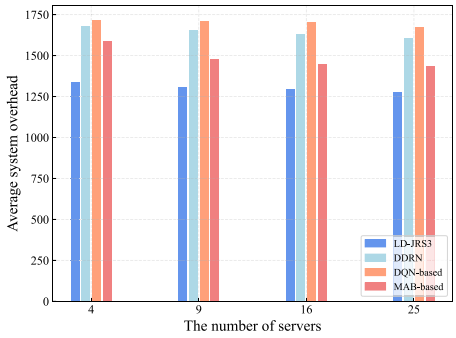


Fig. 17. Impact of different servers on the average system overhead.

Fig. 15 shows the impact of different migration costs on the average system overhead, with migration task data size and offloading task generation rate kept constant in the environment. The generation rate of offloading tasks is set at 3 units per second, and migration costs range within $\{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5\}$. It is clear from the figure that the average system overhead of the random algorithm increases dramatically with the migration cost. Meanwhile, the average system overhead of the NM algorithm remains constant, aligning with typical expectations for this strategy. Additionally, when comparing the other algorithms, our algorithm demonstrates certain advantages. The reasons are explained as follows. Migration cost is a crucial factor influencing the average system overhead and is directly linked to the algorithm's effectiveness within its environment. The LD-JRS3 algorithm manages to balance time and energy overheads through efficient resource scheduling and server selecting strategies. This method not only enhances the overall system performance but also improves resource utilization efficiency, thus minimizing unnecessary time consumption and energy consumption while maintaining QoS.

Fig. 16 shows the impact of different computational capabilities on the average system overhead, with server computational capacities sampled from the set $\{50, 75, 100, 125, 150\}$ GHz. It is evident from the graph that increasing the server's computational capability significantly reduces average system overhead. For servers with insufficient computational capacity, computational latency increases substantially, contributing to the average system overhead. As shown, enhancing the server's computational capability from 50 to 100 GHz decreases the average system overhead by nearly one-half. However, further increments in computational capability beyond 100 GHz result in diminishing returns, with the computational capability contribution to total overhead gradually decreasing and the incremental benefits of performance improvements becoming less significant. This phenomenon can be explained by the law of diminishing returns, where initial increases in computational capability substantially alleviate bottlenecks in processing speed, thereby reducing latency and overhead. Once the system reaches an optimal level of computational capability, additional increases yield smaller improvements in performance, as the system overhead becomes less sensitive to changes in computational capability. The experimental sections of this study are therefore conducted with the server's computational capability set at 100 GHz to optimize the balance between performance and resource usage.

Fig. 17 shows the impact of different number of servers on the average system overhead. In our experiments, we selected the number of servers from the set of $\{4, 9, 16, 25\}$ to evaluate the performance of the different algorithms. The figure compares the LD-JRS3 algorithm with other benchmark algorithms. We find that the average system overhead for the random strategy increases significantly as the number of servers decreases, highlighting its inefficiency in managing fewer resources. In contrast, the LD-JRS3 algorithm demonstrates a more stable ability to maintain lower average system overhead, regardless of the number of servers. This enhanced performance of the LD-JRS3 algorithm is due to its effective resource allocation and server selecting mechanisms, which optimize computational and networking resources better than the comparative strategies, particularly in scenarios where server resources are limited.

Under different network loads and different server capacities, the LD-JRS3 algorithm observes new environmental state information to retrain the generation policy to obtain a certain set of action spaces thus guaranteeing the maximization of the average system utility. The algorithm employs an RNN-based feedback strategy and an MDP-based reinforcement learning strategy. First, reinforcement learning is used to obtain the optimal actions (i.e., including the number of allocated blocks of computational resources and the index of the selected serving server). Further, the feedback strategy is used to determine different priorities, so that network resources (i.e., channel bandwidth) and edge server computational resources are provided proportionally according to the priority coefficients.

This ability to efficiently manage overhead, even with fewer servers, underscores the robustness and adaptability of the LD-JRS3 algorithm.

We scaled up the experimental size of the edge network, where one edge server equipped at one BS can serve 200 mobile users. Fig. 18 shows the convergence of the LD-JRS3 algorithm. Similar to the input size, our proposed algorithm ensures stable convergence.

VI. Conclusion

This article explored the service migration optimization for system overhead minimization in wireless edge networks. Specifically, we formulated it as a nonlinear and nonconvex

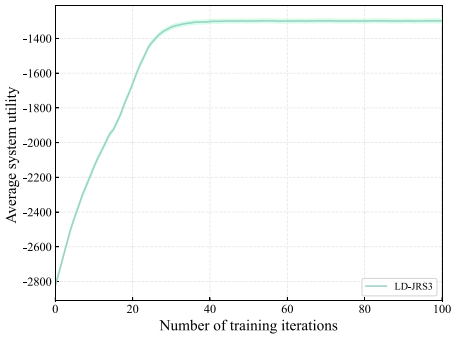


Fig. 18. System utility under the LD-JRS3 algorithm (N = 200).

optimization problem with the aim of minimizing average system overhead based on resource scheduling and server selecting. For this purpose, we proposed an LD-JRS3 algorithm that fully takes advantage of both RNN and SAC algorithm. Simulation results show that our proposed LD-JRS3 algorithm can achieve higher system utility (i.e., lower system overhead) in comparison with other benchmark algorithms, and LD-JRS3 is able to achieve high performance, with 19%, 24% and 11% higher utility than the DDRN, DQN-based, and MAB-based systems, respectively.

In this article, we select the cooperative MBS equipped with edge servers based on utility functions. The consumption of multiple cooperating MBS is an interesting problem that we will consider in future work. The blockchain technology can be provided to reach consensus among individual users. This is an interesting issue of data security and privacy protection, which we will further explore in our subsequent work. DRL combined with frame-stacking is very constructive advice and this is an interesting direction that we will delve into further in our future work. DRL clipping combined with frame stacking techniques can help reduce the computational overhead, which is also an interesting topic for our future work.

REFERENCES

[1] M. Huang, W. Liu, T. Wang, A. Liu, and S. Zhang, "A cloud–MEC collaborative task offloading scheme with service orchestration," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5792–5805, Jul. 2020.

[2] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[3] G. Hong, B. Yang, W. Su, H. Li, Z. Huang, and T. Taleb, "Joint content update and transmission resource allocation for energy-efficient edge caching of high definition map," *IEEE Trans. Veh. Technol.*, vol. 73, no. 4, pp. 5902–5914, Apr. 2024.

[4] Y. Xu, H. Zhang, H. Ji, L. Yang, X. Li, and V. C. M. Leung, "Transaction throughput optimization for integrated blockchain and MEC system in IoT," *IEEE Trans. Wireless Commun.*, vol. 21, no. 2, pp. 1022–1036, Feb. 2022.

[5] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Fast service migration in 5G trends and scenarios," *IEEE Netw.*, vol. 34, no. 2, pp. 92–98, Mar./Apr. 2020.

[6] A. Mukhopadhyay, G. Iosifidis, and M. Ruffini, "Migration-aware network services with edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1458–1471, Jun. 2022.

[7] Y. Chen, Y. Sun, C. Wang, and T. Taleb, "Dynamic task allocation and service migration in edge-cloud IoT system based on deep reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 16742–16757, Sep. 2022.

[8] Y. Peng et al., "Computing and communication cost-aware service migration enabled by transfer reinforcement learning for dynamic vehicular edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 257–269, Jan. 2024.

[9] E. Volnes, T. Plagemann, and V. Goebel, "To migrate or not to migrate: An analysis of operator migration in distributed stream processing," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 1, pp. 670–705, 1st Quart., 2024.

[10] X. Chen et al., "Dynamic service migration and request routing for microservice in multicell mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13126–13143, Aug. 2022.

[11] Z. Liang, Y. Liu, T.-M. Lok, and K. Huang, "Multi-cell mobile edge computing: Joint service migration and resource allocation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 5898–5912, Sep. 2021.

[12] J. Xu, X. Ma, A. Zhou, Q. Duan, and S. Wang, "Path selection for seamless service migration in vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 9040–9049, Sep. 2020.

[13] X. Zhou, S. Ge, T. Qiu, K. Li, and M. Atiquzzaman, "Energy-efficient service migration for multi-user heterogeneous dense cellular networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 890–905, Feb. 2023.

[14] Q. Yuan, J. Li, H. Zhou, T. Lin, G. Luo, and X. Shen, "A joint service migration and mobility optimization approach for vehicular edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 9041–9052, Aug. 2020.

[15] X. Xu, L. Yao, M. Bilal, S. Wan, F. Dai, and K.-K. R. Choo, "Service migration across edge devices in 6G-enabled Internet of Vehicles networks," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 1930–1937, Feb. 2022.

[16] Z. Ning, H. Chen, E. C. H. Ngai, X. Wang, L. Guo, and J. Liu, "Lightweight imitation learning for real-time cooperative service migration," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1503–1520, Feb. 2024.

[17] A. Dalgkitsis, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Data driven service orchestration for vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4100–4109, Jul. 2021.

[18] L. Liu, J. Feng, X. Mu, Q. Pei, D. Lan, and M. Xiao, "Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 15513–15526, Dec. 2023.

[19] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.

[20] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 1468–1476.

[21] J. Wang, J. Hu, G. Min, Q. Ni, and T. El-Ghazawi, "Online service migration in mobile edge with incomplete system information: A deep recurrent actor–critic learning approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 11, pp. 6663–6675, Nov. 2023.

[22] X. Gao, J. Wang, and M. Zhou, "The research of resource allocation method based on GCN-LSTM in 5G network," *IEEE Commun. Lett.*, vol. 27, no. 3, pp. 926–930, Mar. 2023.

[23] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[24] X. Gao, Y. Sun, H. Chen, X. Xu, and S. Cui, "Joint computing, pushing, and caching optimization for mobile-edge computing networks via soft actor–critic learning," *IEEE Internet Things J.*, vol. 11, no. 6, pp. 9269–9281, Mar. 2024.

[25] T. Ergen and S. S. Kozat, "Online training of LSTM networks in distributed systems for variable length data sequences," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 5159–5165, Oct. 2018.

[26] A. R. Heidarpour, M. R. Heidarpour, M. Ardakani, C. Tellambura, and M. Uysal, "Soft actor–critic-based computation offloading in multiuser MEC-enabled IoT—A lifetime maximization perspective," *IEEE Internet Things J.*, vol. 10, no. 20, pp. 17571–17584, Oct. 2023.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016, *arXiv:1511.05952*.

Yuan Yuan (Graduate Student Member, IEEE) was born in 1997. He received the bachelor's degree from Beijing Jiaotong University, Beijing, China, in 2018, where he is currently pursuing the doctoral degree with the National Engineering Research Center of Advanced Network Technologies.

He is mainly engaged in the research of the vehicular network and multiaccess edge computing.

Bin Yang received the Ph.D. degree in systems information science from Future University, Hakodate, Japan, in 2015.

He was a Research Fellow with the School of Electrical Engineering, Aalto University, Espoo, Finland, from November 2019 to 2021. He is currently a Professor with the School of Computer and Information Engineering, Chuzhou University, Chuzhou, China. His research interests include unmanned aerial vehicle networks, mobile edge computing, cyber security, and Internet of Things.

Wei Su was born in October 1978. He received the Ph.D. degree in communication and information systems from Beijing Jiaotong University, Beijing, China, in 2008.

He is currently a Teacher with the School of Electronic and Information Engineering, Beijing Jiaotong University, where he granted the title of a Professor in 2015. He is mainly engaged in researching key theories and technologies for the next generation Internet and has taken part in many national projects, such as National Basic Research Program (also called 973 Program), the Projects of Development Plan of the State High Technology Research, and the National Natural Science Foundation of China. He currently presides over the research project fundamental research on cognitive services and routing of future Internet, a project funded by the National Natural Science Foundation of China.

Jie Ma (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China.

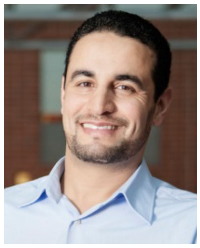
His research interests include future network architecture and network security.

Yihua Peng (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China.

His primary research interests include network security and in-band network telemetry.

Qi Liu received the B.S. degree in information and communication engineering and the Ph.D. degree in communication and information system from Beijing Jiaotong University, Beijing, China, in 2003 and 2009, respectively.

She was a Postdoctoral Researcher with the Electronic Engineering Department, Tsinghua University, Beijing, from 2009 to 2011. She is currently a Professorate Senior Engineer with Smart City Research Institute of China Unicom, Beijing. Her research interests focus on 5G, cooperation of heterogeneous networks, Internet of Vehicles, and high-precision positioning.



Tarik Taleb (Senior Member, IEEE) received the B.E. degree (with Distinction) in information engineering and the M.Sc. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 2001, 2003, and 2005, respectively.

He is currently a Full Professor with the Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, Germany. Prior to that, he is currently a Professor with the Centre for Wireless Communications (CWC)–Networks and Systems Unit, Faculty of Information Technology and Electrical Engineering, The University of Oulu, Oulu, Finland. He is the Founder and Director of the MOSA!C Laboratory (www.mosaic-lab.org), Aalto University, Espoo, Finland, where he was a Professor with the School of Electrical Engineering from October 2014 to December 2021. Prior to that, he was working as a Senior Researcher and a 3GPP Standards Expert with NEC Europe Ltd., Heidelberg, Germany. Before joining NEC and till March 2009, he worked as an Assistant Professor with the Graduate School of Information Sciences, Tohoku University, Sendai, Japan, in a Lab fully funded by KDDI, the second largest mobile operator in Japan. From October 2005 to March 2006, he was a Research Fellow with the Intelligent Cosmos Research Institute, Sendai. He has been also directly engaged in the development and standardization of the Evolved Packet System as a member of 3GPP's System Architecture Working Group. His research interests lie in the field of telco cloud, network softwarization and network slicing, AI-based software-defined security, immersive communications, mobile multimedia streaming, and next-generation mobile networking.

Prof. Taleb served as the General Chair for the 2019 edition of the IEEE Wireless Communications and Networking Conference, Marrakech, Morocco. He was the Guest Editor-in-Chief of the IEEE JSAC SERIES ON NETWORK SOFTWARIZATION AND ENABLERS. He was on the editorial board of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, *IEEE Wireless Communications Magazine*, IEEE JOURNAL ON INTERNET OF THINGS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, and a number of Wiley journals. Till December 2016, he served as the Chair of the Wireless Communications Technical Committee.