Lightweight Imitation Learning for Real-Time Cooperative Service Migration

Zhaolong Ning, Handi Chen, Edith C. H. Ngai, Xiaojie Wang, Lei Guo and Jiangchuan Liu, *Fellow, IEEE*

Abstract—Due to the revolution of communication technology, the rapidly increasing number of mobile devices in edge networks generates various real-time service requests, requiring a considerable volume of heterogeneous resources all the time. However, edge devices with limited resources cannot afford substantial learning cost, while migrating services requires heterogeneous resources, especially for dynamic networks. To address these issues, we first establish a cooperative service migration framework and formulate a bi-objective optimization problem to optimize service performance and cost. By analyzing the optimal migration ratio of service cooperative migration, we propose an offline expert policy based on global states to provide optimal expert demonstrations. To realize real-time service migration based on observable states, we design a lightweight online agent policy to imitate expert demonstrations and leverage meta update to accelerate the model transfer. Experimental results show that our algorithm is exceptional in training cost and accuracy, and has significant superiors in multiple metrics such as the service latency and payment under different workloads, compared to other representative algorithms.

Index Terms—Service migration, resource cooperation, imitation learning, dynamic wireless network.

1 Introduction

Enhanced mobile broadband has driven 5G to become the commercial reality. With the transition to 6G, the rapid expansion of smart devices and the explosive growth of real-time applications emerge state-of-art service requirements, such as holographic communication, digital twin and augmented reality, generating a huge amount of data requiring timely processing [1]. It is reported in [2] that global mobile traffic will reach 1 ZB/month in 2028, which is equivalent to 5 billion users worldwide costing 200 GB/month. The urgent computation capacity requirements are significant challenges to resource-limited edge networks. The incomplete functionality of current devices results in the conflict between restricted timeliness requirements of substantial services and limited edge resources.

1.1 Motivation

The costly expense of updating and extending hardware limits the development of novel service commercialization. Resources, including computation, communication, and caching, are reserved based on requirements declared by service session to guarantee real-time performance. However, service execution requires heterogeneous resources among multiple edge devices, highly depending on global network states. Since information is isolated on independent devices, edge devices cannot observe global states due to limited communication capabilities. However, frequent interactions with the central node (such as base station or other infrastructures with powerful sensors) aggravate the network burden and threaten private information. Consequently, a fundamental problem is how to design lightweight and distributed agent policies to enable devices for autonomous service cooperation with optimal decisions in real time, especially for dynamic edge networks. Satisfying service requirements with high throughput and data volume is important and necessary, motivating us to study this topic.

1.2 Research Challenges

Heterogeneity, ultra-dense and time-sensitivity characteristics in edge networks limit the efficiency of service provision in real time, which is further exacerbated by the unstable communication links and the fragmented edge resources. The research challenges are listed as follows:

- The competition of insufficient edge resources is fiercer among mobile devices with limited energy. Single Service Provider (SP) not only increases the rent burden but also lowers resource utilization efficiency. Thus, how to schedule services and combinatorially manage heterogeneous resources to optimize the Quality of Experience (QoE) of Service Requesters (SRs) deserves to be discussed.
- Users need to be given incentives to provide their resources. Thus, it is necessary to design an efficient pricing mechanism for incentivizing devices and assist SRs by making a satisfying trade-off between the stable but competitive infrastructure resources and the fragmented but available device resources.
- Generally speaking, leveraging learning algorithms as an advanced tool to make decisions intelligently is a promising approach that many studies have

- *Z. Ning, X. Wang (Corresponding author) and L. Guo are with the School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. Email: z.ning@ieee.org, xiaojie.kara.wang@ieee.org, guolei@cqupt.edu.cn.*
- *H. Chen and E. Ngai are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong 999077, China. Email: hardychn@outlook.com, chngai@eee.hku.hk.*
- *J. Liu is with the School of Computing Science, Simon Fraser University, British Columbia, V5A1S6, Canada. Email: jcliu@cs.sfu.ca.*

demonstrated [3], [4]. However, the training cost, communication load, and convergence speed lead to a decline in the quality of real-time service. Designing an online-enabled lightweight learning policy for distributed making decisions is rather challenging.

1.3 Contributions

To solve the above-mentioned issues, we construct a service cooperative migration framework integrating computation, communication and caching resources. The mobile terminals are viewed as SPs to address the shortage of public infrastructure resources. The cooperative service migration problem is formulated as a bi-objective optimization problem. A distributed online agent policy is designed to solve the problem by imitating the expert demonstrations generated by the offline expert policy. The main contributions are summarized as follows:

- We construct an intelligent service cooperative migration framework based on resource combining optimization and design a pricing mechanism to reflect the willingness of service cooperation. The problem is formulated as a bi-objective optimization problem to minimize execution latency and payment. The bi-objective problem is decomposed into selecting executing devices and deciding the optimal migration ratio by analyzing the optimal execution latency.
- To solve the first sub-problem, we propose an imitation Learning-based Online Service cooperative migration policy (LOS). An offline expert policy is proposed to generate expert demonstrations for agents. By analyzing the optimal migration ratio, the results of expert demonstration are proved to be the optimal results.
- We design a lightweight online agent policy to make decisions distributedly in real time by imitating the obtained expert demonstrations. To overcome the staleness of expert demonstrations, we apply meta learning to accelerate transferring model for updating agent policy and reduce the training workload for continuous imitation learning.
- The real-world dataset of Hangzhou (China) is utilized to illustrate the effectiveness of our proposed policy from communication, computation, and caching aspects. Compared with representative schemes, experimental results demonstrate the superiority of our proposed policies in multiple metrics.

The rest of this paper is organized as follows. In Section 2, we review the related work about service migration and imitation learning in detail. Then, we describe the system model and problem formulation in Section 3. In Section 4, we design an imitation learning-based service cooperative migration policy with an offline expert policy and online agent policy. Performance evaluations are discussed in Section 5, followed by a conclusion in Section 6.

2 Related Work

In this section, we review the state-of-art of service migration from 5G to 6G and the applications of imitation learning.

2.1 Service Migration

In recent years, many studies have explored possible solutions for improving service processing efficiency. The studies in [5], [6], [7] explore the optimization of communication resources to improve network performance and promote services in 5G/B5G. Pokhrel *et al.* [8] propose a multi-path transmission control protocol by jointly integrating load balancing and forward error correction for packet loss caused by time-varying heterogeneous wireless paths. However, the improvement of millisecond time-sensitive 6G network performance by only considering the spectral efficiency is limited.

Xia *et al.* [9] study the collaborative caching problem in the resource-limited edge computing environment from the perspective of the SP, and propose an online collaborative edge data caching algorithm based on Lyapunov optimization to optimize the system cost and the QoE. Considering the compatibility of incenting vehicular users and individual rationality, Ning *et al.* [10] propose an efficient partial computing offloading and adaptive task scheduling algorithm and allocate computation resources for maximizing the overall system profit.

The studies in [11], [12], [13] leverage slicing technique to manage fine-grained heterogeneous resources. Chekired *et al.* [11] design a vertical structure of autonomous vehicles based on slicing technique to improve the QoS of autonomous driving applications. Faraci *et al.* [12] integrate communication and computation resources for low-latency application scenarios by controlling computing elements. To satisfy various slicing demands, an iterative auction game-based solution is proposed in [13] to optimize resource utilization and SRs' fairness.

The traditional network architecture cannot handle the millisecond latency requirement by new services. For satisfying time-sensitive demands, the network structure should be flattened for ultra-low latency in B5G and 6G, promoting the development of network virtualization technology. Fu *et al.* [14] propose a tapped water-filling algorithm to integrate computation, communication and caching resources to improve the network energy efficiency. Tang *et al.* [15] propose a novel resource sharing architecture to integrate heterogeneous resources. The proposed architecture can be flexible on device cooperation and resource allocation to minimize energy consumption. Luo *et al.* [16] design a fog-enabled caching, communication, computation resource sharing framework to allocate heterogeneous resources based on the auxiliary graph to minimize energy consumption. However, the ultra-dense time-sensitive network complicates the integration of heterogeneous resources to improve energy efficiency. Other related literature (published from 2020 to 2022) are discussed in Appendix A.

2.2 Imitation Learning

Imitation learning is proposed to enable distributed agents by imitating expert behaviors for making decisions autonomously. In [27], imitation learning methods are classified into engineering-oriented and biological-oriented methods, which are widely leveraged in robotic motion planning, human-computer interaction, autonomous driving and so on. The recent studies about imitation learning are reviewed

TABLE 1
Comparison of service/task migration methods.

Ref.	Domain	Learning method	Dataset	Advantages	Disadvantages
[17]	Games	Actor-critic algorithm	Atari Games and MuJoCo	Learn to reproduce the agent's past good experiences	Get stuck at a sub-optimal policy on a few games
[18]	Games	Stochastic mixing iterative learning	SuperTux Kart game and Mario Bros. game	Solve simple classification problems, without requiring roll outs of full cost-to-go	Fall quite often into gaps
[19]	Games	CoordConv	Mountain car, hopper, and 3 Atari games	Resolve naturally occurring and fundamental problems	The exact choice policy is relatively insignificant
[20]	Autonomous driving	Generative Adversarial Network (GAN)	The open source racing car simulator	Automatically distinguish certain behaviors in human driving and learn a policy that can imitate the human experts	Depend on visual information
[21]	Autonomous driving	A deep encoder-decoder	Real-world driving dataset	Drive on a full-size vehicle	Lack a predictive long-term planning model
[22]	Robotic	Meta learning	Collected tasks	Map a single successful demonstration of a task to an effective policy	Frequently fail on the hardest tasks
[23]	Robotic	ResNet18 "torso"	Collected robotic manipulation tasks	Enable a vision-based robotic manipulation system to generalize to novel tasks in training	Performance on novel tasks varies significantly
[24]	Navigation	Soft actor-critic	Simulation in AirSim, physical experiments on the Jackal	Robust to egocentric viewpoint mismatch	Not explicitly seek to overcome view-point mismatch
[25]	Navigation	Convolution Neural Network (CNN)	Simulation	Resolve ambiguities in the perceptuomotor mapping	The model trained without data augmentation fails completely
[26]	Physics-based control	GAN	MuJoCo	Model free and no interact with the expert	Need more environment interaction

in TABLE 1. Different from reinforcement learning which requires a specified reward function, the paradigm of imitation learning supports agents to be taught complex tasks with a little information, which transforms the training process to fitting expert demonstration distributions [28]. Standard imitation learning involves two participants: experts and agents. Expert demonstration datasets are constructed from optimal decisions made by expert nodes, and then expert policy π^e is sampled. Agents make decisions by mapping expert demonstrations through supervised learning, which is similar to an inverse reinforcement learning process. Since imitation is the process of predicting novel observed states based on stale trajectories, the loss of prediction accuracy increases significantly due to the non-independent and identically distributed (non-i.i.d) data. To reduce the increasing loss, methods such as DAgger [29] and Dart [30] focusing on the interactive access to expert nodes are proposed to enhance the learning performance. The former is an iterative algorithm to train a stationary deterministic policy, while the latter intends to alleviate the covariate shift by injecting artificial noise into the supervisor's policy.

By combining with sensors that can collect and transmit a huge volume of data, edge devices can make decisions quickly and map observable states to actions in real-time based on imitation learning. Wang *et al.* [31] design a distributed policy to address the loss generated by learning the partial observation in an edge network based on the tolerance of incomplete information. To relieve the compound error caused by supervised learning, generative adversarial network is leveraged to design a multi-agent policy combining imitation learning in [32] for distributed edge computing. Yan *et al.* [33] leverage imitation learning to accelerate the recursive process of the branch-and-bound algorithm to reduce the corresponding decision burden. Due to the partial observation of edge users, constructing neural networks with DAgger to predict opponents' actions based on sequential data is a common method to improve the performance of imitation learning [34].

Most existing studies focus on optimizing the speed and accuracy of imitation learning, while little research investigates the learning cost. Considering the computation-intensive applications with massive data in the network, Ryan *et al.* [35] design an architecture-independent imitation learning framework to reduce the computation time and hardware overheads. Anish *et al.* [36] propose an imitation learning framework to construct policies at runtime for task scheduling in heterogeneous many-core platforms and analyze the latency and storage cost, and the error accumulation problem is resolved by DAgger algorithm. Yang *et al.* [37] propose a method to accelerate the training process by self-imitation with its historical actions, and use the confidence-based matching method to ensure accuracy. However, the retraining cost still burdens the edge devices with limited

computation capacity.

Consequently, designing a lightweight imitation learning-based policy that can guarantee learning performance for real-time service migration is the main motivation of this paper.

3 System Model and Problem Formulation

This section first specifies system model, and then formulates the bi-objective optimization problem.

3.1 System Model

As shown in Fig. 1, the dynamic dense edge network can be divided into $\mathcal{R}$ domains according to the infrastructures. The system contains two logical roles, i.e., SRs and SPs. In principle, a physical device can be a SR and a SP simultaneously. To capture dynamic states, service migration is executed in discrete time slots denoted by $t \in \mathcal{T}$. At the beginning of each time slot, devices observe the environment state. Service generated by SRs can be partially migrated to other devices for execution. To respond to the migration requests in real time, dense edge servers with the same domain are utilized as SPs to reduce the communication burden in this system.

Service migration can be divided into three steps as shown in Fig. 2, i.e., input, execution and output. The input of services contains two parts, i.e., service data and data package which is required by the service category. For instance, map data packages are required by the navigation services. SRs decompose the migrated service into two parts for local and migrated execution respectively according to migration ratio $\gamma_i(t)$. Providing services in parallel can reduce cost and increase resource utilization by decentralizing workload.

In time slot t, the devices randomly arrived at the coverage of the infrastructure can be denoted by $\mathcal{D}(t) = \{D_1(t), D_2(t), \cdots, D_{n_t}(t)\}$, where n_t is the number of devices. The generated service of $D_i(t)$ is represented by $S_i(t)$. To indicate different services, we define $\mathcal{K} = \{K_1, K_2, ..., K_k\}$ to represent the service categories.

The characteristics of device $D_i(t)$ are denoted as follows:

Definition 1. ***Device Model:*** *We define device $D_i(t)$ ($D_i(t) \in \mathcal{D}(t)$) as a tuple:*

$$\begin{aligned} D_i(t) =& (S_i(t), D_i^{ca}(t), D_i^{speed}(t), D_i^{comp}(t), D_i^{down}(t), \\ & D_i^{tp}, D_i^{ene}(t), D_i^{loc}(t)), \end{aligned} \tag{1}$$

where $S_i(t)$ is the service tuple generated by device $D_i(t)$ in time slot t. $D_i^{ca}(t)$ denotes the data package category cached in device $D_i(t)$, where $D_i^{ca}(t) \in \mathcal{K} \cup \{0\}$ and $D_i^{ca}(t) = 0$ indicates no content is cached in $D_i(t)$. $D_i^{speed}(t)$ denotes the speed of dynamic device $D_i(t)$ (in meters per second); $D_i^{comp}(t)$ denotes the computation capacity (in CPU cycles per second); $D_i^{down}(t)$ denotes the download rate (in megabyte per second); D_i^{tp} denotes the fixed transmission power for communications (in Watt); $D_i^{ene}(t)$ indicates the remaining energy percentage of $D_i(t)$, and $D_i^{ene}(t) \in [0, 1]$; $D_i^{loc}(t)$ denotes the coordinate of $D_i(t)$, and $D_i^{loc}(t) = (x_i^t, y_i^t)$, where x_i^t and y_i^t are the longitude and latitude of $D_i(t)$ obtained by global positioning system, respectively.

Herein, the characteristics of requested service $S_i(t)$ in Definition 1 are detailed as follows:

Definition 2. ***Service Model:*** *Service $S_i(t)$ is defined as a tuple:*

$$S_i(t) = (K_i, S_i^{size}(t), S_i^{comp}(t), S_i^{out}(t)), \tag{2}$$

where K_i is the service category of $S_i(t)$ and $K_i \in \mathcal{K}$; $S_i^{size}(t)$ denotes the data size of task (in megabyte); $S_i^{comp}(t)$ denotes the required computation resources of task $S_i(t)$ (in CPU cycles); $S_i^{out}(t)$ is the data size of service result (in megabyte).

The characteristics of service category K_i is detailed as:

Definition 3. ***Service Type Model:*** *Service category K_i ($K_i \in \mathcal{K}$) is defined as a tuple:*

$$K_i = (K_i^{size}, K_i^{delay}), \tag{3}$$

where K_i^{size} denotes the data size of the data package required by i (in megabyte); K_i^{delay} denotes the delay tolerance of service K_i (in millisecond).

In addition to user terminals, infrastructure can also be utilized as SP. The characteristics of infrastructure $R(t)$ in time slot t are specified as follows:

Definition 4. ***Infrastructure Model:*** *We define the infrastructure as a tuple:*

$$R(t) = (R^{comp}(t), R^{down}(t), R^{tp}, R^{loc}, R^{ch}), \tag{4}$$

where $R^{comp}(t)$ denotes the available computation resource (in CPU cycles per second); $R^{down}(t)$ is the download rate of infrastructure (in megabyte per second); R^{tp} denotes the transmit power of infrastructure (in Watt); $R^{loc} = (x_r, y_r)$, where x_r and y_r are the longitude and latitude of the infrastructure, respectively; R^{ch} is the number of the sub-channels.

The other main notations are listed in TABLE 2.

3.2 Service Execution Model

The investigated scenario includes two communication modes, i.e., Device-to-Device (D2D) and Device-to-Infrastructure (D2I). The achievable communication rate between two devices can be calculated by the Shannon equation as follows:

$$r_{ij}^{comm}(t) = B_{ij} \log_2(1 + \Gamma_{ij}(t)), \tag{5}$$

where B_{ij} and $\Gamma_{ij}(t)$ represent the bandwidth and Signal to Interference plus Noise Ratio (SINR) in time slot t, respectively. If communicable distance between devices $D_i(t)$ and $D_j(t)$, where $D_i(t), D_j(t) \in \mathcal{D}(t)$, is within the communication range, the communication link can be constructed for data transmission.

To ensure the communication quality, we consider that each user terminal can communicate with only one device at the same time. The SINR between devices $D_i(t)$ and $D_j(t)$ is calculated by $D_i^{tp} D_{ij}^{cg}(t)^2 / \sigma^2$ with no interference, where σ^2 represents the additive white gaussian noise. Correspondingly, if the communicable distance and available sub-channel between device $D_i(t)$ and infrastructure $R(t)$ are satisfied, the communication link can be constructed

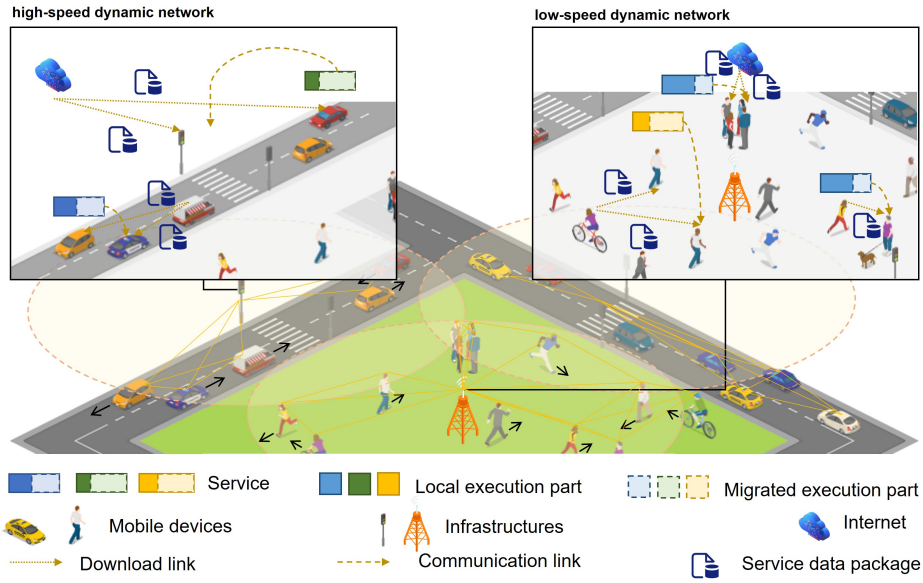


Fig. 1. Illustrative system model of service migration in dynamic networks.

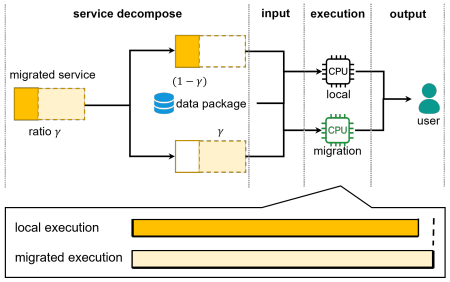


Fig. 2. Illustrative service migration.

based on non-Orthogonal multiple access, and the SINR $\Gamma_{ir}(t)$ can be calculated by:

$$\Gamma_{ir}(t) = \frac{D_i^{tp} D_{ir}^{cg}(t)^2}{(\sigma^2 + \sum_{i' \in \mathcal{D} \setminus i} D_{i'r}^{tp}(t) D_{i'r}^{cg}(t))}. \tag{6}$$

In time slot t, SPs may receive more than one transmission requests from other devices. Independent requests (including task data transmission requests and cached data package transmission requests) follow the first-come-first-service order, and the arrival of transmission requests follows the Poisson distribution. For device SP, there is only one service table, which can accommodate up to N requests. The states of communicable devices can be observed to evaluate the availability at the beginning of each time slot before making decisions for service migration. To avoid exceeding the transmission capacity of devices, if the length of the queue has already reached N, the device is labeled as unavailable for other SRs. The transmitted requests can be constructed as an *M/G/1* queuing model. Transmission waiting delay T_{ij}^{wait} can be calculated by:

$$T_{ij}^{wait} = \frac{(\lambda \bar{T}^{tran} + \lambda \vartheta^2)}{2(1 - \lambda \bar{T}^{tran})}, \tag{7}$$

where variables λ and $\bar{T}^{tran}$ are the intensity of task transmission and average transmission latency between two devices. ϑ^2 denotes the transmission latency variance. The latency of communication $T_{ij}^{comm}(t)$ can be computed by $T_{ij}^{tran}(t) + T_{ij}^{wait}$, where $T_{ij}^{tran}(t)$ represents the latency of task data transmission, which depends on the achievable transmission rate computed in (5).

When K_i-type service cannot be found among available devices, SPs need to download K_i from the Internet and cache it if the remaining storage resources are sufficient. The mobility of devices and the update of cached content on devices produce huge obstacles to predicting data package distribution. Many existing studies have investigated the distribution of caching content in edge networks [38], [39]. However, caching process is not our main consideration. Instead, this work focuses on the obtainment of cached data package. To simplify, we consider distinct data package cached in devices following random distribution. The cached data package can be shared with other communicable devices. Due to the scarcity of the D2I spectral resources, infrastructures can only download the data package from the Internet.

After obtaining the whole input, SPs can provide computation resources for the service execution. Based on the model mentioned above, the latency of executing service $S_i(t)$ on device $D_j(t)$ consists of four parts, namely service data obtainment latency, data package obtainment latency, execution latency, and feedback latency. To clarify the division of service for parallel execution on two devices, $\gamma_i(t)$ is used to indicate the migration ratio of service $S_i(t)$. Hence, $S_i(t)\gamma_i(t)$ and $S_i(t)(1 - \gamma_i(t))$ represent migration part and local part, respectively, $\gamma_i(t) \in [0, 1]$. If $\gamma_i(t) = 1$, service $S_i(t)$ is migrated completely. If $\gamma_i(t) = 0$, service $S_i(t)$ is executed locally. Correspondingly, the resources required by $S_i(t)$ are also divided by ratio $\gamma_i(t)$. We define binary variable $\alpha_{ij}(t)$ to indicate the migrated SP. When $\sum_{j=1}^{n_t} \alpha_{ij}(t) = 0$, the service is migrated to the infrastructure. When $j = i$, the service data and result do not need to be transmitted to SR, i.e., $T_{ii}^{comm}(t)$ and $T_{ii}^{back}(t)$ equal to 0. The binary variable $\beta_{ijh}(t)$ is defined to indicate the data

package sharing device. When $\sum_{h=1}^{n_t}\beta_{ijh}(t)=0$, the data package is obtained by download. Thus, the local execution latency can be computed as follows:

$$\begin{aligned}T_i^{loc}(t) =& T_i^{comp,loc}(t)+T_i^{dp,loc}(t)\\ =&\frac{(1-\gamma_i(t))S_i^{comp}(t)}{D_i^{comp}(t)}+\sum_{h=1}^{n_t}\beta_{iih}(t)(\frac{K_i^{size}}{r_{ih}^{comm}(t)}\\ &+T_{ih}^{wait})(1-\sum_{h=1}^{n_t}\beta_{iih}(t))\frac{K_i^{size}}{D_i^{down}(t)}.\end{aligned} \tag{8}$$

For migration, the latency can be computed as follows:

$$\begin{aligned}T_{ij}^{mig}(t) =& T_{ij}^{comm}(t)+T_{ij}^{comp}(t)+T_{ij}^{dp}(t)\\ =&\sum_{j=1,j\neq i}^{n_t}\alpha_{ij}(t)[\frac{\gamma_i(t)(S_i^{size}(t)+S_i^{out}(t))}{r_{ij}^{comm}(t)}+\\ &T_{ij}^{wait}+\frac{\gamma_i(t)S_i^{comp}}{D_j^{comp}}+\sum_{h=1}^{n_t}\beta_{ijh}(t)(\frac{K_i^{size}}{r_{ih}^{comm}(t)}\\ &+T_{ih}^{wait})+(1-\sum_{h=1}^{n_t}\beta_{ijh}(t))\frac{K_i^{size}}{D_i^{down}(t)}]+(1\\ &-\sum_{j=1,j\neq i}^{n_t}\alpha_{ij}(t))(\frac{\gamma_i(t)S_i^{comp}(t)}{R^{comp}(t)}+\frac{\gamma_i(t)S_i^{size}}{r_{ir}^{comm}}\\ &+\frac{K_i^{size}}{R^{down}(t)}).\end{aligned} \tag{9}$$

Since the local and the migrated parts are executed in parallel at the same time, the total service execution latency can be obtained by:

$$T_i(t)=\max\{T_i^{loc}(t),T_i^{mig}(t)\}, \tag{10}$$

that is the maximum value between local execution latency and migrated execution latency.

3.3 Rent Model

Due to the selfishness of users, a fairness incentive mechanism is necessary to motivate device cooperation. In our work, the unit rent price Φ_j^{comp} of computation resources is time-varying with the state of device $D_j(t)$ ($D_j(t)\in\mathcal{D}(t)$), which is defined by:

$$\Phi_j^{comp}(D_j^{comp}(t),D_j^{ene}(t))=\frac{1}{D_j^{comp}(t)}e^{\frac{\kappa}{D_j^{ene}(t)}}, \tag{11}$$

where parameter κ represents the price coefficient to adjust the impact of available computation capacity and remaining energy on unit rent. Herein, those two factors are negatively correlated with the unit rent to reflect the intention of renting resources for profit. Similar to Definition 4 in [31], the pricing function can be viewed as the product of two parts, (i.e., $1/D_j^{comp}(t)$ and $e^{\kappa/D_j^{ene}(t)}$), to reflect the distinct impacts of computation capacity and remaining available energy, respectively. We consider that the computation capacity (CPU frequency) can be released after completing service execution, while the battery level cannot be recovered until recharging. Considering recharging has been studied in many studies [40], [41], [42] which is fully compatible with the proposed system, the recharging case is not considered in this work. Since the increasing rate of exponential function is constantly greater than that of inverse function (which can be easily analyzed by the derivatives), we select an exponential function to evaluate the impacts of battery level, similar to the definition of the reward function, i.e., equation (4) in [43]. Then, we can derive the price of computation resources of each device. An example of rent with $\kappa=0.5$ is shown in Fig. 3, which illustrates SPs' willingness to provide services for SRs. The x-axis shows

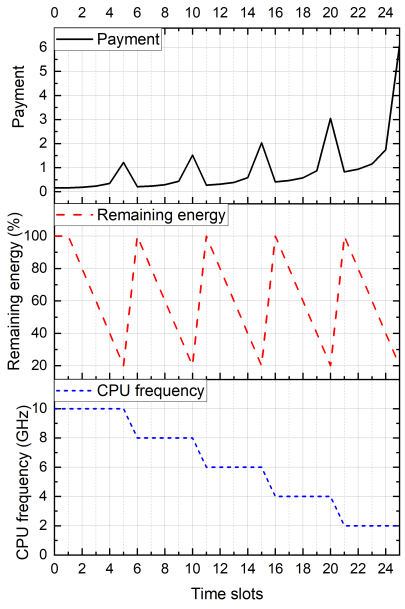


Fig. 3. Example of rent change with available energy and computation capacity.

time slots. The y-axes show payment, remaining energy percentage and available computation resources, respectively. The rent is calculated based on equation (11). We can observe that the unit rent of SP with sufficient computation capacity and the remaining energy is quite low, which tends to provide services for profit. If $D_j^{ene}(t)$ is extremely low (10%), the rent rises dramatically to prevent running out of battery regardless of how many computation resources can be utilized. We consider that the infrastructure normally has a fixed power supply in real world applications. Thus, it is not constrained by the remaining energy. Its rent function Φ_r^{comp} is defined as:

$$\Phi_r^{comp}(R^{comp}(t),1)=\frac{1}{R^{comp}(t)}e^{\kappa}. \tag{12}$$

The specified explanation of (11) and (12) can be found in Appendix B. The pricing relationship between Φ_j^{comp} and Φ_r^{comp} is analyzed in Theorem 1.

Theorem 1. *When the remaining energy $D_j^{ene}(t)$ and available computation capacity $D_j^{comp}(t)$ satisfy $D_j^{ene}(t)\leq\kappa/(\kappa-ln(D_j^{comp}(t)/R^{comp}(t)))$, the computation resource rent of device is not lower than that of the infrastructure, i.e., $\Phi_j^{comp}\geq\Phi_r^{comp}$. If $D_i^{comp}(t)\leq R^{comp}(t)$, regardless of the value of κ, $\Phi_j^{comp}\geq\Phi_r^{comp}$ always holds.*

The key to the proof is analyzing the impact of renting relationship between infrastructure and device caused by network states. Details can be found in Appendix C.

TABLE 2
Main notations

Notation	Description
$R(t)$, $D_i(t)$	Infrastructure and device in time slot t, respectively
$S_i(t)$	Service generated by device $D_i(t)$ in time slot t
$\alpha_{ij}(t)$	The binary decision variable of migrating service $S_i(t)$ from device $D_i(t)$ to device $D_j(t)$
$\beta_{ijh}(t)$	The binary decision variable of transmitting data package required by service $S_i(t)$ from device $D_h(t)$ to device $D_j(t)$
$\gamma_i(t)$	The migration ratio decision variable of service $S_i(t)$
B_{ij}	The bandwidth between devices $D_i(t)$ and $D_j(t)$
$\Gamma_{ij}(t)$	The SINR between devices $D_i(t)$ and $D_j(t)$
ϑ^2	Transmission latency variance
λ	The intensity of task transmission
$T_i^{loc}(t)$, $T_i^{mig}(t)$	The local and migrated execution latency of service $S_i(t)$, respectively
$T_i(t)$, $P_i(t)$	The total execution latency and payment of service $S_i(t)$, respectively
$U_i(t)$	The utility of service $S_i(t)$
Φ_j^{comp}	The unit price of computation resources of device $D_j(t)$
Φ_r^{comp}	The unit price of computation resources of infrastructure $R(t)$
κ	The price coefficient
$\mathbf{E}_l$, $\mathscr{E}_{\mathbf{d}}$	Expert demonstration dataset in l update period and expert trajectory in $\mathbf{d}$ time slots, respectively
$p_i^s(t), p_j^d(t)$	The preferential value of service $S_i(t)$ and device $D_j(t)$, respectively
$\mathbf{\Delta}(t)$	The adjust variable list in time slot t
$L_E([\theta_0^-, \theta_0])$	The loss function of agent policy based on sample E
θ_0^-	The frozen parameter of the initial model
θ_l, ω_l	The agent and meta learning parameters in update period l, respectively

Since the decision is only made at the beginning of a time slot, the SR should pay for the resources of the whole time slot. Hence, the total payment $P_i(t)$ consists of computation and downloading payment, which can be calculated as follows:

$$\begin{aligned} P_i(t) =&\alpha_{ij}(t)[\Phi_j^{comp}(D_j^{comp}(t), D_j^{ene}(t))\tau + \\ &(1-\sum_{h=1}^{n_t}\beta_{ijh}(t))\Phi^{down}K_i^{size}] + (1- \\ &\sum_{j=1}^{n_t}\alpha_{ij}(t))[\Phi_r^{comp}(R^{comp}(t),1)\tau + \\ &\Phi_r^{down}K_i^{size}], \end{aligned} \tag{13}$$

where τ indicates the duration of each time slot.

The corresponding energy consumption can be obtained by $T_i^{comp,loc}(t)e^{comp} + (1-\sum_{h=1}^{n_t}\beta_{iih})T_i^{dp,loc}(t)e^{down} + (T_{ij}^{comm}(t)+\sum_{h=1}^{n_t}\beta_{iih}T_i^{dp,loc}(t))e^{comm}$, where e^{comp}, e^{down} and e^{comm} indicate the unit energy consumption of computation, download and communication operations, respectively.

3.4 Problem Formulation

In order to reduce the impact of time-varying heterogeneous resource states on service cooperative migration performance, we intend to use service latency and migration payment in each time slot as the indicators of service cooperation performance and cost, respectively, which can be formulated as:

$$\begin{aligned} P1: &\min_{\alpha_{ij}(t),\beta_{ijh}(t)} \frac{1}{\mathcal{T}}\sum_{t=1}^{\mathcal{T}}\sum_{i=1}^{s_t}P_i(t), \\ &\min_{\alpha_{ij}(t),\beta_{ijh}(t),\gamma_i(t)} \frac{1}{\mathcal{T}}\sum_{t=1}^{\mathcal{T}}\sum_{i=1}^{s_t}T_i(t), \end{aligned} \tag{14}$$

s.t.

$$\begin{aligned} &C1: T_i(t) \le K_i^{delay}, \\ &C2: T_{ij}^{mig}(t), \le \widehat{T_{ij}^{comm}}(t), \\ &C3: D_i^{ene}(t) - E_i^{cost}(t) \ge 0, D_{i,t} \in \mathcal{D}, \\ &C4: \sum_{i=1}^{n_t}(1-\sum_{j=1}^{n_t}\alpha_{ij}(t)) \le R^{ch}, \\ &C5: \sum_{j=1}^{n_t}\alpha_{ij}(t), \sum_{h=1}^{n_t}\beta_{ijh}(t) \in \{0,1\}, \\ &C6: \gamma_i(t) \in [0,1], \\ &C7: (1-\gamma_i(t)\sum_{j=1}^{n_t}\alpha_{ij}(t))(1-\sum_{j=1}^{n_t}\alpha_{ij}(t)) \in \{0,1\}, \\ &C8: \sum_{i=1}^{n_t}(\alpha_{ij}(t)+\sum_{j'=1}^{n_t}\beta_{ij'j}(t)) \le N. \end{aligned}$$

Herein, we assume that the number of devices within the infrastructure coverage only changes at the beginning of each time slot, similar to [31]. Constraint $C1$ ensures that the average latency of service cannot exceed its tolerable latency to ensure users' QoE. Constraint $C2$ guarantees that each service needs to be completed within the communicable time $\widehat{T_{ij}^{comm}}(t)$. Constraint $C3$ ensures that each SP should not exhaust its remaining energy to prevent service interruption caused by breakdown. The upper bound of D2I communication capacity is limited in $C4$. Constraints $C5$ and $C6$ indicate the value range of binary decision variables and migration ratio, respectively. Constraint $C7$ illustrates when migration ratio $\gamma_i(t) = 0$, there is no SP provides service cooperation, i.e., $\sum_{j=1}^{n_t}\alpha_{ij}(t) = 0$. Constraint $C8$ limits the upper bound of transmission workload.

Theorem 2. *The joint optimization problem in (14) is NP-hard.*

The core idea is reducing and rephrasing $P1$ as a tra-

ditional NP-hard problem, and details can be found in Appendix D.

To solve the formulated optimization problem, this paper considers two situations from the perspective of global and distributed. Then, we propose an imitation learning-based solution named LOS including an offline expert policy and an imitation learning-based online agent policy. The former obtains the optimal scheduling solution based on global information, and the latter makes decisions online by imitating expert behavior trajectories. These two policies will be detailed in the following section.

4 Imitation Learning-based Online Service Cooperative Migration

4.1 Overview of LOS

Variables $\alpha_{ij}(t)$ and $\beta_{iih}(t)$ in the formulated problem are coupled and interdependent in each time slot. To resolve it, we first analyze the optimal latency to decouple decision variables in Subsection 4.2. Then, an imitation learning-based online service cooperative migration policy is designed, and the architecture is illustrated in Fig. 4. We detail an optimal offline expert policy in Subsection 4.3. In Subsection 4.4, we design a lightweight online agent policy which can continuously imitate expert trajectories with a few trajectories.

4.2 Problem Transformation

Since the purpose of problem $P1$ is to minimize the average performance of service cooperative migration, we intend to minimize the average latency and cost of service cooperative migration in each time slot, and $P1$ can be transformed as follows:

$$\begin{aligned} P2: &\min_{\alpha_{ij}(t),\beta_{ijh}(t)} \sum_{i=1}^{s_t} P_i(t), \\ &\min_{\alpha_{ij}(t),\beta_{ijh}(t),\gamma_i(t)} \sum_{i=1}^{s_t} T_i(t), \\ &\text{s.t.} \quad C1 - C8. \end{aligned} \tag{15}$$

Theorem 3. *When cooperative decision variables $\alpha_{ij}(t)$ and $\beta_{ijh}(t)$ are determined, the optimal migration ratio $\gamma_i^*(t)$ and the corresponding latency $T_i^*(\gamma_i^*(t),t)$ to problem $P2$ should satisfy*

$$T_i^{loc}(\gamma_i^*(t),t) = T_i^*(\gamma_i^*(t),t), \tag{16}$$

$$T_i^{mig}(\gamma_i^*(t),t) = T_i^*(\gamma_i^*(t),t). \tag{17}$$

These can be proved by contradiction. Please refer to Appendix E for the details.

According to Theorem 3, $P2$ can be transformed into:

$$\begin{aligned} P3: &\min_{\alpha_{ij}(t),\beta_{ijh}(t)} \sum_{i=1}^{s_t} P_i(t), \\ &\min_{\alpha_{ij}(t),\beta_{ijh}(t)} \sum_{i=1}^{s_t} T_i^{mig}(t), \\ &\min_{\gamma_i(t)} \sum_{i=1}^{s_t} |T_i^{loc}(t) - T_i^{mig}(t)|, \\ &\text{s.t.} \quad C1 - C8. \end{aligned} \tag{18}$$

The two decision variables $\alpha_{ij}(t)$ and $\beta_{ijh}(t)$ deducing payment and latency are coupled with each other. It is noted that the payment depends on the available energy and computation capacity of SP, while the migrated latency is determined by service processing rate. The multi-objective optimization problem exists no single global solution, that is, the solution of $P3$ is a perato optimal solution set [44]. We define a metric utility as $U_i(t) = 1/P_i(t)T_i^{mig}(t)$ to evaluate the pareto optimal solution. Then the joint optimization can be decomposed into two sub-problems as follows:

$$\begin{aligned} P4: &\max_{\alpha_{ij}(t),\beta_{ijh}(t)} \sum_{i=1}^{s_t} U_i(t), \\ &\text{s.t.} \quad C3 - C5, C8. \end{aligned} \tag{19}$$

and

$$\begin{aligned} P5: &\min_{\gamma_i(t)} \sum_{i=1}^{s_t} |T_i^{loc}(t) - T_i^{mig}(t)|, \\ &\text{s.t.} \quad C1, C2, C7. \end{aligned} \tag{20}$$

Sub-problems $P4$ and $P5$ indicate two parts, i.e., migration device selection and migration ratio decision. We propose solutions including offline expert policy (Algorithms 1$\sim$3) detailed in Subsection 4.3 and lightweight online agent policy (Algorithm 4) based on imitation learning detailed in Subsection 4.4 to solve $P4$ and $P5$.

4.3 Offline Expert Policy of LOS

The investigated system involves multiple devices and more than one migrated services at the same time. In time slot t, SRs and SPs can be constructed as two entity sets without intersection, which can be denoted as $\mathcal{S}(t) = \{S_1(t), ..., S_{s_t}(t)\}$ and $\mathcal{D}(t) = \{D_1(t), ..., D_{d_t}(t)\}$, respectively. According to the observed global states, the utility of service migration in each device can be obtained. Thus, the proposed $P4$ can be transformed to the matching problem of maximizing total utility. To solve the mentioned problem, an optimal matching algorithm is proposed in Algorithm 1.

Theorem 4. *The optimal matching result, i.e., maximum utility, can be obtained in Algorithm 1.*

It can be proved by contradiction, and the proof in detail can be found in Appendix F.

We assume that infrastructure as the expert node can obtain the full global states $s(t)$ [34], and construct expert trajectories $\langle s(t), a(t)\rangle$. Executing time slots can be divided into $\mathbf{l}$ batches to update the imitation with $\mathbb{D}$ state-action pairs. Based on the obtained matching result $\alpha_{ij}(t)$ and $\beta_{ijh}(t)$, the range of $\gamma_i^*(t)$ can be obtained according to Theorem 5.

Theorem 5. *Based on the obtained values of $\alpha_{ij}(t)$ and $\beta_{ijh}(t)$, the lower bound of $\gamma_i(t)$ is*

$$\gamma_i^l(t) = \frac{K_i^{delay} - T_i^{dp,loc}(t) - T_i^{comp,loc}(t)}{T_i^{comp,loc}(t)}. \tag{21}$$

When $\widehat{T_{ij}^{comm}}(t) < K_i^{delay}$, the upper bound of $\gamma_i(t)$ is

$$\gamma_i^{u,c}(t) = \frac{\widehat{T_{ij}^{comm}}(t) - T_{ij}^{wait} - T_{ij}^{dp}(t)}{T_{ij}^{comm}(t) + T_{ij}^{comp}(t)}. \tag{22}$$

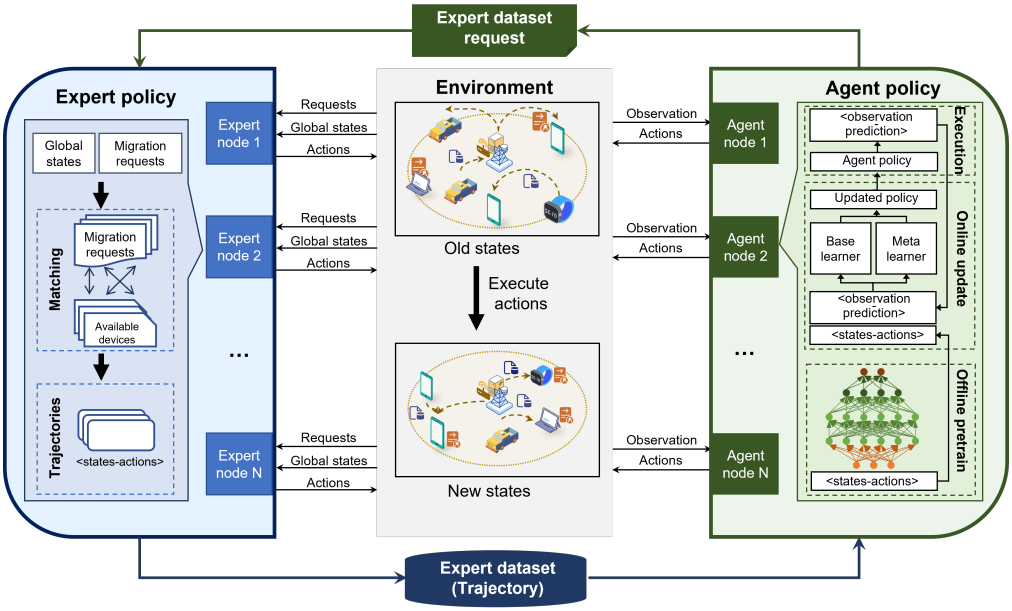


Fig. 4. Structure of the designed algorithm.

Otherwise, the upper bound is

$$\gamma_i^{u,d}(t) = \frac{K_i^{delay} - T_{ij}^{wait} - T_{ij}^{dp}(t)}{T_{ij}^{comm}(t) + T_{ij}^{comp}(t)}. \tag{23}$$

The key to the proof is analyzing the latency constraints based on Theorem 3. Details can be found in Appendix G.

According to Theorem 3, the optimal migrated ratio can be obtained as follows:

$$\gamma_i^*(t) = \frac{T_i^{comp,loc}(t) + T_i^{dp,loc}(t) - T_{ij}^{mig}(t) - T_{ij}^{dp}(t)}{T_{ij}^{comm}(t) + T_i^{comp}(t) + T_{ij}^{comp}(t)}. \tag{24}$$

We can observe that if $T_i^{comp,loc}(t) + T_i^{dp,loc}(t) \leq T_{ij}^{mig}(t) + T_{ij}^{dp}(t)$, $\gamma_i(t) = 0$ based on C6 in $P1$, and $\sum_{j=1}^{n_t} \alpha_{ij}(t) = 0$. Combining Theorem 5, migration ratio can be computed as follows:

$$\gamma_i(t) = \begin{cases} \gamma_i^l(t), & \gamma_i^*(t) \in [0, \gamma^l(t)]; \\ \gamma_i^*(t), & \gamma_i^*(t) \in [\gamma_i^l(t), \min\{\gamma^{u,d}(t), \gamma^{u,c}(t)\}]; \\ \gamma^{u,d}(t), & \gamma_i^*(t) \in [\gamma^{u,d}(t), 1], \gamma^{u,d}(t) < \gamma^{u,c}(t); \\ \gamma^{u,c}(t), & \gamma_i^*(t) \in [\gamma^{u,c}(t), 1], \gamma^{u,c}(t) \leq \gamma^{u,d}(t). \end{cases} \tag{25}$$

After obtaining the whole decisions, the utility can be computed to construct the preference list of each element for matching. The matching process contains two phases, i.e., $Match$ (Algorithm 2) and $Update$ (Algorithm 3). The optimal metric preferential value obtained based on the preference list is defined to evaluate the optimal matching results by $Match$, and the adjusted variables are updated by $Update$ until all services have been matched with migration devices.

Herein, $Match(S_i(t), \mathcal{D}(t))$ represents the matching process to find a suitable executing device from $\mathcal{D}(t)$ for $S_i(t)$, according to the adjust variable list and the preferential value of each element. If $p_i^s(t) + p_j^d(t) = U_{ij}(t)$ is satisfied, $S_i(t)$ will be migrated to $D_j(t)$. Otherwise, $\mathbf{\Delta}_j(t)$ needs to be updated as $\min\{\mathbf{\Delta}_j(t), p_i^s(t) + p_j^d(t) - U_{ij}(t)\}$.

The update process $Update(p^s(t), p^d(t), \mathbf{\Delta}(t))$ intends to update the adjust variable list for modified matching results. For $D_j(t) \in \mathcal{D}(t)$ which has not been matched before, the adjust variable is $\min\{\delta, \mathbf{\Delta}_j(t)\}$, and preferential values of all tasks and vehicles are modified with the updated adjust variable.

Theorem 6. *The time complexity of Algorithm 1 is* $\mathcal{O}(N^3\mathbf{l})$.

Time complexity is analyzed to reflect the algorithm execution efficiency, which is mainly determined by loops. The detailed proof can be found in Appendix H.

According to Theorems 4 and 6, we can observe that although the optimal decision results can be obtained based on expert policy and global states, the complexity is rather high for execution. To realize the real-time service migration in the edge network, we propose a lightweight online agent policy. After completing the expert policy, trajectories of expert r can be obtained as the dataset $\mathbf{E} = \{\mathscr{E}_{\mathbf{d}} = \{\langle s(t), a(t)\rangle\}_{t=1}^{\mathbf{l}}\}_{\mathbf{d}=1}^{\mathbb{D}}$ and transmitted to required agents for policy training.

4.4 Online Agent Policy of LOS

Due to the finite communication capacity, devices cannot observe the global network states, which is rather challenging for service migration in real time. To overcome the obstacle, devices are viewed as distributed agents to make migration decisions by approaching expert demonstrations and imitating expert behaviors. However, excessive expert trajectories generate a huge communication burden (especially in dynamic networks), and expert demonstrations would become stale as time goes by. Thus, the agents need to update their models to prevent performance loss in a while, which is a repetitive process consuming substantial computation resources. To prevent mentioned issues, we leverage meta

Algorithm 1: Pseudo-code of Offline Expert Policy

Input: Available SP set $\mathcal{D}$ and SR set $\mathcal{S}$
Output: Expert trajectory $\mathscr{E}_{\mathbf{d}}$

for *executing time slots* $t < t + \mathbf{l}$ **do**
  Initialize match times $D_j(t).visit$ and $S_i(t).visit$ with 0, $D_j(t) \in \mathcal{D}(t)$ and $S_i(t) \in \mathcal{S}(t)$;
  Initialize the preferential value of each device with $p_j^d = 0, D_j(t) \in \mathcal{D}(t)$ and the adjust variable $\mathbf{\Delta}(t) = \{\mathbf{\Delta}_1(t), ..., \mathbf{\Delta}_{d_t}(t)\}$ with ∞;
  foreach $S_i(t) \in \mathcal{S}(t)$ **do**
    foreach $D_j(t) \in \mathcal{D}(t)$ **do**
      Obtain the optimal migrated ratio based on (25);
      if *C1-C7 are satisfied* **then**
        Calculate the updated utility $U_{ij}(t)$ of $S_i(t)$ and add $U_{ij}(t)$ into the preference list $S_i(t).p$ in decreasing order;
      end
      else
        Calculate updated utility $U_{ij}(t)$ with $\gamma_i(t) = 0$;
      end
    end
    Derive preferential value of each service, $p_i^s(t) = \max\{S_i(t).p\}$, $S_i(t) \in \mathcal{S}$;
  end
  foreach $S_i(t) \in \mathcal{S}$ **do**
    while *true* **do**
      if $Match(S_i(t), \mathcal{D})$ **then**
        Break;
      end
      $Update(p^s(t), p^d(t), \mathbf{\Delta}(t))$;
    end
    $t+ = 1$;
  end
  return $\mathscr{E}_{\mathbf{d}} = \{\langle s(t), a(t)\rangle\}_{t=1}^{\mathbf{l}}$.
end

Algorithm 2: $Match(S_i(t), \mathcal{D}(t))$

Input: $S_i(t), \mathcal{D}(t)$
Output: Matching flag

$S_i(t).visit+ = 1$;
foreach *device* $D_j(t) \in \mathcal{D}(t)$ **do**
  if $D_j(t).visit! = 0$ **then**
    Continue;
  end
  if $p_i^s(t) + p_j^d(t) = U_{ij}(t)$ **then**
    $D_j(t).visit+ = 1$;
    if $D_j(t)$ *has not been allocated to other service* **then**
      $D_j(t).match = S_i(t)$;
      return matching results;
      Break;
    end
  end
  else
    $\mathbf{\Delta}_j(t) = \min\{\mathbf{\Delta}_j(t), p_i^s(t) + p_j^d(t) - U_{ij}(t)\}$;
  end
end
return matching results.

Algorithm 3: $Update(p^s(t), p^d(t), \mathbf{\Delta}(t))$

Input: Preferential values $p^s(t)$ and $p^d(t)$, adjust variable list $\mathbf{\Delta}(t)$
Output: Updated preferential values and adjust variable list

$\delta = \infty$;
foreach $D_j(t) \in \mathcal{D}(t)$ **do**
  if $D_j(t).visit = 0$ **then**
    $\delta = \min\{\delta, \mathbf{\Delta}_j(t)\}$;
    foreach $S_i(t) \in \mathcal{S}(t)$ *and* $S_i(t).visit \neq 0$ **do**
      $p_i^s(t) - = \delta$;
    end
    foreach $D_j(t) \in \mathcal{D}$ **do**
      if $D_j(t).visit \neq 0$ **then**
        $p_j^d(t) + = \delta$;
      end
      else
        $\mathbf{\Delta}_j(t) - = \delta$;
      end
    end
  end
end
return $p^s(t)$, $p^d(t)$ and $\mathbf{\Delta}(t)$.

learning to accelerate imitating updated expert trajectories with a few demonstrations.

The imitation learning process contains two kinds of participants: experts and agents. To reduce the agent policy loss caused by imitating the stale expert dataset, agent policies need to be updated every $\mathbf{l}$ time slots by obtaining new expert trajectories transmitted from the expert node. In addition, when a mobile agent moves into a new domain, the agent policy also needs to be updated to adapt to the new environment by imitating new expert dataset. The update period is represented as $l \in \mathcal{L}$. We leverage DAgger [29] to control the performance loss between two update periods and enhance the generality of agent policy. The training performance and cost evaluation can be found in Appendix I.

Due to the workload caused by multiple devices with high mobility, decentralized policies need to be designed for autonomous decision. Thus, each device needs to learn policies based on observable information and update policies independently to ensure the accuracy of policies. The pseudo-code of agent policy training including three phases is detailed in Algorithm 4 as follows:

4.4.1 Phase 1: Agent network initialization

Before updating the model, an initial large-scale model needs to be pre-trained to provide prior knowledge. After obtaining initial expert demonstration dataset $\mathscr{E}_0$ and expert policy π_0^e, each agent needs to obtain the initial agent model by training the neural network. The agent network estimates action $\hat{a}$ based on the observed states s, and trains its

Algorithm 4: Pseudo-code of Training Online Agent Policy

Input: Expert demonstration dataset $\mathbf{E} = \{\mathscr{E}_{\mathbf{d}}\}_{\mathbf{d}=1}^{\mathbb{D}}$, learning rates ι_b and ι_m;
Output: Agent learning model
1 **Phase 1: Agent network initialization**
2 Initialize θ_0^- and θ_0;
3 Obtain initial expert trajectory sample $\mathscr{E}_0$ from expert node;
4 **for** *Sample* $E \in \mathscr{E}_0$ **do**
5 Estimate action $\hat{a}$ based on states;
6 Evaluate $L_E([\theta_0^-, \theta_0])$ by (26);
7 Update $[\theta_0^-, \theta_0]$ by (27);
8 **end**
9 Observe states and make decisions with agent policy $\pi_0^a(\theta_0^-, \theta_0)$ and calculate the optimal migrated ratio by (25);
10 **Phase 2: Update agent model**
11 Initialize meta parameter ω_1;
12 **for** *update period* $l \in \mathcal{L}$ **do**
13 **Phase 2.1: Prepare data for training**
14 Obtain the updated expert trajectories $\mathbf{E}_l \leftarrow \mathscr{E}_l^*$ from expert nodes;
15 **if** $\mathbf{E}_l$ *is updated by* $\mathscr{E}_l^*$ *or time slot* t_l *satisfy* $t_l \% \mathbf{t} == 0$ **then**
16 $\mathbf{E}_l \leftarrow \mathbf{E}_l \cup \mathscr{E}_{l,t_l}$ to train policy $\hat{\pi}_{l,t_l}^a$;
17 **Phase 2.2: Meta transfer learning**
18 **for** *meta sample* $\mathscr{E}_{e,l} \in \mathbf{E}_l$ **do**
19 Sample $\mathscr{E}_{e,l}$ into $\mathcal{E}_{e,l}^b$ and $\mathcal{E}_{e,l}^m$;
20 **for** $E \in \mathcal{E}_{e,l}$ **do**
21 Optimize θ_l' based on $\mathcal{E}_{e,l}^b$ by (28);
22 **end**
23 Optimize ω_l based on $\mathcal{E}_{e,l}^m$ by (29);
24 Optimize θ_l based on $\mathcal{E}_{e,l}^m$ by (30);
25 **end**
26 Obtain policy $\pi_{l,t_l}^a = p^{(t_l/\mathbf{t})}\pi_{l,0}^a + (1 - p^{(t_l/\mathbf{t})})\hat{\pi}_{l,t_l}^a$;
27 **end**
28 **Phase 3: Make decisions with trained model** π_{l,t_l-1}^a
29 Observe states and make decisions with l-th period agent policy $\pi_l^a(\theta_0^-, \theta_l, \omega_l)$ and calculate the optimal migration ratio by (25) to obtained $\mathscr{E}_{l,t_l} = \{\langle s_{t_l}, a_{t_l}\rangle\}_{t_l=0}^{\mathbf{t}}$;
30 **end**

policy by fitting their observed states and estimated action distributions $\pi^a(\hat{a}|s)$ with that of expert $\pi^e(a|s)$, by the following loss function:

$$L_E([\theta_0^-; \theta_0]) = \mathbb{E}[\|\pi^a(\hat{a}|s) - \pi^e(a|s)\|^2], \tag{26}$$

where θ_0^- represents the frozen parameters of the initial model, which provides a prior knowledge for the updated model. Therefore, the parameter update process is defined as follows:

$$[\theta_0^-, \theta_0] \leftarrow [\theta_0^-, \theta_0] - \iota_b \nabla L_E([\theta_0^-, \theta_0]). \tag{27}$$

4.4.2 Phase 2: Update agent model

In period l, in order to reduce the execution loss, DAgger is leveraged to update the model every $\mathbf{t}$ time slots. As shown in lines 13∼16, if the training dataset is updated by new expert demonstration $\mathscr{E}_l^*$ (line 14) or aggregated by observation $\mathscr{E}_{l,t_l}$ (line 16) (obtained by the previous policy $\pi_{l,t_l-\mathbf{t}}^a$), the agent model will be updated through meta transfer learning.

To accelerate updating model $\hat{\pi}_{l,t_l}^a$, we leverage meta training to learn the scaling and shifting of the model transfer. Meta learning parameter in period l is represented as ω_l. The meta training process transforms $\arg\max_{\theta_l'} \log p(\theta_l'|\mathscr{E}_{l-1}, \mathscr{E}_l)$ to $\log \int_{\omega_l} p(\theta_l|\mathscr{E}_l, \omega_l) p(\omega_l|\mathscr{E}_{l-1}) d\omega_l$ to get ω_l by $\arg\max_{\omega_l} \log p(\omega_l|\mathscr{E}_{l-1})$. The objective of meta-training is to make $\arg\max_{\theta_l} \log p(\theta_l|\mathscr{E}_l, \omega_l)$ approximate to $\arg\max_{\theta_l} \log p(\theta_l|\mathscr{E}_{l-1}, \mathscr{E}_l)$ [45].

The meta update of agents contains two sub-tasks, i.e., the basic learning and the meta learning. In period l, expert demonstration $\mathscr{E}_{e,l}$ are extracted from the data set randomly and then sampled into $\mathscr{E}_{e,l}^b$ for the basic learning model and $\mathscr{E}_{e,l}^m$ for meta-model learning, $\mathscr{E}_{e,l}^b + \mathscr{E}_{e,l}^b = \mathscr{E}_{e,l}$. The temporary parameter θ_l' initialized by θ_{l-1} for fine-tune is updated by:

$$\theta_l' \leftarrow \theta_{l-1} - \iota_b \nabla_\theta L_{\mathscr{E}_{e,l}^b}([\theta_0^-, \theta_{l-1}], \omega_{l-1}). \tag{28}$$

Based on (28), the parameter of meta learning is calculated based on $\mathcal{E}_{e,l}^m$ by:

$$\omega_l \leftarrow \omega_{l-1} - \iota_m \nabla_{\omega_{l-1}} L_{\mathscr{E}_{e,l}^m}([\theta_0^-, \theta_l'], \omega_{l-1}). \tag{29}$$

Then, agent parameter can be updated by:

$$\theta_l \leftarrow \theta_{l-1} - \iota_m \nabla_{\theta_{l-1}} L_{\mathscr{E}_{e,l}^m}([\theta_0^-, \theta_l'], \omega_l). \tag{30}$$

After updating $\hat{\pi}_{l,t_l}^a$, updated policy π_{l,t_l}^a is obtained by integrating expert policy $\pi_{l,0}^a$ and updated policy based on adjusted probability p, which indicates the decay of expert policy obtained in the initial process of period l.

4.4.3 Phase 3: Make decisions with the trained agent policy

After completing l-th training, distributed agents make migrated decisions based on the observed states by policy $\pi_l^a(\theta_0^-, \theta_l, \omega_l)$. Until it moves into the coverage of other infrastructures or up to the $(l+1)$-th update period, the agent repeats phase 2 to update. The update process of the agent model enables lightweight continuous imitating, which can efficiently adapt to the updated demonstration with lower communication with expert nodes while retaining some known knowledge.

Herein, the prediction learning network is constructed based on a fully-connected network. The overall computation complexity of the designed agent policy in the execution process is analyzed by Theorem 7.

Theorem 7. *When the number of the leveraged fully-connected hidden layers is M, the overall complexity of the designed agent policy for each agent during execution process is $\mathcal{O}((\sum_{m=1}^{M} n_m n_{m-1})\mathbf{l})$.*

The core of the time complexity proof is analyzing the network structure. Please refer to Appendix J for details.

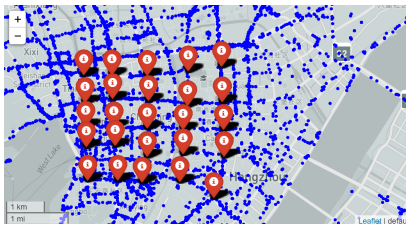


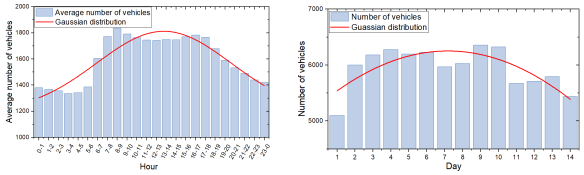
Fig. 5. The mobility traces in one hour.

5 Performance Evaluation

In this section, we demonstrate the effectiveness of the proposed solution based on extensive simulations, including the simulation setup and numerical results.

5.1 Simulation Setup

The real-world trace dataset in Hangzhou (China) includes one month of trace samples from March 1, 2014 to March 31, 2014 with characteristics of coordinates, time, ID and so on, including 16.1 GB recorded information. 25 infrastructure locations are selected to divide Hangzhou into 25 regions, which are detailed in TABLE 3. The mobility traces in one hour of the dataset is shown in Fig. 5. By analyzing, we observe that the traffic flow distribution can be represented by Gaussian distribution in a day and 2 weeks, as shown in Figs. 6(a) and 6(b), respectively. The mean values and standard deviations of vehicles are 1599.3194 and 176.4026 in 24 hours, 5856.8125 and 411.5623 in 2 weeks, respectively.



(a) In 24 hours. (b) In 2 weeks.

Fig. 6. Traffic flow distributions of Hangzhou traffic dataset.

We set the wireless communicable range of infrastructure by 500 m as the expert node to schedule the requests of passing-by vehicles and generate expert datasets for agents. Devices can implement SP and SR simultaneously. The channel gain of devices can be computed as $127 + 30 \times log_2 L_{i,j}$ [46], where distance $L_{i,j}$ can be obtained based on the coordinates of devices. To evaluate service performance under different workloads, the Electroencephalography (EEG) dataset [47] and the statistical online YouTube video service in [48] are utilized to simulate low workload and high workload, respectively. Each medical analysis consists of 4096 samples of EEG data. The data size of each record in EEG is between $[560, 747]$ KB, utilized to simulate the low-workload service. The time slot of each low-workload service is defined as 1 s. The data size of online YouTube service distributes between $[30, 450]$ MB, utilized to simulate the high-workload service. The time slot of each high-workload service is defined as 3 min. According to the statistical results in [48], the data sizes of most YouTube videos are less than 80 MB. The required CPU cycle of each service is 1 and 10 Gigacycles for low and high workload services. Since cached content is affected by various unpredictable subjective factors such as user preferences, different categories of services are viewed as randomly distributed on devices. To differentiate distinct service categories, we define the data size of the required data package as $K \times 1$ MB and $K \times 100$ MB to clarify two scenarios with low and high workloads, respectively. Similar to [49], other parameters are illustrated in Table 4.

To evaluate the performance of our proposed solution, the following performance indicators are considered.

- Achievable QoS: The expected sum of service processing rate, which is defined as:

$$\mathbf{Q}_i = \frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} \frac{1}{s_t} \sum_{i=1}^{s_t} (\gamma_i r_{ij}(t) + (1 - \gamma_i(t)) r_i(t)). \quad (31)$$

- Latency: The average service migration latency of each time slot.
- Energy consumption percentage: The percentage of average consumed energy for service migrating to the overall energy.
- Payment: The average payment of service migration of each time slot.
- Increased Time-To-Live (ITTL): The average ITTL of service migrated execution compared to local execution.

Herein, the achievable QoS is calculated based on the sum of achievable migration and local service processing rates (i.e., $r_{ij}(t)$ and $r_i(t)$). Variable $r_{ij}(t)$ is calculated by summing the migrated computation, communication and data package obtainment rates up. The migrated communication rate can be calculated by (5), and the migrated data package obtainment rate can be obtained by transmitting (5) or downloading ($D_i^{down}(t)$ or ($R_{down}(t)$). To unify the unit of computation rate (megabytes per second), $r_{ij}^{comp}(t)$ is calculated by:

$$\begin{aligned} r_{ij}^{comp}(t) = & \frac{S_i^{size}(t) \sum_{j=1}^{n_t} \alpha_{ij}(t) D_j^{comp}(t)}{S_i^{comp}} \\ & + \frac{S_i^{size}(t)(1 - \sum_{j=1}^{n_t} \alpha_{ij}) R^{comp}(t)}{S_i^{comp}}, \end{aligned} \quad (32)$$

where $j = i$ indicates that the service is executed locally. Correspondingly, $r_i(t)$ is calculated by summing up the local computation and data package obtainment rates. The expert policy and agent policy of LOS are indicated as LOS-E and LOS-A, respectively. LOS-A includes pretrain and meta transfer update phases, denoted by LOS-A-P and LOS-A-MT, respectively. Similar to [45], we construct a three-layer CNN as the base learner of LOS-A, and two-layer full-connected layers fed by features extracted by the base learner are constructed as the prediction learning network. In addition to the proposed LOS, the other representative policies are selected for comparison. The pre-trained network is obtained based on stochastic gradient descent with a learning rate 0.001. Adam is selected as the optimizer.

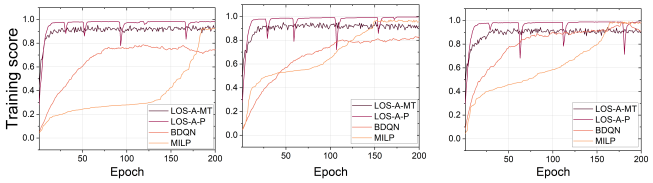
- User-centric Edge Sharing Mechanism (UESM) [50]: It divides the service migration process into several executing units to allocate communication, caching

TABLE 3
Coordinates of infrastructures

ID	Locations	ID	Locations	ID	Locations	ID	Locations	ID	Locations
1	[30.2726,120.1586]	6	[30.2663,120.1958]	11	[30.2607,120.1592]	16	[30.2560,120.1594]	21	[30.2483,120.1597]
2	[30.2726,120.1662]	7	[30.2654,120.1865]	12	[30.2606,120.1668]	17	[30.2563,120.1671]	22	[30.2482,120.1677]
3	[30.2724,120.1757]	8	[30.2667,120.1760]	13	[30.2602,120.1757]	18	[30.2538,120.1756]	23	[30.2478,120.1743]
4	[30.2735,120.1866]	9	[30.2670,120.1664]	14	[30.2600,120.1852]	19	[30.2543,120.1852]	24	[30.2480,120.1845]
5	[30.2745,120.1956]	10	[30.2662,120.1591]	15	[30.2599,120.1961]	20	[30.2538,120.1962]	25	[30.2443,120.1936]

TABLE 4
Simulation parameters

Parameter	Value
Bandwidth of D2I	$50MHz$
Transmit power of device	$0.5W$
Gaussian channel noise	$-96dBm$
computation capacity of device	$5GHz$
computation capacity of infrastructure	$25GHz$
Download rate of device	$5MBps$



(a) $\mathfrak{D} = 600$. (b) $\mathfrak{D} = 900$. (c) $\mathfrak{D} = 1200$.

Fig. 7. Convergence performance with different training data sizes $\mathfrak{D}$.

and computation resources based on the simulated annealing algorithm.

- Beyond Deep Q-Network (BDQN) [51]: It integrates communication, computation and caching states to make decisions based on dueling double deep Q-network algorithm.
- Multi-agent Imitation Learning based computation offloading algorithm for Pervasive edge computing (MILP) [32]: It supports partial service migration based on multi-hop sensing content and leverages imitation learning to minimize the latency.
- Imitation Learning enabled Task Scheduling algorithm (IELTS) [4]: It imitates expert trajectories generated by the branch-and-bound method and migrates the whole service to an SP by satisfying all constraints to minimize latency.
- LOS without Meta Transfer learning (LOS w.o. MT): SRs need to re-train their model in each update period.

5.2 Experimental Results

5.2.1 *Impact of training data sizes*

Fig. 7 illustrates the convergence performance with different training data sizes. Herein, we normalize the total reward of BDQN and MILP algorithms as training scores to evaluate the convergence performance. We can observe that LOS-A-P has the best convergence performance in all cases, while LOS-A-MT is the closest to it. That is because the LOS-A-P is trained based on updating the whole model, while LOS-A-MT trained partial layers by frozen partial parameters and converged faster than BDQN and MILP. LOS-A-P and LOS-A-MT have better convergence performance with different $\mathfrak{D}$ in Figs. 7(a), 7(b) and 7(c), demonstrating the proposed LOS-A adapts to various environments with different training data sizes.

5.2.2 *Impact of update periods*

Figs. 8 and 9 demonstrate the training efficiency of LOS-A, LOS-A w.o. MT, MILP, BDQN. Herein, the expert policy is used as a standard to evaluate the accuracy of other policies. Expert trajectories need to be updated once at the start of a period to prevent performance loss caused by stale data. Agents perform data aggregation three times between two update periods to reduce the errors generated by the stale LOS-E. To ensure the timeliness of expert trajectories, the length of eliminated stale data is equal to that of the updated data in each update period. Fig. 8 evaluates the total training time. The training performance of every DAgger operation between two update periods can be found in Appendix K. As shown in Fig. 8, we can observe the initial operating time of LOS-A and LOS w.o. MT is slightly different since these two policies need the same initialization phase. However, the training time of LOS-A is much lower than that of LOS-A w.o. MT. Hence, the operating time of LOS-A decreases dramatically from update periods 1 to 2. Integrating the accuracy performance of Fig. 9, LOS w.o. MT re-trained agent policies to make decisions directly with updated datasets, wasting prior knowledge and losing accuracy. The accuracy is hovering around 0.866, while LOS-A transferring the old model to the new model has better accuracy compared to LOS w.o. MT. Hence, LOS-A is more suitable for continuous updates required by long-term scenarios.

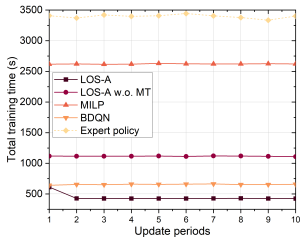


Fig. 8. Total training time with different update periods.

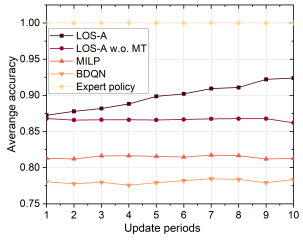
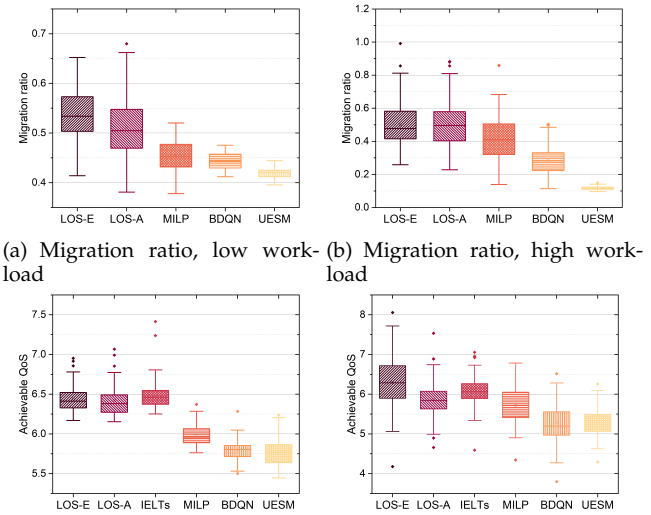


Fig. 9. Accuracy with different update periods.

Figs. 10(a)−10(d) show the distribution of the migration ratio and achievable QoS within 100 update periods under the low and high workloads, respectively. In Figs. 10(a) and 10(b), the errors between the mean values of LOS-E and LOS-A are rather slight, with 0.099 and 0.006



(a) Migration ratio, low workload (b) Migration ratio, high workload

(c) Achivable QoS, low workload (d) Achivable QoS, High workload

Fig. 10. Performance distribution with different algorithms

under low and high workloads, respectively. That valid the imitation accuracy of the proposed LOS-A which makes decisions by imitating the expert trajectories, since the value of optimal migration ratio highly depends on the migration decisions. In Figs. 10(c) and 10(d), the achievable QoS indicates the achievable service processing speed, and the proposed LOS policies (including LOS-A and LOS-E) can obtain achieve the highest QoS, except for IELTs which makes decisions with the single latency-minimization objective without considering the cost of executing services. Overall, Fig. 10 demonstrates LOS-A always maintains efficient performance under low and high workloads from a long-term perspective.

5.2.3 *Impact of the average service data sizes.*

Fig. 11 illustrates the performance of LOS-E, LOS-A, IELTs, MILP, BDQN and UESM with different service data sizes to evaluate the adaptability of distinct workloads. Figs. 11(a)−11(d) evaluate the two objectives from execution performance and cost perspectives under low and high workloads. In Figs. 11(a) and 11(b), we can observe that the latency of LOS-E is the least, while LOS-A is the closest to it. The latency of LOS-A increases from 0.87 s to 3.078 s under low workloads and from 98.907 s to 402.437 s under high workloads. BDQN focuses more on the search for cached content and neglects the transmission workload. However, LOS-A makes a rational trade-off between local and migrated devices by balancing the communication and computation loads with the adjustable migration ratio to adapt to distinct workloads. The trade-off between communication and computation workload is also evaluated in Figs. 11(c) and 11(d), where the average payment of policies declines with the increasing service data size. As data size increases, it can be observed that LOS-A tends to reduce the payment and latency by declining the migration ratio, which is also demonstrated by the comparison among IELTs and other partial migration policies.

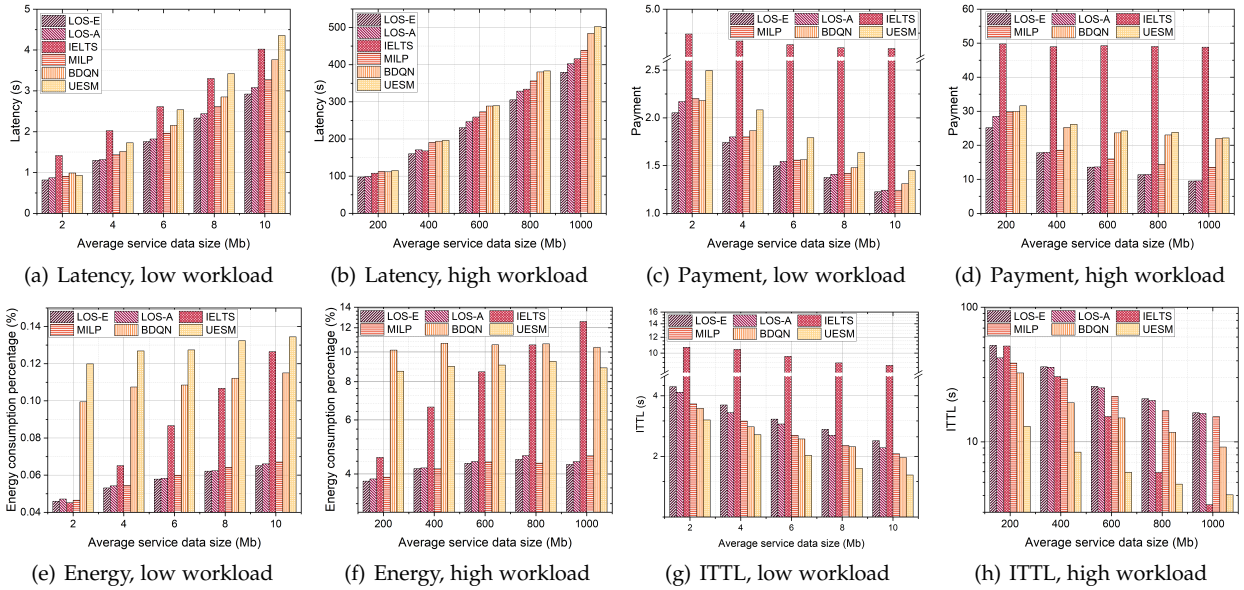
Figs. 11(e) and 11(f) show the average service processing energy consumption percentage under low and high workloads, respectively. We can observe the average energy consumption percentage increases with the rising service data size. The consumed energy of LOS-A slightly increases from 0.047% to 0.066% (under low workloads) and 3.851% to 4.386% (under high workloads). LOS-A well imitates the optimal solutions generated by LOS-E and adapts to the increasing workloads. Figs. 11(g) and 11(h) evaluate the average ITTL of SRs. Under low workload, the ITTL of IELTs is higher than LOS-A due to the saved computation workload. However, the rapid drop of ITTL shown in Fig. 11(h) demonstrates the increasing communication workload exacerbates the cost of local execution. Compared with IELTs, LOS-A partially immigrating service to SPs performs a good adaptability under high workloads.

5.2.4 *Impact of service categories*

The service performance of policies with different numbers of service categories is evaluated in Fig. 12. The experiments were conducted from 3 to 9 service categories with 0.5 service generation ratio to evaluate the algorithm generalization of multiple service categories. With the same experimental conditions, the more caching content categories, the lower cache hit rate among communicable devices. In Figs. 12(a) and 12(b), we measure the average latency with distinct numbers of service categories. The proposed centralized method LOS-E can obtain the optimal migration latency under both low and high workloads, demonstrating LOS-E designed based on matching algorithms obtains better adaptability with the increasing complexity of environment states compared to reinforcement learning-based algorithms (BDQN and MILP). LOS-A, with a timely update policy, has a good imitation performance based on the limited observed states. MILP, imitating deep reinforcement learning based on partial observable states, has a better performance compared to BDQN. With the optimization objective of latency minimization, the latency of MILP is average 0.01 s higher and 0.019 s lower than those of LOS-E and LOS-A under a lower workload, average 6.838 s and 4.791 s higher than those of LOS-E and LOS-A, respectively. That is because of the prediction error of continuously variable migration ratio, although MILP can obtain better imitation to predict integer decisions. The proposed LOS-A obtains the optimal migration ratio by one-step calculation based on the sub-optimal integer variable, which reduces the number of labels directly.

By analyzing results in Figs. 12(c) and 12(d), it is obvious that the payment of LOS-E and LOS-A are less than other algorithms. The payment of LOS-A only increases 0.628 and 28.237 with the number of data package categories from 3 to 9 under low and high workloads, respectively. That not only shows the adaptability of LOS-A to the increasing service categories since LOS-A make a satisfying trade-off between caching content and migration part, but also demonstrates that the expert decision distribution can be accurately imitated and obtain the near-optimal decisions.

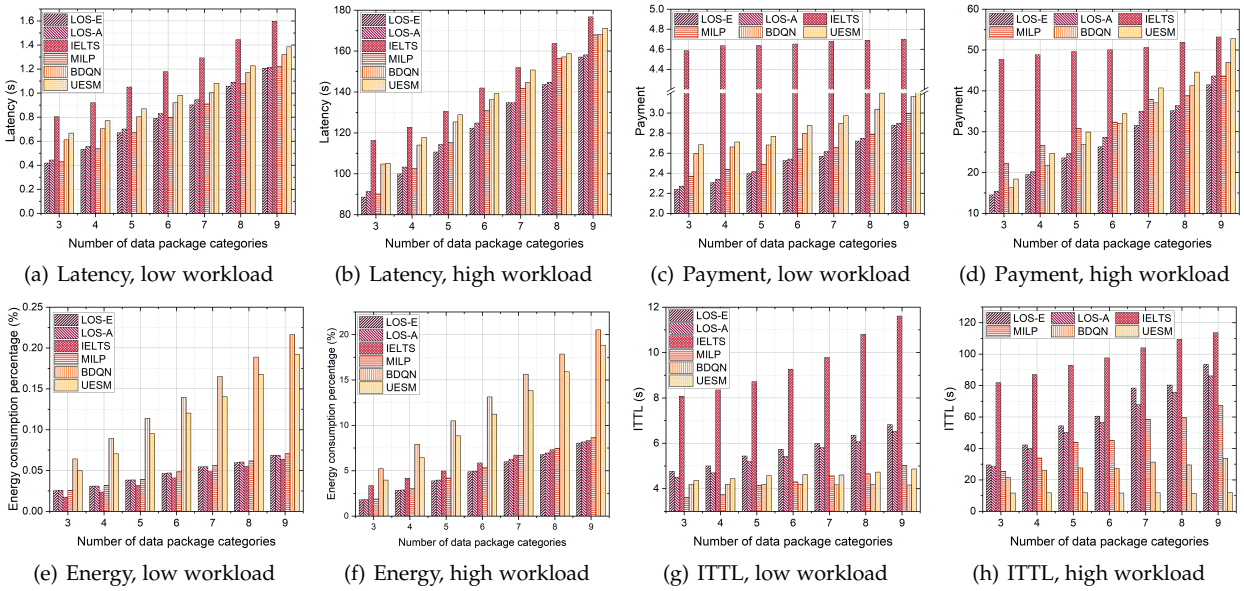
Based on energy consumption evaluated in Figs. 12(e) and 12(f), we can observe that the average energy consumption of LOS-A is rather stable compared with BDQN and UESM under different numbers of service categories. There is a merely small gap from 3 to 9 service categories under distinct workloads, which rises the ITTL evaluated



(a) Latency, low workload (b) Latency, high workload (c) Payment, low workload (d) Payment, high workload

(e) Energy, low workload (f) Energy, high workload (g) ITTL, low workload (h) ITTL, high workload

Fig. 11. Performance with different service data sizes.



(a) Latency, low workload (b) Latency, high workload (c) Payment, low workload (d) Payment, high workload

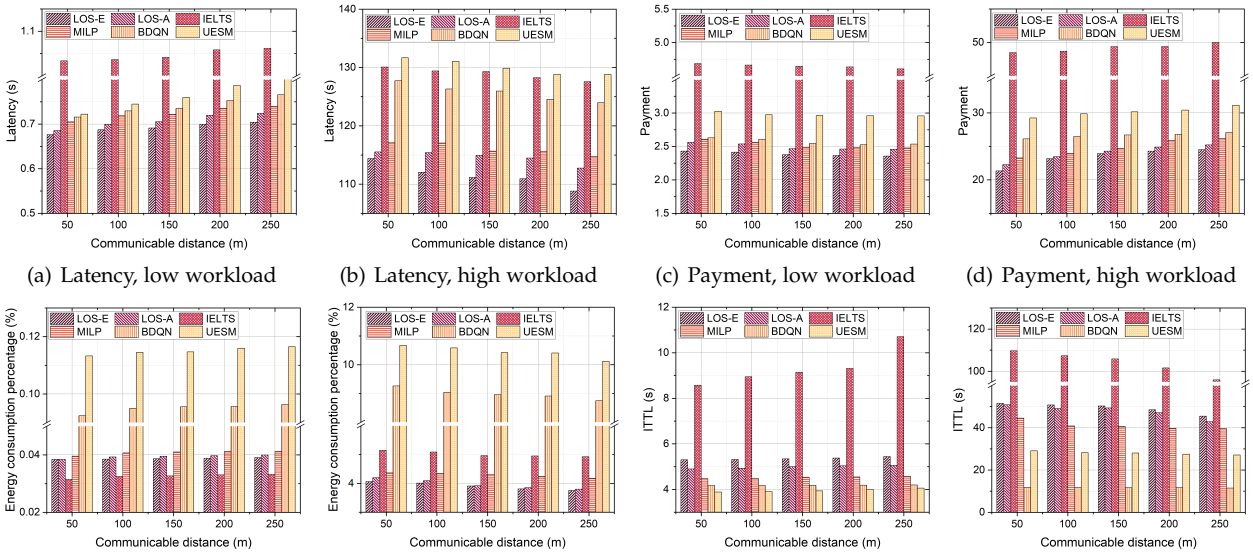
(e) Energy, low workload (f) Energy, high workload (g) ITTL, low workload (h) ITTL, high workload

Fig. 12. Performance with different number of service categories.

in Figs. 12(g) and 12(h). Herein, it is noted that the energy consumption percentage of LOS-A is only average 0.006% higher than that of IELTs. By observing the results in Figs. 12(a)∼12(d), IELTs imitating to branch and bound algorithm is a completed migration algorithm, which dramatically reduces the execution energy consumption while scarifying the execution latency and payment. LOS-A takes both the states of SRs and SPs into account, because LOS-A has a better global fit by imitating the expert trajectories and making a satisfying trade-off among different workloads with partial observation. The performance gain of LOS-A rises with the increasing number of service categories, demonstrating that LOS can efficiently adapt to multiple service category scenarios.

5.2.5 *The impact of communicable distance*

Since the channel gain can be affected by communicable distance, experiments were conducted to evaluate the performance of algorithms under different communicable constraints as shown in Fig. 13. We can observe that with the rising communicable distances, UESM and BDQN, which highly depends on the global states, lack comprehensive consideration of SPs' states compared to LOS-A. While the severity of IELTs is more significant due to the ignorance of ideal resources on local devices. By imitating the state-action distribution of experts, LOS-A obtains better results with the limited observable states. Taken latency performance in Figs. 13(a) and 13(b) under low workloads as instances, the average latency of UESM approaches that of LOS-A, while the gap between UESM and LOS-A is more significant under high workloads when the communicable distance increases.



(a) Latency, low workload (b) Latency, high workload (c) Payment, low workload (d) Payment, high workload

(e) Energy, low workload (f) Energy, high workload (g) ITTL, low workload (h) ITTL, high workload

Fig. 13. Performance with different communicable distances.

That is because UESM dividing service execution into several service units is imposed more stringent restrictions on communicable states, leading to a higher tendency to migrate services to infrastructure with more payment.

Figs. 13(e) and 13(f) measure the energy consumption under low and high workloads, respectively. It is noted that when the communicable distance increases, the energy consumption of LOS-A slightly increases under low workloads while slightly declines under high workloads. SRs are more inclined to migrate services by renting resources while sacrificing slight latency and energy consumption, as validated in Figs. 13(a) and 13(c). Under low workload, the consumed energy gap between LOS-A and MILP is from 0.16% to 0.38%, and the gap increases 2.375 times, with 50 m communicable distance and 250 m communicable distance. That is because the performance of MILP can be affected by the increasing action space. Integrating the ITTL evaluation in Figs. 13(g) and 13(h), LOS-E obtains optimal matching results by comprehensively paying attention to the states of communicable devices and adjusting the mobility in time to provide services. LOS-A can more flexibly adapt to different communicable limitations to extend the life cycle of SRs by approximating the global state-action distribution generated by LOS-E.

6 Conclusion

This paper established a dynamic edge system to support real-time service cooperative migration. With the purpose of minimizing both latency and payment for service execution, we designed a imitation learning-based LOS policy. First, we analyzed the optimal delay of cooperative service execution in parallel, based on which the formulated problem was transformed into SP selection and optimal migration ratio calculation. Through the derivation of the optimal migration ratio, we proposed a matching-based offline expert policy and analyzed the optimality of the proposed expert policy. Based on the obtained expert demonstrations, the distributed agent policy was designed by minimizing the error of state-action pairs distribution to fit the expert policy. Meta update is leveraged to accelerate model transfer for lightweight continuous imitation. Experimental results demonstrated that our algorithm costs lower training time and is superior on various metrics, including average latency, average energy consumption, and average payment under both high and low workload conditions.

7 Acknowledgments

This work was supported by the Natural Science Foundation of China under Grants 61971084, 62025105, 62001073, 62221005 and 62272075, by the National Natural Science Foundation of Chongqing under Grants cstc2021ycjh-bgzxm0039 and cstc2021jcyj-msxmX0031, by the Science and Technology Research Program for Chongqing Municipal Education Commission KJZD-M202200601, by the Support Program for Overseas Students to Return to China for Entrepreneurship and Innovation under Grants cx2021003 and cx2021053, by the UGC General Research Funds No. 17203320 and No. 17209822 from Hong Kong, by the Start-up Grant from The University of Hong Kong.

References

[1] K. B. Letaief, W. Chen, Y. Shi, *et al.*, "The roadmap to 6G: AI empowered wireless networks," *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[2] F. Khan, "Multi-comm-core architecture for terabit-per-second wireless," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 124–129, 2016.

[3] T. Liu, S. Ni, X. Li, *et al.*, "Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2022.

[4] X. Wang, Z. Ning, S. Guo, *et al.*, "Imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 598–611, 2022.

[5] L. Bai, Z. Huang, Y. Li, *et al.*, "A 3D cluster-based channel model for 5G and beyond vehicle-to-vehicle massive MIMO channels," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 8401–8414, 2021.

[6] M. Zhang, Y. Dou, P. H. J. Chong, *et al.*, "Fuzzy logic-based resource allocation algorithm for V2X communications in 5G cellular networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2501–2513, 2021.

[7] F. Tang, Y. Zhou, and N. Kato, "Deep reinforcement learning for dynamic uplink/downlink resource allocation in high mobility 5G HetNet," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 12, pp. 2773–2782, 2020.

[8] S. R. Pokhrel and J. Choi, "Low-delay scheduling for Internet of vehicles: Load-balanced multipath communication with FEC," *IEEE Transactions on Communications*, vol. 67, no. 12, pp. 8489–8501, 2019.

[9] X. Xia, F. Chen, Q. He, *et al.*, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2021.

[10] Z. Ning, P. Dong, X. Wang, *et al.*, "Partial computation offloading and adaptive task scheduling for 5G-enabled vehicular networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[11] D. A. Chekired, M. A. Togou, L. Khoukhi, *et al.*, "5G-slicing-enabled scalable SDN core network: Toward an ultra-low latency of autonomous driving service," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1769–1782, 2019.

[12] G. Faraci, C. Grasso, and G. Schembra, "Design of a 5G network slice extension with MEC UAVs managed with reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2356–2371, 2020.

[13] J. Khamse-Ashari, G. Senarath, I. Bor-Yaliniz, *et al.*, "An agile and distributed mechanism for inter-domain network slicing in next-generation mobile networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[14] S. Fu, J. Gao, and L. Zhao, "Collaborative multi-resource allocation in terrestrial-satellite network towards 6G," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2021.

[15] M. Tang, L. Gao, and J. Huang, "Enabling edge cooperation in tactile internet via 3C resource sharing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2444–2454, 2018.

[16] S. Luo, X. Chen, Z. Zhou, *et al.*, "Fog-enabled joint computation, communication and caching resource sharing for energy-efficient IoT data stream processing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3715–3730, 2021.

[17] J. Oh, Y. Guo, S. Singh, *et al.*, "Self-imitation learning," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 3878–3887, 2018.

[18] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 661–668, 2010.

[19] P. de Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[20] Y. Li, J. Song, and S. Ermon, "InfoGAIL: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[21] J. Hawke, R. Shen, C. Gurau, *et al.*, "Urban driving with conditional imitation learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 251–257, 2020.

[22] Y. Duan, M. Andrychowicz, B. Stadie, *et al.*, "One-shot imitation learning," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[23] E. Jang, A. Irpan, M. Khansari, *et al.*, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Proceedings of the 5th Conference on Robot Learning*, vol. 164, pp. 991–1002, 2022.

[24] H. Karnan, G. Warnell, X. Xiao, *et al.*, "VOILA: Visual-observation-only imitation learning for autonomous navigation," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2497–2503, 2022.

[25] F. Codevilla, M. Müller, A. López, *et al.*, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4693–4700, 2018.

[26] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[27] S. Calinon, *Robot programming by demonstration*. EPFL Press, 2009.

[28] A. Hussein, M. M. Gaber, E. Elyan, *et al.*, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–35, 2017.

[29] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 627–635, 2011.

[30] M. Laskey, J. Lee, R. Fox, *et al.*, "Dart: Noise injection for robust imitation learning," in *Conference on robot learning*, pp. 143–156, PMLR, 2017.

[31] X. Wang, Z. Ning, S. Guo, *et al.*, "Minimizing the age-of-critical-information: An imitation learning-based scheduling approach under partial observations," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[32] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 411–425, 2021.

[33] Z. Yan, P. Cheng, Z. Chen, *et al.*, "Two-dimensional task offloading for mobile networks: An imitation learning framework," *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2494–2507, 2021.

[34] X. Wang, Z. Ning, S. Guo, *et al.*, "Dynamic UAV deployment for differentiated services: A multi-agent imitation learning based approach," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[35] R. G. Kim, W. Choi, Z. Chen, *et al.*, "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2458–2471, 2017.

[36] A. Krishnakumar, S. E. Arda, A. A. Goksoy, *et al.*, "Runtime task scheduling using imitation learning for heterogeneous many-core systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4064–4077, 2020.

[37] B. Yang, J. Zhang, and H. Shi, "Interactive-imitation-based distributed coordination scheme for smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3599–3608, 2021.

[38] N. Deng and M. Haenggi, "The benefits of hybrid caching in gauss–poisson D2D networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1217–1230, 2018.

[39] S. Zhang, P. He, K. Suto, *et al.*, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2018.

[40] C. Wang, J. Li, F. Ye, *et al.*, "NETWRAP: An NDN based real-time wireless recharging framework for wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 6, pp. 1283–1297, 2014.

[41] C. Wang, J. Li, F. Ye, *et al.*, "A novel framework of multi-hop wireless charging for sensor networks using resonant repeaters," *IEEE Transactions on Mobile Computing*, vol. 16, no. 3, pp. 617–633, 2017.

[42] Y. Pan, Q. Chen, N. Zhang, *et al.*, "Extending delivery range and decelerating battery aging of logistics UAVs using public buses," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2022.

[43] X. Wang, Z. Ning, X. Hu, *et al.*, "Optimizing content dissemination for real-time traffic management in large-scale internet of vehicle systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1093–1105, 2019.

[44] Z. Ning, S. Sun, X. Wang, *et al.*, "Blockchain-enabled intelligent transportation systems: A distributed crowdsensing framework," *IEEE Transactions on Mobile Computing*, vol. 21, no. 12, pp. 4201–4217, 2022.

[45] T. Duan, M. Chauhan, M. A. Shaikh, *et al.*, "Ultra efficient transfer learning with meta update for cross subject EEG classification," *ArXiv*, vol. abs/2003.06113, 2020.

[46] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, 2017.

[47] R. G. Andrzejak, K. Lehnertz, F. Mormann, *et al.*, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state," *Physical Review E*, vol. 64, no. 6, p. 061907, 2001.

[48] X. Che, B. Ip, and L. Lin, "A survey of current youtube video characteristics," *IEEE MultiMedia*, vol. 22, no. 2, pp. 56–63, 2015.

[49] Z. Ning, P. Dong, M. Wen, *et al.*, "5G-enabled UAV-to-community offloading: Joint trajectory design and task scheduling," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 11, pp. 3306–3320, 2021.

[50] D. Wu, J. Yan, and H. Wang, "User-centric edge sharing mechanism in software-defined ultra-dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, pp. 1531–1541, 2020.

[51] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2018.

Zhaolong Ning (M'14-SM'18) received the Ph.D. degree from Northeastern University, China in 2014. He was a Research Fellow at Kyushu University from 2013 to 2014, Japan. Currently, he is a full professor with the College of Communication and Information Engineering, the Chongqing University of Posts and Telecommunications, Chongqing, China. His research interests include mobile edge computing, 6G networks, machine learning, and resource management. He has published over 120 scientific papers in international journals and conferences. Dr. Ning serves as an associate editor or guest editor of several journals, such as IEEE Transactions on Industrial Informatics, IEEE Transactions on Social computation Systems, The Computer Journal and so on. He is a Highly Cited Researcher (Web of Science).

Handi Chen received the M.Sc. degrees in Network Engineering in 2022 from the Dalian University of Technology, Dalian, China. She is currently working toward the Ph.D. degree in Department of Electrical and Electronic Engineering, the University of Hong Kong, Hong Kong, China. Her research interests include mobile edge computing, resource allocation and network optimization.

Edith C. H. Ngai (M'08-SM'15) received her Ph.D. from The Chinese University of Hong Kong, Hong Kong in 2007. She was a Postdoctoral Researcher at Imperial College London, United Kingdom from 2007 to 2008. She is currently an Associate Professor with the Department of Electrical and Electronic Engineering, The University of Hong Kong (HKU), Hong Kong. Before joining HKU in 2020, she was an Associate Professor with the Department of Information Technology, Uppsala University, Uppsala, Sweden. Her research interests include the Internet of Things, mobile cloud computing, network security and privacy, data analytics and machine learning for smart cities, and health informatics. She has published more than 100 journals and conference papers in the area. Dr. Ngai was a VINNMER Fellow (2009) awarded by the Swedish Governmental Research Funding Agency VINNOVA.

Xiaojie Wang (SM'22) received the PhD degree from Dalian University of Technology, Dalian, China, in 2019. After that, she was a postdoctor in the Hong Kong Polytechnic University. Currently, she is a full professor with the College of Communication and Information Engineering, the Chongqing University of Posts and Telecommunications, Chongqing, China. Her research interests are wireless networks, mobile edge computing and machine learning. She has published over 50 scientific papers in international journals and conferences, such as IEEE TMC, IEEE JSAC, IEEE TPDS and IEEE COMST.

Lei Guo received the Ph.D. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2006. He is currently a full professor with Chongqing University of Posts and Telecommunications, Chongqing, China. He has authored or coauthored more than 200 technical papers in international journals and conferences. He is an editor for several international journals. His current research interests include communication networks, optical communications, and wireless communications.

Jiangchuan Liu (F'17, IEEE) received the Ph.D. degree from The Hong Kong University of Science and Technology in 2003 (through HKUST's MED program with recommendation from Education Ministry of China and scholarship from Hong Kong Jockey Club; recipient of 2004 Hong Kong Young Scientist Award for the PhD work), Hong Kong, China. He was a Microsoft Research Fellow and worked at Microsoft Research Asia (MSRA) in the summers of 2000, 2001, 2002, 2007, and 2011. He was an EMC-Endowed Visiting Chair Professor with Tsinghua University, Beijing, China, from 2013 to 2016. He was a Professor of Tsinghua Shenzhen Graduate School (2016-2017). He was an Assistant Professor with The Chinese University of Hong Kong, Hong Kong. He is currently a Full Professor (with University Professorship) in the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada. His research interests include multimedia systems and networks, cloud and edge computing, social networking, online gaming, and Internet of Things/RFID/backscatter. Prof. Liu is a fellow of The Canadian Academy of Engineering and an NSERC E.W.R. Steacie Memorial Fellow.