Computer Networks 287 (2026) 112533

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Task completion-oriented service migration for connected autonomous vehicles in multi-server edge computing

Jing Liu [a,b], Jieyi Deng [a,b], Longxin Zhang [c,*], Qiushi Cao [a,b], Wei Hu [a,b], Cen Chen [d], Keqin Li [e]

[a] College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan, 430081, Hubei, China
[b] Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, Wuhan, 430081, Hubei, China
[c] College of Computer Science, Hunan University of Technology, Zhuzhou, 412007, Hunan, China
[d] School of Future Technology, South China University of Technology, Guangzhou, 510641, Guangdong, China
[e] Department of Computer Science, State University of New York, New Paltz, 12561, NY, USA

ARTICLE INFO

Keywords:
Connected autonomous vehicle
Task deadlines
Mobile edge computing
Service migration
User mobility

ABSTRACT

To address the urgent practical challenge of service migration for connected autonomous vehicles (CAVs) in mobile edge computing (MEC), this study aims to maximize the task completion rate, particularly for safety-critical operations. Existing approaches often overlook the completion status of tasks with different priority levels during frequent service migrations and fail to co-optimize multiple constraints such as energy consumption, latency, and offloading cost. Consequently, it remains difficult to reliably complete highly urgent tasks in resource-constrained edge environments. To tackle this issue, we propose a comprehensive two-stage solution: the Improved Task Offloading and Service Migration (ITOSM) algorithm. In the first stage, a weighted evaluation model based on information entropy is constructed by integrating transmission time, execution time, and offloading cost. Tasks are offloaded to the edge server with either the highest or second-highest weighted sum according to their urgency level. In the second stage, service migration decisions are optimized using an improved binary particle swarm optimization (BPSO) algorithm with enhanced local search capability. Experimental results demonstrate that ITOSM outperforms existing methods, achieving up to 10.00% higher completion rates for Extremely Important Tasks (EITs) with strict deadlines. This improvement directly contributes to safer and more reliable CAV operations, highlighting the practical significance of this work for intelligent transportation systems.

1. Introduction

The integration of Internet of Things (IoT), 5G communication, and artificial intelligence (AI) is driving a transformative shift in transportation systems, accelerated by the emergence of connected autonomous vehicles (CAVs). These technologies enhance driving efficiency, safety, and passenger comfort while contributing to reduced emissions and environmental impact [1]. CAVs, such as Baidu's Apollo and Google's Waymo, heavily rely on sensors (e.g., cameras and LiDAR) to collect extensive real-time road environment data. However, the limited computing capacity of on-board units (OBUs) often struggles to process such data efficiently, especially for safety-critical applications that demand ultra-low latency. To address this challenge, mobile edge computing (MEC) has emerged as a key enabler, allowing CAVs to offload compute-intensive or latency-sensitive tasks to nearby edge servers, thereby significantly improving quality of service (QoS) [2]. Nevertheless, vehicle mobility introduces a critical complication: as a CAV moves, the distance to its serving edge server may increase, leading to higher communication latency. To mitigate this, service migration-relocating a CAV's service instance (e.g., virtual machine) to a closer edge server-ensures timely task offloading and execution. However, frequent migrations consume substantial resources, making it difficult to maintain high QoS for all CAVs under constrained edge infrastructure. In practice, orchestrating efficient and reliable service migration for CAVs is a highly complex challenge, as it requires the simultaneous coordination of latency, energy, cost, and task-criticality under dynamic mobility and resource constraints. In this paper, we address the optimization problem of service migration in MEC for CAVs.

Existing research on service migration has made meaningful progress in addressing isolated aspects of this problem, such as latency reduction [3,4] or energy-aware migration [5]. However, the real-world scenario presents a multi-faceted optimization challenge that

* Corresponding author.
E-mail address: longxinzhang@hut.edu.cn (L. Zhang).

https://doi.org/10.1016/j.comnet.2026.112533
Received 16 December 2025; Received in revised form 15 June 2026; Accepted 1 July 2026
Available online 8 July 2026

Table 1
Comparisons of our work with other researches.

Researches	Objectives	Constraints	Service migration
Ding et al. [3]	minimize energy consumption and latency	time and resource	Yes
Ibtissam et al. [4]	minimize the long-term average energy	risk and availability	Yes
Wang et al. [5]	minimize the long-term expected discounted sum total cost		Yes
Shahryari et al. [6]	minimize cost and maximize QoS	time and resource	Yes
Peng et al. [7]	maximize the services' satisfaction degree of delay	time	Yes
Li et al. [8]	minimize the long-term task processing latency	energy, time and resource	Yes
Bozkaya [9]	minimize the task completion delay	time	Yes
Liu et al. [10]	minimize the average completion time	energy	No
Li et al. [11]	minimize the latency and energy consumption	energy, time and resource	Yes
Zhou et al. [12]	minimize the average energy consumption	time and resource	Yes
Li et al. [13]	minimize the average task completion latency	time, power and resource	No
Xu et al. [14]	minimize the latency and energy consumption	resource	Yes
Yu and Zhang [15]	minimize the overall service cost	time	Yes
Chen et al. [16]	minimize the long-term system delays	resource	Yes
Qin et al. [17]	minimize the cost	time, resource, height, distance, QoS	No
Wang et al. [18]	maximize the total system utility	resource	Yes
Li et al. [19]	minimize the average task completion delay	time and resource	Yes
Xing et al. [20]	minimize average execution delay	time and energy consumption	Yes
Ours	maximize number of completed tasks	energy, cost, time and resource	Yes

remains inadequately addressed: the joint optimization of task completion, energy efficiency, deadline satisfaction, and offloading cost under heterogeneous and resource-limited edge environments. Current approaches often focus on singular objectives or simplified constraints, which limits their applicability in realistic CAV operations where multiple system requirements must be balanced concurrently [6]. Table 1 shows the differences between our work with other related researches across three key dimensions: optimization objectives, constraints, and whether service migration is supported. The core of the problem lies in designing a migration strategy that can co-optimize multiple competing constraints-migration energy consumption, computational offloading costs, task latency, and overall task completion rates—while operating within strict resource limits and ensuring timely execution of safety-critical tasks. This system-level coordination is particularly vital for CAV applications, as it directly influences both operational safety and service reliability.

To tackle this integrated challenge, we propose a solution—the Improved Task Offloading and Service Migration (ITOSM) algorithm. Our approach is designed to function as a comprehensive two-stage decision framework that holistically addresses the multi-constrained nature of CAV service migration. In the first stage, a weighted offloading strategy evaluates candidate edge servers using a composite metric based on information entropy, which incorporates transmission time, execution time, and offloading cost. Tasks are then assigned according to urgency, selecting servers with the highest or second-highest weighted score to enhance robustness. In the second stage, an enhanced binary particle swarm optimization (BPSO) mechanism is introduced to optimize migration decisions, with improved local search capability to refine solutions in later iterations.

Contributions of this paper are as following:

- We introduce a composite utility function derived from information-theoretic principles, which synthesizes transmission latency, execution time, and offloading cost into a scalarized score. This metric enables a principled ranking of candidate edge servers, with task urgency dictating whether the highest or second-highest ranked server is selected—a design that enhances robustness against localized resource fluctuations.
- We introduce ITOSM, a two-stage algorithmic framework that systematically coordinates task offloading and service migration under multiple practical constraints including energy, cost, latency, and task urgency.
- Through extensive simulations, we demonstrate that ITOSM achieves superior performance in task completion rates—especially for Extremely Important Tasks (EITs) with stringent deadlines—while co-optimizing energy and cost efficiency. This validates ITOSM as a viable and effective engineering solution for enhancing the safety and reliability of CAV operations in intelligent transportation systems.

The remainder of this article is structured as follows. Section 2 reviews and compares related research. Section 3 details our system model and problem formulation. In Section 4, we present the proposed algorithm. Section 5 evaluates the performance of our method through comprehensive simulations. Finally, Section 6 concludes the paper and outlines future research directions.

2. Related work

In mobile edge computing, user mobility-induced latency is addressed through task migration and service migration. Task migration transfers computational tasks to alternative edge servers, while service migration relocates service instances (e.g., virtual machines) to maintain seamless service delivery.

2.1. Task migration

To mitigate the latency caused by user mobility, several studies have explored task migration strategies. For example, a deep reinforcement learning method has been proposed to optimize task migration while accounting for migration energy consumption [10]. Another approach leverages a deep Q-network (DQN) to learn migration policies from historical data without requiring prior knowledge of user mobility patterns [21]. Further work considers both user mobility and task deadlines, introducing a heuristic scheme that dynamically selects offloading locations to improve deadline compliance [22]. In vehicular networks, an Internet of Vehicles (IoV)-oriented migration method under sixth-generation (6G) networks has also been developed, aiming to balance energy and time costs during migration to enhance user experience [14].

However, these approaches overlook the role of service instances. In CAV scenarios, tasks rely on dedicated service instances; when a vehicle moves beyond server coverage, the instance must be migrated to ensure continuity. Several studies address task offloading in vehicular contexts [13,23,24], but they do not consider service migration. This distinction motivates our focus on service migration as a holistic solution for continuous CAV operations.

2.2. Service migration

Service migration in MEC has been widely studied with diverse optimization objectives. Peng et al. [7] propose a cost-aware method that jointly considers communication and computing expenses to maximize delay satisfaction in vehicular edge networks. Zhou et al. [12] introduce an online algorithm named EGO, which combines particle swarm optimization and Lyapunov optimization to minimize energy consumption under latency and interference constraints. Liu and Sun [25] design a software-defined networking-based service migration mechanism for UAV (unmanned aerial vehicle)-assisted vehicular edge computing scenarios, minimizing task delay while meeting long-term energy constraints. Bozkaya [9] employs digital twin technology to reduce task completion time through predictive service migration. Ouyang et al. [26] adopt an online learning framework to reduce migration cost and latency while enhancing personalized service performance. Li et al. [11] introduce the EOSM offloading and migration strategy, which alleviates overload issues across multiple edge devices to minimize access delay and energy consumption under multiple constraints of capacity, time, and energy. Li et al. [8] propose a joint resource allocation and service migration algorithm based on a branching deep Q-network (branching DQN) for intelligent buoy-enabled maritime mobile edge computing networks, with the objective of minimizing the long-term task processing delay of maritime IoT users and ensuring service continuity. Xing et al. [20] construct a decision model for task offloading and migration in mobile edge computing, aiming to minimize the average execution delay under different mobility patterns while satisfying both time and energy constraints.

Despite these advances, existing service migration schemes exhibit three key limitations: optimization targets are limited to single metric, such as latency, energy, or cost, neglecting task urgency and completion rates; task completion maximization under multiple constraints (migration energy, offloading costs, differentiated deadlines) remains unaddressed; the unique requirements of CAV applications, particularly EITs with stringent deadlines, are under explored. In this paper, we explicitly target task completion maximization under multiple practical constraints. Our method achieves significant gains in completion rates, especially for EITs, directly enhancing CAV safety and reliability.

3. Models and problem definition

This section first summarizes the notations used in this paper in Table 2, then introduces several important concepts, presents the system model, and finally formulates the service migration optimization problem for CAVs in heterogeneous MEC environments.

3.1. Important conceptions

A task: it is a computational work that a computing system needs to deal with, such as obstacle detection.

A time slot: continuous time is partitioned into multiple discrete intervals of equal or variable length, and each interval is called a time slot.

Uploading: it is the process of transmitting data from end devices (e.g., CAVs) to remote servers (e.g., edge servers or cloud).

Downloading: it is the process of transmitting data from remote servers back to end devices.

Offloading: it is the process of transferring computational tasks from devices (e.g., CAVs) with limited resource to remote computing nodes (e.g., servers) with rich resource for execution.

Service migration: it is the behavior of relocating a running service instance (e.g., virtual machine) from one computing node to another to keep service continuity.

Local processing: it is the behavior that a task is directly executed on the device that generated the task.

Table 2
Notations.

Notation	Definition
M	Number of edge servers
N	Number of CAVs
K	Number of tasks generated by each CAV
ES_j	The jth edge server
VU_i	The ith CAV
VM_i	Service instance for CAV VU_i
v_k^i	The kth task of CAV VU_i
$loc_{i,k}$	The execution position of task v_k^i
$\varphi_{i,j,k}$	Offloading decision for task v_k^i on ES_j
$z_{i,j',j,k}$	Migration decision of the service instance for CAV task v_k^i
$\lambda_{i,k}$	Size of input data for task v_k^i
$O_{i,k}$	Size of output data for task v_k^i
$C_{i,k}$	Size of workload for task v_k^i
$W_{i,k}$	Type of task v_k^i
$D_{i,k}$	Deadline of task v_k^i
$r_{i,j,k}$	The data transmission rate between ES_j and VU_i
W_j	Wireless bandwidth
p_i	Transmission power of CAV VU_i
$h_{i,j}$	Channel gain
σ	Noise power
μ	Channel path loss index
$d_{i,j,k}$	Distance between the edge server ES_j and CAV VU_i
B	Wired Bandwidth
S_i	Size of the service instance VM_i for CAV VU_i
$N_{i,j,k}$	Number of tasks executed on ES_j
f_i^{max}	Maximum computing capacity of CAV VU_i
f_j^{max}	Maximum computing capacity of edge server ES_j
$T_{i,k}^l$	Local execution latency of task v_k^i
$T_{i,k}^m$	required latency of v_k^i on an edge server
$T_{i,k}$	The actual time to execute v_k^i
$t_{i,j,k}^u$	Transmission latency of task v_k^i to edge server ES_j
$t_{i,j,k}^e$	Task execution latency of v_k^i on edge server ES_j
$t_{i,j,k}^m$	Task migration latency from ES_j' to ES_j to execute v_k^i
$t_{i,j,k}^d$	The latency to transfer back the computational result of task v_k^i
$E_{i,j,k}$	Migration energy consumption when CAV VU_i executes task v_k^i
E_b	Energy consumption per bit for migration
$\theta_{f_{i,j,k}}$	Offloading unit price corresponding to computing power $f_{i,j,k}$
C_i^b	Offloading cost budget for CAV VU_i
E^b	Energy consumption budget for migration
$y_{p,i,k}^t$	the element in the ith row and k column of velocity Y_p^t
$x_{p,i,k}^t$	the element in the ith row and k column of position Y_p^t
$pbloc_{p,i,k}^t$	the element in the ith row and k column of position $pbloc_p^t$
$gbloc_{i,k}^t$	the element in the ith row and k column of position $gbloc^t$
$X.F$	the fitness value of position X

3.2. System model

We consider a heterogeneous MEC system consisting of four main components: mobile CAVs, base stations, a wireless access network, and edge servers. Each base station is co-located with an edge server, providing computational resources within its geographic coverage area. Edge servers are heterogeneous in terms of computing capacity and memory, each with finite resources. CAVs are mobile and may traverse multiple edge server coverage zones during operation. Each CAV carries a set of tasks that must be processed while in motion. Upon entering the service area of an edge server, a CAV can communicate via the wireless access network and offload tasks to that server for execution. Let $ES = \{ES_1, ES_2, \ldots, ES_M\}$ denote the set of edge servers, $VU = \{VU_1, VU_2, \ldots, VU_N\}$ the set of CAVs, and $V_i = \{v_1^i, \ldots, v_k^i, \ldots, v_K^i\}$ the set of tasks associated CAV VU_i, where $1 \le k \le K$ and $1 \le i \le N$. Here M, N, K represent the total number of edge servers, CAVs, and tasks per CAV, respectively. Edge server resources are allocated in the form of virtual machines (VMs), each representing a service instance deployed on a server. In this paper, the terms service and service instance are used interchangeably. Each CAV VU_i employs a dedicated service instance VM_i to execute all its tasks; that is, every task of a given CAV must be processed by its own service instance. We assume that all service instances are available across edge servers and operate independently of one another.

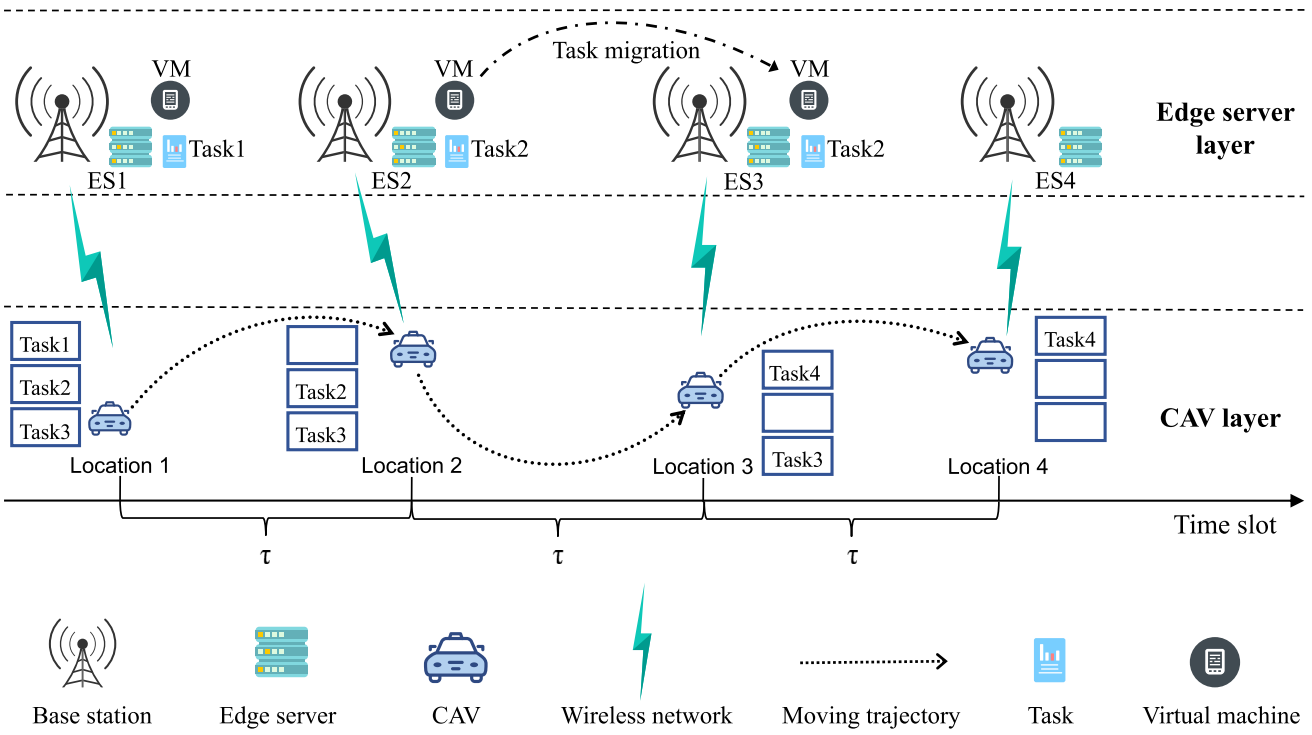


Fig. 1. An example of an end-to-end proposed system architecture.

Fig. 1 illustrates an example of the proposed end-to-end system architecture. The system consists of four base stations, four edge servers, and a connected autonomous vehicle (CAV). During the first time slot τ at location 1, the CAV has three tasks to handle, and task 1 is offloaded to edge server $ES1$ for execution. In the second time slot at location 2, task 2 is first offloaded to $ES2$ and then migrated, along with its service instance, to $ES3$. Finally, during the third time slot at location 3, task 3 is executed locally on the CAV itself.

CAV tasks vary significantly in their characteristics and requirements. For instance, collision prevention is highly delay-sensitive; failure to complete such a task within its deadline may lead to severe safety consequences. In contrast, video streaming is considerably less delay-sensitive, and missing its deadline typically does not result in critical outcomes. To effectively manage service quality for CAV users, we categorize all tasks into three priority levels based on urgency: Extremely Important Tasks (EIT), assigned a priority value of 3; Moderately Important Tasks (MIT), assigned a value of 2; and Non-critical Tasks (NIT), assigned a value of 1. Thus, a higher priority value corresponds to greater task urgency.

Each task v_k^i of CAV VU_i is characterized by a five-tuple $(\lambda_{i,k}, C_{i,k}, O_{i,k}, W_{i,k}, D_{i,k})$. Here, $\lambda_{i,k}$ (in bits) denotes the input data size, $C_{i,k}$ (in CPU cycles/bit) the required CPU cycles per bit for execution, and $O_{i,k}$ the output data size. The parameter $W_{i,k} \in \{1, 2, 3\}$ indicates the task type (priority level), and $D_{i,k}$ (in seconds) represents the task deadline. The value of $D_{i,k}$ is determined according to the urgency associated with the task type $W_{i,k}$.

3.3. CAV mobility model

We adopt a discrete mobility model for CAVs similar to those used in [3,27,28]. In this model, the trajectory of each CAV is represented as a sequence of discrete position coordinates. The total driving time T is divided into equal time slots of duration τ. The interval between two consecutive locations corresponds to one time slot, meaning that a CAV moves from its current position to the next within a single slot. Each time a CAV reaches a new location, it may begin executing a task or initiate a service migration. The execution time of any task is constrained not to exceed τ. Consequently, the discrete movement trajectory of a CAV can be equivalently viewed as the timeline along which its tasks are processed.

Let $g\left(v_k^i\right) = \left(lat_{v_k^i}, lng_{v_k^i}\right)$ denote the latitude and longitude coordinates of CAV VU_i at the start of task v_k^i. Since each base station is co-located with its edge server, we write $g(j) = \left(lat_j, lng_j\right)$ for the geographic coordinates of the jth base station and the associated edge server ES_j. The variable $loc_{i,k} = j$ indicates that task v_k^i of CAV VU_i is executed on edge server ES_j, where the service instance VM_i is deployed, with $j \in \{1, 2 \ldots, M\}$. If $loc_{i,k} = 0$ the task is executed locally on the CAV itself. Thus, each task may be processed either locally or offloaded to an edge server. To explicitly capture the offloading decision, we define a binary variable $\varphi_{i,j,k}$ such that $\varphi_{i,j,k} = 1$ if task v_k^i is offloaded to edge server ES_j for execution, and $\varphi_{i,0,k} = 1$ if it is executed locally on VU_i.

3.4. Communication model

We consider a multi-user MEC environment in which simultaneous task offloading from multiple CAVs to the same edge server would, in general, introduce co-channel interference. Following common practice in related literature [3,29–33], we adopt orthogonal frequency division multiple access (OFDMA) for uplink transmissions. OFDMA ensures orthogonality among subchannels, thereby eliminating co-channel interference and allowing concurrent task offloading from multiple CAVs. This article assumes channel reciprocity and utilizes the same transmit power for both uplink and downlink transmissions, which is commonly adopted in MEC literature [3,33] to focus on the task offloading and service migration optimization, as the symmetric data rate does not influence the relative performance comparison of proposed strategies. Consequently, the data transmission rate $r_{i,j,k}$ between CAV VU_i and edge server ES_j for task v_k^i is computed by

$$r_{i,j,k} = W_j log_2\left(1 + \frac{p_i h_{i,j}^2}{\sigma^2 d_{i,j,k}^{\mu}}\right), \tag{1}$$

where W_j is the wireless bandwidth allocated by ES_j, p_i is the transmit power of VU_i, $h_{i,j}$ is the channel gain, σ^2 is the noise power, and $d_{i,j,k}$ is

the Euclidean distance between VU_i and ES_j at the time v_k^i is executed. The exponent μ is the path-loss factor.

3.5. Migration model

Each edge server operates within a limited service range and has finite computing resources. When a CAV moves beyond the coverage of its current edge server, service interruption may occur. Moreover, as the number of tasks offloaded to an edge server increases, the computational resources allocated to each task decrease, which can degrade the quality of service and user experience. To maintain service continuity and ensure satisfactory performance, a viable approach is to migrate the service instance associated with a CAV's tasks—via a wired backhaul network—to another edge server that is geographically closer to the vehicle's direction of travel. Migration involves transferring both the pending task's input data and the service instance itself to the target server, thereby incurring a migration delay. Conversely, if sufficient computational resources are available on the current server to execute a task, migration is unnecessary; in this case, only the computational results need to be forwarded to the edge server nearest the CAV.

Let $z_{i,j',j,k} \in \{0,1\}$ denote the migration decision of the service instance VM_i for executing task v_k^i of CAV VU_i, where $z_{i,j',j,k} = 1$ indicates that VM_i is migrated from the original edge server $ES_{j'}$ to the new edge server ES_j, and $z_{i,j',j,k} = 0$ otherwise. The service instance VM_i of CAV VU_i can reside on at most one edge server when executing v_k^i. We assume that each CAV starts executing a new task upon arriving at a new location, and the execution time of any task does not exceed the time slot τ.

Based on above analysis, the migration time, denoted by $t_{i,j,k}^m$, can be expressed as:

$$t_{i,j,k}^m = z_{i,j',j,k} \frac{\lambda_{i,k} + S_i}{B} + \left(1 - z_{i,j',j,k}\right) \frac{O_{i,k}}{B}, \tag{2}$$

where B is the communication channel bandwidth between edge servers, and S_i is the size of the service instance VM_i of CAV user VU_i.

3.6. Computational model

We assume that multiple service instances can be deployed concurrently on the same edge server within a time slot, enabling parallel execution of tasks from different CAVs. The server divides its total computing capacity among all active offloaded tasks. Following the resource allocation model in [34], the computing capacity allocated to each task is assumed to be inversely proportional to the number of tasks being processed simultaneously on the server. In other words, as more CAV tasks are offloaded to an edge server, the share of computational resources per task decreases. Unlike [35], which considers potential failures under overload, our model adopts a simplified scenario that does not account for overload-related failures. If the computational resources assigned to a task on a given server are insufficient to meet its requirements, the associated service instance can be migrated to another edge server with more available capacity. Let $N_{i,j,k}$ denote the total number of tasks (including v_k^i) executed on edge server ES_j when CAV VU_i processes task v_k^i. This quantity is expressed as: $N_{i,j,k} = \sum_{i' \in N} H\{loc_{i',k'} = j\}$, where $H\{\cdot\}$ is the indicator function that returns 1 if the condition holds and 0 otherwise. Here, $loc_{i',k'} = j$ indicates that task $v_{k'}^{i'}$ is offloaded to ES_j. Let f_j represent the total computing capacity (CPU frequency) of edge server ES_j. Then, the computing capacity allocated to task v_k^i on ES_j is given by $\frac{f_j}{N_{i,j,k}}$. In the following, we derive the time required to complete a CAV task under this resource allocation model.

When task v_k^i of CAV VU_i is executed locally on the vehicle (i.e., the CAV itself), the execution time, denoted by $T_{i,k}^l$, is computed as:

$$T_{i,k}^l = \frac{\lambda_{i,k} C_{i,k}}{f_i}, \tag{3}$$

where f_i represents the computing capacity (CPU frequency) of VU_i.

When v_k^i is executed on edge server, the required time, denoted by $T_{i,k}^m$, is computed as:

$$T_{i,k}^m = t_{i,j',k}^u + \left((1 - z_{i,j',j,k}) \times t_{i,j',k}^e + z_{i,j',j,k} \times t_{i,j,k}^e\right) + t_{i,j,k}^m + t_{i,j,k}^d, \tag{4}$$

where $t_{i,j',k}^u$, $t_{i,j',k}^e$, $t_{i,j,k}^e$, and $t_{i,j,k}^d$ represent the data upload time, the data execution time, the data execution time, and the data return time, respectively. These components are described in detail in the following.

Data upload time: When a CAV task is fully offloaded to an edge server, its input data must be uploaded via wireless transmission to the edge server where the corresponding service instance is deployed. The transmission time, denoted by $t_{i,j',k}^u$, for uploading the input data $\lambda_{i,k}$ of task v_k^i to edge server $ES_{j'}$ (where the service instance VM_i resides) is expressed as:

$$t_{i,j',k}^u = \frac{\lambda_{i,k}}{r_{i,j',k}}. \tag{5}$$

Data execution time: The execution time, denoted by $t_{i,j',k}^e$, for executing v_k^i on $ES_{j'}$ is computed as:

$$t_{i,j',k}^e = \frac{\lambda_{i,k} C_{i,k}}{\frac{f_{j'}}{N_{i,j',k}}} = \frac{\lambda_{i,k} C_{i,k} N_{i,j',k}}{f_{j'}}, \tag{6}$$

where $\frac{f_{j'}}{N_{i,j',k}}$ is the computing capacity allocated to VU_i by $ES_{j'}$ for executing v_k^i.

Data return time: For any CAV task executed on an edge server, the final computational result (i.e., output data) must be sent back to the CAV from another edge server, incurring a data return time. The output data return time, denoted by $t_{i,j,k}^d$, from edge server ES_j for task v_k^i is computed by

$$t_{i,j,k}^d = \frac{O_{i,k}}{r_{i,j,k}}. \tag{7}$$

Therefore, the actual time to complete task v_k^i of CAV VU_i is given by

$$T_{i,k} = \varphi_{i,j,k}\, T_{i,k}^m + \left(1 - \varphi_{i,j,k}\right) T_{i,k}^l. \tag{8}$$

3.7. Energy model

In edge computing environments, CAVs are equipped with on-board power supplies to ensure stable energy availability. Since the energy consumed for computation and task offloading is negligible compared to the propulsion energy required for driving, we do not model the energy consumption of the CAVs themselves. However, when a service migration occurs, both the pending task and its associated service instance must be transferred to a new edge server. This process incurs substantial energy overhead, particularly on the source edge server due to data transmission, which is non-negligible in practice [10]. Therefore, following [10], we explicitly incorporate the energy consumption of service migration as a constraint in our optimization framework.

The migration energy consumption, denoted by $E_{i,j,k}$, for migrating task v_k^i of CAV VU_i and its associated service instance from the original edge server $ES_{j'}$ to the new edge server ES_j, is expressed as:

$$E_{i,j,k} = z_{i,j',j,k} \left(\lambda_{i,k} + S_i\right) E_b + \left(1 - z_{i,j',j,k}\right) O_{i,k} E_b, \tag{9}$$

where E_b denotes the energy consumed by the original edge server for transmitting one bit of data. The first term in Eq. (9) represents the energy consumed for migrating the input data and service instance of a CAV task to a new edge server, while the second term corresponds to the energy consumed for transmitting the output data without migration. If a CAV task is executed locally, no migration occurs, and the migration energy consumption is zero.

The total migration energy consumption, denoted by E^t, for CAV tasks and their associated service instances can be calculated as:

$$E^t = \sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{j=1}^{M} E_{i,j,k}, \tag{10}$$

and it must not exceed the given energy consumption budget E^b; i.e., $E^t \le E^b$.

3.8. Cost model

A CAV task can be executed either locally on the vehicle or offloaded to an edge server. Local execution incurs no offloading cost, whereas offloaded tasks require the CAV to lease computing resources from the edge server, with the cost depending on both the leased capacity and the execution time. In a heterogeneous edge environment, the computing capacity available to a CAV varies with its location and the current load on edge servers. Let the computing capacity allocated to CAV VU_i by edge server ES_j for task v_k^i be denoted as $f_{i,j,k}$, where $f_{i,j,k} = \frac{f_j}{N_{i,j,k}}$, and let the corresponding unit offloading price for $f_{i,j,k}$ be $\theta_{f_{i,j,k}}$. Then, the total cost C_i^t incurred by CAV VU_i for offloading tasks to edge servers is expressed as

$$C_i^t = \sum_{k=1}^{K} C_{ik}, \tag{11}$$

$$C_{ik} = \left(1 - z_{i,j',j,k}\right)\theta_{f_{i,j',k}} t^e_{i,j',k} + z_{i,j',j,k}\theta_{f_{i,j,k}} t^e_{i,j,k}, \tag{12}$$

where C_i^t is the offloading cost of task v_k^i. The first term in Eq. (12) represents the offloading cost when the service instance is not migrated, while the second term corresponds to the offloading cost when migration occurs. For each VU_i, the total offloading cost C_i^t must not exceed its allocated budget C_i^b; that is, $C_i^t \le C_i^b$.

3.9. Problem description

In this paper, we aim to develop an efficient service migration strategy to maximize the number of successfully completed CAV tasks in a heterogeneous MEC system, subject to multiple practical constraints, including CAV offloading costs, migration energy consumption at edge servers, and task deadlines. Let $\mathcal{N}$ denote the number of successfully completed CAV tasks, defined as:

$$\mathcal{N} = \sum_{i=1}^{N}\sum_{k=1}^{K} H\{T_{i,k} \le D_{i,k}\}. \tag{13}$$

For clarity, we denote the problem addressed in this paper as the Service Migration Optimization Problem (SMOP) and formulate it as follows:

$$\begin{aligned}
P1: \quad & \max_{Z,\varphi} \ \mathcal{N}, \\
\text{s.t.} \quad & C_1: \ 0 \le f_i \le f_i^{max}, 1 \le i \le N, \\
& C_2: \ 0 \le f_j \le f_j^{max}, 1 \le j \le M, \\
& C_3: \ \textstyle\sum_{j=1}^{M} \varphi_{i,j,k} \le 1, \varphi_{i,j,k} \in \{0,1\}, \\
& \qquad 1 \le i \le N, 1 \le k \le K, \\
& C_4: \ \textstyle\sum_{j=1}^{M} z_{i,j',j,k} \le 1, z_{i,j',j,k} \in \{0,1\}, \\
& \qquad 1 \le i \le N, 1 \le k \le K, 1 \le j' \le M, \\
& C_5: \ E^t \le E^b, \\
& C_6: \ C_i^t \le C_i^b, 1 \le i \le N.
\end{aligned} \tag{14}$$

In problem $P1$, the decision variables Z denotes the set of all service migration decisions, and φ denotes the set of all task offloading decisions for CAVs. C_1 and C_2 specify the computing capacity limits for CAVs and edge servers, respectively. Constraint C_3 ensures that each CAV task is offloaded to at most one device—either the local CAV or an edge server. Constraint C_4 states that each service instance can be migrated to at most one target server at a time. Constraint C_5 bounds the total energy consumption of service migrations by the given energy budget for edge servers. Constraint C_6 restricts the total offloading cost of each CAV to remain within its allocated cost budget.

4. Methodology

This section presents the Improved Task Offloading and Service Migration (ITOSM) algorithm, which is designed to solve the formulated service migration optimization problem.

4.1. Task offloading strategy

In CAV edge computing scenarios, task offloading is essential for reducing onboard computational burden and ensuring real-time performance. When multiple edge servers are available, selecting the nearest server by geographic distance—a common approach—often fails to achieve optimal results due to factors such as heavy load and high costs. To address this, we propose a task offloading strategy based on weighted multi-criteria evaluation, which balances multiple conflicting objectives such as low latency and cost efficiency. A key enhancement is the adoption of the entropy weight method to allocate objective weights to performance metrics, eliminating subjective bias and improving the scientific rigor of offloading decisions.

Let q_{j1}, q_{j2} and q_{j3} be the execution time, the transmission time and the offloading cost of the CAV task associated with candidate edge server j, respectively, where $j = 1, 2, \ldots, \mathcal{H}$ and $\mathcal{H}$ represents the total number of candidate edge servers within the CAV's communication range. For task v_k^i, its execution time q_{j1} is computed by Eq. (6). Its transmission time q_{j2}, which refers to the sum of uplink transmission delay (from CAV to edge server) and downlink transmission delay (from edge server to CAV), is derived from Eqs. (5) and (7). Its offloading cost q_{j3} is obtained by Eq. (12). Notably, these three metrics have different dimensions and orders of magnitude, which would result in unfair evaluation if directly used for weighted computation. To eliminate dimensional discrepancies and guarantee the fairness of multi-criteria evaluation, the min–max normalization method is used to map the values of these metrics to the interval $[0, 1]$. We have

$$b_{jn} = \frac{\max_{1\le i\le\mathcal{H}} q_{in} - q_{jn}}{\max_{1\le i\le\mathcal{H}} q_{in} - \min_{1\le i\le\mathcal{H}} q_{in}}, 1 \le n \le 3, \tag{15}$$

where b_{jn} is the normalized value of the nth metric for candidate edge server j. For all three metrics, a smaller original value corresponds to a larger b_{jn}, indicating a better performance of the edge server in this metric.

Next, b_{jn} is converted into a probability matrix to compute the information entropy, which is the core of the entropy weight method. The probability value l_{jn} of the nth metric for edge server j is computed by

$$l_{jn} = \frac{b_{jn}}{\sum_{j=1}^{\mathcal{H}} b_{jn}}, \tag{16}$$

where $0 \le l_{jn} \le 1$ and $\sum_{j=1}^{\mathcal{H}} l_{jn} = 1$, ensuring that l_{jn} meets a probability distribution.

According to the probability matrix, the information entropy denoted by E_n of the nth metric is computed to quantify its discriminative power—the ability to distinguish the performance differences among all candidate edge servers. We have

$$E_n = -\frac{1}{\ln\mathcal{H}} \sum_{j=1}^{\mathcal{H}} l_{jn} \ln l_{jn}, \tag{17}$$

where $0 \le E_n \le 1$. A smaller E_n indicates higher data dispersion of the metric, meaning the metric has stronger discriminative power and greater contribution to the offloading decision.

Then, the objective weight denoted by δ_n for each metric is obtained from E_n, reflecting the relative importance of the nth metric in the multi-criteria evaluation. We have

$$\delta_n = \frac{1 - E_n}{\sum_{n=1}^{3}(1 - E_n)}, \tag{18}$$

where $1 - E_n$ is the information utility of the nth metric. The sum of all weights is 1. By doing this, it can avoid the subjectivity of manual weight setting and make the offloading decision more reliable and scientific.

Finally, the comprehensive evaluation score denoted by s_j for each candidate edge server j is attained by the weighted sum of the three original metrics and their weights. We have

$$s_j = \sum_{n=1}^{3} q_{jn} \times \delta_n, 1 \le j \le \mathcal{H}. \tag{19}$$

This score integrates the execution time, transmission delay, and offloading cost to assess the performance of each candidate edge server. A larger s_j indicates that the candidate edge server j has better overall performance regarding execution efficiency, transmission speed, and cost control.

Algorithm 1 Entropy-weight-based multi-criteria task offloading strategy

Require: $VU, V_i, W_{i,k}, ES, \lambda_{i,k}, C_{i,k}, O_{i,k}, S_i, B$
Ensure: The offloading strategy for CAV task v_k^i
1: **for** $VU_i \in VU$ **do**
2: **for** $v_k^i \in V_i$ **do**
3: According to the geographic coordinate of VU_i at the time of executing task v_k^i, obtain all candidate edge servers available at the current location
4: **if** $W_{i,k} = 1$ **then**
5: $\varphi_{i,0,k} \leftarrow 1$, namely, v_k^i is executed on local CAV VU_i
6: **end if**
7: **if** $W_{i,k} > 1$ **then**
8: **for** each candidate server of v_k^i **do**
9: compute the comprehensive evaluation score for the candidate edge server by Equation (19)
10: **end for**
11: **if** $W_{i,k} = 3$ **then**
12: $\varphi_{i,j,k} \leftarrow 1$, namely, offload v_k^i to the candidate server ES_j with the largest evaluation score
13: **else**
14: **if** there exist at least two candidate edge servers **then**
15: $\varphi_{i,j,k} \leftarrow 1$, namely, offload v_k^i to the candidate server ES_j with the second largest evaluation score
16: **else**
17: offload v_k^i to the only available server ES_j
18: **end if**
19: **end if**
20: **end if**
21: **end for**
22: **end for**

Algorithm 1 shows the task offloading strategy based on the entropy weight method. The algorithm first computes the candidate edge servers currently available based on the geographic locations of vehicles at the time of task execution for all CAV tasks (lines 1–3). Then, it determines whether to offload tasks to an edge server according to the priorities of tasks. If the priority of a task is 1, the task has low real-time requirement and is executed locally on the CAV (lines 4–6). If the priority of the task is larger than 1, the task needs to be offloaded to an edge server to guarantee quality of service. For tasks requiring offloading, the algorithm uses the entropy weight method to determine indicator weights and avoid subjective bias. Specifically, when the priority is greater than 1, the algorithm computes a comprehensive evaluation score for each candidate edge server via Eq. (19) (lines 8–10). For high priority tasks with a priority of 3, the algorithm selects the candidate server with the highest score to ensure optimal execution performance (lines 11–12). For medium priority tasks with a priority of 2, if at least two candidate servers are available, the server with the second highest score is chosen to prevent overloading the optimal server, thus balancing single-task performance with overall system load (lines 13–15). Otherwise, the task is offloaded to the only available server (lines 16–18). Finally, the algorithm outputs the offloading decision of all CAV tasks.

4.2. The service migration strategy

To maintain continuous service and ensure a high-quality user experience as a CAV moves across different edge-server coverage areas, service migration is essential. Effective migration enables tasks with varying urgency levels to be completed in a timely manner, thereby significantly improving both safety and the overall driving experience. The discrete binary particle swarm optimization (BPSO) algorithm is widely adopted for solving combinatorial optimization problems, owing to its strong global search capability and its tendency to converge to high-quality solutions [36]. The algorithm is inspired by the collective foraging behavior of bird flocks, in which individuals share information to collaboratively locate optimal regions. In BPSO, each particle represents a candidate solution and is characterized by its position and velocity. Throughout the iterative process, particles continuously update their positions and velocities, guiding the swarm toward the global optimum. However, BPSO is known to exhibit limited local search ability in later iterations, which can hinder fine-tuning of the solution. In this work, we retain the global exploration strength of BPSO while enhancing its local exploitation capability, resulting in an improved service migration strategy. The details of the proposed approach are elaborated below.

4.2.1. Constructions of the particles and fitness function

The solution to our studied optimization problem is the service migration decisions of all tasks for all CAVs. Thus, every particle can be represented by the service migration decisions of all CAV tasks. We use the following binary decision matrix X_p to represent the p-th particle in the population

$$X_p = \begin{bmatrix} x_{1,1} & \cdots & x_{1,k} & \cdots & x_{1,K} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ x_{i,1} & \cdots & x_{i,k} & \cdots & x_{i,K} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ x_{N,1} & \cdots & x_{N,k} & \cdots & x_{N,K} \end{bmatrix}, \tag{20}$$

where each element $x_{i,k} \in X_p$ represents the migration decision of the service instance for task v_k^i of CAV VU_i, $x_{i,k} \in \{0, 1\}$, $1 \le i \le N$, $1 \le k \le K$. Also, X_p is the position of the p-th particle.

To evaluate the quality of a solution (i.e., the binary decision matrix), we devise a fitness function which is computed by

$$F = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(D_{i,k} - T_{i,k}\right)^2 + \rho\left(\max\left(0, E^t - E^b\right) + \max\left(0, \sum_{i=1}^{N}\left(C_i^t - C_i^b\right)\right)\right), \tag{21}$$

where ρ is a penalty factor for violating the energy and cost constraints, and set to be 10000. When task v_k^i is not completed, its completion time $T_{i,k}$ is set to be zero. The more tasks are completed in time, the smaller value of F is.

Each particle has a velocity that can change its position. We use the matrix Y_p to represent the velocity of the pth particle in the population as

$$Y_p = \begin{bmatrix} y_{1,1} & \cdots & y_{1,k} & \cdots & y_{1,K} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ y_{i,1} & \cdots & y_{i,k} & \cdots & y_{i,K} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ y_{N,1} & \cdots & y_{N,k} & \cdots & y_{N,K} \end{bmatrix}, \tag{22}$$

where each element $y_{i,k} \in Y_p$ is random number from the interval $[-y_{max}, y_{max}]$.

4.2.2. Rules for updating the position and velocity of particles

In the proposed service migration strategy, the search for optimal solutions is driven by the iterative updating of particle positions and velocities. To balance global exploration and local exploitation, different update rules are applied in different phases of the search process. During early iterations, emphasis is placed on global exploration to

quickly identify promising regions of the solution space. For this phase, we adopt the standard BPSO update rules to guide particle movement. In later iterations, the focus shifts to refining the solutions by enhancing local search capability. To this end, we redesign the update rules to improve exploitation in the neighborhood of the current best solution. The update rules for the early exploration phase are detailed as follows.

Let $pbloc_p^t$ and $gbloc^t$ be separately the best previous position of the particle p and the best previous position of all particles (i.e., the best previous position globally) after t iterations, Y_p^t be the velocity of the particle p at the $t-$th iteration. Then, the element $y_{p,i,k}^t \in Y_p^t$ is updated by

$$y_{p,i,k}^t = \omega \times y_{p,i,k}^{t-1} + \alpha_1 \times \beta_1 \times \left(pbloc_{p,i,k}^{t-1} - x_{p,i,k}^{t-1}\right) + \alpha_2 \times \beta_2 \times \left(gbloc_{i,k}^{t-1} - x_{p,i,k}^{t-1}\right), \tag{23}$$

where ω is the inertia factor, α_1 and α_2 are the learning factors, β_1 and β_2 are random numbers from the interval $[0, 1]$, and $x_{p,i,k}^{t-1} \in X_p^{t-1}$. The first term on the right-hand side represents the momentum from the previous velocity, the second term captures the attraction toward the particle's own historical best (cognitive component), and the third term reflects the attraction toward the swarm's historical best (social component).

To prevent divergence and maintain numerical stability, each velocity component is bounded by a maximum speed y_{max}:

$$-y_{max} \le y_{p,i,k}^t \le y_{max}. \tag{24}$$

During the search process, it has been observed that using a dynamically varying value for the factor ω yields better search capabilities compared to a fixed value. In the initial search phase, a higher value of ω is desirable to facilitate strong global search capabilities. However, in the later search phase, a smaller value of ω is needed to enhance the local search abilities of particles and promote faster convergence of the solution. To address this, we employ the linear decreasing weight (LDW) strategy to update ω throughout the search process as

$$\omega^t = \frac{(\omega_{ini} - \omega_{end}) \times (G - t)}{G} + \omega_{end}. \tag{25}$$

The symbols ω_{ini} and ω_{end} represent the inertia factors at the initial and end phases, respectively. Apparently, the inertia factor ω^t is a variable that decreases as the number of iterations t increases.

The velocity of a particle is used to update its position and is transformed by the function Sig into a probability that the particle changes its position:

$$Sig\left(y_{p,i,k}^t\right) = \frac{1}{1 + \exp\left(-y_{p,i,k}^t\right)}, \tag{26}$$

where $Sig\left(y_{p,i,k}^t\right) \in (0, 1)$, representing that the probability that the position takes the value 1.

Denote X_p^t to be the position of particle p at the tth iteration, and then $x_{p,i,k}^t \in X_p^t$ is updated by

$$x_{p,i,k}^t = \begin{cases} 1, & \text{if } r < \text{Sig}\left(y_{p,i,k}^t\right); \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

r is randomly generated number from the interval $[0, 1]$.

In the later iterations, as the particle's position approaches a fixed solution, the change in velocity becomes smaller and smaller, approaching zero. At this stage, the primary influences on the velocity are the particle's "population perception" and "self-perception". Therefore, we modify the rule for updating the velocity of particle as

$$y_{p,i,k}^t = \alpha_1 \times \beta_1 \times \left(pbloc_{p,i,k}^{t-1} - x_{p,i,k}^{t-1}\right) + \alpha_2 \times \beta_2 \times \left(gbloc_{i,k}^{t-1} - x_{p,i,k}^{t-1}\right). \tag{28}$$

Likewise, we need to transform the velocity into a probability by the function Sig as

$$\text{Sig}\left(y_{p,i,k}^t\right) = \begin{cases} 1 - \frac{2}{1+\exp\left(-y_{p,i,k}^t\right)}, & \text{if } y_{p,i,k}^t \le 0; \\ \frac{2}{1+\exp\left(-y_{p,i,k}^t\right)} - 1, & \text{otherwise.} \end{cases} \tag{29}$$

After redesigning the particle velocity, the particle position is also needed a little change. When $y_{p,i,k}^t$ is greater than or equal to 0, $x_{p,i,k}^t$ is updated by

$$x_{p,i,k}^t = \begin{cases} 1, & \text{if } r \le \text{Sig}\left(y_{p,i,k}^t\right); \\ x_{p,i,k}^{t-1}, & \text{otherwise .} \end{cases} \tag{30}$$

When $y_{p,i,k}^t$ is less than 0, $x_{p,i,k}^t$ is updated by

$$x_{p,i,k}^t = \begin{cases} 0, & \text{if } r \le \text{Sig}\left(y_{p,i,k}^t\right); \\ x_{p,i,k}^{t-1}, & \text{otherwise .} \end{cases} \tag{31}$$

4.2.3. The service migration strategy

Algorithm 2 shows the service migration strategy based on the BPSO algorithm. The algorithm starts with initializing the parameters of the particle swarm, as well as the position and velocity of each particle (lines 1–2). Then, it computes the fitness function value of all particles using Eq. (21) and sets the global best position to be the particle with the smallest fitness function value (line 3–4). During the iterative process, the algorithm chooses different velocity and position update rules depending on whether the current iteration number is less than or equal to $G \times \alpha$, to achieve a balance between global exploration in the early phases and local search in the later phases (lines 5–13). After each update, the fitness function value of the particle is computed (line 14). If the particle meets the cost and energy constraints and its fitness function value is better than its individual historical optimum, the individual optimal position is updated (lines 15–19). Also, if it is better than the global optimal fitness function value, the global optimal position is updated (lines 20–23). For particles that violate the constraints, the algorithm reproduces new particles that satisfy the constraints as substitutes (lines 24–26). After the iteration terminates, the algorithm returns the global optimal position as the final service migration decision (lines 27–30).

4.3. The ITOSM algorithm

Algorithm 3 outlines the main operations of the ITOSM algorithm. It first generates trajectories for all CAVs using a mobility model, and initializes the offloading decisions for all tasks as well as the migration decisions for service instances of all tasks to zero (lines 1–2). Then, it sequentially calls Algorithm 1 to attain the task offloading decisions for all CAV tasks, and Algorithm 2 to calculate the migration decisions for all service instances (lines 3–4). Finally, by comparing the deadline and the actual completion time of each task, the algorithm computes the number of successfully completed tasks (line 5).

Fig. 2 shows the execution process of the proposed ITOSM algorithm.

4.4. The time complexity analysis of the proposed algorithms

The time complexity of generating the moving trajectories of CAVs is $O(MNK)$. The time complexity of initializing the particle population of size P is $O(PNK)$. The time complexity of Algorithm 2 is $O(PGNK + MNK) = O(PGNK)$ (usually $M < G$), which is mainly decided by the while loop. The time complexity of computing the number of successfully executed is $O(NK)$. The time complexity of Algorithm 1 is $O(HNK)$, where H is the maximum number of candidate servers for a task and it is a very small value. Therefore, the time complexity of ITOSM is $O(MNK) + O(PNK) + O(NK) + O(PGNK) + O(HNK) = O(PGNK)$.

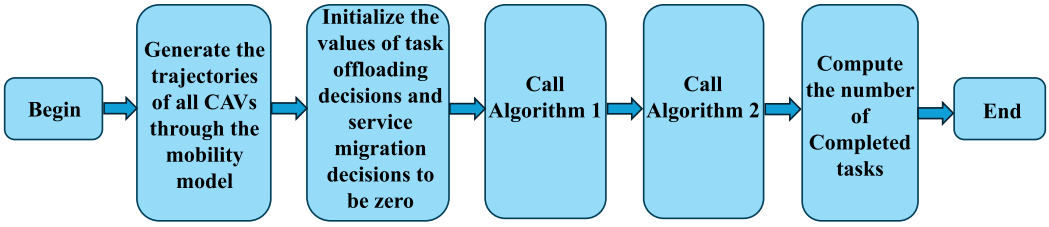


Fig. 2. The flowchart of ITOSM.

Algorithm 2 The service migration strategy

Require: $\lambda_{i,k}, W_{i,k}, D_{i,k}, C_{i,k}, O_{i,k}, W_j, p_i, h_{i,j}, \varphi_{i,j,k}, \sigma, \mu, B, f_i, f_j, E^b, C_i^b$.
Ensure: The globally best previous position $gbloc$.
Initialize the parameters P, G, ω_{ini}, ω_{end}, c_1, c_2, y_{max};
Initialize the velocity and position of all P particles in the particle population;
Calculate the fitness value F of all particles by Eq. (21);
Initialize the globally best previous position to be the position with the smallest fitness value;
while $t \le G$ **do**
  for $p \leftarrow 1$ to P **do**
    if $t \le G \times \alpha$ **then**
      Update velocity Y_p^t via Eq. (23);
      Update position X_p^t via Eqs. (26)(27);
    else
      Update velocity Y_p^t via Eq. (28);
      Update position X_p^t via Eqs. (29)-(31);
    end if
    Calculate the fitness function value F of the particle X_p^t by Eq. (21);
    if particle X_p^t does not violate the energy and cost constraints $E^b, Cost_i^b$ **then**
      if $(X_p^t.F < pbloc_p^t.F)$ **then**
        $pbloc_p^t.F \leftarrow X_p^t.F$; /* $X_p^t.F, pbloc_p^t.F$ are the position fitness function values*/
        $pbloc_p^t \leftarrow X_p^t$;
      end if
      if $(X_p^t.F < gbloc^t.F)$ **then**
        $gbloc^t.F \leftarrow X_p^t.F$;
        $gbloc^t \leftarrow X_p^t$;
      end if
    else
      Generate particles that satisfy the specified constraints;
    end if
  end for
  t++;
end while
$gbloc \leftarrow gbloc^t$;

Algorithm 3 The ITOSM algorithm

Require: $M, N, K, \lambda_{i,k}, W_{i,k}, D_{i,k}, C_{i,k}, O_{i,k}, W_j, p_i, h_{i,j}, \sigma, \mu, B, f_i, f_j, E^b, C_i^b$.
Ensure: The successfully completed task number $\mathcal{N}$.
Generate the trajectories of all CAVs through the mobility model;
Initialize the values of offloading decisions of all tasks and the values of migration decisions of all services to be zero;
Call Algorithm 1 to compute the offloading decisions of all CAV tasks;
Call Algorithm 2 to compute the migration decisions of service instances of all CAV tasks;
Compute the number $\mathcal{N}$ of successfully completed tasks by comparing the deadline and completed time of each CAV task;
return $\mathcal{N}$.

5. Experiments

This section evaluates the effectiveness and efficiency of the proposed ITOSM algorithm. We first describe the experimental setup and baseline methods, followed by a presentation and analysis of the results. Table 3 summarizes the important acronyms that are used in the experiment.

Table 3
Summary of important acronyms.

Acronyms	Full name
CAV	Connected and Automated Vehicle
GPS	Global Positioning System
CPU	Central Processing Unit
QoS	Quality of Service
ITOSM	Improved Task Offloading and Service Migration
TOSM	Task Offloading and Service Migration
BPSO	Binary Particle Swarm Optimization
RA	Random Assignment
AM	Always Migration

5.1. Experimental setting

All experiments are conducted on a simulator running on a computer with the 64-bit Windows 11 operating system, 16 identical processing cores: AMD Ryzen 9 7940HX CPU @2.40 GHz, python 3.12.3 and torch 2.6.0. The simulator is a python program, and we use it to simulate the system model introduced in Section 3.

Edge server deployment: Edge servers are deployed using randomly generated two-dimensional coordinates. They are primarily distributed along the X-axis with small inter-server distances to form an dense edge network ($M = 40$), while slight random perturbations are introduced in the Y-axis to avoid collinearity. CAV trajectories are generated in the same coordinate plane using the improved random walk model.

Communication assumptions: We adopt a distance-dependent wireless transmission model (path loss exponent $\mu = 1.5$) for task offloading and result return, and a fixed wired backhaul ($B = 4$ Mbps) for service migration. All key wireless parameters are listed in Table 4.

Resource heterogeneity: Edge servers have heterogeneous computing capacities from 0.1 GHz to 1.0 GHz mapped to different service prices. CAVs have varying local computing capacities from 0.02 GHz to 0.05 GHz. Tasks differ in input/output size, workload, deadline, and priority (EIT, MIT, or NIT).

Service migration handling: Migration decisions fall into three types: no migration, migration, and local execution. Migration delay and energy consumption are calculated via Eqs. (2) and (9). After each decision round, update the number of tasks on each server.

CAV-edge interaction: We adopt a "move–associate–compute–migrate–return" workflow. CAV positions are generated by the improved random walk model; transmission rates are calculated via Eq. (1); completion time via Eq. (8); and feasibility is checked against cost and energy constraints (Eqs. (10) and (11)).

Geographical settings: The simulation area size is $4R_{es} \times 2R_{es}$ (where $R_{es} = 8$ m is the communication coverage radius of an edge server). Forty edge servers are linearly deployed along the X-axis with a spacing of $2R_{es}$ – random$(1, 3)$ m, and their Y-coordinates are fixed at 0. Around each primary server, 1 to 4 offset servers are

randomly generated, with an X-offset of random$(-2, 2)$ m and a Y-offset of random$(-1, 1)$ m.

Mobility model: Accurately simulating real CAV trajectories is challenging due to their complexity. Existing studies typically adopt one of two approaches: using real GPS trajectory datasets (e.g., Geolife [37]) [3,5], or employing mobility models such as the random walk model [10,38,39]. Since real datasets often lack alignment with our required parameters (e.g., predefined edge server locations) and contain missing key information, we generate CAV movement data using an improved random walk model, which produces discrete position sequences that reflect typical vehicular mobility patterns. The number of trajectory points for a single CAV is $K + 1$ (including the initial position), where $K = 10$ is the number of tasks for a single CAV. The CAV moves with a fixed positive step in the X-direction. The step length ranges from random$(11, 26)$ meters, corresponding to a time slot duration of $\tau = 5$ seconds, resulting in an average speed of approximately 2.2 to 5.2 m/s. In the Y-direction, the CAV moves with random positive or negative steps. The step length ranges from random$(0, 2)$ meters, with a stopping probability of $1/3$. This simulates slight lane changes or obstacle avoidance by the CAV on perpendicular roads. A boundary check is added: the CAV's moving trajectory must not exceed the coverage area of the edge servers, nor coincide with the coordinates of any server.

All simulation parameters are listed in Table 4, and their values are selected based on related literature [3,33]. For each task, the input data size is uniformly sampled from $[100, a_1]$ KB, where a_1 varies from 250 to 700 in steps of 150. The output data size is drawn from $[40, a_2]$ KB, with a_2 ranging from 100 to 280 in steps of 60. The workload is randomly chosen from $[100, a_3]$ cycles/bit, where a_3 increases from 120 to 300 in steps of 60. Edge server CPU frequencies are uniformly selected from $[0.1, 1.0]$ GHz, while CAV onboard CPU frequencies are chosen from $[0.01, 0.05]$ GHz. The time-slot duration is set to $\tau = 5$ seconds. Task deadlines are differentiated according to urgency: Extremely Important Tasks have a deadline of $\tau/3$, Moderately Important Tasks a deadline of $\tau/2$, and Non-critical Tasks a deadline of τ.

5.2. Baseline schemes

Existing service migration studies rarely consider differentiated task deadlines and multi-dimensional resource constraints simultaneously. Furthermore, most algorithms focus on where to migrate services rather than whether migration should occur. Therefore, direct comparisons with prior migration-centric methods are less meaningful for our problem context. Accordingly, we select the following baselines to evaluate the performance of ITOSM:

- BPSO: The standard discrete binary particle swarm optimization algorithm [36], employed for its strong global search capability.
- TOSM: The task-oriented service migration algorithm introduced in [40] (renamed here for convenience).
- Random Assignment (RA): A heuristic in which the migration decision for each task of each CAV is determined randomly. A value is drawn uniformly from $[0, 1]$; if it exceeds 0.5, the service is migrated; otherwise, it remains on the current server.
- Always Migrate (AM): A simple policy that migrates the service instance of every task to the edge server geographically closest to the corresponding CAV.

Our proposed ITOSM enhances TOSM by refining its task offloading strategy, while TOSM itself improves upon BPSO by strengthening local exploitation. Together, these baselines span a spectrum from meta-heuristic optimization to rule-based heuristics, enabling a comprehensive evaluation of ITOSM under the considered multi-constraint scenario.

Table 4
Simulation parameters.

Parameters	[Varied range] (Increment)
Number of particles P	50
Number of the maximum iterations G	100
Initial inertia weight ω_{ini}	1.0
End inertia weight ω_{end}	0.6
Learning factors α_1, α_2	2
Maximum particle velocity y_{max}	3
Number of CAVs N	{20, 30, 40, 50}
Number of edge servers M	40
Number of tasks for each CAV K	10
Size of input data $\lambda_{i,k}$	[250, 700](150)(KB)
Workload $C_{i,k}$	[120, 300](60)(cycles/bit)
Size of output data $O_{i,k}$	[100, 280](60)(KB)
Time slot τ	5s
CPU frequency of edge server f_j	$\{0.1, 0.2, \ldots, 1.0\}$ GHz
CPU frequency of CAV f_i	$\{0.02, 0.03, 0.04, 0.05\}$ GHz
Wireless bandwidth W_j	1 Mbps
Wired bandwidth B	4 Mbps
Transmission power of CAV p_i	1W
Channel gain $h_{i,j}$	10^{-3}
Noise power σ^2	10^{-9}W
Migration energy consumption budget E^b	{80, 120, 160, 200, 240}GJ(10^9J)
Number of candidate servers for each task H	[2, 6]
Offloading cost budget $Cost_i^b$	{10, 20, 30, 40, 50}
the path-loss factor μ	1.5
the penalty factor ρ	10 000
r, β_1 β_2	[0, 1]
γ	0.8
Operating system	64-bit Windows 11
CPU	AMD Ryzen 9 7940HX CPU @2.40GHz
Software	python 3.12.3 and torch 2.6.0

5.3. Case study

We use an example to show how different offloading and migration decisions are made by ITOSM and other baselines, and how these decisions affect performance and QoS.

We consider a mobile edge computing scenario consisting of three edge servers (ES1, ES2, ES3), one CAV, and three tasks with different priority levels (EIT, MIT, and NIT). The time slot duration is set to 5 s. The particle swarm size is 5, and the maximum number of iterations is 20. The wireless bandwidth is 1 Mbps, the wired bandwidth is 4 Mbps, the CAV transmission power is 1 W, the channel gain is 10^{-3}, the noise power is 10^{-9} W, and the path loss exponent is 1.5. The service instance size is 90 KB, and the local computing capacity of the CAV is 5000 cycles/s. The migration energy budget is 5.0, the CAV offloading cost budget is 20.0, and the unit energy consumption coefficient for migration or forwarding is 0.008 /KB.

The CAV sequentially passes through three locations $(1, 0)$, $(4, 0)$, and $(6, 0)$, generating one task at each location. The set of candidate edge servers for each task is $\{\text{ES1}, \text{ES2}, \text{ES3}\}$. The input parameters of the three tasks are as follows:

- **v1 (EIT)**: input 240 KB, output 60 KB, workload 520 cycles/bit, deadline 1.6667 s;
- **v2 (MIT)**: input 190 KB, output 55 KB, workload 360 cycles/bit, deadline 2.5 s;
- **v3 (NIT)**: input 130 KB, output 80 KB, workload 180 cycles/bit, deadline 5.0 s.

The parameters of the three edge servers are as follows:

- **ES1**: located at $(0, 0)$, computing capacity 0.3 GHz, unit price 3.0;
- **ES2**: located at $(4, 0)$, computing capacity 1.2 GHz, unit price 12.0;
- **ES3**: located at $(8, 0)$, computing capacity 0.45 GHz, unit price 4.5.

Table 5
Experimental results of all algorithms, including final offloading server, pre-migration offloading server, migration decision, and task completion status.

Algorithm	Task	Task type	Final offloading server	Pre-Migration Offloading server	Migration decision	Completion time	Completed
ITOS M	v1	EIT	ES2	ES2	No Migration	1.2607	Yes
ITOS M	v2	MIT	ES3	ES2	Migration	1.9997	Yes
ITOS M	v3	NIT	Local	Local	No Migration	4.6800	Yes
TOSM	v1	EIT	ES1	ES1	No Migration	3.7084	No
TOSM	v2	MIT	ES1	ES1	No Migration	2.1849	Yes
TOSM	v3	NIT	Local	Local	No Migration	4.6800	Yes
BPSO	v1	EIT	ES1	ES1	No Migration	3.7084	No
BPSO	v2	MIT	ES1	ES1	No Migration	2.1849	Yes
BPSO	v3	NIT	Local	Local	No Migration	4.6800	Yes
RA	v1	EIT	ES1	ES1	No Migration	3.7084	No
RA	v2	MIT	ES1	ES1	No Migration	2.1849	Yes
RA	v3	NIT	Local	Local	No Migration	4.6800	Yes
AM	v1	EIT	ES1	ES1	No Migration	3.7084	No
AM	v2	MIT	ES2	ES1	Migration	1.2286	Yes
AM	v3	NIT	ES2	ES2	No Migration	0.4982	Yes

Table 6
Performance comparison of all five algorithms across task completion number and QoS.

Algorithm	Completed/All	Completed EIT	Completed MIT	Completed NIT	QoS
ITOSM	3/3	1	1	1	2.5386
TOSM	2/3	0	1	1	0.9503
BPSO	2/3	0	1	1	0.9503
RA	2/3	0	1	1	0.9503
AM	2/3	0	1	1	7.0445

Table 7
The number of tasks completed under the variation of task input data size.

Task input data size interval	ITOSM	TOSM	BPSO	RA	AM	ITOSM VS. TOSM	ITOSM VS. BPSO	ITOSM VS. RA	ITOSM VS. AM
[100,250]	191	181	170	110	83	5.52%	12.35%	73.64%	130.12%
[100,400]	161	150	137	85	69	7.33%	17.52%	89.41%	133.33%
[100,550]	145	137	125	68	65	5.84%	16.00%	113.24%	123.08%
[100,700]	127	118	104	61	61	7.63%	22.12%	108.20%	108.20%

Tables 5 and 6 present the computational results of five algorithms ITOS M, TOSM, BPSO, RA and AM on three tasks with different priority levels. As can be seen from the tables, only ITOSM completes all three tasks, including the EIT task with the most stringent deadline, whereas TOSM, BPSO, RA, and AM each complete only two tasks, and none successfully handles the EIT task. Specifically, ITOSM offloads the EIT task v1 to the more powerful ES2 without migration with completion time of 1.2607 s, reasonably migrates the MIT task v2 from ES2 to ES3 with completion time of 1.9997 s, and executes the NIT task v3 locally with completion time of 4.6800 s, thereby ensuring that all tasks meet their deadlines. In contrast, the other algorithms generally offload v1 to the weaker ES1 without migration, resulting in a significantly longer completion time of 3.7084 s and consequently a missed deadline. In terms of QoS, AM achieves the highest value of 7.0445, ITOSM ranks second with 2.5386, while TOSM and RA score only 0.9503. Although AM obtains a higher QoS, this comes at the cost of failing to complete the EIT task (v1 misses its deadline). In summary, ITOSM ensures that all tasks—especially the urgent ones—are successfully completed while still achieving a competitive QoS, demonstrating its superior ability to balance task completion and quality of service.

5.4. Experimental results and analysis

In our experiments, we use the number of completed tasks and QoS as the performance evaluation metrics of algorithms.

5.4.1. Impact of the input data size, the output data size, and the workload of tasks on task completion

Table 7 presents the variation in the number of completed tasks as the input data size increases from $[100, 250]$ to $[100, 700]$ (in KB) in increment of 150 KB, with the number of CAVs fixed at 20. Each reported value is the average over multiple independent runs. The results indicate that all algorithms—ITOSM, TOSM, BPSO, RA, and AM—exhibit a decline in task completion as the input data size grows. This trend is expected because larger input data prolong the task execution time, while task deadlines remain constant. Notably, ITOSM consistently achieves a higher number of completed tasks compared to TOSM, BPSO, RA, and AM across all input-size ranges. In particular, ITOSM surpasses TOSM, the second-best performer, demonstrating its enhanced capability to handle larger data volumes under stringent timing constraints. This improvement is primarily attributed to two factors. First, ITOSM's entropy-weighted offloading strategy enables it to select more suitable edge servers for tasks of different urgency levels, outperforming TOSM's rule-based approach. Second, the enhanced BPSO search mechanism with adaptive local exploitation allows ITOSM to converge to higher-quality migration decisions than standard BPSO, RA, and AM.

Table 8 illustrates the impact of output data size on task completion, where the size range increases from $[40, 100]$ to $[40, 280]$ (in KB) in step of 60 KB. The results reveal a consistent decline in the number of completed tasks for all algorithms—ITOSM, TOSM, BPSO, RA, and AM—as the output data size grows. This trend is attributable to the longer data-return time required for larger output volumes, while task deadlines remain fixed. Across all tested ranges, ITOSM maintains a higher task completion count than TOSM, BPSO, RA, and AM, demonstrating its robustness under increasing output-data overhead. The same advantages hold here, confirming the robustness of ITOSM's offloading and migration mechanisms.

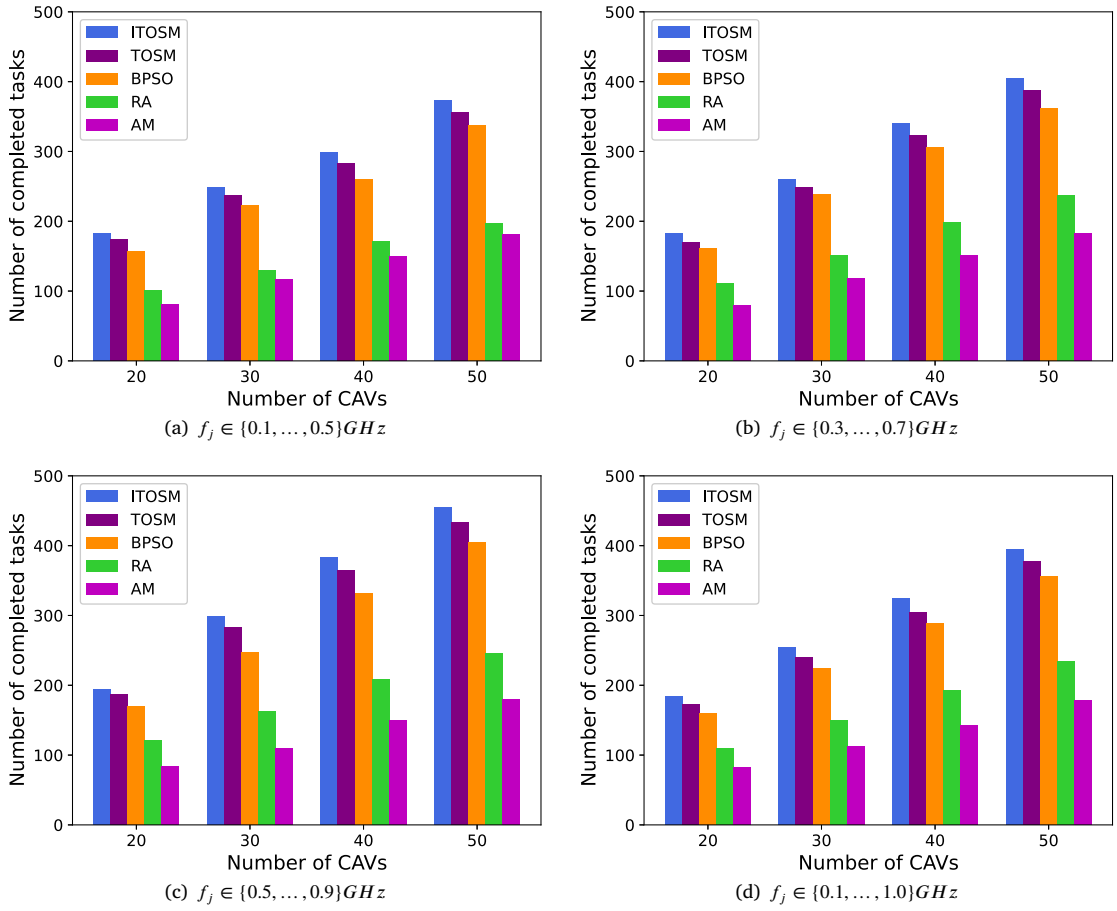
Table 9 shows how the number of completed tasks varies as the task workload increases from $[40, 120]$ to $[40, 300]$ (in CPU cycles/bit),

Table 8
The number of tasks completed under the variation of task output data size.

Task output data size interval	ITOSM	TOSM	BPSO	RA	AM	ITOSM VS. TOSM	ITOSM VS. BPSO	ITOSM VS. RA	ITOSM VS. AM
[40,100]	193	183	168	110	82	5.46%	14.88%	75.45%	135.37%
[40,160]	172	163	152	108	81	5.52%	13.16%	59.26%	112.35%
[40,220]	160	151	141	100	81	5.96%	13.48%	60.00%	97.53%
[40,280]	156	147	136	94	80	6.12%	14.71%	65.96%	95.00%

Table 9
The number of tasks completed under changing task workload demands.

Task workload interval	ITOSM	TOSM	BPSO	RA	AM	ITOSM VS. TOSM	ITOSM VS. BPSO	ITOSM VS. RA	ITOSM VS. AM
[40, 120]	195	185	170	105	83	5.41%	14.71%	85.71%	134.94%
[40, 180]	175	164	150	104	80	6.71%	16.67%	68.27%	118.75%
[40, 240]	162	151	139	83	81	7.28%	16.55%	95.18%	100.00%
[40, 300]	149	138	128	82	79	7.97%	16.41%	81.71%	88.61%



(a) $f_j \in \{0.1, \ldots, 0.5\} GHz$ (b) $f_j \in \{0.3, \ldots, 0.7\} GHz$

(c) $f_j \in \{0.5, \ldots, 0.9\} GHz$ (d) $f_j \in \{0.1, \ldots, 1.0\} GHz$

Fig. 3. Number of completed tasks with varying computing capacities of edge servers and different numbers of CAVs.

in increment of 60. The results indicate a clear downward trend in task completion for all methods—ITOSM, TOSM, BPSO, RA, and AM—as workload rises. This decline occurs because higher computational demand extends task execution time, whereas task deadlines remain fixed. Throughout the workload range, ITOSM consistently completes more tasks than TOSM, BPSO, RA, and AM, confirming its effectiveness under increasingly demanding computational conditions. ITOSM's advantage is especially pronounced under increasing computational demand, where its adaptive strategy matters most.

5.4.2. Impact of the computing capacity of edge servers and the number of CAV tasks on task completion

In a heterogeneous multi-user edge computing system, the computing capacity allocated to each task on an edge server depends on the number of concurrent tasks executing on that server. To evaluate system scalability, we examine task completion under varying numbers of CAVs and different edge-server computing capacities. The number of CAVs is increased from 20 to 50 in step of 10, while the task parameters are fixed as follows: input data size in $[100, 250]$ KB, output data size in $[40, 100]$ KB, and workload in $[40, 120]$ cycles/bit.

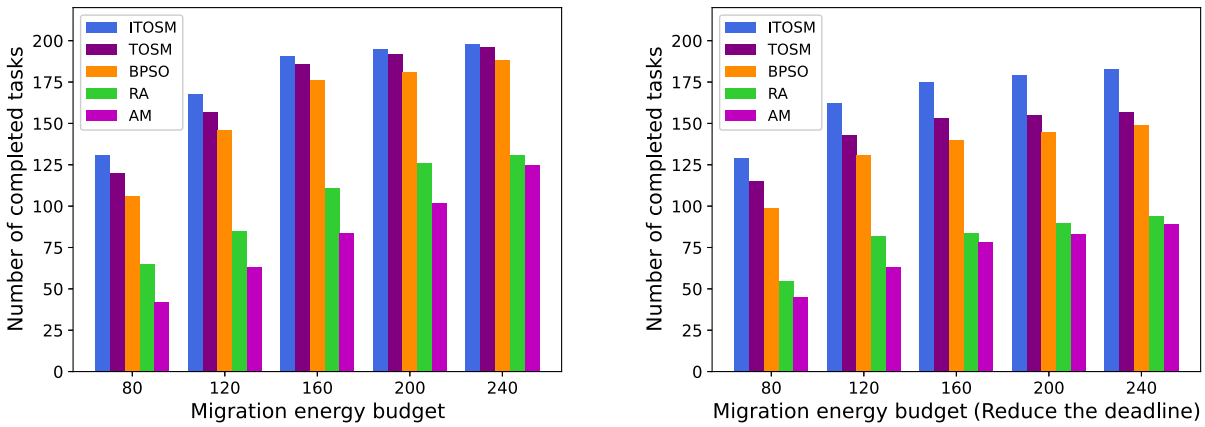


Fig. 4. The number of completed tasks with the variation of migration energy budget: (a) original deadline, (b) deadline reduced.

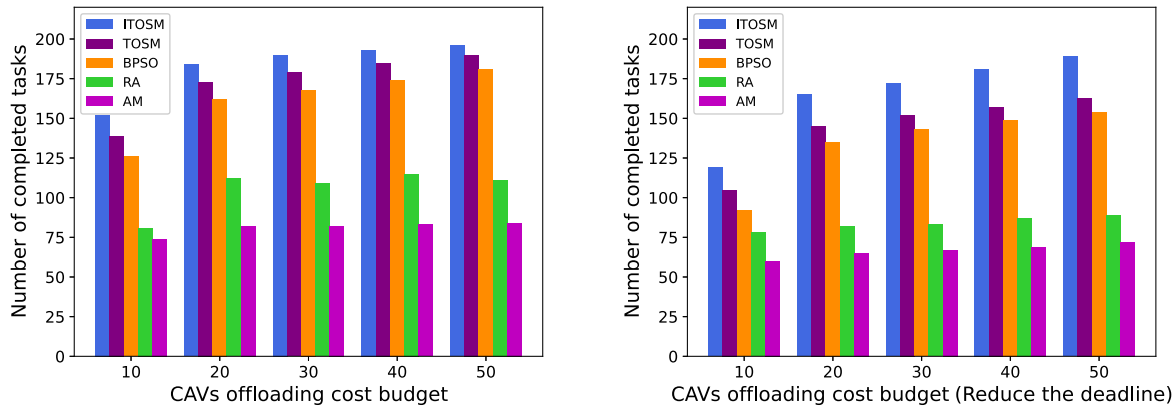


Fig. 5. The number of completed tasks with the variation of CAV offloading cost budget: (a) original deadline, (b) deadline reduced.

Figs. 3(a)–(d) show the total number of completed tasks as the number of CAVs increases for different computing capacities of edge servers. Two clear trends emerge across all subfigures: (1) Effect of CAV count—The number of completed tasks generally rises with increasing CAV numbers for all algorithms (ITOSM, TOSM, BPSO, RA, AM), which is expected because more CAVs introduce more tasks into the system. (2) Effect of server capacity: A higher computing capacity (i.e., higher CPU frequency) of edge servers leads to more tasks being completed by every algorithm, as greater capacity reduces per-task execution time. Throughout these experiments, ITOSM consistently outperforms the four baseline algorithms, demonstrating its superior ability to leverage available resources and schedule tasks effectively under varying system scales and server capabilities. These results consistently demonstrate the effectiveness of ITOSM's two-stage decision framework.

5.4.3. Impact of the CAV offloading cost and migration energy on task completion

We also study the effect of the CAV offloading cost and migration energy on task completion.

Fig. 4 illustrates how the number of completed tasks varies as the migration energy budget increases from 80 to 240 units under two cases: original deadline and reduced deadline, with other parameters held constant. Fig. 4(a) shows that all algorithms (ITOSM, TOSM, BPSO, RA, and AM) exhibit improved task completion as the energy budget grows. This is because a larger energy budget enables more service migrations, allowing a greater number of tasks to meet their deadlines. Fig. 4(b) illustrates the same relationship when task deadlines are reduced. Again, as the energy budget increases, all algorithms exhibit a general upward trend in terms of task completion. Nevertheless, the overall performance of every algorithm drops noticeably compared to the original case in Fig. 4(a), and the maximum number of tasks completed by ITOSM decreases from 198 to 183. On the whole, ITOSM consistently achieves significantly higher task completion than the four baseline methods, demonstrating its effectiveness in utilizing migration energy to maximize task throughput under resource-constrained conditions.

Fig. 5 shows the number of completed tasks as the CAV offloading cost budget increases from 10 to 50 in step of 10 under two cases: original deadline and reduced deadline, with all other parameters held constant. Fig. 5(a) shows that all algorithms—ITOSM, TOSM, BPSO, RA, and AM—achieve higher task completion under larger cost budgets. This trend is expected, as a higher budget allows more tasks to be offloaded to edge servers, thereby increasing the number of tasks that can be processed within their deadlines. Across all budget levels, ITOSM consistently completes more tasks than the baseline algorithms, highlighting its cost-effective offloading and migrating strategies. Fig. 5(b) shows the same relationship under reduced deadlines. All algorithms still improve the number of completed tasks with larger budgets, but each completes fewer tasks than in Fig. 5(a) with original deadlines. For example, when the budget is 50, ITOSM drops the number of completed tasks from 196 to 189, TOSM from 190 to 163, BPSO from 181 to 154, RA from 111 to 89, and AM from 84 to 72. Therefore, tighter deadlines degrade overall performance, yet ITOSM remains superior, showing robustness under timing constraints.

Together, Figs. 4 and 5 demonstrate that ITOSM makes more efficient use of both energy and monetary resources compared to existing approaches, leading to consistently better service delivery under constrained edge-computing environments. This improvement stems from ITOSM's superior task offloading strategy over TOSM, and its superior search strategy over BPSO, RA, and AM.

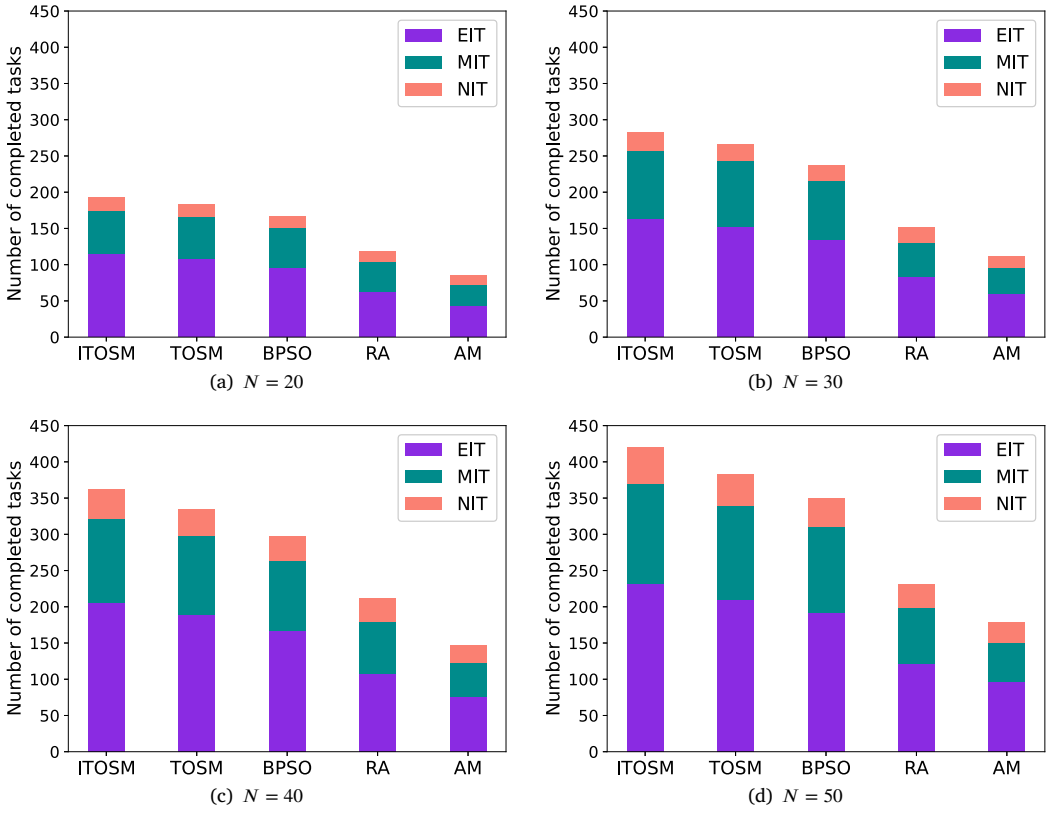


Fig. 6. The number of different urgent tasks to be completed under changing the number of CAVs from 20 to 50.

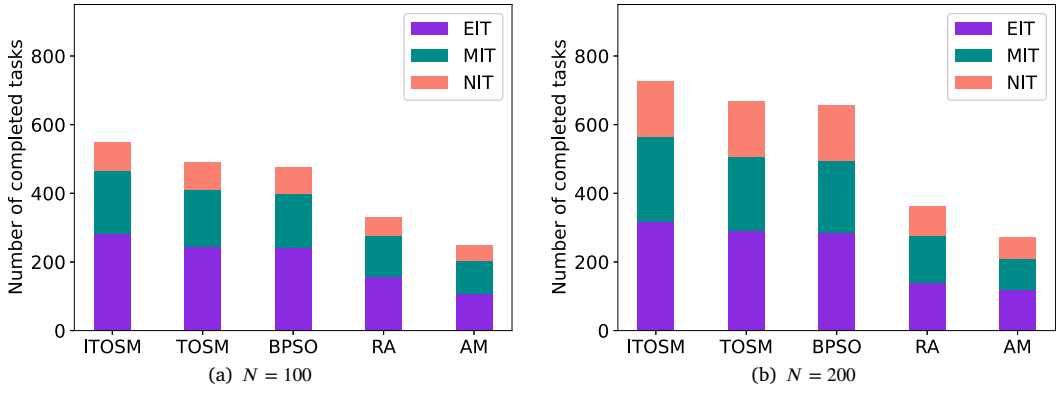


Fig. 7. The number of different urgent tasks to be completed under changing the number of CAVs from 100 to 200.

5.4.4. Impact of the edge server number on task completion

We study the effect of the number of edge servers on task completion.

Table 10 reports the number of completed tasks as the number of edge servers increases from 40 to 100 in increment of 20, with all other parameters held constant. The results indicate that all compared algorithms—ITOSM, TOSM, BPSO, RA, and AM—achieve higher task completion when more edge servers are available. This improvement stems from the greater aggregate computing capacity provided by additional servers, which enables more concurrent task executions. Across all server-count settings, ITOSM maintains a substantially higher task completion rate than the baseline algorithms, demonstrating its effectiveness in leveraging distributed edge resources to maximize system throughput. Once again, ITOSM's adaptive offloading and BPSO-based migration prove superior in leveraging distributed edge resources.

5.4.5. The factors having impact on the QoS

We will further investigate factors that affect QoS. To address that point, we give a new calculation method of QoS as

$$QoS = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(D_{i,k} - T_{i,k} \right) \times W_{i,k}. \tag{32}$$

We only calculate QoS for completed tasks.

Table 10
Impact of the number of edge servers on the number of completed tasks.

Algorithms	$M = 40$	$M = 60$	$M = 80$	$M = 100$
ITOSM	185	188	192	195
TOSM	175	177	179	181
BPSO	160	163	167	170
RA	110	116	115	114
AM	83	85	86	88

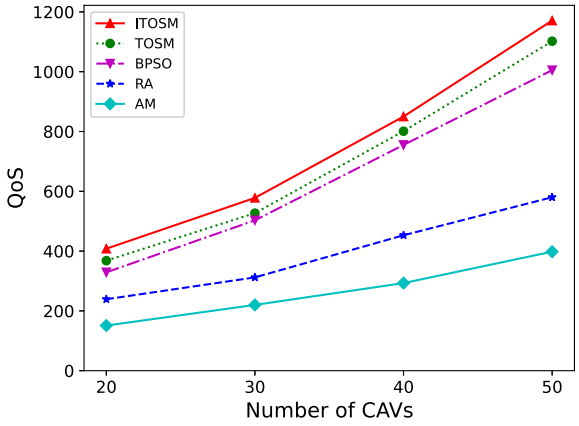


Fig. 8. QoS of the system under changing the number of CAV users.

Figs. 6(a)–(d) show that the number of completed tasks by five algorithms (ITOSM, TOSM, BPSO, RA, AM) for EIT, MIT, and NIT under 20, 30, 40, and 50 CAVs, respectively. Overall, as the number of CAVs increases, the number of completed tasks for all algorithms and task types gradually rises, reflecting the increased system load caused by more CAVs. Across all CAV scales, the proposed ITOSM algorithm consistently completes significantly more tasks than the other four baseline algorithms. When there are 50 CAVs, for EIT, ITOSM completes 231 tasks, compared to 210, 191, 121, and 96 tasks completed by TOSM, BPSO, RA, and AM, respectively. For MIT, ITOSM completes 139 tasks, substantially higher than the 130 tasks of TOSM and 120 tasks of BPSO. For NIT, ITOSM achieves 50 tasks, again outperforming the other algorithms. It is worth noting that the advantage of ITOSM over the other algorithms is most pronounced for emergency tasks, while the improvement is relatively smaller for non-critical tasks. Furthermore, as the number of CAVs increases, the absolute performance gap between ITOSM and the other algorithms widens, indicating that ITOSM maintains good scalability even under higher system loads. This performance gap is directly attributable to ITOSM's urgency-aware offloading and its enhanced search mechanism, which jointly prioritize EITs.

Figs. 7(a) and (b) show task completion under 100 and 200 CAVs. As CAV number increases, all algorithms complete more tasks. ITOSM consistently outperforms the baselines across all task types. Similar to Fig. 6, ITOSM's advantage is greatest for emergency tasks and widens with larger CAV scales, demonstrating robust scalability under high system loads.

Fig. 8 illustrates the QoS corresponding to the task-completion results presented in Fig. 6. The results show that ITOSM delivers significantly higher QoS than the baseline algorithms (TOSM, BPSO, RA, and AM). This improvement is directly attributed to ITOSM's superior task-completion performance, particularly for time-critical tasks with stringent deadlines. Higher QoS translates into enhanced safety and better user experience for CAV operations, underscoring the practical value of the proposed algorithm in real-world intelligent transportation systems.

5.4.6. *The stability of our proposed algorithm*

Fig. 9 shows that the ITOSM algorithm can calculate a stable solution, which indicates that our algorithm has a high confidence.

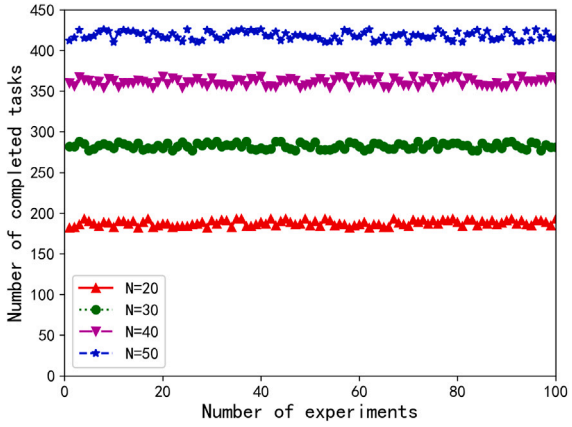


Fig. 9. The number of finished tasks by ITOSM when the number of CAVs changes.

In summary, all experimental results demonstrate the superior performance of our proposed ITOSM.

6. Conclusion

This paper addresses the problem of maximizing the number of completed tasks in mobile edge computing through service migration, under constraints of task urgency, migration energy consumption, and CAV offloading costs. To solve this problem, we propose the ITOSM algorithm, built upon an enhanced discrete BPSO framework. Experimental results show that ITOSM outperforms existing strategies in both task completion rate and QoS. For instance, when evaluating the impact of task input data size on the task completion rate, ITOSM achieves a minimum improvement of 5.52% and a maximum improvement of 133.33% over the baselines. Under our experimental settings (synthetic mobility model, OFDMA-based communication, and continuous service availability), ITOSM demonstrates superior performance in task completion compared to the baseline algorithms. However, these conclusions are drawn from simulation results; real-world deployment may introduce additional challenges such as heterogeneous server coverage and network congestion. For future work, we plan to validate the proposed algorithm using publicly available real-world GPS trajectory datasets to further assess its generalizability. Additionally, we will explore trace-driven simulations based on actual CAV mobility data.

CRediT authorship contribution statement

Jing Liu: Writing – review & editing, Validation, Supervision, Resources, Methodology, Conceptualization. **Jieyi Deng:** Writing – original draft, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Longxin Zhang:** Validation, Supervision, Resources, Methodology, Funding acquisition, Conceptualization. **Qiushi Cao:** Writing – original draft, Visualization, Validation, Methodology, Investigation, Conceptualization. **Wei Hu:** Validation, Supervision, Software, Methodology. **Cen Chen:** Validation, Supervision, Resources, Methodology. **Keqin Li:** Writing – review & editing, Supervision, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to express their sincere gratitude to the editors and the referees for their valuable time and constructive comments which help improve the quality of the manuscript greatly. This work was partially funded by the Natural Science Foundation of Hunan Province, China (Grant No. 2023JJ50204), the Scientific Research Foundation of Hunan Provincial Education Department, China (Grant No. 25A0422). the National Natural Science Foundation of China (Grant No. 62472181), and the Guangdong Basic and Applied Basic Research Foundation (Grant No. 2024A1515010220).

Data availability

Data will be made available on request.

References

[1] R. Coutinho, A. Boukerche, Guidelines for the design of vehicular cloud infrastructures for connected autonomous vehicles, IEEE Wirel. Commun. 26 (4) (2019) 6–11.
[2] L. Pacheco, H. Oliveira, D. Rosário, E. Cerqueira, L. Villas, T. Braun, Service migration for connected autonomous vehicles, in: 2020 IEEE Symposium on Computers and Communications, ISCC, 2020, pp. 1–6.
[3] Y. Ding, C. Liu, K. Li, Z. Tang, K. Li, Task offloading and service migration strategies for user equipments with mobility consideration in mobile edge computing, in: 17th IEEE Intl Conf on Parallel and Distributed Processing with Applications, 2020.
[4] I. Labriji, E.C. Strinati, E. Perraud, F. Joly, Dynamic migration strategy for mobile multi-access edge computing services, in: IEEE Wireless Communications and Networking Conference, WCNC 2022, Austin, TX, USA, April 10-13, 2022, IEEE, 2022, pp. 710–715.
[5] S. Wang, R. Urgaonkar, M. Zafer, T. He, K.K. Leung, Dynamic service migration in mobile edge computing based on Markov decision process, IEEE/ACM Trans. Netw. PP (99) (2019).
[6] S. Shahryari, F. Tashtarian, S. Hosseini-Seno, CoPaM: Cost-aware VM placement and migration for mobile services in multi-cloudlet environment: An SDN-based approach, Comput. Commun. 191 (2022) 257–273.
[7] Y. Peng, X. Tang, Y. Zhou, J. Li, Y. Qi, L. Liu, H. Lin, Computing and communication cost-aware service migration enabled by transfer reinforcement learning for dynamic vehicular edge computing networks, IEEE Trans. Mob. Comput. 23 (1) (2024) 257–269.
[8] H. Li, Y. Wang, M. Pan, S. Li, W. Guan, Deep reinforcement learning-based joint resource allocation and service migration for smart-buoy-enabled maritime multiaccess edge computing networks, IEEE Internet Things J. 12 (24) (2025) 52687–52705.
[9] E. Bozkaya, Digital twin-assisted and mobility-aware service migration in mobile edge computing, Comput. Netw. 231 (2023) 109798.
[10] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, K. Li, Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2021) 1603–1614.
[11] J. Li, D. Zhao, Z. Shi, L. Meng, W. Gaaloul, Z. Zhou, Energy-efficient online service migration in edge networks, IEEE Internet Things J. 11 (18) (2024) 29689–29708.
[12] X. Zhou, S. Ge, T. Qiu, K. Li, M. Atiquzzaman, Energy-efficient service migration for multi-user heterogeneous dense cellular networks, IEEE Trans. Mob. Comput. 22 (2) (2023) 890–905.
[13] Y. Li, L. Li, P. Fan, Mobility-aware computation offloading and resource allocation for NOMA MEC in vehicular networks, IEEE Trans. Veh. Technol. 73 (8) (2024) 11934–11948.
[14] X. Xu, L. Yao, M. Bilal, S. Wan, F. Dai, K.R. Choo, Service migration across edge devices in 6G-enabled internet of vehicles networks, IEEE Internet Things J. 9 (3) (2022) 1930–1937.
[15] H. Yu, Q. Zhang, Hybrid learning based service migration for cost minimization with deadlines in multi-user mobile edge computing systems, Comput. Netw. 242 (2024) 110249.
[16] Z. Chen, S. Huang, G. Min, Z. Ning, J. Li, Y. Zhang, Mobility-aware seamless service migration and resource allocation in multi-edge IoV systems, IEEE Trans. Mob. Comput. 24 (7) (2025) 6315–6332.
[17] P. Qin, M. Li, K. Wu, Y. Fu, Cost minimization resource allocation with service instance caching and task migration for UAV mobile edge computing, IEEE Trans. Netw. Sci. Eng. 13 (2026) 1738–1753.
[18] H. Wang, J. Du, C. Jiang, J. Wang, M. Debbah, Z. Han, Graph-aware temporal encoder-based service migration and resource allocation in satellite networks, IEEE Trans. Wirel. Commun. 25 (2026) 8260–8276.
[19] C. Li, Z. Zhang, B. Wang, M. Lei, S. Liu, A. Li, S. Wan, Joint service migration and resource allocation for DNN tasks using SA-DDQN-DDPG in vehicular edge computing, ACM Trans. Intell. Syst. Technol. 17 (1) (2026).
[20] S. Xing, X. Feng, L. Yue, J. Zhang, M. Li, T. Zheng, A joint optimization strategy of computing offloading and service migration based on multi-user with mobility consideration, IEEE Open J. Commun. Soc. 7 (2026) 1137–1152.
[21] C. Zhang, Z. Zheng, Task migration for mobile edge computing using deep reinforcement learning, Future Gener. Comput. Syst. 96 (JUL.) (2019) 111–118.
[22] B. Qiao, C. Liu, J. Liu, Y. Hu, K. Li, K. Li, Task migration computation offloading with low delay for mobile edge computing in vehicular networks, Concurr. Comput. Pr. Exp. 34 (1) (2022).
[23] B. Kumar, A. Bhardwaj, D. Prasad Sahu, Task offloading for CAVs edge computing environment: Taxonomy, critical review, and future road map, ACM Comput. Surv. 58 (8) (2026).
[24] B. Kumar, A. Bhardwaj, D. Sahu, Machine learning techniques for task offloading in connected autonomous vehicles, in: 2024 IEEE Third International Conference on Power Electronics, Intelligent Control and Energy Systems, ICPEICES, 2024, pp. 400–405.
[25] S. Liu, H. Sun, Dynamic service migration and resource allocation for UAV-assisted vehicular edge computing, in: 2025 in International Conference on Intelligent Computing, Springer Nature Singapore, 2025, pp. 173–184.
[26] T. Ouyang, R. Li, X. Chen, Z. Zhou, X. Tang, Adaptive user-managed service placement for mobile edge computing: An online learning approach, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 1468–1476.
[27] P. Wang, T. Ouyang, G. Liao, J. Gong, S. Yu, X. Chen, Edge intelligence in motion: Mobility-aware dynamic DNN inference service migration with downtime in mobile edge computing, J. Syst. Archit. 130 (2022) 102664.
[28] Z. Zhou, S. Yu, W. Chen, X. Chen, CE-IoT: Cost-effective cloud-edge resource provisioning for heterogeneous IoT applications, IEEE Internet Things J. 7 (9) (2020) 8600–8614.
[29] C. Yi, J. Cai, Z. Su, A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications, IEEE Trans. Mob. Comput. 19 (1) (2019) 29–43.
[30] L. Tan, Z. Kuang, L. Zhao, A. Liu, Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing, IEEE Trans. Wirel. Commun. 21 (3) (2021) 1960–1972.
[31] L. Wang, A. Liu, N.N. Xiong, S. Zhang, T. Wang, M. Dong, SD-SRF: An intelligent service deployment scheme for serverless-operated cloud-edge computing in 6G networks, Future Gener. Comput. Syst. 151 (2024) 242–259.
[32] W. Ma, L. Mashayekhy, Privacy-by-Design Distributed Offloading for Vehicular Edge Computing, Association for Computing Machinery, 2019, 101C110.
[33] K. Li, A game theoretic approach to computation offloading strategy optimization for non-cooperative users in mobile edge computing, IEEE Trans. Sustain. Comput. PP (2018) 1–1.
[34] X. Li, A computing offloading resource allocation scheme using deep reinforcement learning in mobile edge computing systems, J. Grid Comput. 19 (3) (2021) 35.
[35] P.A. Apostolopoulos, G. Fragkos, E.E. Tsiropoulou, S. Papavassiliou, Data offloading in UAV-assisted multi-access edge computing systems under resource uncertainty, IEEE Trans. Mob. Comput. 22 (1) (2021) 175–190.
[36] J. Kennedy, R. Eberhart, A discrete binary version of the particle swarm algorithm, in: 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, vol. 5, 1997, pp. 4104–4108.
[37] Y. Zheng, X. Xie, W. Ma, GeoLife: A collaborative social networking service among user, location and trajectory, IEEE Data Eng. Bull. 33 (2) (2010) 32–39.
[38] J. Plachy, Z. Becvar, E.C. Strinati, Dynamic resource allocation exploiting mobility prediction in mobile edge computing, in: 2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications, PIMRC, 2016.
[39] F. Tang, C. Liu, K. Li, Z. Tang, K. Li, Task migration optimization for guaranteeing delay deadline with mobility consideration in mobile edge computing, J. Syst. Archit. 112 (8) (2020) 101849.
[40] Q. Cao, Service migration strategy considering task deadline in edge computing, in: 2023 Asia-Pacific Conference on Image Processing, Electronics and Computers, IPEC, IEEE, 2023, pp. 453–461.