Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

# A novel vehicular task deployment method in hybrid MEC

Xifeng Xu[1], Yunni Xia[1*], Zeng Feng[2], Fan Li[3], Hong Xie[1], Xiaodong Fu[4] and Mengdi Wang[5]

**Abstract**
With the skyrocketing need for low-latency services on the Internet of Vehicles (IoV) and elastic cross-layer resource provisioning, multi-access edge computing (MEC) is considered a high-potent solution, which evolves from cloud and grid computing to meet the above needs in IoV scenarios. Instead of considering single-point and monolithic IoV tasks, in this paper, we consider the IoV applications to be with structural properties and the supporting environment to be with a hybrid cloud-edge architecture. We develop a scheduling method that offloads tasks to the eNode or cloud according to their estimations of latest starting time. Simulative results clearly demonstrate that our method beat existing solutions in terms of average completion time, average waiting time, and in-time completion rate.

**Keywords:** Internet of Vehicles, Multi-access edge computing, Task scheduling, Quality of service

## Introduction
With the rapid development of Internet-of-Vehicles (IoV) technology [1] and the increasing popularity of intelligent vehicles [2], more and more computational-intensive IoV applications are becoming available. Intelligent vehicles can sense the vehicle's line-of-sight by deploying sensors, including onboard cameras, radar, etc [3]. With this information, applications including vehicle-road/vehicle-person interaction, road condition awareness and collaborative dispatching, video or high-precision map distribution, etc. can be provided [4].

However, due to the limited resources of intelligent vehicles, it is difficult for them to handle computational-intensive applications completely locally on their own [5]. Therefore, the intelligent vehicles transmit this sensor information over the network to more powerful servers for artificial intelligence analysis processing of applications such as video analysis [6] and high-definition maps [7]. After the computation is completed, the server returns the results to the vehicle control system to provide services for these applications.

With the dramatic shift towards applications based on artificial intelligence, machine learning and deep learning, real-life intelligent vehicle applications are divided into interdependent subtasks, and the processing complexity between tasks is achieved by introducing a topology between them [8]. Take the example of a road condition analysis application, which can be requested by an intelligent vehicle. Each task represents a part of the road condition analysis process and there are certain dependencies between the tasks, such as road profile information and other vehicle information. The execution of tasks in the application should be ordered, as processing tasks may require output data from other tasks.

Cloud computing is a viable solution in the IoT that can provide resources in a cost-effectively and elastically [9]. But as the number of intelligent vehicles increases dramatically, a large amount of data is generated that needs to be uploaded and processed by the cloud. For the cloud, network bandwidth is often limited, and scheduling too many service requests can lead to network congestion or even system crashes. For intelligent cars, most tasks are latency-sensitive, and the cloud is so far away that it takes a long time to complete [10], reducing the quality of service, and even for applications like smart obstacle avoidance will lead to safety accidents [11]. So cloud

*Correspondence: xiayunni@hotmail.com
[1] College of Computer Science, Chongqing University, Chongqing, China
Full list of author information is available at the end of the article

computing alone is not enough to solve this problem and requires the use of a more rapid communication solution.

For this problem, Multi-access Edge Computing (MEC), which brings compute and storage resources to the edge of the mobile network, giving mobile devices the ability to run computational-intensive and latency-sensitive applications [12], may be a good choice [13, 14]. It is a distributed architecture that greatly reduces the data transmission time by communicating with the vehicle closer, and it also provides computing resources to meet the needs of the application [15], which can meet the computation-intensive and latency-sensitive applications of intelligent vehicles. Although MEC architectures are much more responsive than cloud computing, meeting the quality of service (QoS) [16] for intelligent vehicles is still a big challenge. Because intelligent vehicles maintain high mobility and a large number of intelligent applications.

In this paper, we aim at providing an offloading strategy for optimizing execution efficiency and service quality of intelligent vehicle applications. The main contributions of this paper are: 1) A cloud-edge hybrid architecture is designed to allow more flexible scheduling of tasks, which is especially adaptable to high load situations. 2) Load balancing between different eNodes is considered so that tasks are not all gathered in the eNode with the best performance for execution. At the same time, the priority order for task scheduling is ordered according to the acceptable flexible latest beginning time of tasks, which improves the overall in-time completion rate. 3) We consider that the application is a combination of multiple tasks and there are certain dependencies between tasks that need to be executed in a certain order, which is closer to reality, but increases the complexity of the problem.

We conducted a series of experiments based on a well-known dataset of real-world edge environments and showed that our proposed algorithm MTS-MEoC outperforms the benchmark algorithms in the metrics of average completion time, in-time completion rate, and average waiting time, indicating that it can provide better QoS.

## Related Work

MEC in IoV to aims to leverage nearby external resources to execute computational-intensive and latency-sensitive intelligent vehicle applications. In recent years, the problem of MEC-oriented task offloading has attracted extensive attention and research interest.

Zhang et al. [17] studied the problem of task offloading and resource allocation in vehicular heterogeneous networks and clustered the QoS of vehicles by improved K-means algorithm before having Q-Learning algorithm for allocation to meet the joint demand of capacity and delay. Liu et al. [18] studied the computational offloading and resource allocation problem of vehicular edge computing and formulated it as a semi-Markov process. They solved this problem through Q-learning and deep reinforcement learning methods. Ye et al [19] studied the problem of uneven data processing demand due to uneven distribution of vehicles in time and space, designed a hybrid fog architecture consisting of a fog computing radio access network and vehicle fog computing, and proposed a heuristic algorithm enhanced by deep learning to optimize computational offloading. Huang et al. [20] proposed a computational offloading algorithm based on meta-learning, by learning historical MEC task offloading data and then adapting a small number of training samples to the current scenario, the resulting algorithm can generate offloading decisions more efficiently. Xu et al. [21] proposed a drone-assisted task offloading approach using deep reinforcement learning techniques to assist smart buildings and devices with communication problems in emergency situations. Peng et al. [22] formulate the task offloading algorithm in edge computing as a mostly integer linear programming problem and propose a decentralized reactive approach that learns dynamically when requests arrive. Huang et al. [23] used deep reinforcement learning techniques to solve dynamic optimization problems for perceptual energy control and computational offloading in the information age. Wu et al. [24] proposed a heterogeneous Markov decision process to model the inter-slice resource allocation process and the intra-slice task scheduling process hierarchically in IoV with network slicing capabilities. They designed the corresponding hierarchical deep reinforcement learning architecture to jointly optimize the inter-slice resource allocation and intra-slice task scheduling problems. These methods of learning can achieve good scheduling strategies to improve the quality of service. However, vehicles have high requirements for latency, and conventional training methods can lead to high latency.

Wei et al. [25] defined the task offloading problem with received energy and completion delay constraints in IoV as a mixed integer nonlinear programming problem. They designed an algorithm for joint optimization of energy consumption and task delay is designed to optimize the selection decision, resource allocation, unloading ratio and transmission power. You et al. [26] used computational resources and experimental demands as constraints on task scheduling by pricing network latency sensitivity ranking based on the Vickrey-Clark-Groves auction algorithm for solving the problem, which effectively reduces the network latency. Deng et al. [27] present an optimization problem for minimizing the completion time at a specified cost in IoV. Several algorithms were improved based on the alternating direction method of multipliers (ADMM) algorithm while introducing an augmented Lagrangian function to iteratively improve the minimized task completion time. Lakhan et al. [28] designed a novel collaborative vehicle fog cloud network based on container microservices, proposing a mobile-aware task offloading method to determine the optimal offloading time and a collaborative task scheduling method responsible for task sequencing and scheduling to reduce communication and computational costs under a given completion time constraint. Chen et al. [29]propose a dynamic task offloading algorithm based on deep reinforcement learning taking into account both MEC and cloud.

Ying et al. [30] considered task scheduling for MEC applications based on directed acyclic graphs (DAG) and proposed a maximum reliability offloading algorithm that decomposes for a given constraint and debits new dynamic adjustments to maximize execution reliability for given energy consumption and delay constraint. Sahni et al. [31] proposed a joint dependent task offloading and flow scheduling heuristic algorithm for minimizing task completion time considering the dependencies between tasks and conflicts of network flow. Zhang et al. [32] mitigate the concurrent request scheduling problems based on directed acyclic graphs in an online manner. They use a Markov decision process model to decompose requests into subtasks, assign schedules based on different states of subtasks, and use reinforcement learning methods to make decisions for each step. Most existing studies of application offloading in IoV do not take into account the fact that smart vehicle applications also consist of multiple subtasks with certain dependencies between tasks.

## System Model and problem formulation

In this section, we first explain the system model under the cloud edge hybrid architecture, including the network model, vehicle application model, transmission model, and computation model.

### Network Model

A hybrid cloud-edge system is considered, as shown in Fig. 1, which consists of a cloud, multiple eNodes with multiple servers, and some intelligent vehicles with applications. The eNodes all have their different locations and wireless communication coverage, and we denote the set of these eNodes in a certain range as $N = \{N_1, N_2, ..., N_K\}$, the set of the $k^{th}$ eNode servers as

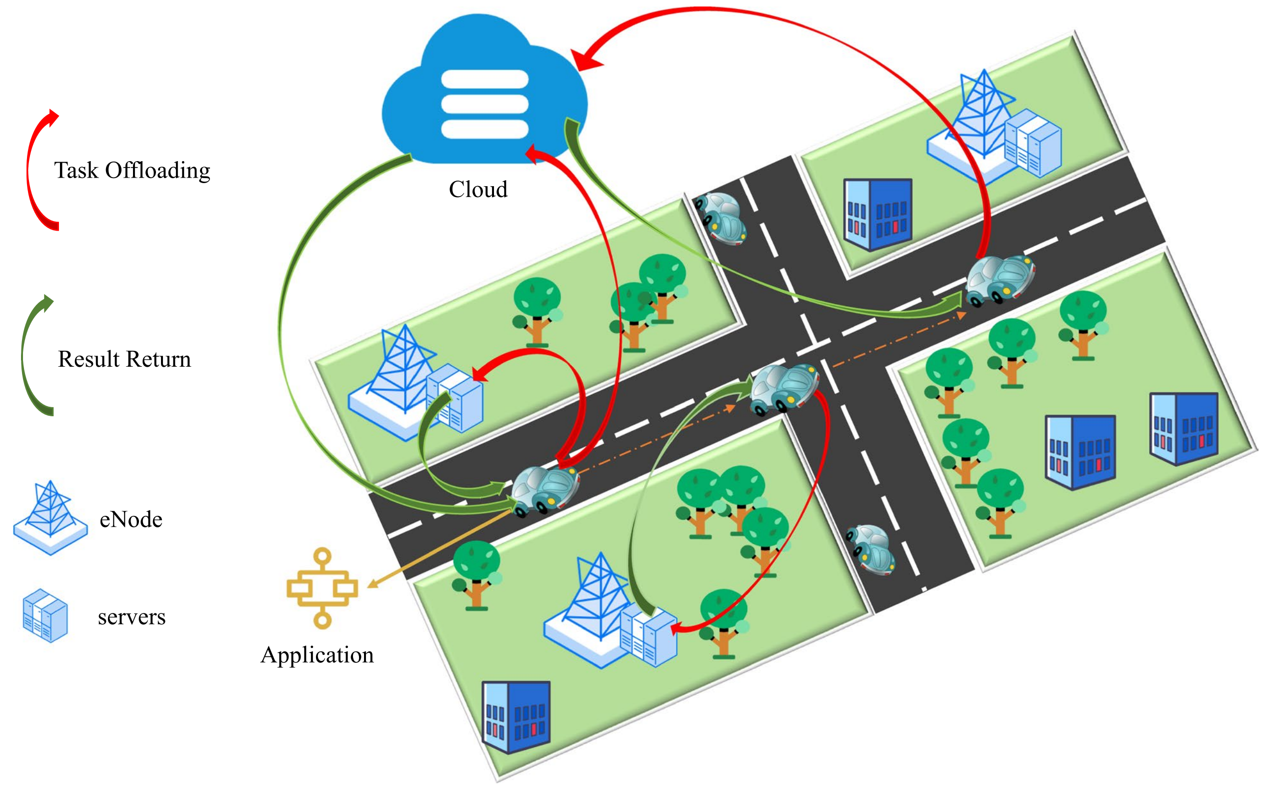


**Fig. 1** System model

$S_k = \{S_{k,1}, S_{k,2}, ..., S_{k,L}\}$. In the system, the intelligent vehicles travel on the road and the set of intelligent vehicles is indicated by $V = \{V_1, V_2, ..., V_h\}$. Each intelligent vehicle has computational-intensive and latency-sensitive applications that need to be offloaded to an eNode server or a server in the cloud for execution. The set of these applications is denoted as $N = \{A_1, A_2, ..., A_I\}$. An application can be structured with *j* tasks and the set of tasks belonging to the same application $A_i$ is $A_i = \{T_{i,1}, T_{i,2}, ..., T_{i,J}\}$. In this paper, we consider scenarios where scheduling and computation are done on a task-by-task basis. Each task can be divided into any eNode which the intelligent vehicle can communicate with, and tasks belonging to the same application do not have to be assigned to the same eNode, e.g. $T_{1,1}$ can be scheduled to $N_1$ for execution, while $T_{1,2}$ can be scheduled to $N_2$ for execution. The intelligent vehicle can only schedule tasks to the eNodes that are communicable at the current moment. Considering the mobility of the intelligent vehicle, if the intelligent vehicle moves out of the communication range of the eNode, then the intelligent vehicle will no longer be able to offload tasks to that eNode. Whereas the communication range of the cloud server is full coverage of the scenario. The notations used in this paper are summarized in Fig. 2.

## Application Model

We consider that the tasks in the application of intelligent vehicles are dependent on each other, so we represent the applications as directed acyclic graphs shown in Fig. 3. A directed acyclic graph represents a class of intelligent vehicle applications. In the same directed acyclic graph, the task pointed by the arrow needs to be executed after the task at the end of the arrow is executed. For instance, $T_{1,2}$ and $T_{1,3}$ can be executed only after $T_{1,1}$ is executed in the figure, and $T_{1,8}$ needs to be executed after both $T_{1,6}$ and $T_{1,7}$ have been executed. The dependencies between tasks of the same application cause them to require the results of other tasks to be executed before they can be executed. There are no dependencies between different applications, so there is no restriction on the order of execution between their tasks.

## Transmission Model

The completion of each application needs to go through three stages, which are scheduling the task to the designated server, executing the task at the designated server, and transmitting the task execution result from the server back to the intelligent vehicle. The application is not completed until the last task execution result is transmitted back to the intelligent vehicle. Since the data of task results are usually small and can be neglected, the finishing time of the application comprises task data transmission time and task computation time.

### *eNode Transmission*

We assume that the wireless communication between intelligent vehicles and eNodes is based on non-orthogonal multiple access technology [33, 34], so in this system, the bandwidths between different eNodes and intelligent vehicles are different and their communication does not interfere with each other. The bandwidth between eNode $N_k$ and intelligent vehicle $V_h$ is denoted by $b_{h,k}$, the channel gain between intelligent vehicle $V_h$ and eNode $N_k$ is denoted by $g_{h,k}$, and the transmission power between intelligent vehicle $V_h$ and eNode $N_k$ is denoted by $p_{h,k}$. Therefore, the transmission rate between intelligent vehicle $V_h$ and eNode $N_k$ can be given by

$$R_{h,k} = b_{h,k} log\left(1 + \frac{g_{h,k}p_{h,k}}{N_0}\right) \tag{1}$$

where $N_0$ indicates the Gaussian noise power inside the channel and $t_{k,h}$ the transmission time of input data $d_k$ from intelligent vehicle $V_h$ and eNode $N_k$. Consequently, the transmission duration $t_{k,h}$ is

$$t_{h,k} = \frac{d_{i,j}}{b_{k,h} log\left(1 + \frac{g_{h,k}p_{h,k}}{N_0}\right)} \tag{2}$$

### *Cloud Transmission*

Similarly, when the tasks are transmitted from the intelligent vehicle $V_h$ to the cloud, the corresponding transmission rate is:

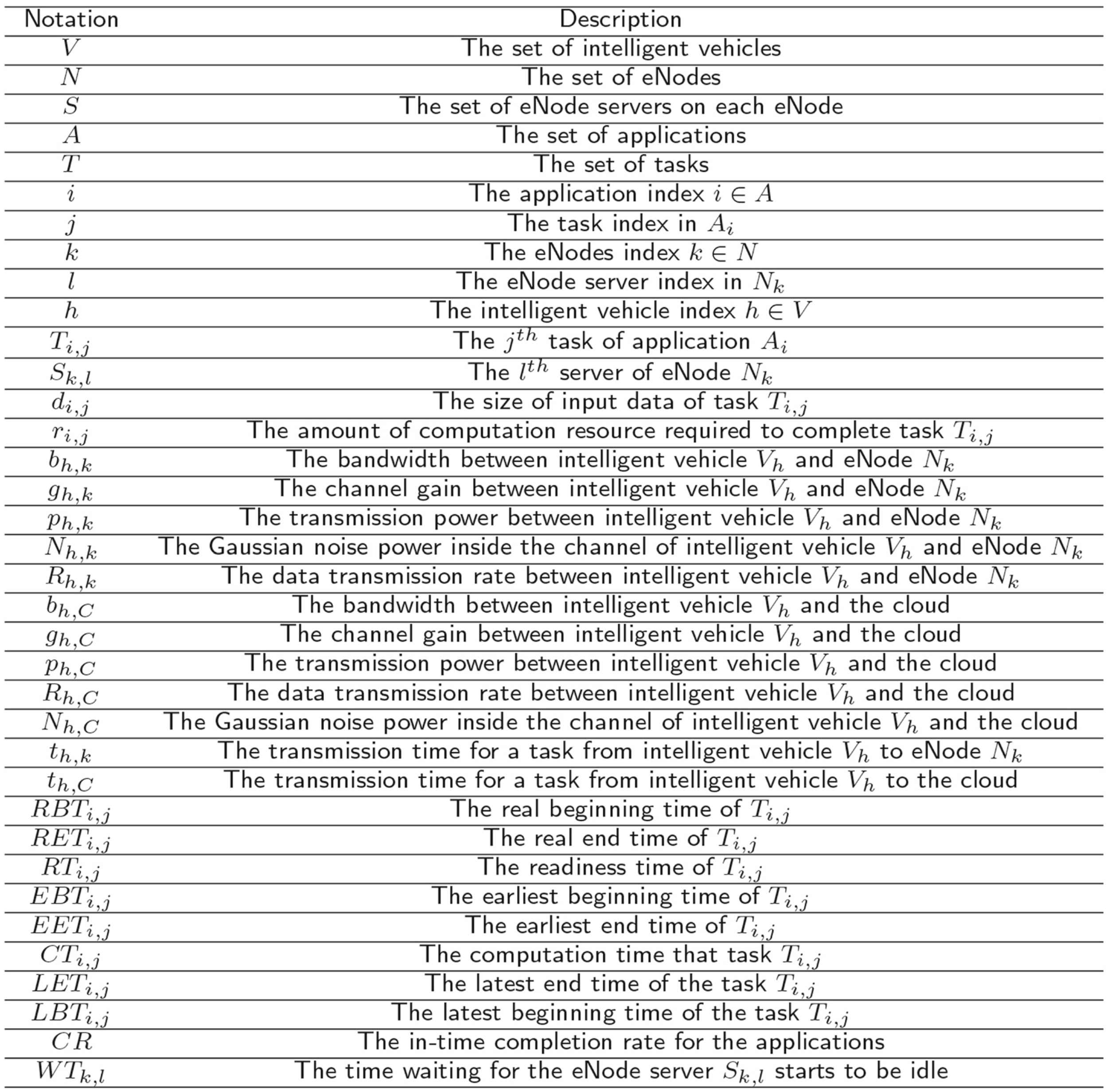
$$R_{h,C} = b_{h,C} log\left(1 + \frac{g_{h,C}p_{h,C}}{N_0}\right) \tag{3}$$

where $b_{h,C}$ indicates the bandwidth between an intelligent vehicle $V_h$ and the cloud, $g_{h,k}$ represents the channel gain between intelligent vehicle $V_h$ and the cloud, and $p_{h,k}$ the transmission power between intelligent vehicle $V_h$ and the cloud. The transmission time $t_{k,C}$ for a task with size $d_k$ to be transferred from intelligent vehicle $V_h$ to the cloud is

$$t_{h,C} = \frac{d_{i,j}}{b_{h,C} log\left(1 + \frac{g_{h,C}p_{h,C}}{N_0}\right)} \tag{4}$$

## Computation Model

Due to limited computing resources, we assume that each server on the eNode can only execute one task at the same time, and each task can only be executed by one server.



| Notation | Description |
|---|---|
| $V$ | The set of intelligent vehicles |
| $N$ | The set of eNodes |
| $S$ | The set of eNode servers on each eNode |
| $A$ | The set of applications |
| $T$ | The set of tasks |
| $i$ | The application index $i \in A$ |
| $j$ | The task index in $A_i$ |
| $k$ | The eNodes index $k \in N$ |
| $l$ | The eNode server index in $N_k$ |
| $h$ | The intelligent vehicle index $h \in V$ |
| $T_{i,j}$ | The $j^{th}$ task of application $A_i$ |
| $S_{k,l}$ | The $l^{th}$ server of eNode $N_k$ |
| $d_{i,j}$ | The size of input data of task $T_{i,j}$ |
| $r_{i,j}$ | The amount of computation resource required to complete task $T_{i,j}$ |
| $b_{h,k}$ | The bandwidth between intelligent vehicle $V_h$ and eNode $N_k$ |
| $g_{h,k}$ | The channel gain between intelligent vehicle $V_h$ and eNode $N_k$ |
| $p_{h,k}$ | The transmission power between intelligent vehicle $V_h$ and eNode $N_k$ |
| $N_{h,k}$ | The Gaussian noise power inside the channel of intelligent vehicle $V_h$ and eNode $N_k$ |
| $R_{h,k}$ | The data transmission rate between intelligent vehicle $V_h$ and eNode $N_k$ |
| $b_{h,C}$ | The bandwidth between intelligent vehicle $V_h$ and the cloud |
| $g_{h,C}$ | The channel gain between intelligent vehicle $V_h$ and the cloud |
| $p_{h,C}$ | The transmission power between intelligent vehicle $V_h$ and the cloud |
| $R_{h,C}$ | The data transmission rate between intelligent vehicle $V_h$ and the cloud |
| $N_{h,C}$ | The Gaussian noise power inside the channel of intelligent vehicle $V_h$ and the cloud |
| $t_{h,k}$ | The transmission time for a task from intelligent vehicle $V_h$ to eNode $N_k$ |
| $t_{h,C}$ | The transmission time for a task from intelligent vehicle $V_h$ to the cloud |
| $RBT_{i,j}$ | The real beginning time of $T_{i,j}$ |
| $RET_{i,j}$ | The real end time of $T_{i,j}$ |
| $RT_{i,j}$ | The readiness time of $T_{i,j}$ |
| $EBT_{i,j}$ | The earliest beginning time of $T_{i,j}$ |
| $EET_{i,j}$ | The earliest end time of $T_{i,j}$ |
| $CT_{i,j}$ | The computation time that task $T_{i,j}$ |
| $LET_{i,j}$ | The latest end time of the task $T_{i,j}$ |
| $LBT_{i,j}$ | The latest beginning time of the task $T_{i,j}$ |
| $CR$ | The in-time completion rate for the applications |
| $WT_{k,l}$ | The time waiting for the eNode server $S_{k,l}$ starts to be idle |

**Fig. 2** Summary of Key Notations

Since there are dependencies between tasks and eNodes have coverage limits, there are two conditions that need to be satisfied for each task to start its execution. Condition one is that all tasks pointing to it in the directed acyclic graph have been completed, and condition two is that there are free servers in the communicable eNodes of the intelligent vehicle this task is located in to execute it.

We use a Boolean variable $e^{i}_{m,j}$ to denote whether the task $T_{i,m}$ in the directed acyclic graph points to $T_{i,j}$, it can be given as

$$e^{i}_{m,j} = \begin{cases} 1, & \textit{if task } T_{i,m} \textit{ points to } T_{i,j} \textit{ in the DAG} \\ 0, & \textit{otherwise} \end{cases} \tag{5}$$

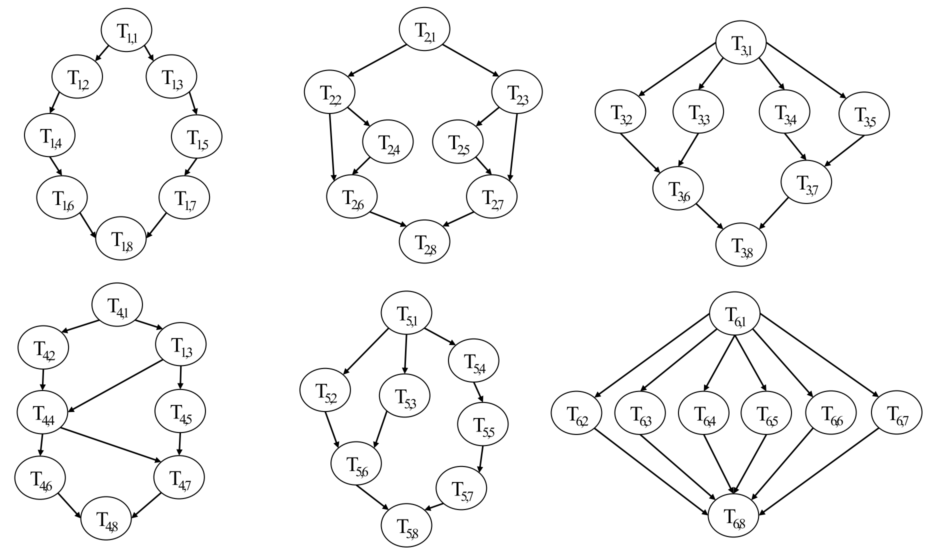


**Fig. 3** Application structure

If $e^{i}_{m,j} = 1$ , the task $T_{i,j}$ needs to be executed after the execution of $T_{i,m}$ is finished.

We denote the real end time of the task $T_{i,j}$ by $RET_{i,j}$. When all the tasks pointing to the task $T_{i,j}$ in the directed acyclic graph are completed, the task $T_{i,j}$ is ready to wait for the idle server to execute it, and its readiness time $RT_{i,j}$ can be expressed as

$$RT_{i,j} = \max_{e^{i}_{x,j}=1} RET_{i,x} \tag{6}$$

where $pre(T_{i,j})$ denotes the set of all tasks pointing to task $T_{i,j}$ in the directed acyclic graph.

If the task $T_{i,j}$ will be scheduled by the algorithm to be executed on eNode server $S_{k,l}$, and server $S_{k,l}$ is idle when the task $T_{i,j}$ is ready, the task can start execution at the readiness time. If there is task $T_{x,y}$ being executed on the server $S_{k,l}$ when the task is ready, the task needs to wait until the server $S_{k,l}$ is idle before execution, so the earliest beginning time of the task $T_{i,j}$ can be expressed as

$$EBT_{i,j} = max\{RT_{i,j}, RET_{x,y}\} \tag{7}$$

where $RET_{x,y}$ denotes the real ending time of the task executed on the server before task $T_{i,j}$.

The computation time $CT_{i,j,k,l}$ of task $T_{i,j}$ on eNode server $S_{k,l}$ is

$$CT_{i,j,k,l} = \frac{d_{i,j}}{c_{k,l}} \tag{8}$$

where $d_{i,j}$ indicates the size of input data of task $T_{i,j}$ and $c_{k,l}$ the computing speed of $S_{k,l}$.

To be similar, the computation time $CT_{i,j,C}$ of task $T_{i,j}$ deployed on the cloud server is

$$CT_{i,j,C} = \frac{d_{i,j}}{C} \tag{9}$$

where $C$ indicates the computing speed of the cloud server.

With the knowledge of the application completion time constraints and the dependencies between the tasks, we can calculate the latest ending time and the latest beginning time of the task $T_{i,j}$ accordingly, and they can be expressed as

$$LET_{i,j} = \min_{T_{i,x} \in suc(T_{i,j})} (LET_{i,x} - \min(CT^{min}_{i,x}, CT_{i,x,C})) \tag{10}$$

$$LBT_{i,j} = LET_{i,j} - \min(CT^{min}_{i,x}, CT_{i,x,C}) \tag{11}$$

where $suc(T_{i,j})$ represents the set of all tasks pointed to by $T_{i,j}$ in the directed acyclic graph and $CT^{min}_{i,x}$ represents the shortest computation time of task $T_{i,x}$ among all the servers that can communicate with the intelligent vehicle.

The earliest ending time $EET_{i,j}$ of task $T_{i,j}$ can be expressed as

$$EET_{i,j} = EST_{i,j} + \min(t_{h,k} + CT_{i,j}, t_{h,C} + CT_{i,j,C}) \tag{12}$$

We denote whether the application $A_I$ is completed in time by $IT_i$, and it can be given as

$$IT_i = \begin{cases} 1, & if\ A_i\ is\ completed\ in\ time \\ 0, & otherwise \end{cases} \tag{13}$$

Then we can obtain the in-time completion rate $P$ for this batch of applications as

$$CR = \frac{\sum_{i=1}^{I} A_i}{I} \tag{14}$$

## Problem Formulation

In this section, we define the problem of minimizing the completion time under the condition that the completion time constraint is satisfied as much as possible.

We use $x(T_{i,j}, S_{k,l})$ to denote whether task $T_{i,j}$ is scheduled to be executed on eNode server $S_{k,l}$, it can be given as

$$x(T_{i,j}, S_{k,l}) = \begin{cases} 1, & if\ task\ T_{i,j}\ is\ scheduled\ to\ server\ S_{k,l} \\ 0, & otherwise \end{cases} \tag{15}$$

Therefore, the real ending time $RET_i$ of application $A_i$ scheduled to eNodes can be denoted as

$$RET_i = RET_{i,J} = RT_{i,J} + \sum_{k=1}^{K} \sum_{l=1}^{L} [x(T_{i,J}, S_{k,l}) CT_{i,j,k,l} + t_{h,k} + WT_{k,l}] \tag{16}$$

and if scheduled to the cloud can be denoted as

$$RET_i = RET_{i,J} = RT_{i,J} + CT_{i,j,C} + t_{h,C} \tag{17}$$

where $WT_{k,l}$ represents the waiting time to wait for the server to be idle if the task is ready but the server still has tasks in progress.

We denote the completion time constraint of application $A_i$ as $CTC_i$. The application completes in time requires the real ending time to be less than the completion time constraint. Furthermore, the problem can be formulated as

$$\begin{aligned} &(\mathbf{P1})\ \min\left(\frac{\sum_{i=1}^{I} RET_i}{\alpha CR}\right) \\ &s.t.\ RET_i \le CTC_i \qquad \forall A_i \in A \\ &\quad dist(T_{i,j}, S_{k,l}) \le Rad_k \quad \forall x(T_{i,j}, S_{k,l}) = 1 \\ &\quad Count_{k,l} \le 1 \qquad \forall t \in T \\ &\quad RET_{i,m} \le RBT_{i,j} \qquad \forall e^i_{m,j} = 1 \end{aligned}$$

where $\alpha$ denotes a factor of completion rate $CR$. $dist(T_{i,j}, S_{k,l})$ denotes a distance from task $T_{i,j}$ to server $S_{k,l}$, $Rad_k$ denotes the coverage of eNode $N_k$, and $dist(T_{i,j}, S_{k,l})$ needs to be less than $Rad_k$ for the task $T_{i,j}$ to be delivered to server $S_{k,l}$. $Count_{k,l}$ denotes the number of tasks on the server at the moment $t$, and only one task can be executed at the same time. $RBT_{i,j}$ denotes the real beginning time of the task $T_{i,j}$.

If the mobility of intelligent vehicles is not considered and there is only one eNode and all intelligent vehicles can communicate with this eNode, then this problem can be reduced to a job-shop scheduling problem (JSP), which is NP-hard and difficult to solve directly [35].

## Proposed Algorithm MTS-MEoC

In this section, to solve the above problem, we proposed a method of multiple applications scheduling to eNode or cloud for execution (MTS-MEoC). The algorithm offloads tasks to the appropriate server on a suitable communicable eNode or the server in the cloud for execution.

First of all, to prioritize the order among tasks, we use a flexible latest beginning time $FLBT_{i,j}$ to indicate the urgency of task $T_{i,j}$, which can be denoted as

$$FLBT_{i,j} = \min_{T_{i,m} \in suc(T_{i,j})} (LET_{i,m} - ET^{max}_{i,m}) \tag{18}$$

where $ET^{max}_{i,m}$ indicates the maximum execution duration of task $T_{i,m}$ on all servers of eNode, and the $FLBT_{i,j}$ of the last task $T_{i,j}$ of the application $A_i$ is calculated when $LET_{i,m}$ is $CTC_i$ in Eq. (18). Once the application reaches this time, it means that the risk of application timeout is greatly increased and the priority of this task needs to be raised so that the task is executed as soon as possible to guarantee the in-time completion rate of the application. Meanwhile, sorting by FLBT allows applications arriving at different moments to be sorted according to the same standard and does not affect the priority of the task by the difference in the ordered arrival time.

At the beginning of the algorithm, all ready tasks (i.e., tasks that have reached RT) are placed in the task queue waiting to be scheduled. The first task of the application is the moment when the application arrives. The tasks in the task queue are first sorted by FLBT, and each task is scheduled for execution in this order. We schedule each task by selecting the idle server that can get the minimum EET. If there is no free server available for communication at the current moment, it is necessary to wait for the server to finish executing its current task. However, if the task reaches FLTB, or if the waiting time is about to reach FLTB, the task needs to be put into the waiting queue, and the tasks in the waiting queue are given priority in each algorithmic scheduling. Tasks in the waiting queue are prioritized to the fastest server to ensure that the task is completed no later than the latest ending time. They will choose the server that can get the minimum EET for scheduling, and if that server is not available task $T_{i,j}$ will also be queued into that server $S_{k,l}$ first to wait for priority processing, while at the same time when other tasks choose a server, the ready time of server $S_{k,l}$ needs to be added to the execution time of task $T_{i,j}$. And if the task will exceed the latest ending time no matter which server is selected, the task will be scheduled to the cloud for execution, freeing up quality eNode resources for other applications to ensure the overall QoS. The details of the algorithm are shown in Algorithm 1.

```
1: Initialize the task queue $TQ$, waiting queue $WQ$.
2: repeat
3:    Add the tasks with $RT$ less than the current time to $TQ$
4:    Calculate all $FLBT$s according to Eq(18)
5:    Sort $TQ$ and $WQ$ according to FLBTs.
6:    for task $T_{i,j}$ in $WQ$ do
7:       Calculate $LET_{i,j}$ according to Eq(10);
8:       for all servers that can communicate do
9:          Calculate $EET_{i,j}$ according to Eq(12);
10:         if min$EET_{i,j} > LET_{i,j}$ then
11:            Schedule $T_{i,j}$ to cloud and remove it from $TQ$;
12:         else
13:            Schedule $T_{i,j}$ to the corresponding server and remove it from $WQ$;
14:         end if
15:      end for
16:   end for
17:   repeat
18:      Choose task $T_{i,j}$=$TQ$.head;
19:      for all servers that can communicate and are idle do
20:         Calculate $EET_{i,j}$ according to Eq(12);
21:         Get the server $S_{k,l}$ corresponding to the minimum $EET$;
22:         if the server is from the cloud then
23:            Schedule $T_{i,j}$ to cloud and remove it from $TQ$;
24:         else
25:            if server $S_{k,l}$ is idle then
26:               Schedule $T_{i,j}$ to $S_{k,l}$ and remove it from $TQ$;
27:            else
28:               Add $T_{i,j}$ to $WQ$ and remove it from $TQ$;
29:            end if
30:         end if
31:      end for
32:   until $TQ = \Phi$
33: until all the applications are completed
```

**Algorithm 1** MTS-MEoC

## Experiment

In this section, we conduct a series of studies on the average application completion time, average application waiting time, in-time application completion rate, and utilization of servers based on the cloud-edge hybrid architecture.

### Experiment settings

We conduct the experiments based on the public and widely-used EUA dataset [36]. It contains the locations of the 125 edge servers (base stations) in the Melbourne central business district area in Australia. We use the base station information from the EUA as the information of our eNodes with a communication range of 400-800 meters and a certain number of servers in each eNode. The computation speed of each eNode server is $1\times10^8$ – $3\times10^8$ kB/s and that of the cloud is $1\times10^9$ kB/s.

At the start time of the experiment, there will be a certain number of intelligent vehicles traveling in a straight line according to their respective speed and direction, and their speed sizes range from 36 km/h to 72 km/h. There will be applications on the intelligent vehicles that need to be unloaded, and the task size of the applications is between $2\times10^6$–$5\times10^6$ kB. The completion time constraint $CTC_i$ of the application $A_i$ is 0.5 – 0.7s. The dependencies of tasks in different applications are not the same and are classified into six cases illustrated in Fig. 3.

The intelligent vehicles communicate with the eNodes and cloud through wireless. The channel bandwidth between intelligent vehicle $V_h$ and eNode $N_k$ $b_{h,k}$ is 500–800 kHz, and that between intelligent vehicle $V_h$ and cloud $b_{h,C}$ is 50 kHz. The power gain $g_{h,k}$ is $6.5\times10^{-4}$ at a reference distance of one meter and the transmission power $p_{h,k}$ is 0.1 W. The noise power of the system $N_0$ is $10^{-10}$ W.

We consider the following algorithms as the peers:

1)Random Selection Algorithm (RSA): In this algorithm, every task will be scheduled to a random eNode and a random server of this eNode.

2)Greedy Algorithm (GA): The greedy algorithm schedules every task to the eNode with the shortest transmission time, and then chooses the idle server with the fastest computation speed.

3)Multiple Applications Multiple Tasks Scheduling (MAMTS) [37]: In this algorithm, the tasks in the same application will be scheduled to the eNode with the shortest overall execution time, and then the tasks in this application will select the server according to a priority.

In our experiments, we compare the in-time completion rate, average completion time, average waiting time, and server utilization of MTS-MEoC with other benchmark algorithms for different load cases.

### Result and Comparison

Our experiments used an environment with five servers at each of the five eNodes. A variety of different numbers of intelligent vehicles were tested, and each vehicle had an application required to offload for execution.

As shown in Fig. 4, different kinds of applications are represented by different shapes, such as triangles, prototypes, and squares. The applications scheduled to the same eNode are represented by the same color. We can see that the RSA algorithm schedules some applications that are very far from the eNode to that eNode, and the picture has more cross lines of different colors. While GA prefers to select applications closer to the eNode for scheduling. MAMTS assigns more balanced applications to each eNode. MTS-MEoC has multiple color lines attached to the same Application because it can assign different tasks in the directed acyclic graph of an application to different eNodes.

#### *Comparison of average completion time*

**Results:** Figure 5 illustrates the average completion time of all applications for our proposed MTS-MEoC and the other three methods for different numbers of applications. In the comparison, our proposed method achieves the lowest average completion time (on average, 24.4% lower than MAMTS; 37.0% lower than GA and 63.1% lower than RSA ). With a small number of applications,

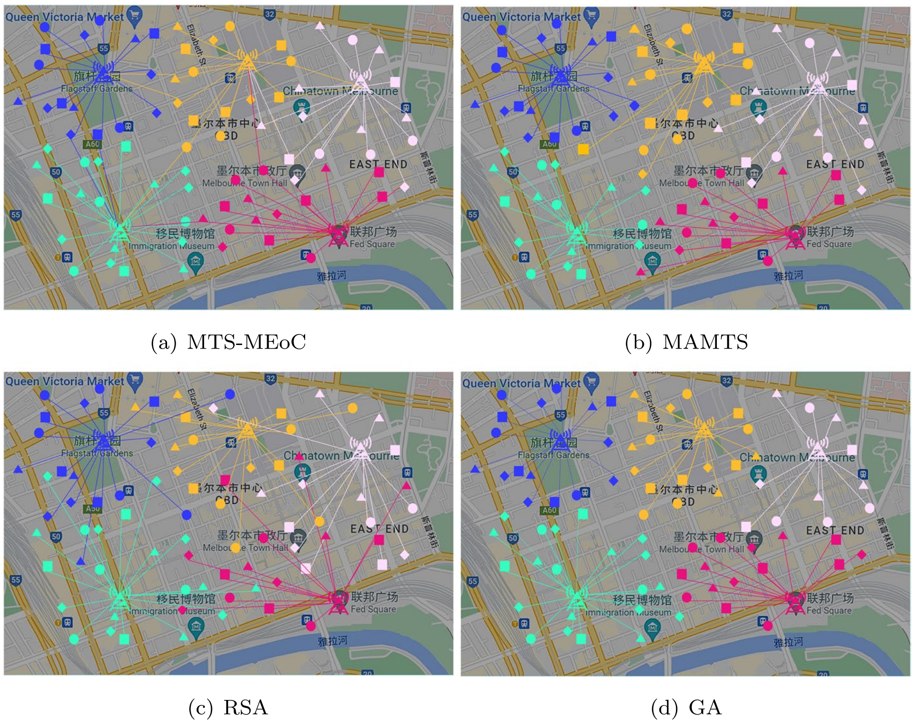


**Fig. 4** How applications are scheduled with different algorithms

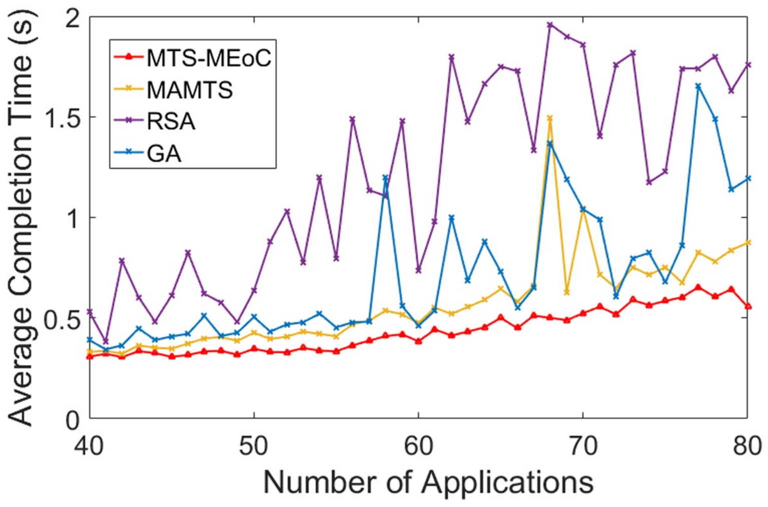


**Fig. 5** The average completion time

all three algorithms, except the RSA, possess a certain degree of stability, but as the amount of applications increases and the servers all enter into high-load usage, the performance of the MAMTS and GA algorithms shows some volatility, while MTS-MEoC still guarantees a low completion time.

**Analysis:** MTS-MEoC has the lowest average completion time mainly for two reasons: 1) It takes into account the hybrid cloud-edge architecture, so it is more flexible for task scheduling and can avoid some key tasks waiting for idle servers as much as possible; 2. It takes into account the load of eNodes and schedules tasks to eNodes with lower load to execute, making full use of server resources.

### *Comparison of in-time completion rate*

**Results:** Figure 6 shows the application's In-time completion rate, which corresponds to the percentage of applications that are finished within the specified completion time constraints. MTS-MEoC achieved the highest task completion rate, (on average 40.3% higher than MAMTS, 41.9% higher than GA, and 143% higher than RSA). When the load is low, MTS-MEoC and MAMTS can basically complete all applications in time. As the server load rises, both have some applications that cannot be completed in time, but MTS-MEoC still has the highest completion rate, which means that MTS-MEoC can provide high-quality services to more intelligent vehicle users.

**Analysis:** GA and RSA do not prioritize tasks, so some urgent tasks are not completed, resulting in a lower in-time completion rate. GA algorithm mainly pursues the fastest execution of tasks, so the performance is better when the load is high compared to RSA and MAMTS. Our proposed algorithm MTS-MEoC and MAMTS both prioritize the tasks, so the more urgent tasks are processed first, which makes the task completion rate higher when the server load is low. But MAMTS has too many tasks performed on part of the eNodes after high server load without a load balancing strategy, so the completion rate becomes low. MTS-MEoC, however, offloads tasks to eNodes with a lower load to execute them as appropriate and also to the cloud, so it is more flexible and has a higher completion rate.

### *Comparison of average waiting time*

**Results:** Figure 7 demonstrates the average waiting time before the application completes execution, the time when the task is ready after all the tasks pointing to it in the directed acyclic graph have been completed but there are no idle servers available for scheduling. In the comparison, MTS-MEoC obtains the lowest average completion time (on average 31.2% lower than MAMTS, 71.9% lower than RSA, and 37.4% lower than GA), which means that MTS-MEoC can provide a better quality of service for intelligent vehicle applications.

**Analysis:** The lower average waiting time of MTS-MEoC is mainly due to the fact that tasks are scheduled to be executed on eNodes with low loads so that the waiting time can be significantly reduced when there are too many tasks arriving at the same time. Also, more reasonable matching of tasks to servers can reduce the application waiting time to some extent.

### *Comparison of servers utilization*

**Results:** Figure 8 demonstrates the utilization of all servers throughout the process of providing computing services to intelligent vehicles. The utilization of MTS-MEoC is 13.3% lower than MAMTS, but 47.5% higher than GA and 170.3% higher than RSA. The utilization of the four algorithms is roughly at a stable level regardless of the load.

**Analysis:** Idle servers result in lower utilization, so utilization roughly reflects the level of an algorithm for server utilization. But it is not necessarily good if the utilization is high, because the utilization is also higher if the server with less computing power is always used. So although our algorithm has a lower utilization than

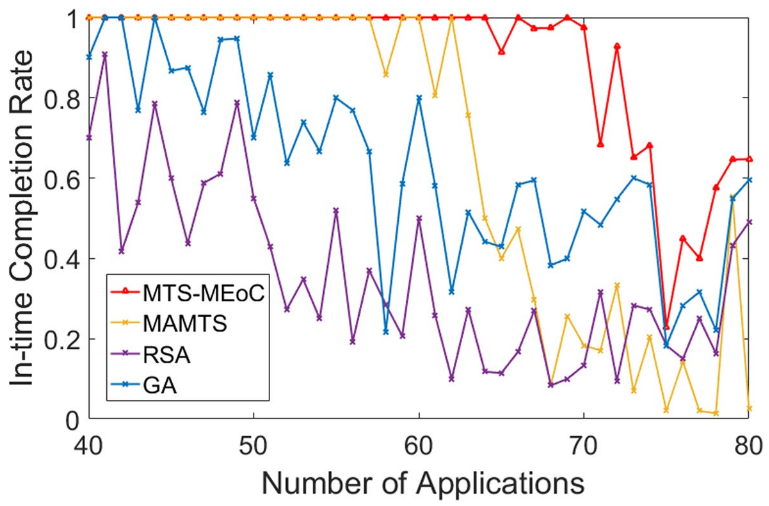


**Fig. 6** The in-time completion rate

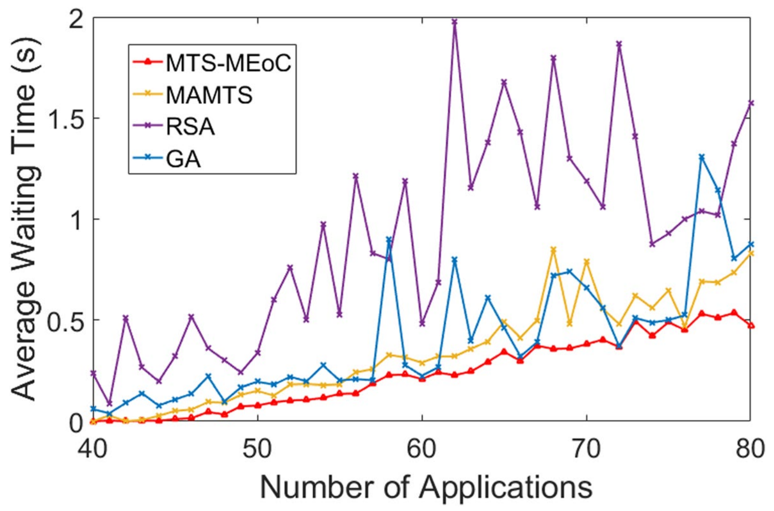


**Fig. 7** The average waiting time of applications

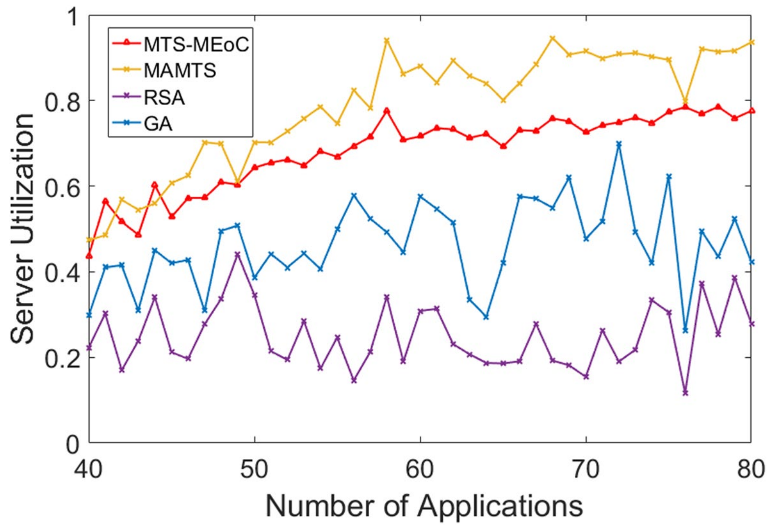


**Fig. 8** The servers utilization

MAMTS, the average completion time and waiting time of the previous algorithm can be combined to show that our algorithm uses the server more rationally and provides better quality of service with lower utilization. With the same quality of service, lower utilization also means cost savings.

## Conclusions and future work

In this paper, we consider the application offloading problem for intelligent vehicles based on a hybrid cloud-edge environment. We propose the algorithm MTS-MEoC considering the dependencies of tasks in the application. MTS-MEoC is experimentally proven to outperform its peers in terms of average completion time, in-time completion rate, and average waiting time. MTS-MEoC is able to provide a better quality of service to intelligent vehicle users.

In future work, we will consider the following aspects: 1) The application offloading problem cannot be reduced to a static optimization problem in reality, and we should consider the dynamic arrival of the application over time in our subsequent research; 2) We will use some data mining methods to implement load sensing and movement prediction, which will enable better task scheduling.

**Abbreviations**
IoV: Internet of Vehicles; QoS: Quality of Service; MEC: Multi-access Edge Computing; DAG: Directed acyclic graphs.

**Acknowledgements**
The authors will be grateful to the editor and anonymous referees for their valuable comments and suggestions.

**Authors' contributions**
Xifeng Xu performed the evaluation experiments and wrote the manuscript. Yunni Xia designed the study. Zeng Feng, Fan Li, and Xiaodong Fu proposed a new framework for the system, and Hong Xie proposed algorithm modifications. Mengdi Wang provided all the pictures. All authors participated actively in discussions and reviewed the manuscript. The author(s) read and approved the final manuscript.

**Funding**
This work is supported by the National Science Foundation of China (NSFC) with the granting No. 62172062 and the Graduate Student Research and Innovation Foundation of Chongqing, No.CYS21062.

**Availability of data and materials**
The EUA dataset is available at the URL https://github.com/982543386/eua-dataset.

## Declarations

**Ethics approval and consent to participate**
Not applicable.

**Competing interests**
The authors declare that they have no competing interests.

**Author details**
[1]College of Computer Science, Chongqing University, Chongqing, China. [2]Discovery Technology (Shenzhen)Limited, Shenzhen, China. [3]Key Laboratory of Fundamental Synthetic Vision Graphics and Image Science for National Defense, Sichuan University, Chengdu, China. [4]Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming, China. [5]College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, China.

Received: 29 July 2022 Accepted: 2 November 2022
Published online: 08 December 2022

**References**
1. Huang J, Tong Z, Feng Z (2022) Geographical poi recommendation for internet of things: A federated learning approach using matrix factorization. Int J Commun Syst. https://doi.org/10.1002/dac.5161
2. Dureja A, Sangwan S (2021) A review: efficient transportatio-future aspects of iov. Evolving Technologies forComputing, Communication and Smart World, p 97–108
3. Hakimi A, Yusof KM, Azizan MA, Azman MAA, Hussain SM (2021) A survey on internet of vehicle (iov): applications & comparison of vanets, iov and sdn-iov. ELEKTRIKA J Electr Eng 20(3):26–31
4. Santhakumar G, Whenish R (2022) Internet of Vehicles. Springer International Publishing, Cham, pp 259–281
5. Chen Y, Zhao F, Chen X, Wu Y (2022) Efficient multi-vehicle task offloading for mobile edge computing in 6g networks. IEEE Trans Veh Technol 71(5):4584–4595. https://doi.org/10.1109/TVT.2021.3133586
6. Varsha P, Priyadharshini D, Swetha S et al (2021) Video analysis of vehicle and pedestrian using neural network. Ann Romanian Soc Cell Biol 4727–4733
7. Bao Z, Hossain S, Lang H, Lin X (2022) High-definition map generation technologies for autonomous driving: a review. arXiv preprint arXiv:2206.05400
8. Liu Z, Liwang M, Hosseinalipour S, Dai H, Gao Z, Huang L (2022) RFID: towards low latency and reliable DAG task scheduling over dynamic vehicular clouds. CoRR abs/2208.12568. https://doi.org/10.48550/arXiv.2208.12568
9. Jie D, Zhao Y, Liu Y, Qi L, Hu C (2014) Cloud-assisted analysis for energy efficiency in intelligent video systems. J Supercomput 70(3):1345–1364
10. Cao B, Sun Z, Zhang J, Gu Y (2021) Resource allocation in 5g IoV architecture based on SDN and fog-cloud computing. IEEE Trans Intell Transp Syst PP(99):1–9
11. Mohapatra H, Rath AK, Panda N (2022) IoT infrastructure for the accident avoidance: an approach of smart transportation. Int J Inf Technol 14(2):761–768
12. Chen Y, Gu W, Xu J et al (2022) Dynamic task offloading for digital twin-empowered mobile edge computing viadeep reinforcement learning. China Commun
13. Shetty RS (2021) Multi-Access Edge Computing in 5G. Apress, Berkeley, pp 69–102
14. Abdullah MFA, Yogarayan S, Razak SFA, Azman A, Amin AHM, Salleh M (2022) Edge computing forvehicle to everything: a short review. F1000Research 10(1104):1104
15. Chen Y, Zhao F, Lu Y, Chen X (2021) Dynamic task offloading for mobile edge computing with hybrid energy supply. Tsinghua Sci Technol. https://doi.org/10.26599/TST.2021.9010050
16. Luo X, Zhou M, Li S, Xia Y, You Z-H, Zhu Q et al (2017) Incorporation of efficient second-ordersolvers into latent factor models for accurate prediction of missing qos data. IEEE Trans Cybern 48(4):1216–1228
17. Zhang H, Luan Q, Zhu J, Fangwei LI, Amp N (2018) Task offloading and resource allocation in vehicleheterogeneous networks with MEC. Chinese J Internet of Things 2(3):36–43
18. Liu Y, Yu H, Xie S, Zhang Y (2019) Deep reinforcement learning for offloading and resource allocation in vehicleedge computing and networks. IEEE Trans Veh Technol 68(11):11158–11168
19. Ye T, Lin X, Wu J, Li G, Li J (2020) Processing capability and qoe driven optimized computation offloadingscheme in vehicular fog based f-ran. World Wide Web 23(4):2547–2565
20. Huang L, Zhang L, Yang S, Qian LP, Wu Y (2020) Meta-learning based dynamic computation task offloadingfor mobile edge computing networks. IEEE Commun Lett 25(5):1568–1572
21. Xu J, Li D, Gu W et al (2022) Uav-assisted task offloading for IoT in smart buildings and environment via deep reinforcement learning. Build Environ 222. https://doi.org/10.1016/j.buildenv.2022.109218
22. Peng Q, Xia Y, Wang Y, Wu C, Luo X, Lee J (2020) A decentralized reactive approach to online taskoffloading in mobile edge computing environments. In: International Conference on Service-OrientedComputing. Springer, p 232–247
23. Huang J, Gao H, Wan S et al (2023) Aoi-aware energy control and computation offloading for industrial IoT. Futur Gener Comput Syst 139:29–37
24. Wu W, Dong J, Sun Y, Yu FR (2022) Heterogeneous markov decision process model for joint resourceallocation and task scheduling in network slicing enabled internet of vehicles. IEEE Wireless Commun Lett 11(6):1118–1122. https://doi.org/10.1109/LWC.2022.3152177
25. Huang W, Xiong NN, Mumtaz S (2021) Joet: Sustainable vehicle-assisted edge computing for internet of vehicles. arXiv:2108.02443. https://doi.org/10.48550/arXiv.2108.02443
26. You M, Zhou H, Zhuang Y (2020) Research on application of auction algorithm in internet of vehicles taskscheduling under fog environment. In: Proceedings of the 2020 the 4th International Conference on Innovationin Artificial Intelligence, p 242–249
27. Deng Y, Chen Z, Yao X, Hassan S, Wu J (2019) Task scheduling for smart city applications based onmulti-server mobile edge computing. IEEE Access 7 :14410–14421
28. Lakhan A, Memon MS, Elhoseny M, Mohammed MA, Qabulio M, Abdel-Basset M et al (2022) Cost-efficient mobility offloading and task scheduling for microservices iovt applications in container-based fogcloud network. Cluster Comput 25(3):2061-2083
29. Chen Y, Gu W, Li K (2022) Dynamic task offloading for internet of things in mobile edge computing via deep reinforcement learning. Int J Commun Syst. https://doi.org/10.1002/dac.5154
30. Ying S, Li J, Xiguang W (2020) Dag-based task scheduling in mobile edge computing. 2020 7th International conference on information science and control engineering (ICISCE). https://doi.org/10.1109/ICISCE50968.2020.00095

31. Sahni Y, Cao J, Yang L, Ji Y (2020) Multihop offloading of multiple dag tasks in collaborative edge computing. IEEE Internet of Things J 8(6):4893–4905
32. Zhang Y, Li R, Zhou Z, Zhao Y, Li, R (2021) Deep reinforcement learning for dag-based concurrent requestsscheduling in edge networks. In: International Conference on Wireless Algorithms, Systems, and Applications. Springer, p 359–366
33. Li K, Zhao J, Hu J et al (2022) Dynamic energy efficient task offloading and resource allocation for noma-enabled IoT in smart buildings and environment. Build Environ. https://doi.org/10.1016/j.buildenv.2022.109513
34. Chen Y, Xing H, Ma Z et al (2022) Cost-efficient edge caching for noma-enabled iot services. ChinaCommun
35. Li Y, Luo G, Wu B (2019) Flexible job shop scheduling based on genetically modified neighborhood hybrid algorithm. In: 2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA). pp 337–342. https://doi.org/10.1109/ICAICA.2019.8873501
36. Lai P, He Q, Abdelrazek M, Chen F, Hosking J, Grundy J et al (2018) Optimal edge user allocation inedge computing with variable sized vector bin packing. In: International Conference on Service-OrientedComputing. Springer, p 230–245
37. Liu Y, Wang S, Zhao Q, Du S, Zhou A, Ma X et al (2020) Dependency-aware task scheduling invehicular edge computing. IEEE Internet of Things J 7(6):4961–4971

## Publisher's Note