TYPE Original article
DOI http://dx.doi.org/10.21271/ZJPAS.37.6.9

OPEN ACCESS

*Corresponding author
Sarkawt H. Abdulqadir
Sarkawt.hamaali@gpu.edu.iq

RECEIVED :11 /01 /2025,
ACCEPTED :02/07/ 2025
PUBLISHED :31/ 12/ 2025

KEYWORDS:
VFC, IoV, Task scheduling, Task offloading, Energy consumption, Response time.

Dynamic Task Scheduling with Mobility Consciousness in Vehicular Fog Computing Environments

Sarkawt H. Abdulqadir [1] , Hemin Ahmed Hatam[2], Abdullah Hameed M.Salih[3], Nahro Nooraldeen abbas[4], Mohammed Hassan Husain[1].

1Department of Information Technology, Kalar Technical Institute, Garmian Polytechnic University, Kurdistan Region, Iraq.

2 Computer Science Department, College of Science, University of Garmian, Kurdistan Region, Iraq.

3 General Directorate of Education in Garmian - Directorate of Education in Kalar, Ministry of Education in Kurdistan Region of Iraq.

4 Department of Computer Engineering, Kifri Technical College, Garmian Polytechnic University, Kurdistan Region, Iraq.

ABSTRACT

While Vehicular Fog Computing (VFC) is a hot research area for boosting processing power in connected vehicles (Internet of Vehicles), efficiently assigning the best resources is tricky. This is because new services like augmented reality and self-driving cars are exploding in popularity. These services require lightning-fast responses, powerful computing, and all while dealing with constantly moving vehicles. The biggest difficulty is guaranteeing tasks are completed super-fast (strict latency) as vehicles move in and out of range of roadside stations (RSUs). This research tackles the challenge of scheduling tasks for connected vehicles. It considers both how long a task takes to complete (latency) and how much battery power it uses (energy consumption). The system makes decisions about whether to run the task on the vehicle itself, a nearby roadside station (RSU), or even another RSU further down the road based on the vehicle's planned route. This approach aims to find a balance between getting tasks done quickly and keeping the vehicles running for as long as possible. Initially, we use a special technique called Markov renewal process (MRP) to capture how vehicle moves around over time. This method helps us understand the vehicle's mobility patterns. We then prove some technical properties about our goal of minimizing both latency and energy consumption. This lets us develop a fast and effective method (greedy heuristic) to find a good solution. On top of that, we propose a more powerful approach that combines two existing optimization techniques (Moth-flame and particle swarm) to tackle scheduling tasks in very large networks with moving vehicles. We tested our new scheduling method against existing ones, and computer simulations showed it to be very effective in solving task scheduling problems. The simulation outcomes demonstrate that our suggested MFO-PSO approach reduces average delays in tasks by 23.4% and uses 17.8% less energy than typical scheduling methods.

1.Introduction

As more and more vehicles connect to the internet (Internet of Vehicles), they're becoming smarter and able to handle more complex tasks that require a lot of computing power and need to be done very quickly. These tasks include features that help drivers, like collision avoidance, keeping track of traffic flow, managing car functions automatically, and even in-car entertainment (Liu et al., 2022). Nowadays, Modern vehicles are generating a rapidly growing amount of data for their applications. These applications need to be completed very quickly (latency-sensitive) and require a lot of processing power (computation intensive) (Liu et al., 2021b). With limited processing power and battery life, these vehicles struggle to run complex applications, hindering their ability to deliver high-quality services (Arthurs et al., 2021).

Cloud computing stepped up as a solution for the Internet of Vehicles (IoV) to manage its ever-growing demands. Vehicles can now shed heavy processing tasks by sending them wirelessly to powerful cloud data centers. The current cloud-based system, with data centers far from vehicles, hinders the offloading of critical tasks. This results in slow data transfer and hinders the performance of latency-sensitive applications in the Internet of Vehicles (IoV). As a solution to cloud computing limitations, fog computing has appeared as a promising approach for Internet of Thing and connected vehicles (Husain et al., 2024b). It tackles the issue of slow data transfer (latency) by bringing processing power closer to the devices, ultimately improving service quality. Building on the concept of fog computing, vehicular fog computing (VFC) is a new research area in intelligent transportation systems. VFC implements RSU systems which extend cloud services directly to vehicles to let them access Internet of Vehicles (IoV) network services (Martinez et al., 2020).

Workload management stands as the primary obstacle that operates in VFC environments. Car systems generate specific tasks that privately transmit to processing RSUs found within their vicinity. Research on vehicle task scheduling in VFC networks has existed in abundance but lacks sufficient investigation of scheduling efficiency under mobile vehicle movement with uncertain parking durations. Research on this topic needs additional investigation. Knowledge about vehicle movement behavior in fog networks enhances the efficiency of scheduling system processes. After sending requests to the appropriate RSU server, a vehicle might transition between RSU regions. Because vehicles can switch between roadside units (RSUs), some tasks might fail if they haven't finished processing when the car moves out of range. This forces the vehicle to resend the tasks to the new RSU, causing delays. While the completed tasks eventually reach the cloud and then the vehicle, this whole process takes extra time (Saeed et al., 2024, Asaad et al., 2024).

This work explores a novel task scheduling approach for vehicular fog computing (VFC) that considers how vehicles move. In VFC, vehicles send tasks to roadside units (RSU) for processing. Our research question is how to decide which tasks should be processed by the vehicle itself and which ones should be assigned to RSUs along the vehicle's route, considering the challenges mentioned earlier (task failure due to vehicle movement and extra time to get results from the cloud). Current research on mobility-aware offloading in VFC has some limitations. These strategies often rely on factors like vehicle speed, but they might only consider one possible route (unidirectional) or focus on tasks handled by a single roadside unit (RSU) (Misra and Bera, 2019, Raza et al., 2021). In a network where vehicles move with unpredictable speeds due to traffic, they can take many different routes, not just a single one in one direction. To efficiently assign tasks in this network, we need to figure out how long vehicles will stay within a certain area and which paths they'll likely take. Knowing how the vehicles will move is crucial for making the most of the network's resources (Mohammed and Sciences, 2022).

Running complex calculations for car features uses a lot of power on the vehicle's processor. Offloading these tasks to nearby RSU servers can help save energy, but these servers might not be powerful enough to handle everything without causing delays. To handle the demands of car-based applications (IoV), this system should consider both how much energy it uses and how long it takes to complete tasks. This paper proposes a new approach to scheduling tasks for connected vehicles (VFC). Unlike existing methods, ours takes into account both the tasks' deadlines and how the vehicles move around. This helps to reduce both the energy used by the vehicles and the time it takes to complete the tasks. We represent vehicle movement using a MRP to forecast both the duration of their stops and their routes. Following this, we frame the goal of task scheduling as a sub-modular function. Then, we propose a greedy algorithm to achieve the best possible outcome. However, we propose a more sophisticated method (hybrid optimization approach) for handling a large number of connected vehicles (large-scale networks). This new method combines the strengths of two existing techniques (Moth-flame Optimization and particle swarm optimization) to tackle the challenge of scheduling tasks while considering how vehicles move (mobility-aware). Here are the primary contributions we've made:

- We can express the problem of scheduling tasks considering vehicle movement as a complex mathematical equation (mixed integer non-linear programming problem). This equation helps us find the best way to complete tasks as quickly as possible (minimize latency) while also reducing the energy used by the vehicles.
- The model uses a technique called a Markov renewal process (MRP) to understand how vehicles move. This helps predict where a vehicle might be going next and how long it will stay in a particular location.
- We convert the original goal (objective function) into a more manageable form that leverages the benefits of submodular optimization. This allows us to find a good solution efficiently.
- This paper introduces a new method for scheduling tasks in large networks that considers how mobile devices move around. The method combines two existing techniques: MFO and PSO.

This paper is organized like this: First, we'll discuss what other researchers have done in task scheduling for vehicles using fog computing (Part 2). Then, we'll explain how our system works, including the challenges we're considering (Part 3). Next, we'll introduce a basic solution for scheduling tasks while vehicles move around (Part 4). Part 5 explains a more advanced solution that builds on a technique called MFO-PSO. We'll show how well our methods work in Part 6, and finally, summarize everything in Part 7.

2.RELATED WORK

Fog computing task scheduling has been a major research focus recently. One particularly interesting area is scheduling tasks for vehicles connected to the internet (Internet of Vehicles) (Hamdi et al., 2022). To understand existing research, we've categorized it into four main approaches: minimizing latency, conserving energy, balancing both latency and conserving energy, and adapting to how vehicles move (mobility-aware). We'll now explore these categories in more detail.

2.1Latency Aware Scheduling

For vehicles using a fog network, it's crucial to minimize delays in processing tasks. A system which executes tasks before deadlines allow the user to achieve satisfaction. Both studies (Zhang et al., 2020) and (Liu et al., 2019a) concentrate their research on vehicular network improvement. Using efficient resource allocation methods, the authors of reference (Zhang et al., 2020) showed how to minimize transmission time across vehicular fog networks. The method applies Perron-Frobenius theory in conjunction with weighted minimum mean error to achieve its purpose. The study of (Liu et al., 2019a) analyzes ways to reduce network delay time within vehicle edge computing systems. Their methodology includes creating an offloading scheme based on matching theory principles. Researchers used (Sun et al., 2020) concepts to establish a task scheduling mechanism that operates in vehicular edge networks. The system functions to minimize processing time duration and source resource utilization simultaneously. Researchers developed a distinctive algorithm by merging elements from both the partheno genetic algorithm and a hybrid one for this purpose.

Different approaches for vehicular network task management appeared in two distinct studies. The authors of Reference (Wang et al., 2020) sought to maximize vehicle benefits through delaying processing times in vehicular edge computing environments. A distributed algorithm allowed their system to reach optimal results by using game-based approach for task distribution. Research (Zhou et al., 2019b) created an allocation system which handles vehicular fog computing tasks. Through contract matching theory the mechanism sought to reduce delays by finding optimal task assignments.

2.2Energy-efficient scheduling

The schedule of vehicle tasks must analyze their energy requirements. The scheduling process must focus on minimizing the energy waste because vehicles consume energy during both communication operations and computational processes. The authors in (Qin et al., 2022) developed a strategy to minimize energy expenses in vehicle fog networks during the task offloading process. Their system transforms offloading tasks into three implementable sub tasks which include breaking workloads into smaller elements along with selecting tasks for offloading and finally designating resources as needed. A combination of greedy approach, matching theory and Lyapunov optimization serves as the three different solutions for these sub problems. The authors in (Zhou et al., 2019a) developed a resource allocation mechanism to decrease energy consumption among devices in vehicle edge network (user equipment). Through the alternating direction method of multipliers (ADMM) technology the mechanism enables its operations.

Using their previous work as base (Abdel-Basset et al., 2020) the researchers created a fog environment focused task scheduling system. The developed system focuses on reducing overall energy consumption across IoT-fog operation systems. They reach this objective through utilization of the marine predator's optimization algorithm as their special algorithm. (Zhai et al., 2020) developed an offloading system that functions in an SDN platform designed for vehicles (IoV). The proposed system design works to reduce power consumption levels in edge network devices. A systematic approach for lowering the energy intake of cars in vehicle fog environments was presented by (Liu et al., 2019b) through relocating computational operations to adjacent computing systems. A different scheduling method for industrial IoT tasks described in (Hazra et al., 2022) allows devices to achieve minimum energy efficiency. The technique depends on matching theory principles from economic science.

2.3Delay and Energy-Conscious Scheduling

The VFC network demands balancing speed of task execution to decrease delay against battery preservation to enhance energy efficiency. The limited power capability of vehicles along with urgent task requirements creates difficulties for simultaneously controlling these competing elements. The authors in (Raza et al., 2021) explored how to optimize efficiency levels in mobile edge computing (MEC) networks. The strategy incorporates mobile device properties to guide strategic task and resource distribution decisions. The authors developed computational efficiency improvements through game theory which regulates computation delay and energy consumption duration.

In reference (Luo et al., 2021) the authors present an approach for dispersing network computations within vehicular edge computing. Task assignment with particle swarm optimization detects the optimal timing and power draw relationship between delay and energy usage. The writers in (Tang et al., 2022) designed a new approach to task offloading in mobile edge networks. Using a greedy algorithm for minimization provides an efficient method which shortens response times throughout all industrial operations and delivers total speed improvement. Research scientists carried out detailed investigations regarding mobile edge computing offloading procedures (Chen et al., 2023). Researchers suggested a proposal which implements game theory strategies as a basis for sending tasks to edge servers. This technique minimizes service time as well as protects user device battery capacity.

2.4Mobility-Conscious Scheduling

The scheduling process for VFC networks requires consideration of vehicle movement between RSU server areas because it affects task execution efficiency. The approach enables better efficiency in task completion. The research conducted by (Lin et al., 2020) addresses the issue of fast task processing in vehicular fog computing networks. The system successfully operates through analyzing vehicle movement throughout the task assignment process. The general heuristic algorithm of their approach pairs tasks with nearby fog base stations based on vehicle movement to reach minimum total processing time. The flexibility of software-defined networks in vehicular fog computing (Misra and Bera, 2019) became the foundation for creating an offloading strategy which analyzes vehicle movement patterns. The strategy achieves computational delay reduction through an efficient method which is known as greedy heuristic. The authors of Reference (Lakhan et al., 2022) describe a task scheduling method for IoV that bases its operations on vehicle mobility factors. The approach embraces a general heuristic technique which simultaneously reduces both data transfer communication expenses and application running power requirements in vehicles.

The study approaches a new scheduling problem which aims to distribute tasks between vehicles operating in vehicular fog networks. The research takes into account both the period of time vehicles remains under RSU coverage (sojourn time) alongside their navigation paths between RSUs. Such considerations make scheduling tasks with both delay minimization and energy-saving considerations exceptionally complex. This research expands upon existing approaches because it analyzes both RSU range residency durations (sojourn time) and RSU-to-RSU vehicle movements. The scheduling process becomes more effective due to this approach. Scientists study machine learning and deep learning for server task allocation but (Yan et al., 2020, Xiong et al., 2020) explain these methods are both data-intensive and train slowly. We selected alternative approaches compared to deep learning because our main goal focuses on limited resource servers.

3.MODEL OF SYSTEM AND FORMULATION OF PROBLEM

The next part focuses on both key elements of our proposed system which include the problem formulation and the system model.

3.1Network Model

The scenario involves an urban area deployment of Road Side Units (RSUs) which is depicted in Fig. 1. The RSUs exist in strategic positions near roads of intelligent cities while receiving management from a central base station. Several RSUs function as roadside stations (RSUs) within an identified range from 1 to an unseen total number noted as RS , where $RS = \{1, 2, \ldots, rs, \ldots, RS\}$. The RSUs receive planned deployment to prevent their operational areas from intersecting with each other. The overall power capability of roadside stations (RSUs) depends on their CPU processing speed together with their memory storage capacity and their network interface connectivity performance (Ali et al., 2020). The central processing unit (CPU) and its processing capabilities of RSU are the only focus for this analysis. Each roadside station (RSU), identified by a unique ID (rs_id), has its own vehicular fog computing (VFC) server. This server's processing power is measured in cycles per second (fC_{rs}). In addition to the network, we're also considering applications designed for vehicles, known as Internet of Vehicles (IoV) applications. There's a set called VE that encompasses all these applications, with each application having its own unique identifier ranging from 1 to a total number represented by VE , where $VE = \{1, 2, \ldots, ve, \ldots, VE\}$. Imagine each vehicle only runs one application at a time. To keep things clear, we'll simply use ve to denote both the vehicular and its current application. These vehicles can then connect to roadside stations (RSUs) to access the services they need. Vehicular can connect to the nearest roadside station within their range. Each roadside units have a built-in computer (fog node) to handle vehicle requests. All these RSUs are then connected to a central hub (base station). We're using specific symbols (refer to Table 1 for details) to represent these elements for easier explanation. Vehicles submit application requests to the wireless station at predetermined intervals. The wireless station then collects and analyzes these requests. Considering the processing power of nearby roadside servers (RSU fog servers), the capabilities of each vehicle's computer (vehicle capacity), and how the vehicles are moving, the base station assigns applications task to either the vehicular itself or the most suitable roadside unit server along the vehicular route. Details about each vehicle's application are explained in section 3-2. Figure 1 shows that each vehicle runs a single application that's broken down into separate tasks. As shown in the figure, the blue vehicle $ve1$ has an application with ten separate tasks. It asks the base station to decide where to run these tasks. Based on $ve1$ movement, how long it will stay in range of each server (sojourn time), and the available communication and processing power, the base-station assigns 3-tasks to roadside unit $server_1$, 5-tasks to RSU $server_2$, and the remaining 2-tasks to roadside unit $server_3$.

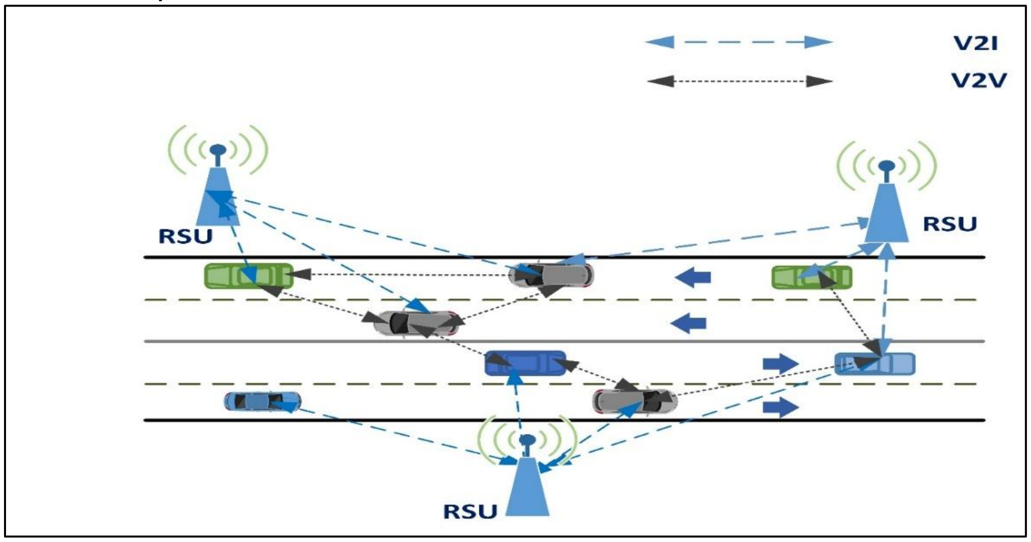


Figure 1: Fog enabled IoV Framework (Hou et al., 2020)

Table 1: A brief overview of symbols and abbreviations.

Symbol	Description
rs, RS	Contains set RSUs RS the RSU rs
ve, VE	Contains set VE the vehicle ve
RS	The overall count of RSUs
VE	Number of vehicles within the coverage area of the RSU
AP_{ve}	The collection of tasks for application ve
AP_{ve}^{le}	The collection of tasks to be performed in vehicle ve
AP_{ve}^{oe}	The collection of tasks to be carried out in roadside units (RSU)
SZ_{ve}	The applications size in bits
CY_{ve}	The total number of clock cycles needed for application ve
$dline_{ve}$	The time limit for application ve
$xe_{ve,ie}^{rs}$	An application variable is needed to represent the assignment of the ie^{th} task to server rs
BW_{ve}^{rs}	The amount of data that can be transferred between the vehicle ve and the RSU at rs
pw_{ve}^{rs}	The rate at which data is transmitted through the channel connecting vehicle ve and RSU rs
PO_{ve}^{rs}	The power used for transmission by vehicle ve
GA_{ve}^{rs}	The strength of the connection between RSU rs and vehicle ve
tt_{ve}^{me}	The amount of time it takes for the vehicle ve to send its application data AP_{ve} to a receiver
fC_{rs}	The computing capability of the rs^{th} RSU server
XE_{gk}	Transition state gk
TT_{gk}	The moment when transition state gk occurs
Δ_{ve}^{rs}	The duration that vehicle ve spends in the RSU rs area
PM_{ve}	The probability of transition in the Markov Chain related to vehicle ve
HB_{ve}^{rs}	The likelihood distribution of the duration that vehicle ve stays within RSU region rs
RE^{maxx}	The highest quantity of resources accessible within the VFC system
EG_{ve}^{avll}	The amount of energy present within vehicle ve

3.2Task Model

In section explains a model used to represent tasks within an Internet of Vehicles (IoV) applications. Each application (AP_{ve}) may be described as a set of three elements: data size (SZ_{ve}), required processing cycles (CY_{ve}), and deadline ($dline_{ve}$). These elements apply to all applications within the set VE . where $AP_{ve} = \{ SZ_{ve}, CY_{ve}, dline_{ve}\}, ve\ \epsilon\ VE$. Many applications can be broken down into individual tasks that can be assigned to different servers to work on at the same time. Imagine a self-driving car that needs to analyze multiple images. We can split these images into groups and process them all at once on separate computers (Liu et al., 2019b). The system decides whether to run each app task on the vehicle itself or on a nearby roadside server (RSU server) depending on how the vehicle is moving. The segmented tasks of application ve are represented as $AP_{ve} = \{AP_{ve,1}, AP_{ve,2}, \dots, AP_{ve,i}, \dots, AP_{ve,n}\}$, where ne signifies the no. of tasks of applications ve and the ie^{th} tasks of ve^{th} vehicular application may be expressed with $AP_{ve,ie}$. To control which task is done by the vehicles and roadside units (RSUs), we start by defining a variable called $xe^{rs}_{ve,ie}$.

$$xe^{rs}_{ve,ie} = \begin{cases} 1, \text{ if task } ie \text{ of the } ve^{th} \text{ application is allocated to } rs \\ \text{either vehicles processor or roadside unit server} \\ 0, \text{ otherwise} \end{cases} \quad (1)$$

Wherein rs in $x^{rs}_{ve,ie}$ specifies both the RSU servers and vehicle. This sentence explains how tasks are categorized based on where they are run. Tasks that stay on the vehicle due to the vehicle's movement are denoted by AP^{le}_{ve}. Tasks that are sent to roadside units (RSU servers) are denoted by AP^{oe}_{ve} i.e., $AP_{ve} = AP^{le}_{ve} \cup AP^{oe}_{ve}$.

3.3Mobility Model

Vehicles interact with an RSU by sending a request within a specific time slot and then move on to another the coverage area RSU. The MRP allows us to predict the vehicles movement patterns ($\{(XE_{gk}, TT_{gk})$: $gk >= 0\}$) To forecast the trajectory of IoVs and the duration they spend at RSUs, utilizing XE_{gk} to denote the gk^{th} transitional state and TT_{gk} for the gk^{th} transition time moment. The time a vehicle ve spends within the coverage area of a roadside unit RSU rs is called its dwell time. We can estimate this dwell time (denoted by Δ^{rs}_{ve}) by analyzing traffic data. The Markov chain's transition probability for vehicle ve is denoted as PM_{ve}. HB^{rs}_{ve} represents the duration distribution of the semi-Markov process indicating how long vehicle ve remains within RSU rs coverage before transitioning to another RSU.

1) Mobility Prediction: The Markov renewal process helps us understand how long a vehicle ve might stay in a specific state (RSU rs) during its travel period (TT). The process determines the unknown dwell time (sojourn duration) Δ^{rs}_{ve} of vehicles within RSUs based on research (Somesula et al., 2021, Hayder and Husain, 2018). Calculations based on the following mathematical formula yield an average estimation for vehicle sojourn time when in a state:

$$\mu^{rs}_{ve} = \int_0^{\infty} ye\ HB^{rs}_{ve}\text{(Tran and Pompili)}\ dy \quad (2)$$

The HB^{rs}_{ve} (Tran and Pompili) defines the probability for ve vehicles to remain in particular rs states for sojourn time periods. This statistical indicator determines the typical period μ^{rs}_{ve} which vehicles remain in state rs during their route between two RSUs.

The vehicle system operates with the ability to move in different traveling directions. The trajectory of vehicle ve along a route within a duration TT is denoted as ZE = $\{ze^1_{ve}, ze^2_{ve}, \dots, ze^{ke}_{ve}\}$, where ke >= 1. The initial condition of path ke is represented as $ze^{ke,0}_{ve}$ = ze^{ke}_{ve} , i.e., This implies that the initial state of path ke coincides with the initial condition of all paths associated with vehicle ve . Therefore, the ke^{th} path for vehicle ve is denoted as ze^{ke}_{ve} = $\{ze^{ke,0}_{ve}, ze^{ke,1}_{ve}, \dots, ze^{ke,be}_{ve}\}$, where be>0 or $be = 0$. The notation $ze^{ke,1}_{ve}$ specifically refers to the ke^{th} path of vehicle ve when it encounters the first roadside unit (RSU). It is noteworthy that a vehicular can traverse back to the same roadside unit zone multiple times within a single path. Consequently, the vehicles trajectory, denoted as ze_{ve}^{ke}, is represented as a multi set. The vehicular remains in the begin state to a duration of tt_{ve}^{oe} before initiating the first transition. Across all possible vehicle trajectories, the initial transition is anticipated to occur at the following specific time instant:

$$TT_{ve}^{ke,1} = \int_0^\infty y\, e\, HB_{ve}^{ze_{ve}^{ke,0}} \text{(Tran and Pompili)}\, dy \quad (3)$$

Given that $TT_{ve}^{ke,0}$ is zero and $TT_{ve}^{ke,1}$ maintains a constant value all paths of vehicle ve, the ke^{th} path je^{th} transition can be anticipated to occur as follows:

$$TT_{ve}^{ke,je} = TT_{ve}^{ke,je-1} + \mu_{ve}^{ze_{ve}^{ke,je-1}} \quad (4)$$

where $je <= be$. The definitive transition takes place at the conclusion of time instant TT.

The time spent by a task traversing the ke^{th} path ze_{ve}^{ke} is denoted by $\Delta_{ve}^{ke} = \Delta_{ve}^{ze_{ve}^{ke,0}}, \Delta_{ve}^{ze_{ve}^{ke,1}}, \ldots, \Delta_{ve}^{ze_{ve}^{ke,b}}\}$.

The sojourn time is $\Delta_{ve}^{z_{ve}^{ke,je}} = TT_{ve}^{ke,je} - TT_{ve}^{ke,je-1}$. Therefore, the sojourn time can be explained in detail as follows:

$$\Delta_{ve}^{ze_{ve}^{ke,je}} = \begin{cases} TT_{ve}^{ke,1}, & je = 0 \\ [\mu_{ze_{ve}^{ke,je}}^{rs}], & 1 <= je <= be - 1 \\ [TT - \sum_{we=0}^{be-1} \Delta_{ve}^{ze_{ve}^{ke,w}}], & je = be \end{cases} \quad (5)$$

The function $gk = [uf]$ is defined as follows: If uf < 0, then qe = 0 and if uf >= 0, then $qe = uf$.

The probability of vehicle ve selecting path ze_{ve}^{ke} is defined as:

$$PM_{ve}^{ke} = \prod_{je=0}^{be-1} PM_{ze_{ve}^{k,j}},\, ze_{ve}^{ke,je+1} \quad (6)$$

Consequently, the average time of stay of vehicular ve at roadside unit rs through the time interval TT is represented by:

$$\Delta_{ve}^{rs} = \sum_{ke\in KE} PM_{ve}^{ke} \quad \sum_{we=0}^{d} \Delta_{ve}^{ze_{ve}^{ke,je}} \quad 1_{(z_{ve}^{ke,we}=rs)} \quad ,\forall\, rs \in RS \quad (7)$$

3.4Model of Communication

The section focuses on model of communication employed for transmitting tasks from Vehicles (IoV) to the Roadside Unit (RSU) server. we posit these entities communicating wireless with each other. According to reference (Liu et al., 2020), we posit that the wireless channel between vehicles and roadside units utilizes the (OFDMA) mechanism for communication. The defined data transfer rate, denoted by ρw_{ve}^{rs}, represents the channel's transmission speed between vehicles ve and RSUs rs is detailed in the following manner:

$$\rho w_{ve}^{rs} = BW_{ve}^{rs} \log\left(1 + \frac{PM_{ve}^{rs} G_{ve}^{rs}}{N_0}\right) \quad (8)$$

Wherein BW_{ve}^{rs} signifies the communication ability of the link connecting the vehicle designated ve and the roadside unit located at rs, PO_{ve}^{rs} represents the strength of the radio signal transmitted by vehicle ve directed towards the RSU located at rs, and N_0 represents the background noise level. Hence, the following equation defines how long it takes for vehicle ve to transmit (offload) its task ie to the roadside unit (RSU) at rs:

$$tm_{ve,ie}^{rs} = \frac{SZ_{ve,ie}}{\rho w_{ve}^{rs}}, \forall\, ie \in AP_{ve}^{oe}, \forall\, rs \in RS \quad (9)$$

Where AP_{ve}^{oe} signifies the collection of tasks to be offloaded to RSUs. The notation $sz_{ve,ie}$ denotes the size of the ie^{th} task belonging to the ve^{th} applications, which encompasses both output and input data of the task. To simplify things, we're assuming the data transfer speed is the same in both directions (uplink and downlink) between the vehicle and the roadside unit. This lets us define the total time it takes for the vehicle ve to send its tasks to the RSU rs unit is defined as:

$$tm_{ve}^{rs} = \sum_{ie\in AP_{ve}^{oe}} \sum_{rs\in RS} x_{ve,ie}^{rs} * \frac{sz_{ve,ie}}{\rho w_{ve}^{rs}}, \forall\, ve \in VE \quad (10)$$

Hence, this is how we can calculate the energy used when the application's tasks are sent to the appropriate RSU server as follows:

$$EG_{ve}^{ce} = PO_{ve}^{rs} * \sum_{ie\in A_{ve}^{oe}} \sum_{rs\in RS} xe_{ve,ie}^{rs} * \frac{sz_{ve,ie}}{pw_{ve}^{rs}}, \forall\, ve \in VE, \quad (11)$$

Where AP_{ve}^{oe} signifies the collection of tasks of ve^{th} application which is assigned to roadside units. ρw_{ve}^{rs} denotes the transmission rate experienced by vehicle ve over the communication channel established with RSU rs. PO_{ve}^{rs} denotes the transmission power allocated to this channel.

3.5Computation Model

This section explains how applications for connected vehicles (IoV) are processed on roadside servers (RSU servers) in our system. It's important to note that these RSU servers have limited processing power and can only handle one task at a time.

1) Local Computing: Vehicles first send their application requests to a central hub (base station). The base station then figures out which tasks the vehicle can handle itself and which ones need to be sent to nearby roadside servers (RSU fog servers) for processing. Here's how long it takes the vehicle to execute task ie of applications ve that the base station assigned to it:

$$tt_{ve,ie}^{le} = \frac{ce_{ve,ie}}{fC_{ve}}\ , \forall\ ie \in ve \qquad (12)$$

Where $ce_{ve,ie}$ denotes the number of cycles required for its execution the ie^{th} task of ve^{th} application. Additionally, fC_{ve} denotes the computational capacity of the ve^{th} vehicle. A subset of tasks assigned to vehicle ve is processed locally within a vehicle instead of offloading to roadside units. The aggregate computation time required to perform these tasks locally in vehicles ve is determined as follows:

$$tt_{ve}^{le} = \sum_{ie\in AP_{ve}^{le}} \frac{c_{ve,i}}{fC_{ve}}, \forall\ ve \in VE \qquad (13)$$

2) RSU Computing: Vehicles relying on the Internet of Vehicles (IoV) can send their applications wirelessly to nearby RSU fog servers for processing. This decision depends on how fast the vehicles are moving and how long they'll be in range of each server. These fog servers, with their powerful processors, can handle the requests and send the outcomes back to the vehicle. The base-station figures out which tasks are best suited for these RSU servers along the vehicle's route. Here's how long it typically takes to complete all the tasks assigned to these RSU servers:

$$tt_{ve}^{\overline{ee}} = \sum_{ie\in AP_{ve}^{le}} \sum_{rs\in RS} (x_{ve,ie}^{rs} * \frac{c_{ve,ie}}{fC_{ve}})\ , \forall\ ve \in VE \qquad (14)$$

Where, $ce_{ve,ie}$ denotes the necessary no. of cycles to perform ie^{th} task of the ve^{th} applications, while fC_{rs} signifies the computational capability of the rs^{th} server

Hence, this sentence signals the definition of a formula that will determine the total amount of processing time required to handle the entire application AP_{ve} is defined as follows:

$$tt_{ve}^{ee} = Max(tt_{ve}^{le}, tt_{ve}^{\overline{ee}}) \qquad (15)$$

This sentence focuses on the energy use for running tasks on the vehicle's processor. The formula for this energy consumption is provided below:

$$EG_{ve}^{le} = \sum_{ie\in A_{ve}^{le}} ke\ fC_{ve}^{2} * c_{ve,ie}, \forall\ ve \in VE \qquad (16)$$

Where AP_{ve}^{le} signifies the collection of tasks of ve^{th} application and ke denote the coefficient contingent upon the capacity of the vehicle's onboard processor chip (Shen et al., 2022).

3.6Problem Formulation

The total time it takes to complete task i of application ve includes both the time it takes to send the data and the time it takes to process it. This total service time can be calculated as follows:

$$tt_{ve,ie} = tm_{ve,ie}^{rs} + tw_{ve,ie}^{rs} + tt_{ve,ie}^{ee} \qquad (17)$$

Accordingly, when task ie of vehicle ve is scheduled locally within the vehicle itself, the associated transmission time, denoted by $tm_{ve,ie}^{rs}$, is zero. Furthermore, the time of stay of vehicle ve in previous roadside units and waiting time of ie^{th} task of applications ve in the assigned server rs are collectively represented by $tw_{ve,ie}^{rs}$. The waiting time, in turn, is defined as the time difference between the task's arrival time and the commencement of its execution. Upon task ie assigned to vehicle ve, the task's waiting time excludes the sojourn time. The notation $tt_{ve,ie}^{ee}$ represents the execution time for task ie of application ve. The execution of $tt_{ve,ie}^{ee}$ can occur either within the confines of vehicle ve or within the Roadside Unit (RSU) designated as rs. The overall energy usage of the vehicle for completing the application's tasks is the combination of the transmission energy for the offloaded tasks and the computation energy for the local tasks, as defined below:

$$EG_{ve} = EG_{ve}^{ce} + EG_{ve}^{le}\ , \forall\ ve \in VE \qquad (18)$$

This system aims to find the best balance between two things: how long it takes to complete tasks for vehicles' applications (service delay) and how much energy the vehicles use (energy consumption). This balancing act needs to happen within a set time period and consider limitations on both processing power (delay) and available energy (resource constraints). We'll explore this trade-off between delay and energy, and here's how we define a combined measure of success (joint utility function):

$\mathcal{U}\mathrm{F}\,(ve,\,ie) = \frac{1}{\lambda^{tt} tt_{ve,ie} + \lambda^{EG} EG_{ve,ie}}$, $\forall\ ie \in ve$ (19)

The service time required to execute task ie for vehicle ve is denoted by $tt_{ve,ie}$. Similarly, $EG_{ve,ie}$ represents the energy consumption by vehicle ve when processing task ie. When the task is offloaded to Roadside Unit (RSU) rs, the energy expenditure encompasses both transmission energy, denoted by $EG_{ve,ie}^{ce}$, and the local processing energy $EG_{ve,ie}^{le}$ of vehicle ve is disregarded. Conversely, if the task is executed locally on the vehicle, only the local processing energy $EG_{ve,ie}^{le}$ is considered, and the transmission energy $EG_{ve,ie}^{ce}$ is zero. The parameter λ^{tt} serves as a weighting factor for the service time associated with each application, while λ^{EG} represents a weighting factor for the energy consumption incurred by each vehicle. The derivation of Equation 19 draws inspiration from the work presented in (Zhu et al., 2018), as per the suggested system model. The parameters λ^{tt} and λ^{EG} must assume values between 0 and 1, with the additional constraint that their sum equals unity. In accordance with user specifications, the values are constrained to fall within a range between zero and one. In contrast to conventional approaches, our proposed model assigns equal significance to both delay and energy considerations. This is reflected in the equal weighting of the λ^{tt} and λ^{EG} parameters, both set to 0.5. Consequently, the utility function (UF) encompassing every task performed by the vehicular may be formally expressed according to:

$UF = \sum_{ve\in VE} \sum_{i\in ve} \sum_{rs\,\in RS} \mathcal{U}F\,(ve, ie)$ (20)

Our goal is to find a balance between minimizing the time it takes to complete tasks and reducing the energy used by the vehicles. This balance is achieved by maximizing the following utility function:

$Max. UF$ (21)

$s.t.\ tt_{ve,ie} \leq dline_{ve,ie},\ \forall\ ie$ (22)

$\sum_{ie\in AP_{ve}^{oe}} x_{ve,ie}^{rs} * tt_{ve,ie} \leq \Delta_{ve}^{rs},\ \forall\ rs \in RS$ (23)

$\sum_{rs\,\in RS} x_{ve,ie}^{rs} \leq 1,\ \forall\ ve \in VE$ (24)

$\sum_{ve\,\in VE} \sum_{ie\in AP_{ve}^{oe}} x_{ve,i}^{rs} * fC_{rs} \leq RS^{maxx},\ \forall\ rs \in \mathrm{RS},\ \forall\ ve \in VE$ (25)

$EG_{ve}^{ce} + EG_{ve}^{le} \leq EG_{ve}^{avll},\ \forall\ ve \in \mathrm{VE}$ (26)

$|VE_{rs}| <= C_{rs},\ \forall\ rs\in RS$ (27)

$xe_{ve,ie}^{rs},\ ie \in \{0,\ 1\},\ \forall\ rs \in RS,\ \forall\ ve \in VE$ (28)

$\lambda^{tt} + \lambda^{EG} = 1$ (29)

The aim of the objective function is to assign tasks from each vehicle's applications to locally available resources or those in RSUs, with the goal of maximizing the utility function UF. In simpler terms, rule 22 says the vehicle must finish its assigned task before the deadline. Rule 23 ensures the vehicle completes all its requests before it needs to leave the area. Rule 24, each task needs to be assigned to just one processing unit, be it a vehicle or a roadside station (RSU). Rule 25, the workload assigned to an RSU cannot be greater than its processing capabilities. Rule 26, a vehicle's energy use for offloading tasks and running them on a RSU must be less than its total battery power. Rule 27, the number of vehicles connected to an RSU can't be higher than the number of communication channels it has available. Rule 28, this rule defines the key choices we need to make (likely about offloading tasks) and Rule 29, the importance of how long tasks take to run (service time) and how much energy the vehicle uses needs to be balanced is 1.

The equation 21 describes a task scheduling problem that is difficult to solve exactly. This is because it involves both integer and non-linear components, making it a mixed-integer non-linear programming (MINLP) problem. Indeed, it has been demonstrated to be NP-hard, implying that no efficient algorithm is known to solve it in all cases (Zhu et al., 2018). We define a function to measure how good a schedule is (objective function). This function is shown to have two important properties: it always increases when you add more tasks (monotone), and it satisfies a specific technical condition (submodular) explained in the next section. Finding the best possible schedule (maximization) can be tackled using a simple approach (greedy heuristic) that guarantees at least 63% of the optimal value (Xia et al., 2019). In the next section, we'll explain how to use a greedy approach to find a good solution for maximizing this submodular problem.

4.SUB MODULAR OPTIMIZATION

4.1Preliminaries

We'll start by introducing the fundamental ideas of submodular functions and matroids. These concepts are important for understanding what follows.

4.1.1 Sub Modular Function

Consider we have a set function, denoted by gk, that is defined on the elements belonging to the set NE, i.e., gk: $2^{\mathrm{NE}} \rightarrow$ RS. In order for function gk to be submodular, it must satisfy these conditions:

gk (XE ∪ YE) + gk(XE ∩ YE) ≤ gk(XE) + gk(Tran and Pompili), ∀XE, YE ⊆ NE (30)

gk (XE) ≤ gk (Tran and Pompili), ∀XE ⊆ YE ⊆ NE (31)

Let $gk(XE|je) = gk(XE \cup \{je\}) - gk(XE)$ be the marginal value gk regarding the element je. Additionally, it's stated that gk is monotonic if gk (XE) ≤ gk (Tran and Pompili), for all $XE \subseteq YE \subseteq NE$.

4.1.2Matroid

A matroid is characterized as (NE, IE), where IE is a subset of 2^{NE}, representing a collection of subsets of NE. ME is considered a matroid when it fulfills the subsequent conditions.

- $IE \neq \phi$
- If $XE \subseteq YE, YE \in IE$, then $XE \in IE$
- If $XE, YE \in IE$, and $|XE| < |YE|$, then there exists e ∈ $YE \backslash XE$, such that $XE \cup F\{ee\} \in IE$

We're starting by defining a set called NE. This set will contain all the tasks that can be allocated to the servers RSU, likes:

NE = {xe_1^0,1, . . ., xe_{VE}^0, ne, xe_1^1,1, . . ., xe_{VE}^1, ne, xe_1^{RS},1, . . ., xe_{VE}^{RS}, ne}

Where $xe_{ve,ie}^{rs}$ indicates that task ie belonging to vehicle ve is allocated to the server RSU rs. signifies that task ie belonging to vehicle ve is processed on the vehicle's own computer unit (local server) rather than being sent to an RSU server. All the tasks NE can be grouped into separate collections into VE′ = VE ∗ RS disjoint sets i.e., NE = $\cup_{pe=1}^{\mathrm{VE}\prime} NE_{pe}$, and $NE_{pe} \cap NE_{pe'}$ = ϕ, for any $pe \neq pe'$, where NE_{pe} = {$xe_{ve,1}^{rs}$, ...$xe_{ve,ne}^{rs}$} with $pe = \sum_{rs'=1}^{rs-1}(\mathrm{rs}' * \mathrm{VE}) + ve$. Let $NE_{pe,ie} = x_{ve,ie}^{rs}$.

We also define a matroid ME = (NE, IE), where NE is the ground set, and IE ⊆ 2NE is the independent sets of NE, i.e.,

$$\begin{aligned}
&\mathrm{IE} = \{\mathrm{XE} \subseteq \mathrm{NE}: |\mathrm{XE} \cap NE_{pe,ie}| \leq 1,\ pe = 1,2,\dots,NE,\\
&tt\,(\mathrm{XE} \cap NE_{pe,ie}) \leq dline_{ve,ie},\\
&tt\,(\mathrm{XE} \cap NE_{pe}) \leq \Delta_{ve}^{rs},\\
&|\mathrm{XE} \cap \mathrm{NE}\,| \leq RS^{maxx}, \qquad (32)\\
&\mathrm{EG}\,(\mathrm{XE} \cap NE_{pe}) \leq EG_{ve}^{avll},\\
&\sum_{pe=1}^{VE\prime} |\mathrm{XE} \cap NE_{pe,ie}| \leq \mathrm{Crs}, \forall \mathrm{rs}\}
\end{aligned}$$

Where $XE \cap NE_{pe,ie}$ indicates the server assigned to a specific task ie belonging to a particular vehicle ve. Where tt ($XE \cap NE_{pe}$) signifies the total time taken to service the tasks assigned to a specific vehicle ve at a particular roadside unit RSU rs and $EG\,(XE \cap NE_{pe})$ signifies the amount of energy the vehicle ve uses to complete a specific task ie.

If we replace the original limitations in problem 21 with the rules of a matroid, $ME = (NE, IE)$, we can express the goal (fitness function) as maximizing a function with certain properties (monotone submodular) while considering limitations (matroid constraints). Here's the specific formulation:

$$\max_{XE \in 2^{NE}} \mathbf{UF}(XE) \qquad (33)$$

$$s.t.\ \mathrm{XE} \in \mathrm{IE} \qquad (34)$$

The upcoming section explains that a greedy approach is the most effective way to maximize a function that meets certain criteria (monotone submodular) while adhering to specific restrictions (matroid constraints).

4.2Mobility-Aware Task Scheduling Optimization via a Greedy Algorithm

Equation 33 describes a task scheduling problem. We can find a pretty approximation solution to this problem efficiently using a simple method called the greedy algorithm. While greedy algorithms are not guaranteed to produce globally optimal solutions in general, they are highly efficient and perform well for problems where the objective function is monotone submodular and the constraints form a matroid, as in our case (Xia et al., 2019). This method guarantees to find a solution that's at least a certain percentage $\mathbf{(}1 - \frac{1}{ee}\mathbf{)}$ of the optimal solution, even in the worst-case. Because a greedy algorithm is efficient, it takes a reasonable amount of time to compute the answer. The outlined steps of a greedy strategy are outlined in Algo. (1). At the first stage, $\overline{NE}_{pe}$ is configured for every $pe = 1, 2, \ldots, VE'$. $\overline{NE}$ is specified in the line 2. Lines 5-7 implement an iterative greedy approach. In each iteration, the elements are selected from the ground set YE and added to the solution set XE. This process continues until the stopping criteria are met: either the size of set XE reaches a predefined limit (number of servers) or there are no elements in YE that offer any improvement (marginal gain becomes zero).

However, the greedy approach works well for smaller problems, but as vehicles and no. of tasks grow in a single time period (time slot), it becomes difficult to use. That's why for large-scale scheduling problems in vast networks, we need a more powerful scheduling task algorithm.

Algorithm 1: Algorithm for Task Scheduling Based on Greedy Strategy

```
Input: Collection of tasks $TT$, collection of Vehicles $VE$, time of stay Δ for all vehicles.
Output: $XE$: Schedule list
1: Initializing $\overline{NE}_{pe}$ = ϕ, ne = 1, 2, . . ., VE′
2: $\overline{NE}$ = $\bigcup_{pe=1}^{VE'} \overline{NE}_{pe}$
3: XE = $\overline{NE}$ , YE = NE
4: Reiterate
5: Select the item $ee\hat{} = x_{ve,ie}^{rs}$, i.e., $ee\hat{} = argmax_{ee' \in Y}$ UF $(XE \mid ee')$
6: XE = XE ∪ $\{ee\hat{}\}$
7: YE = YE − $\{ee\hat{}\}$
8: until |XE| ≤ $RS^{maxx}$ or UF $(XE|ee') \neq 0$
9: return XE
```

Based on what we've discussed so far, we're proposing a new approach for scheduling tasks that combines different optimization techniques. We'll explain this hybrid approach in detail in the next section.

5.MFO-PSO TASK SCHEDULING ALGORITHM

While there's no guarantee of finding the absolute best solution, studies show that meta-heuristic optimization algorithms are great at tackling tough problems (NP-hard) because they're fast, easy to implement, and adaptable to different situations (Abd Elaziz et al., 2021, Abdel-Basset et al., 2020). However, metaheuristic algorithms attract popularity to solve urgent real-scale challenges but they embody specific drawbacks. The premature convergence issue affecting particle swarm optimization (PSO) traps it inside the optimal local area within complex search domains (Wang et al., 2018). Conversely, Moth-Flame Optimization (MFO) includes an integrated feature which helps escaping from local optima to reach the truly optimal solution (global optimum). The metropolis acceptance rule enables achievement of this objective. MFO escapes local traps but its search for the best solution occurs at a slow pace (Liu et al., 2021a). The proposed optimization algorithm-unites PSO and MFO features to produce effective task scheduling solutions.

5.1Particle Swarm Optimization

The population in PSO continually modifies its positions and velocities through self-best-tracking combined with neighbor-sourced tracking data (Wang et al., 2018). In PSO, the number of particles searching for the best solution is represented by SF. Each particle, identified by an index ie $(ie = 1, 2, \ldots, |SF|)$, has a current position x_i and a speed ve_i at influences how it moves. Each particle in PSO keeps track of a solution for the task scheduling problem. This solution considers the number of tasks assigned to each vehicle at a specific point in time ($VE * ne$ elements), and the order in which vehicles tackle these tasks. Imagine a scenario in PSO where we have five tasks to schedule. One possible solution, represented by the list [5, 4, 0, 1, 2], dictates which task goes to which server. Here, the number represents the server assigned to the task. For instance, the first task goes to server 5, the second to server 4, and so on. A value of 0 means the task is assigned to the vehicle. The velocity of a particle tells us how much the assignments of tasks to different RSU servers will change. For instance, imagine a velocity vector is [one, zero, one, two, three]. This means that during a process that involves repetition, the first task is planned to be moved to a different RSU server with Id 6. This calculation is done by adding 1 to the current assignment (5) and then taking the remainder when divided by the total no. of servers (m). The locations and speeds of these particles are constantly modified using certain formulas to help them find the best solution.

$$ve_{ie+1} = \alpha v_{ie} + c_1 rs_1 (xe^{le}_{ie} - xe_{ie}) + c_2 rs_2 (xe^{gk}_{ie} - xe_{ie}) \quad (35)$$

$$xe_{ie+1} = xe_{ie} + ve_{ie+1} \quad (36)$$

Algorithm 2: Algorithm for Task Scheduling (MFO-PSO)

Input: collection of tasks TT, collection of vehicles VE, time of stay Δ for all vehicle

Output: XE: Schedule list

```
1: Initializing the parameters of PSO $ce1$, $ce2$, $rs1$, $rs2$, and inertia $\alpha$
2: Initializing the no. of iterations $max_{itr}$
3: Initializing the parameters of MFO rate of cooling $tempp$ and the temperature $tt$
4: for each $ie$ = 1 to SF do
5:        Initializing the particle of velocity $ve_{ie}$ and position $xe_{ie}$
6: end for
7: while $itr < max_{itr}$ do
8:        for each $ie$ = 1 to SF do
9:        9:     Calculation of objective value $F_{ie}$ of the $ie^{th}$ particle by Equation (21)
10:       end for
11:       Modify set of global location population $xe^{gk}$
12:       for each $ie$ = 1 to SF do
13:       Apply Equation (35) to update the new particle velocity $ve_{ie+1}$
14:       Apply Equation (36) to update the new particle position $xe_{ie}$
15:       if $F_{ie+1} \geq F_{ie}$ then
16:       Apply the new modify $xe_{ie+1}$
17:       else
18:       MFO ($x_{ie+1}$, $ve_{ie+1}$, $F_{ie+1}$, $tt$,) using Algo. (3)
19:       end if
20:       end for
21:       Modify the temperature $tt \leftarrow tempp * tt$
22:       $iter \leftarrow iter + 1$
23: end while
24: $XE \leftarrow xe^{gk}$
25: return $XE$
```

Where, ce_1 and ce_2 are constant values. rs_1 and rs_2 represent random numbers generated for the calculation. xe^{le}_{ie} refers to the particle's personal best position it has found so far, and xe^{gk}_{ie} denotes the best position discovered by the entire group (global best).

In order to address the large-scale task scheduling problem in dynamic vehicular fog, a balance between exploration and exploitation is necessary. The novelty of our hybrid MFO-PSO algorithm is in its manner of achieving this combination: by taking advantage of the exploration ability of MFO, where the system is capable of avoiding premature convergence, and the exploitation power of PSO, where rapid convergence to good positions is encouraged. The exploration phase driven by MFO expands the search space, helping the system to avoid local optima, while the exploitation phase of PSO fine-tunes the solutions in identified promising regions. This synergy guarantees the algorithm to be diversified and convergent at the same time, achieving a trade-off degree with that is very desired for real-time and energy-efficient vehicular task scheduling.

5.2Moth-flame Optimization

The MFO algorithm avoids getting stuck on subpar solutions (local optima) by considering situations that make things worse (worsen the fitness value). This is achieved through a built-in mechanism called a jumping mechanism. As part of the iterative process, the algorithm creates a new potential solution and then calculates how much better or worse it is compared to the previous solution. The new state is guaranteed acceptance only if it's an improvement. Otherwise, the chance of accepting it depends on a probability based on the amount of improvement/decline and a temperature parameter ($ee^{-difference/tt}$), we only automatically accept improvements to the solution ($difference > 0$). Otherwise, there's a chance we might accept a slightly worse option, depending on how much worse it is ($difference$) and a factor that controls how likely we are to accept less beneficial changes, called temperature tt. We chose a hybrid approach, combining PSO and MFO for our task scheduling algorithm. PSO excels at finding efficient solutions quickly, while MFO's strength lies in escaping local optimasituations where the algorithm gets stuck on a subpar solution. This combination aims to achieve both efficient exploration of the solution space and avoidance of getting trapped in suboptimal solutions.

Algorithm 3: Algorithm of Moth-flame Optimization (MFO)

```
Input: Particle Modified velocity $ve_{ie}$ and position $xe_{ie}$, Temperature $tt$
1: $difference = F_{ie+1} - F_{ie}$
2: Create a random decimal no. $rs$ between zero and one.
3: if $rs < ee^{-difference/tt}$ then
4:     Update the particle's velocity $ve_{ie+1}$ and position to $xe_{ie+1}$
5: else
6:     Maintain the particles current velocity $ve_{ie}$ and position $xe_{ie}$ .
7: end if
```

We present a designed hybrid algorithm for scheduling task problem that considers mobility, details in Algorithm 2. The Algorithm receives two inputs: a set of tasks denoted by TT and a set of RSUs represented by RS. It then calculates a scheduling plan, XE, as its output. The first lines (1-7) of the algorithm set up important variables for both the PSO and MFO algorithms. For PSO, this includes how many particles are in the swarm, where they are initially positioned, and how fast they move. MFO uses temperature and how it cools down as part of its optimization process, and these are also set up at the beginning. The algorithm repeats a loop for a specific number of times (defined in line 8). Within this loop (lines 9-12), it evaluates the performance of each potential solution (particle) in the swarm. It also tracks the best solution found global best xe_{ie}^{gk} and updates it if a better particle is encountered. The algorithm iterates through each possible solution (particle) in the swarm. For each particle, it updates its movement position and velocity in lines 14-15. If this update improves particle's performance compared to before, the algorithm adopts the new location (lines 16-17). In other words, the algorithm keeps track of promising areas of the search space explored by the particles. Otherwise, the algorithm adopts the MFO technique (detailed in Algorithm 3). This technique leverages the metropolis admission rule to modify or maintain the particles velocity and position (as described in lines 2-6 of Algo. (3)). Algo. (2) keeps looping until a specific condition is met. Inside the loop, the temperature is adjusted in each round. Once finished, the algorithm outputs the best overall schedule found among all the particles.

6.PERFORMANCE EVALUATION

This part evaluates the effectiveness of the suggested task scheduling algorithm that considers vehicle movement (mobility-aware). It details the simulation setup, performance measures used, and algorithms used for comparison.

6.1Simulation Settings

We tested how well the MFO-PSO algorithm performs by running it in a $Jupyter$ Notebook using Python. The Table 2 lists the settings used in our simulation. We designed a simulation of vehicular fog networks which had one base station and ranging numbers of roadside units (RSUs) - from 3 to 15 RSUs with one base station. Our simulation included 10 to 50 on-board processors among the vehicles which operate in the network. A simulation of vehicular fog computing network was created through random placement of roadside units (RSUs) within the network area. RSUs link to a single base station known as the BS. The simulation parameters adopt numerical values noted in (Shen et al., 2022, Tran and Pompili, 2018). The transmission power of each roadside unit (RSU) stands at 50 dBm. RSUs were assumed to operate with data transmission bandwidth between 50 to 100 megahertz. Each roadside unit (RSU) installs a CPU which functions at a speed varying between 10 and 15 gigahertz. Every vehicle within the network needed onboard processors with CPU frequencies between 2 to 5 gigahertz. Simulation tasks have been assumed to be of moderate size between 0.5 to 5 megabytes throughout the simulation. Table 2 contains all the information regarding parameters that were implemented in our MFO-PSO algorithm. The MFO-PSO algorithm simulation operates with a population of 50. The other parameters specific to PSO (rs_1, rs_2, ce_1, and ce_2 are set based on reference (Li et al., 2022) and tested with specific values: 2.5, 2.5, 2.1, and 2.1 respectively. Additionally, for each particle in the swarm, the inertia weight takes on different values between 0.6 and 1.0. For the MFO portion of the MFO-PSO algorithm, we set the cooling rate parameter ($tempp$) to gradually decrease between 0.96 and 1.0. The algorithm starts with a high initial temperature of 1000 (as suggested in (Liu et al., 2021a)) and runs for 1000 iterations to find optimal solutions. To ensure their proposed mechanisms were reliable, the authors tested them using existing, publicly available code from related research (papers (Somesula et al., 2021, Poularakis and Tassiulas, 2016)). They adapted this code to work within the framework of their proposed network model. To assess the proposed algorithm's reliability, we ran multiple simulations (50 repetitions) within the simulation environment. Simulations featured an average performance evaluation for the reported results. Within one standardized simulation our new algorithm underwent comparison testing with dominant established algorithms.

Table 2: Parameters for simulation.

Parameters	**Values**
No. of vehicles	20-500
No. of RSUs	10-50
Vehicle's CPU frequency	2-5 GHz
Mobility model	MRP
Average data size of the task	0.5-5 MB
CPU frequency of the Fog server	10-15 GHz
Transmission power of RSU server	50 dbM
Necessary computing size for the task (10^9 cycles)	0.5-5
Permissible task deadline	50-200 ms
RSU server bandwidth	50 MHz
MFO-PSO Parameters	**Values**
inertia (α)	0.6-1.0
$tempp$	0.8
ce_1	2.1
ce_2	2.1
rs_1	2.1-2.5
rs_2	2.1-2.3
Population size	50
Iterations	1000

6.2Benchmark Algorithms

The proposed paper presents MFO-PSO as a novel task scheduling algorithm. Our research compares MFO-PSO to six well-known scheduling algorithms which include: MATO (Liu et al., 2019a), MACTER (Raza et al., 2021), ORADRL (Yan et al., 2020), RADQN (Xiong et al., 2020), and RSU scheduling as well as random scheduling. Below are the descriptions of each method.

1) **MATO (Liu et al., 2019a):** selects the optimal task offloading destination from nearby fog servers which includes RSUs and other vehicles by considering mobility constraints for the vehicle. Range-based vehicles using this system can move inside the zone that one RSU oversees.
2) **MACTER (Raza et al., 2021):** addresses task offloading from vehicles by managing operations within networks which employ VFC along with fog servers and cloud resources. The system applies game theory so it decides where to place computational resources between vehicle processors and RSUs allowing offloading either internal or external processing. A drawback of this method exists because it ignores the actual movement patterns which vehicles might exhibit.
3) **ORADRL (Yan et al., 2020):** implements deep reinforcement learning (DRL) for server task assignment yet disregards user vehicle movements. The DRL system takes its decisions without factoring in future positions of devices.
4) **RADQN (Xiong et al., 2020):** This system allocates resources at the network edge to handle tasks from vehicles. It uses a deep learning technique called DQN to make these decisions. However, unlike some approaches, it doesn't take into account how the vehicles will be moving around (mobility) when assigning tasks.
5) **RSU Scheduling:** This strategy for managing tasks in a network for connected vehicles sends all tasks to RSU along a vehicle's path, instead of letting the vehicles handle them themselves.
6) **Random Scheduling:** Vehicles are assigned tasks at random, and these tasks can be sent for processing to either a nearby RSU or another vehicle.

6.3Performance Metrics

The effectiveness of the proposed approach is measured against existing solutions using the following performance metrics.

a) ***Service Latency***: This total vehicle task latency is calculated using Eqn. 17. It includes both local execution time (service latency) and remote execution time (communication latency and RSU server computation time) depending on where the task is processed.

b) ***Energy Consumption**:* We can determine the vehicle's energy usage for all its onboard computations using Eqn. 18.

c) ***Quality of service**:* We evaluate the system's performance (QoS) by considering both how long tasks take to complete (service delay) and how much energy the vehicles use (Eqn. 21). This allows us to find a balance between low energy consumption and fast task completion, ultimately reflecting how well the system meets the needs of vehicle users.

d) ***Success Rate**:* The success rate reflects what portion of tasks at the base station are completed on time. It's calculated as the number of tasks finished before their deadline divided by the total number of tasks.

6.4Mobility Model

To test our new scheduling method that considers how vehicles move around, we used real-time data on vehicle movements in San Francisco. This data, which is publicly available from Crawdad (Husain et al., 2024a), includes the locations of 500 vehicles in the San Francisco Bay Area, capturing their locations for a duration of 30 days. The simulation involved 50 vehicles while deploying 15 carefully located Roadside Units (RSUs) across the roads. The RSUs have been placed according to their X and Y coordinate positions. In our work, a base station forecasts how long each vehicle will dwell at each RSU and assigns jobs to both considering how the vehicles move around.

6.5Results and Discussion

The following section explores the performance of our proposed algorithm compared to established methods. We tested the system under three different conditions: how it handles changes in (1) tasks assigned, (2) number of roadside units (RSU), and (3) number of vehicles.

1) Effect of No. of Tasks: This section examines how well our MFO-PSO task scheduling method performs when the number of tasks changes. We increased the tasks from 50 to 250, while keeping the no. of roadside units (RSUs) fixed at 10. The results are shown in Fig. 2. Fig. 2a illustrates that as the task count increases, so does the service delay for all algorithms. This is because the RSU servers become overloaded. However, the proposed MFO-PSO method consistently experiences lower service delays compared to other approaches. The new scheduling method considers how vehicles move and how long they stay at different locations.

By taking these factors into account when assigning tasks to roadside servers, it can significantly decrease the time it takes to complete those tasks. Our MFO-PSO method significantly outperforms existing algorithms in terms of service time reduction. Compared to Random, RSU, ORADRL, RADQN, MACTER and MATO, it achieves improvements of up to 61.8%, 56%, 48%, 41%, 33%, and 19.2%, respectively. However, it's important to note that as no. of tasks increases in our networks, the overall energy usage of vehicles also rises (as illustrated in Figure 2b). While the overall energy usage for all vehicle increases as the number of tasks grows (mentioned previously in Fig. 2b), it's important to note that our proposed MFO-PSO algorithm prioritizes minimizing energy consumption as part of its design. This translates to lower energy usage compared to existing methods. In fact, MFO-PSO shows significant improvements of up to 66.4%, 69.6%, 57%, 62.8%, 16% and 51.6% compared to Random, RSU, ORADRL, RADQN, MACTER and MATO, respectively.

Fig. 2c shows the impact of increasing tasks on the quality of service (QoS) in our vehicular fog computing (VFC) network. As the number of tasks grows, the QoS unfortunately tends to decrease. The algorithm being proposed achieves superior performance other approaches in terms of Quality of Service (QoS) because its function prioritizes minimizing both energy usage and service time of the vehicles. Fig. 2d shows that as the number of tasks increases, the success rate for all algorithms goes down. However, the algorithm being proposed consistently achieves a higher success rate compared with the others. The suggested algorithm achieves a high success rate because including task service time in its objective function prioritizes completing tasks on time.

2) Effect of No. of Vehicles: This subsection explores how the MFO-PSO task scheduling algorithm performs when the number of vehicles changes. We test it with a fixed number of 10 RSUs (Roadside Units) but vary the vehicle count from 10 to 50. The results are visualized in Figure 3. The data in Fig. 3a shows that adding more vehicles to the system increases the time it takes to complete tasks. This makes sense because, as we already mentioned, vehicles can have varying workloads. More vehicles on a road mean more work for the roadside servers (RSUs) that handle tasks. Since these servers have limited processing power, the service time for each task increases with more vehicles. The MFO-PSO approach significantly reduces task service time compared to other algorithms. This is because it intelligently assigns tasks to roadside servers (RSUs) considering both how vehicles move and how long they stay within range of an RSU. This optimized distribution minimizes task processing time. The MFO-PSO algorithm significantly outperforms existing methods in reducing task service time. Compared to traditional approaches like RSU, Random, and MATO, MFO-PSO achieves improvements of up to 44.2%. It also demonstrates substantial enhancements over more recent algorithms like RADQN, ORADRL, and MACTER, with gains of up to 38.4%. Fig. 3b confirms the expected rise in overall energy consumption with more vehicles. However, the proposed MFO-PSO algorithm tackles this challenge. By prioritizing tasks based on a vehicle's available energy, it significantly reduces individual vehicle energy consumption. Moreover, the MFO-PSO algorithm achieves significant improvements in reducing energy consumption compared to existing methods. It outperforms approaches like RSU, Random, and MATO by up to 69.4%. MFO-PSO outperforms RADQN, ORADRL, and MACTER by achieving remarkable breakdown of up to 64.2%.

The network Quality of Service (QoS) displays modifications because of heightened vehicle numbers as shown in Fig. 3c. As the no. of vehicles in the network (VFC) grows, QoS generally declines. However, the proposed algorithm stands out by achieving consistently high QoS compared to other approaches. Unlike traditional scheduling methods, the proposed algorithm prioritizes both minimizing service delays and reducing vehicle energy consumption. This is achieved by considering each vehicle's mobility patterns. As a result, it efficiently improves the overall Quality of Service (QoS). While Fig. 3d reveals a general drop in task success rates with more vehicles, the proposed algorithm bucks this trend. It consistently achieves higher success rates compared to other scheduling mechanisms, even under varying vehicle loads. The MFO-PSO algorithm prioritizes tasks with stricter deadlines by factoring service delay into its decision-making process (utility function). This ensures tasks are allocated according to their urgency, leading to the proposed method's remarkably excellent success rate.

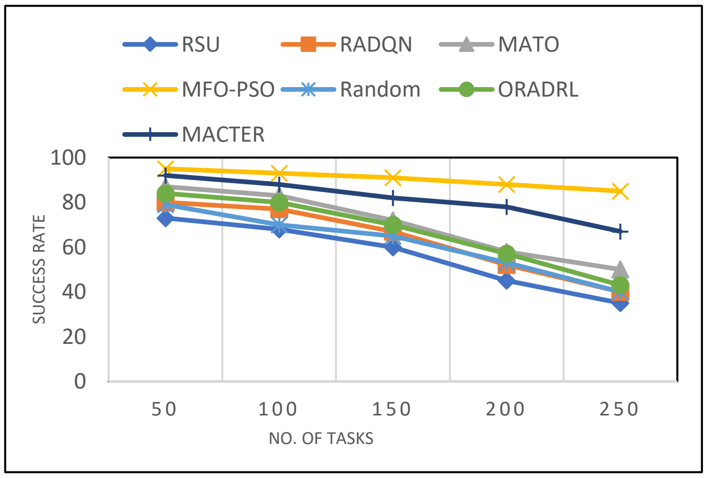
3) Effect of No. of RSUs: This section explores the performance of our MFO-PSO scheduling algorithm with a fixed number of vehicles (30) and tasks (150) but varying numbers of roadside servers (RSUs) (from 3 to 15). The outcomes are illustrated in Fig. 4. Fig. 4a shows a clear trend, as the number of roadside servers (RSUs) increases, task service time decreases for all algorithms. However, the proposed MFO-PSO consistently achieves the lowest service time compared to others. The MFO-PSO algorithm excels at minimizing task service time. It achieves this by strategically assigning tasks to roadside servers (RSUs) based on two crucial factors: vehicle mobility and how long a vehicle stays within range of an RSU (sojourn time). While Fig. 4b confirms that adding more roadside servers (RSUs) reduces overall energy consumption, the MFO-PSO algorithm takes further advantage of this trend. It outperforms existing methods by up to 32.2% in minimizing service time (as shown in Figure 4a). This is achieved by strategically assigning tasks to RSUs based on vehicle movement and dwell time. Compared to approaches like RSU, Random, and MATO, MFO-PSO demonstrates significant improvements in both service time and energy efficiency. The proposed algorithm achieves lower energy consumption for vehicles, as this was our primary objective. Compared to existing algorithms, MFO-PSO significantly reduces energy consumption by 61.6%, 66.4%, 50.4%, 57%, 16% and 41.4%, for Random, RSU, ORADRL, RADQN, MACTER and MATO, respectively.

The authors test their new algorithm by looking at how well it delivers service, as measured in Fig. 4c. The quality of service gets better as they add more RSUs to the network. The proposed algorithm stands out for its high quality of service (QoS) compared to others. This is because it considers both minimizing service times and reducing energy consumption for the vehicles, which are two key factors for good service. Fig. 4d shows that the more Radio Service Units (RSUs) there are in the network, the more tasks are completed successfully. The suggested MFO-PSO algorithm achieves a particularly high success rate compared to others. This is because it prioritizes assigning tasks in a way that minimizes service time while still meeting deadlines.

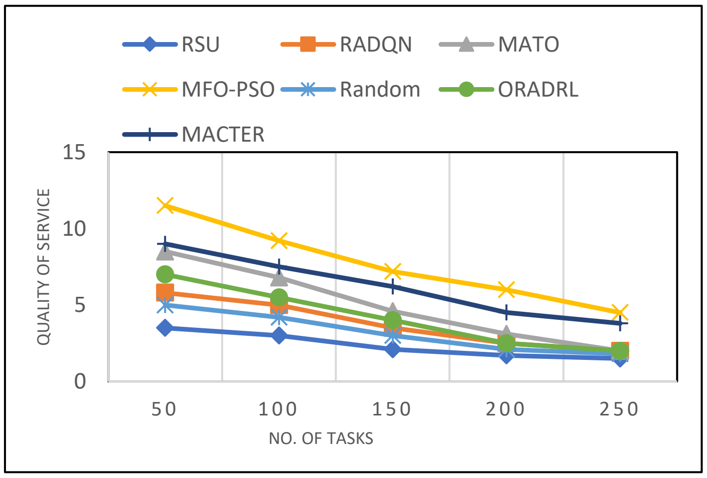
By using MFO-PSO users can obtain an innovative method for task scheduling. The optimization solution evaluates three essential elements: vehicle mobility patterns alongside resource presence elements as well as task completion deadlines. MACTER and MATO recognize only achievable tasks applying present resources in a single RSU region whereas new approaches do not share this limitation. A task assignment will fail because these algorithms do not support vehicle movement when vehicles operate outside RSU areas after receiving their assignments. RADQN and ORADRL function exclusively with tasks which remain within a single RSU space since they don't take into account vehicle movement. The result becomes failed tasks coupled with delays combined with wasted energy because of this method.

With the MFO-PSO method the researchers address the problems caused by vehicle movement and available resources beyond singular RSU boundaries. This lets it assign tasks more effectively, reducing both the time it takes to complete tasks and the energy used by the vehicles compared to older methods. Overall, the proposed algorithm gives better task allocation and makes the entire network run smoother.

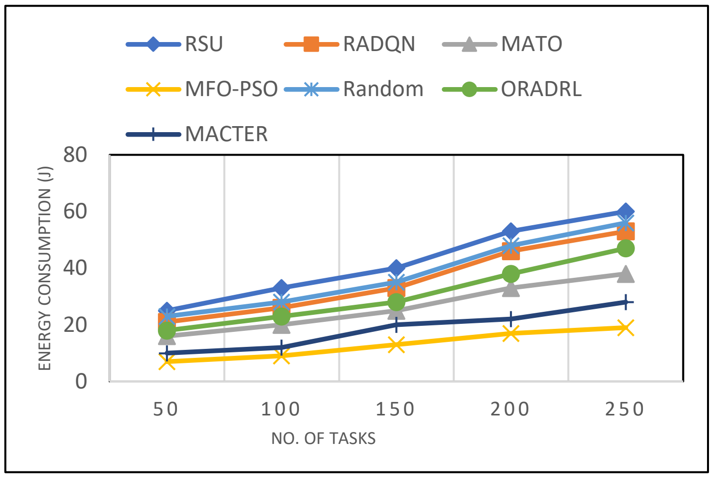
To facilitate a clearer comparison of the proposed algorithm with existing benchmarks, Table 3 presents a consolidated view of the performance metrics evaluated in Figures 2 through 4. The results confirm that the proposed MFO-PSO consistently outperforms other baseline algorithms across varying numbers of tasks, vehicles, and RSUs. Notably, MFO-PSO achieves higher success rates and QoS, while significantly reducing energy consumption and service latency, making it the most efficient scheduling approach for vehicular fog computing environments.



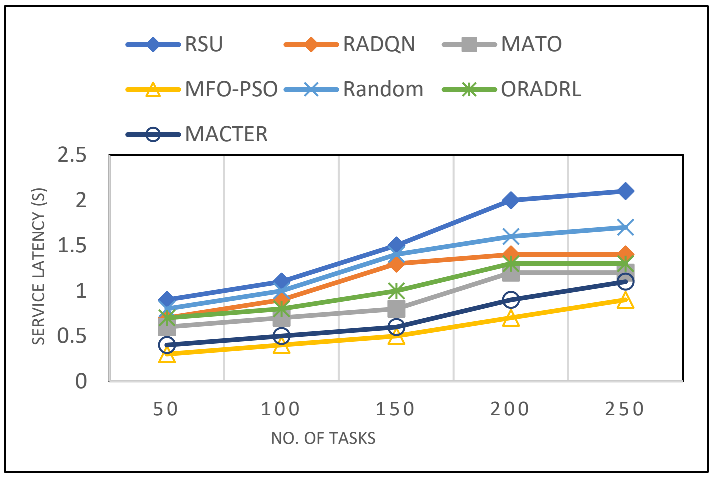
(a)



(b)

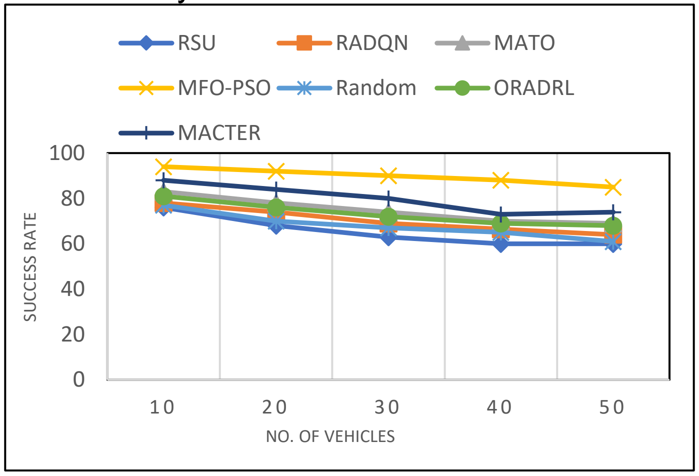


(c)

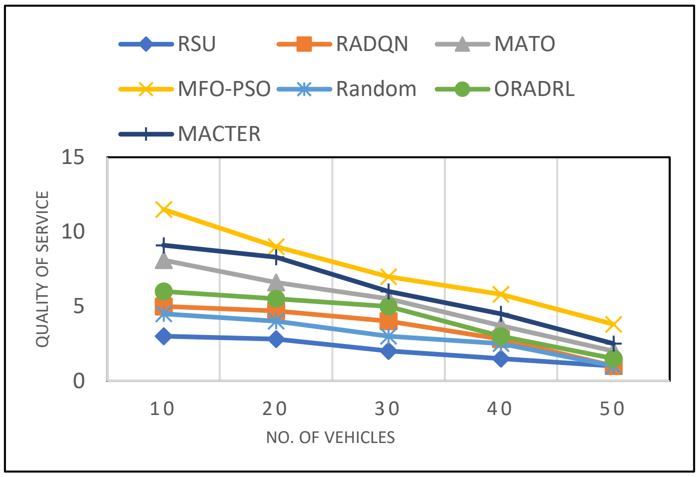


(d)

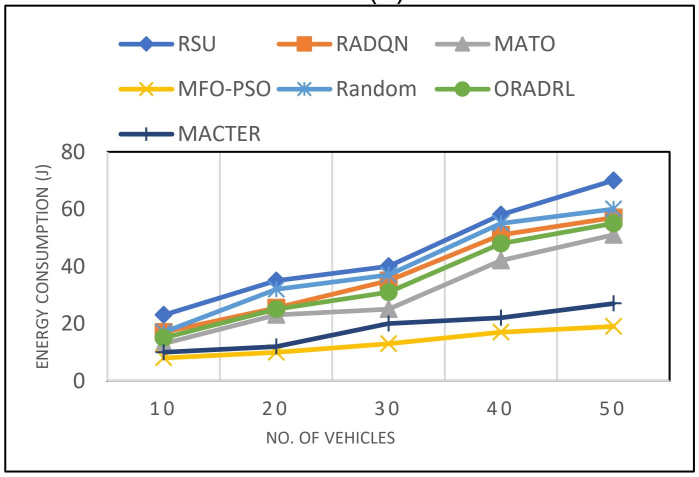
Figure 2: Compare the performing of suggested MFO-PSO algorithms with other exist algorithms concerning Number of Tasks in relation to (a) Success Rate, (b) QoS, (c) Energy Consumption, and (d) Service Latency



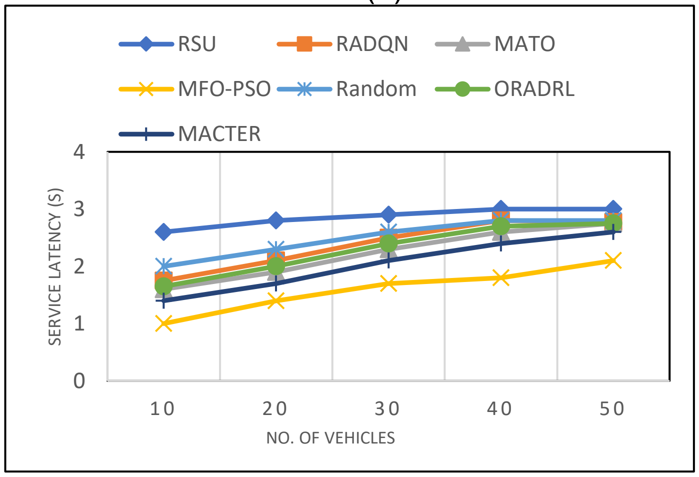
(a)



(b)

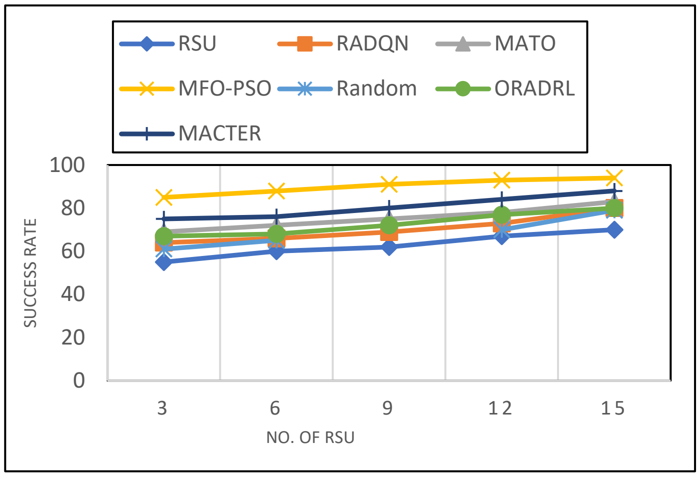


(c)

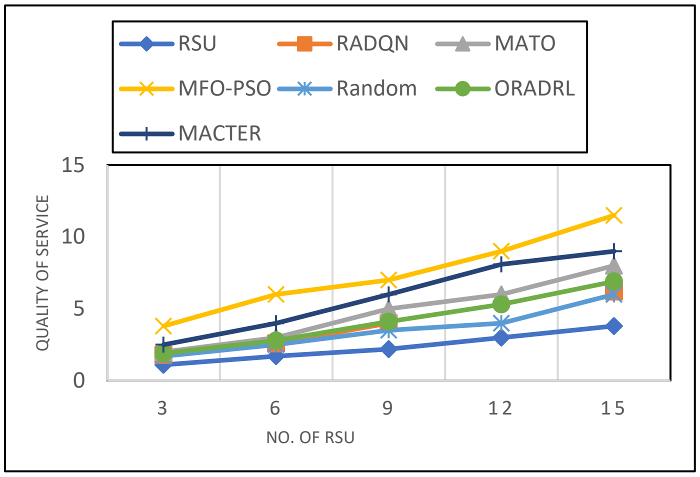


(d)

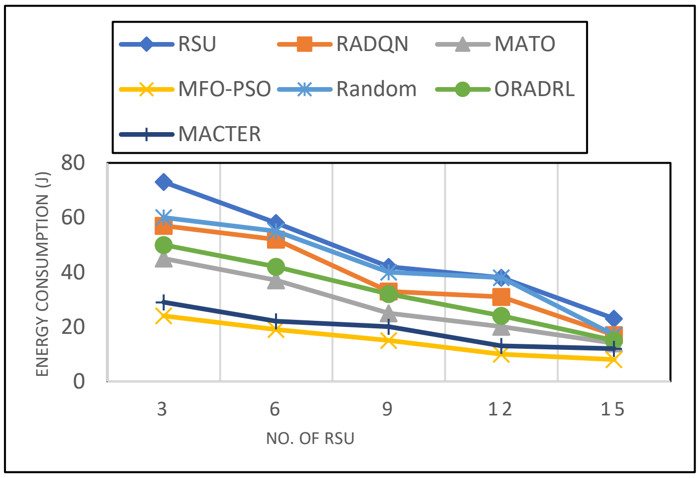
Figure 3: Compare the performing of suggested MFO-PSO algorithms with other exist algorithms concerning No. of vehicles in relation to (a) Success Rate, (b) QoS, (c) Energy Consumption, and (d) Service Latency



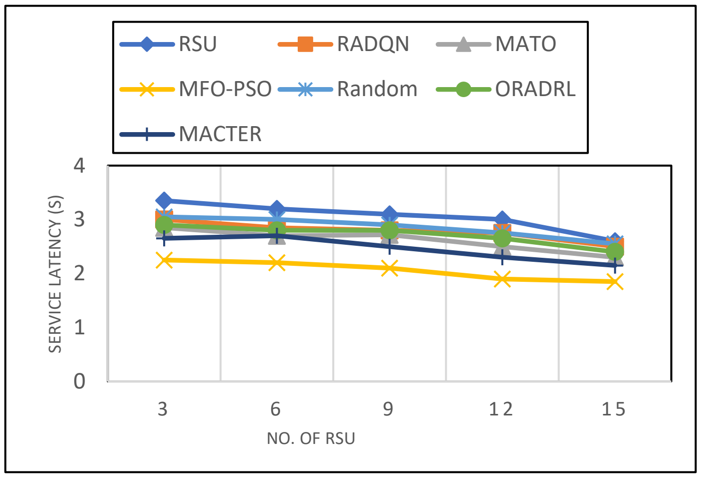
(a)



(b)



(c)



(d)

Figure 4: Compare the performing of suggested MFO-PSO algorithms with other exist algorithms Concerning No. of RSUs in relation to (a) Success Rate, (b) QoS, (c) Energy Consumption, and (d) Service Latency

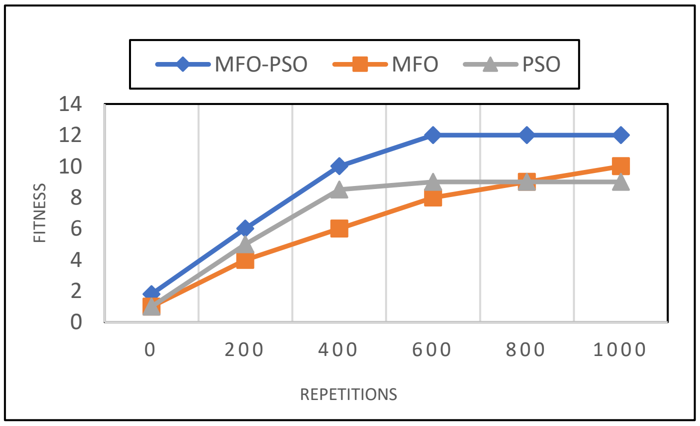


Figure 5: Comparison of convergence between the suggested MFO-PSO method and MFO and PSO algorithms.

Table 3: Comprehensive Performance Comparison of Scheduling Algorithms

Scenario / Metric	MFO-PSO	MATO	MACTER	ORADRL	RADQN	RSU	Random
Varying No. of Tasks (50)							
Success Rate (%)	95	87	92	84	80	73	79
QoS	11.5	8.5	10	7	5.8	3.5	5
Energy Consumption (J)	7	10	11.0	18	23	25	23
Service Latency (s)	0.3	0.6	0.4	0.7	0.7	0.9	0.8
Varying No. of Tasks (100)							
Success Rate (%)	93	82	88	80	77	68	70
QoS	9.2	6.8	7.5	5.5	5	3	4.2
Energy Consumption (J)	9	20	12	23	26	33	28
Service Latency (s)	0.4	0.7	0.5	0.8	0.9	1.1	1
Varying No. of Tasks (150)							
Success Rate (%)	91	72	82	70	67	60	65
QoS	7.2	4.6	6.2	4	3.5	2.1	3
Energy Consumption (J)	13	23	20	28	33	40	35
Service Latency (s)	0.5	0.8	0.6	1	1.3	1.5	1.4
Varying No. of Tasks (200)							
Success Rate (%)	88	58	78	57	52	45	53
QoS	6	2.1	4.5	3.5	2.5	2.1	1.7
Energy Consumption (J)	17	33	22	38	40	53	48
Service Latency (s)	0.7	1.2	0.9	1.3	1.4	2	1.6
Varying No. of Tasks (250)							
Success Rate (%)	85	50	67	43	40	35	40
QoS	4.5	2	3.8	2	2	1.8	1.5
Energy Consumption (J)	19	38	28	47	53	60	56
Service Latency (s)	0.9	2.2	1.1	1.3	1.4	2.1	1.7
Varying No. of Vehicles (10)							
Success Rate (%)	94	83	88	81	78	76	77
QoS	11.5	8.1	9.1	6	5	3	4.5
Energy Consumption (J)	8	13	10	15	17	23	17
Service Latency (s)	1	1.6	1.4	1.65	1.75	2.6	2
Varying No. of Vehicles (20)							
Success Rate (%)	92	78	84	76	74	68	70
QoS	9	6.6	8.3	5.5	4.7	2.8	4
Energy Consumption (J)	10	23	12	25	25	35	32
Service Latency (s)	1.4	1.9	1.7	2	2.1	2.8	2.3
Varying No. of Vehicles (30)							
Success Rate (%)	90	74	80	72	69	63	67
QoS	7	5.5	6	5	4	2	3
Energy Consumption (J)	13	25	20	31	35	40	37
Service Latency (s)	1.7	2.3	2.1	2.4	2.4	2.9	2.6
Varying No. of Vehicles (40)							
Success Rate (%)	88	70	73	69	66	60	65
QoS	5.8	3.7	4.5	3	2.3	1.5	2.5
Energy Consumption (J)	17	42	22	48	51	58	55
Service Latency (s)	1.8	2.6	2.4	2.7	2.8	3	2.8
Varying No. of Vehicles (50)							
Success Rate (%)	85	68	74	68	64	60	61
QoS	3.8	2	2.5	1.5	1.5	1	1
Energy Consumption (J)	19	51	27	55	57	70	60
Service Latency (s)	2.1	2.7	2.6	2.73	2.8	3	2.8
Varying No. of RSUs (3)							
Success Rate (%)	85	69	75	67	64	55	61

QoS	3.8	2	2.5	1.9	1.6	1.1	1.2
Energy Consumption (J)	24	45	29	50	37	73	60
Service Latency (s)	2.25	2.85	2.65	2.9	3	3.35	3.05
Varying No. of RSUs (6)							
Success Rate (%)	88	72	76	68	66	60	65
QoS	6	3	4	2.8	2.8	1.7	2.3
Energy Consumption (J)	19	37	22	42	52	58	55
Service Latency (s)	2.2	2.8	2.7	2.8	2.85	3.2	3
Varying No. of RSUs (9)							
Success Rate (%)	91	75	80	72	69	62	69
QoS	7	5	6	4.1	4	2.2	3.5
Energy Consumption (J)	15	25	20	32	33	42	40
Service Latency (s)	2.1	2.72	2.5	2.8	2.8	3.1	2.9
Varying No. of RSUs (12)							
Success Rate (%)	93	78	84	77	73	67	70
QoS	9	6	8.1	5.3	5.3	3	4
Energy Consumption (J)	10	20	13	24	31	38	38
Service Latency (s)	1.9	2.65	2.3	2.63	2.65	3	2.75
Varying No. of RSUs (15)							
Success Rate (%)	94	83	88	80	80	70	79
QoS	11.5	8	9	6.9	6.2	3.8	6
Energy Consumption (J)	8	14	12	15	17	23	17
Service Latency (s)	1.85	2.4	2.15	2.4	2.5	2.6	2.55

6.6Sensitivity Analysis

Sensitivity analysis is like a check-up for algorithms. It sees how the results change when you tweak the starting information (input parameters). This helps us understand which pieces of information are most important (critical factors) for the algorithm to perform well, and how much the final results can be swayed by changes in the data (Karimiafshar et al., 2021).

This section dives into how sensitive the MFO-PSO algorithm is to different settings. We'll be looking at how changes in parameters from both the PSO (Particle Swarm Optimization) and MFO (Multi-Objective Firefly Optimization) parts of the algorithm, like c_1, c_2, α, and $tempp$, (shown in Table 4), affect the overall performance. The better the MFO-PSO algorithm performs, the higher its "fitness value" is. We found that this value goes down as we increase a setting called the "trust parameter" c_1 from 2.1 to 2.5. This is because a higher c_1 pushes the algorithm to focus on areas it already knows are good (local search), which can miss even better solutions elsewhere. So, a higher trust parameter c_1 leads to a lower fitness value, which means the algorithm performs worse. On the other hand, the MFO-PSO algorithm seems to do better (higher fitness value) when we increase the social constant c_2. In the MFO-PSO algorithm, a setting called the "cognitive constant" c_1 controls how much each virtual particle relies on its own past success. If c_1 is too high, the particles get stuck prioritizing their own best results, rather than exploring new possibilities together. This can lead to the algorithm missing better solutions overall. Similarly, the "social constant" c_2, affects how much the particles are influenced by the optimal solution found so far (the global best). If c_2 is too high, the particles all rush towards that one solution too quickly, and they might miss even better options out there. That's why, to achieve a compromise between individual exploration and learning from the best overall solution, the values for c_1 and c_2 are set to 2.1 each in the MFO-PSO algorithm.

The MFO-PSO algorithm controls the memory function through the inertia weight parameter of each particle. The inertia weight determines the extent to which past momentum of a particle affects its upcoming directional choices. This setting enables users to manage examination of new territories together with focused investigation of successful areas. The MFO-PSO method was evaluated through diverse inertia value (α) adjustments between 0.6 and 1.0 by the researchers. The algorithm produced less successful results as the inertia parameters increased. The authors conducted an evaluation of MFO-PSO by modifying parameter tempp from 0.6 to 1.0. Results in Table 4 show that an MFO-PSO algorithm with 0.8 rate of cooling achieves optimal exploitation-exploration trade-off. Lower values of tempp (0.6) provide extensive exploration at the start so the algorithm can evade local optima trap. The algorithm devotes more time to area exploitation when tempp reaches levels near 1.0 (cooling rate reduces) which enhances its speed of convergence.

Table 4: Examination of sensitivity.

Parameters	Value	Fitness	Parameters	Value	Fitness
ce_1	2.1	12	Inertia (α)	0.6	12
	2.2	12		0.7	12
	2.3	11.8		0.8	12
	2.4	11		0.9	11.9
	2.5	11		1.0	11.9
ce_2	2.1	12	$tempp$	0.6	11.9
	2.2	11.7		0.7	12
	2.3	11		0.8	12
	2.4	11		0.9	11.9
	2.5	10.9		1.0	11.6

6.7Convergence of Proposed MFO-PSO

The way MFO-PSO approaches its optimal solution throughout time is shown in Fig. 5. The authors assessed the efficiency of their MFO-PSO algorithm for a mobile task scheduling problem. This involved comparing the fitness values obtained by MFO-PSO with those achieved by the standard MFO and PSO algorithms. The specific scheduling problem is defined in Equation 21. Fig. 5 shows that PSO converges quickly, but gets stuck at a lower quality solution (lower fitness value). MFO, on the other hand, converges slowly but reaches a higher quality solution (higher fitness value). The proposed MFO-PSO combines the strengths of both, achieving faster convergence than MFO and reaching a higher fitness value than PSO.

6.8Limitations

This research proposes a new way to schedule tasks for vehicles in a network that uses fog computing (vehicular fog network). This method considers how the vehicles move around and the available computing resources. To efficiently assign tasks to resources, the approach uses a technique called hybrid MFO-PSO, which combines two existing optimization algorithms. However, there are some real-world challenges to consider before using it in practice. First issue is the variety of tasks vehicles might have and the different capabilities of roadside unit (RSU) servers in a real network. As an example, some tasks like virtual reality applications demand a lot of processing power and might specifically need fog servers equipped with GPUs (graphical processing units). In order for the proposed algorithm to work well in these diverse environments (heterogeneous environments), some changes are necessary. Second, this research treated servers as if they could only work on a single job at once. However, it might be beneficial to explore allowing servers to handle multiple tasks simultaneously.
Third limitation of this work is that it doesn't take into account how communication delays can be affected by real-world obstacles. Buildings, terrain variations, and other structures can weaken signals (signal attenuation), which can significantly slow down data transfer in vehicular fog networks. Fourth, the greedy algorithm’s optimality is restricted to submodular objectives. In future, it would be useful to study how the system reacts in situations where mobility is not always constant (e.g., where roads are crowded and vehicles move unpredictably). Fifth, this research only explores running tasks on either a vehicle's processor or a roadside server. It doesn't consider the possibility that these vehicle processors could be underutilized, suggesting

there might be an opportunity to improve efficiency. However, the study doesn't fully explore the possibility of using processing power from nearby vehicles. Vehicular fog networks are essentially a collection of various computing devices on wheels, and leveraging this extra resource could significantly improve the network's performance. Taking these limitations into account would be important for implementing the proposed solution in real-world scenarios.

7.CONCLUSION

This research introduces a smart way to schedule tasks for vehicles in a network that uses fog computing (IoV). It considers how vehicles move around (mobility-aware) to minimize both the delay tasks experience and the energy vehicles use. To achieve this, they first turn the task scheduling problem into a challenging mathematical equation that accounts for both whole numbers (integer) and factors that aren't straight lines (non-linear). The researchers then use a technique called a Markov renewal process (MRP) to understand how long vehicles stay in one spot (sojourn time) and how they move around the network. To solve the task scheduling problem for smaller networks, they use a well-known approach called a greedy algorithm, which is guaranteed to find a solution that's at least 63% as good as the absolute best solution. For bigger networks, the researchers designed a new scheduling method that combines two existing techniques: moth-flame optimization and particle swarm optimization. They then test how well this new method performs by comparing it to established scheduling algorithms. The new scheduling method shows significant improvement over the second-best option. It reduces task service delay by 15% and cuts vehicle energy consumption by 16%. Experimental results prove that the hybrid algorithm outperforms PSO, MFO and greedy strategies, dropping average delay by 23.4% and energy use by 17.8%. In future work of this research, we plan to explore how Roadside Units (RSUs) can work together to distribute resources even more effectively within vehicular fog networks. This could potentially lead to even better overall performance.

REFRENCES

ABD ELAZIZ, M., ABUALIGAH, L. & ATTIYA, I. J. F. G. C. S. 2021. Advanced optimization technique for scheduling IoT tasks in cloud-fog computing environments. 124, 142-154.

ABDEL-BASSET, M., MOHAMED, R., ELHOSENY, M., BASHIR, A. K., JOLFAEI, A. & KUMAR, N. J. I. T. O. I. I. 2020. Energy-aware marine predators algorithm for task scheduling in IoT-based fog computing applications. 17, 5068-5076.

ALI, I. M., SALLAM, K. M., MOUSTAFA, N., CHAKRABORTY, R., RYAN, M. & CHOO, K.-K. R. J. I. T. O. C. C. 2020. An automated task scheduling model using non-dominated sorting genetic algorithm II for fog-cloud systems. 10, 2294-2308.

ARTHURS, P., GILLAM, L., KRAUSE, P., WANG, N., HALDER, K. & MOUZAKITIS, A. J. I. T. O. I. T. S. 2021. A taxonomy and survey of edge cloud computing for intelligent transportation systems and connected vehicles. 23, 6206-6221.

ASAAD, K. A., SAFFER, A. A., ABDULQADIR, S. H., PASHA, S. A. J. Z. J. O. P. & SCIENCES, A. 2024. ECR-IoT: Energy-efficient and cluster-based routing method for WSN-based IoT using Harris hawk's and whale optimization algorithms. 36, 43-60.

CHEN, J., DENG, Q., YANG, X. J. J. O. P. & COMPUTING, D. 2023. Non-cooperative game algorithms for computation offloading in mobile edge computing environments. 172, 18-31.

HAMDI, A. M. A., HUSSAIN, F. K. & HUSSAIN, O. K. J. F. G. C. S. 2022. Task offloading in vehicular fog computing: State-of-the-art and open issues. 133, 201-212.

HAYDER, W. A. & HUSAIN, M. H. J. K. J. O. A. R. 2018. Intelligent application implementation model for automated agent negotiation. 3, 68-74.

HAZRA, A., ADHIKARI, M., AMGOTH, T. & SRIRAMA, S. N. J. I. S. J. 2022. Fog computing for energy-efficient data offloading of IoT applications in industrial sensor networks. 22, 8663-8671.

HOU, X., REN, Z., WANG, J., CHENG, W., REN, Y., CHEN, K.-C. & ZHANG, H. J. I. I. O. T. J. 2020. Reliable computation offloading for edge-computing-enabled software-defined IoV. 7, 7097-7111.

HUSAIN, M. H., AHMADI, M. & MARDUKHI, F. J. I. A. 2024a. A GWO-MFO-based Resource Allocation in Vehicular Fog Computing with Latency Constraints and Energy Reduction.

HUSAIN, M. H., AHMADI, M. & MARDUKHI, F. J. W. P. C. 2024b. Vehicular Fog Computing: A Survey of Architectures, Resource Management, Challenges and Emerging Trends. 136, 2243-2273.

KARIMIAFSHAR, A., HASHEMI, M. R., HEIDARPOUR, M. R. & TOOSI, A. N. J. I. T. O. S. C. 2021. An energy-conservative dispatcher for fog-enabled IIoT systems: When stability and timeliness matter. 16, 80-94.

LAKHAN, A., MEMON, M. S., MASTOI, Q.-U.-A., ELHOSENY, M., MOHAMMED, M. A., QABULIO, M. & ABDEL-BASSET, M. J. C. C. 2022. Cost-efficient mobility offloading and task scheduling for microservices IoVT applications in container-based fog cloud network. 1-23.

LI, X., CHEN, T., YUAN, D., XU, J. & LIU, X. J. I. T. O. S. C. 2022. A novel graph-based computation offloading strategy for workflow applications in mobile edge computing. 16, 845-857.

LIN, C., HAN, G., QI, X., GUIZANI, M. & SHU, L. J. I. T. O. V. T. 2020. A distributed mobile fog computing scheme for mobile delay-sensitive applications in SDN-enabled vehicular networks. 69, 5481-5493.

LIU, J., AHMED, M., MIRZA, M. A., KHAN, W. U., XU, D., LI, J., AZIZ, A. & HAN, Z. J. I. I. O. T. J. 2022. RL/DRL meets vehicular task offloading using edge and vehicular cloudlet: A survey. 9, 8315-8338.

LIU, P., AN, K., LEI, J., ZHENG, G., SUN, Y. & LIU, W. J. I. I. O. T. J. 2021a. SCMA-based multiaccess edge computing in IoT systems: An energy-efficiency and latency tradeoff. 9, 4849-4862.

LIU, P., LI, J. & SUN, Z. J. I. A. 2019a. Matching-based task offloading for vehicular edge computing. 7, 27628-27640.

LIU, T., LI, J., SHU, F., HAN, Z. J. I. T. O. N. S. & ENGINEERING 2019b. Optimal task allocation in vehicular fog networks requiring URLLC: An energy-aware perspective. 7, 1879-1890.

LIU, Y., WANG, S., ZHAO, Q., DU, S., ZHOU, A., MA, X. & YANG, F. J. I. I. O. T. J. 2020. Dependency-aware task scheduling in vehicular edge computing. 7, 4961-4971.

LIU, Y., ZHANG, H., LONG, K., ZHOU, H. & LEUNG, V. C. J. I. T. O. V. T. 2021b. Fog computing vehicular network resource management based on chemical reaction optimization. 70, 1770-1781.

LUO, Q., LI, C., LUAN, T. H. & SHI, W. J. I. T. O. S. C. 2021. Minimizing the delay and cost of computation offloading for vehicular edge computing. 15, 2897-2909.

MARTINEZ, I., HAFID, A. S. & JARRAY, A. J. I. I. O. T. J. 2020. Design, resource management, and evaluation of fog computing systems: a survey. 8, 2494-2516.

MISRA, S. & BERA, S. J. I. T. O. V. T. 2019. Soft-VAN: Mobility-aware task offloading in software-defined vehicular network. 69, 2071-2078.

MOHAMMED, M. A. J. Z. J. O. P. & SCIENCES, A. 2022. A comprehensive survey on congestion control techniques and the research challenges on VANET. 34, 50-78.

POULARAKIS, K. & TASSIULAS, L. J. I. T. O. C. 2016. On the complexity of optimal content placement in hierarchical caching networks. 64, 2092-2103.

QIN, P., FU, Y., TANG, G., ZHAO, X. & GENG, S. J. I. T. O. V. T. 2022. Learning based energy efficient task offloading for vehicular collaborative edge computing. 71, 8398-8413.

RAZA, S., WANG, S., AHMED, M., ANWAR, M. R., MIRZA, M. A. & KHAN, W. U. J. I. I. O. T. J. 2021. Task offloading and resource allocation for IoV using 5G NR-V2X communication. 9, 10397-10410.

SAEED, N. K., ASAAD, K. A., SAFFER, A. A. J. Z. J. O. P. & SCIENCES, A. 2024. Optimized Resource Allocation in Vehicular Fog Computing Environments Using Hybrid MOSP Algorithm. 36, 118-131.

SHEN, Q., HU, B.-J. & XIA, E. J. I. T. O. V. T. 2022. Dependency-aware task offloading and service caching in vehicular edge computing. 71, 13182-13197.

SOMESULA, M. K., ROUT, R. R. & SOMAYAJULU, D. V. J. C. N. 2021. Contact duration-aware cooperative cache placement using genetic algorithm for mobile edge networks. 193, 108062.

SUN, J., GU, Q., ZHENG, T., DONG, P., VALERA, A. & QIN, Y. J. I. A. 2020. Joint optimization of computation offloading and task scheduling in vehicular edge computing networks. 8, 10466-10477.

TANG, C., ZHU, C., ZHANG, N., GUIZANI, M. & RODRIGUES, J. J. J. I. I. O. T. J. 2022. SDN-assisted mobile edge computing for collaborative computation offloading in industrial Internet of Things. 9, 24253-24263.

TRAN, T. X. & POMPILI, D. J. I. T. O. V. T. 2018. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. 68, 856-868.

WANG, D., TAN, D. & LIU, L. J. S. C. 2018. Particle swarm optimization algorithm: an overview. 22, 387-408.

WANG, Y., LANG, P., TIAN, D., ZHOU, J., DUAN, X., CAO, Y. & ZHAO, D. J. I. I. O. T. J. 2020. A game-based computation offloading method in vehicular multiaccess edge computing networks. 7, 4987-4996.

XIA, W., QUEK, T. Q., ZHANG, J., JIN, S. & ZHU, H. J. I. T. O. W. C. 2019. Programmable hierarchical C-RAN: From task scheduling to resource allocation. 18, 2003-2016.

XIONG, X., ZHENG, K., LEI, L. & HOU, L. J. I. J. O. S. A. I. C. 2020. Resource allocation based on deep reinforcement learning in IoT edge computing. 38, 1133-1146.

YAN, J., BI, S. & ZHANG, Y. J. A. J. I. T. O. W. C. 2020. Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach. 19, 5404-5419.

ZHAI, Y., SUN, W., WU, J., ZHU, L., SHEN, J., DU, X. & GUIZANI, M. J. I. T. O. I. T. S. 2020. An energy aware offloading scheme for interdependent applications in software-defined IoV with fog computing architecture. 22, 3813-3823.

ZHANG, K., PENG, M. & SUN, Y. J. I. I. O. T. J. 2020. Delay-optimized resource allocation in fog-based vehicular networks. 8, 1347-1357.

ZHOU, Z., FENG, J., CHANG, Z. & SHEN, X. J. I. T. O. V. T. 2019a. Energy-efficient edge computing service provisioning for vehicular networks: A consensus ADMM approach. 68, 5087-5099.

ZHOU, Z., LIU, P., FENG, J., ZHANG, Y., MUMTAZ, S. & RODRIGUEZ, J. J. I. T. O. V. T. 2019b. Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach. 68, 3113-3125.

ZHU, C., TAO, J., PASTOR, G., XIAO, Y., JI, Y., ZHOU, Q., LI, Y. & YLÄ-JÄÄSKI, A. J. I. I. O. T. J. 2018. Folo: Latency and quality optimized task allocation in vehicular fog computing. 6, 4150-4161.