Manage On-Road Services with Chains: A Distributed Vehicular Task Offloading Approach Based on Blockchain Technology

Wenjun Zhang, Xinlu Mao, Xiao Chen, Chao Zhu*

***Abstract*—In resource-constrained vehicular environments, offloading tasks to edge servers simultaneously from multiple client vehicles leads to resource competition, causing a cycle of increased task processing latency. Designing a task offloading strategy that balances the latency of new offloaded tasks and those already being executed on edge servers is a crucial challenge. Additionally, centralized task offloading strategies based on global information require the design of complex communication mechanisms to collect task and computational workload information in real time, which is not desirable in vehicular environments due to the dynamic changes in the locations of client vehicles and the varying task offloading demands across time and space. To avoid such inconvenience, we propose a distributed blockchain-based task offloading strategy, Moscato, to address the resource competition issue in multi-client vehicular task offloading. In Moscato, client vehicles are transformed into consensus nodes within the blockchain, leveraging the existing consensus mechanisms for non-real-time global information sharing. Moreover, we designed an algorithm combining Federated Learning (FL) and Deep Q-Network (DQN) to address the task offloading problem, considering various task profiles and dynamic traffic situations. We collected real server workload and task processing latency data and conducted comparative simulations based on the collected dataset. Through comparison with state-of-the-art methods, we demonstrate that Moscato can effectively balance the resource competition between new and executing tasks.**

***Index Terms*—Vehicular Fog Computing, Federated Learning, Task Offloading, Resource Scheduling, Blockchain**

I. Introduction

With the popularization of intelligent vehicles [1], [2], many emerging smart driving applications, such as assistant driving and autonomous driving, are becoming increasingly popular. These emerging applications, which include time-critical and data-intensive computing tasks (e.g., real-time object recognition), require more computational resources than traditional vehicular ones. However, due to limitations in the physical space available, weight restrictions, and budget constraints of most vehicles, their computational capacities may not be high enough to handle these computation-intensive tasks.

Recently, Vehicular Fog Computing (VFC) [3] has emerged as a viable solution to these problems. VFC combines the Internet of Vehicles (IoV) and Multi-access Edge Computing (MEC), transforming roadside infrastructure, such as traffic lights equipped with ample space and power, into fog nodes that serve nearby client vehicles [4]. With one-hop vehicle-to-vehicle (V2V) communication, computational tasks generated by client vehicles can be offloaded to nearby fog nodes for much more efficient processing.

Although fog nodes have more computational resources than individual client vehicles, they may become insufficient when multiple client vehicles concurrently offload tasks. Any newly offloaded task will compete for resources with the tasks being executed, negatively impacting the processing latency of all tasks hosted on the fog node. Therefore, designing an appropriate offloading strategy that meets the requirements of newly offloaded tasks and minimizes the negative impact on the executing tasks is critically essential.

Recent studies show that most of the state-of-the-art vehicular task offloading strategies [5]–[7] are based on the assumption of global information transparency, where each client vehicle is presumed to have full knowledge of the real-time operating status of the fog nodes—that is, whether a fog node has sufficient capacity to handle a task offloaded from a client vehicle within a given time frame. However, this assumption is not desirable in vehicular networks. Firstly, the network topology in urban vehicular environments is highly dynamic; therefore, a complex communication mechanism must be designed to update the status of all involved entities in real-time. Secondly, the spatiotemporal variation and the diversities of tasks generated from client vehicles make it difficult to observe the operating status of fog nodes.

To design vehicular task offloading strategies without the assumption of global information transparency, blockchain can provide a global-level solution in a connected vehicle environment by maintaining a distributed ledger between various entities. In a blockchain network, consensus nodes achieve blockchain state consistency through consensus protocols and peer-to-peer (P2P) networking protocols [8]. Based on this observation, we design Moscato, a blockchain-based task offloading strategy to reduce task processing latency and improve task performance, considering the impact of resource competition caused by concurrent tasks. In Moscato, client vehicles are turned into consensus nodes. In the initial stage, once a task is offloaded, a new block recording the task's details (e.g., task ID, timestamp, data size, and required computational resources) would be created and added to the blockchain ledger. When a client vehicle generates a new task, it can review the entire blockchain ledger to obtain the latest information about the executing tasks on fog nodes. Based on this information, the client vehicle can decide whether to

W. Zhang is with the Department of Computer Science, University of Helsinki, Helsinki, Finland.

X. Mao, X. Chen, and C. Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. (Chao Zhu is the corresponding author: chao@bit.edu.cn)

offload the new task. If offloading is chosen, a new block would be created and attached to the ledger; otherwise, the task would be executed locally without creating any block. This method allows task offloading by utilizing the blockchain's established broadcast system to retrieve global information in a non-real-time manner.

Considering the various task requirements and dynamic traffic situations, traditional model-based algorithms are insufficient to capture the complexity of the task offloading problem. Therefore, we employ an experience-based Deep Q Network (DQN) [9], where each client vehicle determines its own offloading decision while accounting for the impact of task concurrency. On the other hand, the current task scheduling methods adopted by computer operating systems are highly complex, and traditional mathematical approaches find it challenging to quantify the *Concurrency Overhead*, which is the negative impact caused by adding a new task under different fog node workloads (i.e., the latency of the new task and the extended latency of executing tasks). To address this challenge, we utilize a Convolutional Neural Network (CNN) [10] to predict concurrency overhead. To align with the distributed decision-making mechanism, we introduce a cluster-based Federated Learning (FL) [11] to train the concurrency overhead model. Specifically, fog nodes are clustered based on their varying workloads, with each fog node training a CNN prediction model with its historical concurrency overhead data, and the models are then aggregated within their respective clusters. By transmitting the model instead of raw concurrency overhead data, FL can indirectly expand the training set, maintaining the model's accuracy while reducing the amount of data transmission.

The key contributions of this work are summarized below:

- We propose Moscato, a novel vehicular task offloading strategy aiming at balancing the latency of new offloaded task and that of executing tasks. Notably, we are the first to introduce blockchain technology to tackle the issue of resource competition in vehicular task offloading.
- We designed a FL and DQN-Integrated (FL-DQN) algorithm to address the task offloading problem, taking into account the various task profiles and dynamic traffic situations.
- We gather the concurrency overhead dataset by executing two different tasks (i.e., image processing and Fibonacci sequence calculations) on real servers, and use this dataset for simulations. The results demonstrate the effectiveness of the proposed task offloading strategy.

The rest of this paper is organized as follows. Section II discusses the related works. Section III presents the system architecture of the proposed Moscato. Section IV elaborates on the system model and the optimization problem. Section V provides the details of the designed FL-DQN algorithm. Section VI details the profiles of involved functions and shows the evaluation results. Section VII concludes the paper.

II. Related Work

A. Vehicular Task Offloading

In recent years, significant research efforts have been dedicated to exploring task offloading strategies within vehicular networks, particularly driven by the emergence of the IoV and the MEC. These advancements have necessitated the development of sophisticated approaches to optimize the trade-offs between computational efficiency and communication overhead in increasingly dynamic and complex vehicular environments. Chen et al. [12] proposed a multihop task offloading decision model designed specifically for IoV environments. Their model employs a bat algorithm to optimize resource allocation, effectively addressing the challenges posed by the dynamic and rapidly changing nature of vehicular networks. Liu et al. [13] developed a mobility-aware framework that considers both the movement patterns and connectivity of autonomous vehicles, building on the need to adapt task offloading strategies to the high mobility of vehicles. Cui et al. [14] focused on offloading computationally intensive tasks, such as simultaneous localization and mapping (SLAM), for autonomous vehicles, demonstrating the potential of edge computing to handle computation-intensive demanding tasks. Xiao et al. [15] explored the collaborative computation approach for perception task offloading in autonomous driving, leveraging shared computational resources to enhance both safety and reliability in autonomous driving scenarios. Yang et al. [16] examined the integration of edge intelligence within 6G wireless systems to support autonomous driving, providing a comprehensive analysis of the design challenges and proposed solutions for leveraging 6G capabilities to improve real-time data processing, decision-making, and overall vehicle operation at the network edge. Sun et al. [17] introduced an adaptive learning-based task offloading strategy for vehicular edge computing systems, utilizing online learning and multi-armed bandit algorithms to dynamically adjust offloading decisions in response to changing vehicular environments.

Collectively, these studies underscore the pivotal role of innovative task offloading strategies in enhancing the performance and efficiency of vehicular networks. As autonomous and connected vehicle technologies continue to evolve, the need for advanced computational models and optimized offloading strategies becomes increasingly critical in meeting the complex demands of the IoV.

B. Distributed Decision-making in Vehicular Environment

Distributed decision-making in vehicular environments is increasingly critical as autonomous vehicles and connected infrastructures demand efficient, reliable, and secure methods to manage and process vast amounts of data. In this section, we investigate two primary distributed decision-making approaches: Blockchain and Federated Learning (FL), each of which offers distinct advantages in enabling decentralized decision-making within vehicular networks.

Blockchain in Vehicular Networks. Blockchain technology has garnered significant attention for its potential to provide secure, transparent, and decentralized solutions in various applications, including vehicular networks. Jain et al. [18] provided a comprehensive review of blockchain applications in autonomous vehicles, highlighting how blockchain can enhance security, facilitate distributed ledger systems, and support autonomous vehicle operations. Abishu et al. [19] proposed a consensus mechanism for blockchain-enabled energy

trading in the Internet of Electric Vehicles (IoEV), leveraging blockchain to manage and secure energy transactions. Jabbar et al. [20] proposed a blockchain-based framework that integrates smart contracts and secure payment mechanisms, enhancing the safety and reliability of V2X interactions. Li et al. [21] introduced a blockchain-based framework for vehicle position correction in vehicular networks, utilizing blockchain to share GPS error information and improving the accuracy of vehicle positioning through distributed error correction.

Federated Learning in Vehicular Networks. Federated Learning (FL) has emerged as a promising approach for distributed decision-making in vehicular networks, enabling vehicles to collaboratively learn and improve models without sharing raw data. This decentralized approach is particularly valuable in preserving data privacy while optimizing resource usage across the network. Xiao et al. [22] addressed the challenge of vehicle selection and resource optimization in federated learning for vehicular edge computing, selecting the most suitable vehicles for participation in FL tasks and thereby optimizing local model accuracy and resource allocation. Chen et al. [23] introduced a Byzantine-fault-tolerant decentralized FL method tailored for autonomous vehicles, enhancing the robustness and reliability of FL systems by mitigating the effects of faulty or malicious nodes. Li et al. [24] advanced FL by incorporating multi-agent deep reinforcement learning (MARL) for resource allocation in V2V communications, leveraging MARL to dynamically allocate resources. Chai et al. [25] proposed a hierarchical blockchain-enabled FL algorithm for knowledge sharing in IoV, ensuring that knowledge sharing among vehicles is both secure and efficient.

To the best of our knowledge, we are the first to leverage blockchain's consensus mechanism for non-real-time sharing of information related to on-road services. In our proposed Moscato framework, vehicular task offloading can utilize blockchain's established communication mechanisms to achieve non-real-time global information sharing. Additionally, through clustered FL, Moscato can expand the training dataset without transmitting raw data, thereby improving the load prediction accuracy of on-road servers and supporting more precise offloading decisions.

III. Architecture Design

In this section, we first define the terms and present the architecture of the blockchain ledger. Then, we outline the process of task offloading in Moscato.

A. Related Terms

Client Vehicles: Intelligent vehicle entities equipped with emerging applications are defined as *client vehicles (CVs)*. The computing power of CVs is inconsistent, ranging from high-performance computing vehicles such as the Tesla to ordinary vehicles with only cameras and a simple Android system.

Tasks: Computing requirements arise while running the CVs application, which is described as *tasks*. An application may contain multiple tasks with different computational resource requirements. For example, when a CV performs augmented reality-based driver assistance, it generates tasks such as object recognition and road planning, where the former requires more computational resources than the latter. Note that the tasks generated by the application could be processed locally or offloaded to a nearby edge server.

TABLE I: Block Types

(a) Resource Block

	Field	Description
BlockHeader	previousBlockHeader	Hash value of the previous block
	timestamp	Timestamp of the block creation
	currentBlockHash	Hash value of the current block
Transactions	fogNodeIDSet	IDs of fog nodes
	fogNodeLocationSet	Locations of fog nodes
	fogNodeCapacitySet	Computational capacities of fog nodes
	concurrencyOverhead	Concurrent overhead model

(b) Task-start Block

	Field	Description
BlockHeader	previousBlockHeader	Hash value of the previous block
	timestamp	Timestamp of the block creation
	currentBlockHash	Hash value of the current block
Transactions	taskID	ID of the task
	dataSize	Data size of the task
	requireComp	Computational resource requirement
	beginingTime	Timestamp of the task start
	hostID	ID of the host fog node

(c) Task-end Block

	Field	Description
BlockHeader	previousBlockHeader	Hash value of the previous block
	timestamp	Timestamp of the block creation
	currentBlockHash	Hash value of the current block
Transactions	taskID	ID of the task
	endTime	Timestamp of the task completed
	hostID	ID of the host fog node

Fog Nodes: Due to a limited amount of onboard resources, CVs could offload their tasks to nearby edge servers, which are called *fog nodes*. In Moscato, we just consider stationary fog nodes, which are the computing nodes co-located with traffic signs, roadside billboards, lights, or any other stationary infrastructures.

Zone Heads: Currently, the urban areas of modern cities are completely covered by cellular networks, making them naturally suitable for being divided into multiple *service zones*. Within a specific service zone, the base station located in the center of the area, called *zone head*, is selected to coordinate all the in-zone CVs and fog nodes. Under the existing cellular registration mechanism, a vehicle always notifies the zone head when it enters or leaves the zone. Moreover, the in-zone fog nodes periodically report their locations and the amount of resources to the zone head.

B. Blockchain Ledgers

When a new task is decided to be offloaded, it preemptively takes computational resources from tasks that are currently executing, resulting in fog node's computational resource competition. In order to avoid blind competition, we set up a blockchain architecture, acting as a *blockchain ledger*, to share

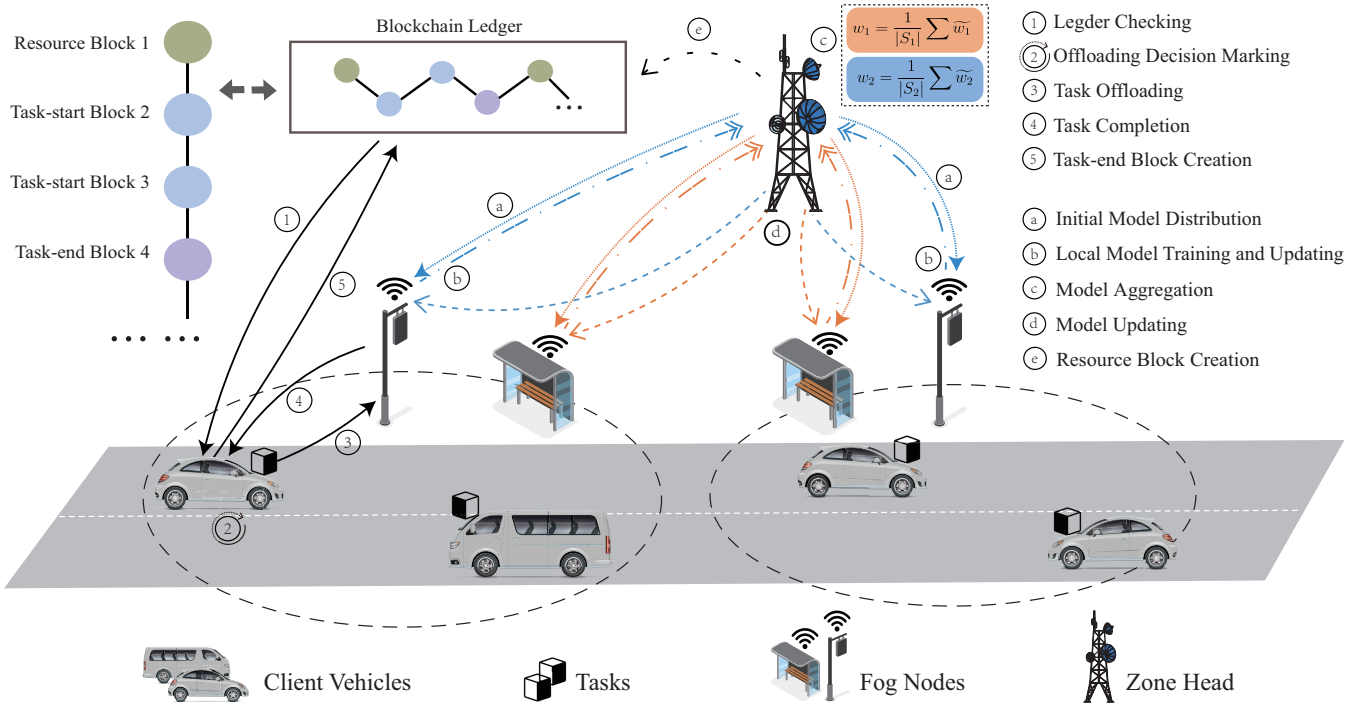


Fig. 1: System Overview of Moscato.

the global information of in-zone tasks, helping to manage the computational resources of fog nodes. The blockchain ledger is a chain of blocks, with each block recording basic information of related terms. In Moscato, CVs and the zone heads are treated as independent consensus nodes, capable of creating blocks and performing on-blockchain actions.

1) Block Types: As illustrated in Table I, three types of blocks are designed in Moscato.

- *Resource Block*: Resource blocks are created by zone head, containing information about the IDs, locations and initial computational capacities of all fog nodes within the zone. Notably, we define a specific neural network named *Concurrency Overhead Model* to describe the impact of computational resource competition (i.e., the latency of the new task and the extended latency of executing tasks), which is also included in the resource block.
- *Task-start Block*: Task-start blocks are created by CVs. They aim to mark that a task is starting its execution. The information about an offloaded task, such as the task ID, starting time, required computational resources (i.e., number of floating-point operations required), and its host fog node's ID, is contained in a task-start block.
- *Task-end Block*: CVs also create task-end blocks, aiming at marking that a task has ended its execution. The ID of a completed task is contained in a task-end block.

2) Blockchain formulation and updating: In Moscato, all in-zone consensus nodes share and maintain the regional blockchain ledger in the following manners.

- *Block Generation*: Once a demand such as a task is decided to be offloaded, the corresponding consensus node will create a new block. Note that the first block in the blockchain ledger is a resource block created by the zone head.
- *Consensus Mechanism*: The integrity and consistency of the blockchain are maintained through a consensus algorithm to ensure that all consensus nodes agree on the generation and verification of blocks via a peer-to-peer (P2P) protocol. In Moscato, a Proof of Model Quality (PoMQ) [26] consensus mechanism is adopted, which does not rely on computationally intensive traditional Proof of Work (PoW). Instead, it reaches consensus by validating the quality of task offloading decision models submitted by each node. A model is included in the blockchain consensus process only if it meets the predefined quality standards, thus effectively preventing malicious nodes from submitting invalid offloading decision models.
- *Blockchain Structure Maintenance*: Consensus nodes maintain the continuity and stability of the blockchain by constantly attaching new read-only blocks to the blockchain structure. Each new block is attached to the end of the blockchain and connected to the previous block via a cryptography hash.
- *Distributed storage*: All in-zone consensus nodes jointly maintain and store one blockchain ledger in a distributed manner. This distributed storage mechanism leverages data replication to ensure reliability, thereby preventing single points of failure from impacting the entire system. Note that, when a vehicle leaves the current zone, any unfinished offloaded tasks would be discarded. The leaving vehicle reports its departure status to the current zone's zone head node, and the unfinished tasks are marked as lost and removed from the blockchain.

C. Task Offloading Process

Fig. 1 illustrates the process of the task offloading process. The whole process consists of 5 operations.

1. *Ledger Checking*: Once a new task is generated, the host CV checks the blockchain ledger from start to finish. By reading the latest resource block, the host CV can identify the ID of the nearest fog node and obtain its resource information. By reading the task-start and task-end blocks, the host CV can gather information about its competitors, which are the tasks currently being executed on the fog node.
2. *Offloading Decision Making*: After obtaining the relevant information, the CV decides whether to offload the newly generated task. Although offloading the new task may benefit the CV itself (i.e., reducing the service latency), the need to preempt computational resources can adversely affect tasks being executed. Therefore, making the right offloading decision is challenging for optimizing the revenue of all participants. To this end, we employ experience-based DQN to address this problem, where each CV makes its own offloading decision while considering the impact of resource competition (i.e., concurrency overhead). By leveraging historical experience for training, DQN enables CV to obtain an effective offloading strategy without complex mathematical modeling.
3. *Task Offloading*: Once the CV decides to offload the newly generated task, it will transmit it to the fog node over one-hop V2V communication links, such as Wi-Fi. Once the task is completed and transferred, the fog node processes it in real-time. Simultaneously, a new task-start block will be created and attached to the blockchain. Note that if the CV decides not to offload the new task, the task will be executed locally, and no block will be created.
4. *Task Completion*: Once an offloaded task is completed, the fog node will send the processed results to its host CV.
5. *Task-end Block Creation*: In the end, the CV creates a new task-end block to indicate the completion of the task and attach it to the blockchain.

D. Concurrency Overhead Model Obtaining

Due to the large number of fog nodes and their varying computational capacities, learning the concurrency overhead model in a centralized manner is not feasible. On the other hand, the number of training samples of a single fog node may be too small, which may lead to poor accuracy of the trained model. We apply FL instead of centralized or onboard learning to address this issue.

To address the device heterogeneity resulting from the computational capacities of fog nodes, we classify fog nodes equipped with varying levels of workloads into different clusters, where each cluster performs model aggregation independently. In Moscato, the zone heads perform the clustered FL mechanism.

Fig. 1 illustrates the concurrency overhead model learning process, which consists of 5 operations.

a. *Initial Model Distribution*: At the beginning, the zone head broadcasts the initial concurrency overhead model to all fog nodes in the zone. We assume that the concurrency overhead model initially distributed is untrained.
b. *Local Model Training and Uploading*: When the fog node receives the model distributed from the zone head, it starts training the concurrency overhead model using its own training set. The training dataset includes the resource competition relationship between the newly offloaded task and tasks currently being executed, specifically the computation latency of the newly offloaded task and its adverse impact on tasks presently being executed.
c. *Model Aggregation*: After the model training is completed, fog nodes will upload their local models to the zone head for aggregation, where methods such as weighted averaging of the local models are involved.
d. *Model Updating*: After the model aggregation, the global concurrency overhead model will be updated. Model Updating includes two parts. The zone head first distributes the new model to all the in-zone fog nodes for continuous training. Then, the zone head creates a new resource block, where the updated concurrency overhead model is included.
e. *Resource Block Creation*: Once the model is updated, the zone head creates a new resource block, where the updated concurrency overhead model is included.

Note that, as time progresses, the blockchain ledger of the service zone may accumulate excessive length, resulting in increased storage space requirements and block retrieval latency. To address this, Moscato employs a periodic data pruning mechanism whereby the zone head compacts the blockchain every hour. Rather than completely destroying the old blockchain, the algorithm condenses it by summarizing expired blocks and archiving data with a snapshot block. This snapshot block retains essential historical data to maintain continuity while reducing storage demand.

IV. Task Offloading Problem Formulation

In this section, we formulate the task offloading problem in a vehicular environment. Table II shows the related symbols and explanations.

A. Tasks and Offloading Decisions

1) Executing Tasks: For simplicity, we only consider multiple CVs connecting to a single fog node in Moscato. We use v, τ, and Θ to denote the CV, task, and fog node, respectively.

We use i and e to denote the ID of a specific new generated task and task already being executed on the fog node, respectively. Suppose when a new task τ_i is generated by its host CV v_i at timestamp t, there are already N_e tasks executing on the fog node. We use τ_e to denote the executing task, which can be described as:

$$\tau_e = \{C_e, T_e\}, \forall e \in (1, N_e), \tag{1}$$

where C_e represents the required floating-point operations and T_e represents the start time of task τ_e. The number of concurrent tasks in a fog node is N_e.

TABLE II: Notations and Descriptions

Notations	Descriptions
τ_i, i	New task and its ID
τ_e, e	Executing task and its ID
C_i, C_e	Required floating-point operations of the new task τ_i and the executing task τ_e
T_i, T_e	Beginning timestamp of the new task τ_i and the executing task τ_e
S_i	Data size of the new task τ_i
v_i	Host CV of the new task τ_i
x_i	Offloading decision of the new task τ_i
Θ	Host fog node
$Pos_i(t), Pos_\Theta$	Locations of the CV (at timestamp t) and the fog node
$d_\Theta^{v_i}(t)$	Distance between the CV v_i and the fog node at timestamp t
$r_{comm,i}$	Communication data rate of the new task τ_i
$l_{comm,i}$	Communication latency of the new task τ_i
N_e	Number of executing tasks
F_{v_i}, F_Θ	Computational capacities of the CV v_i and the fog node
$f_\Theta^i(t), f_\Theta^e(t)$	Fog node's computational resources allocated to the new task τ_i and the executing task τ_e at timestamp t
$l_{comp,i}$	Computation latency of the new task τ_i
ζ_e^t	Number of finished floating-point operations of the executing task τ_e at timestamp t
Δ_e	Remaining computation latency of the executing task τ_e
l_i	Service latency of the new task τ_i
U_i	Offloading reward of the new task τ_i

2) New Task and Offloading Decision: When a new task τ_i is generated, its host CV v_i needs to decide whether to process it locally or offload it to the nearest fog node. The task τ_i is described as:

$$\tau_i = \{S_i, C_i, T_i\}, \tag{2}$$

where S_i, C_i and T_i are the data size, required floating-point operations, and the starting time of task τ_i, respectively.

We use a binary indicator $x_i \in \{0, 1\}$ to denote whether CV v_i to offload the task τ_i, where $x_i = 0$ indicates local processing and $x_i = 1$ indicates that the task would be offloaded to the nearest fog node. Note that, in Moscato, each CV makes its own offloading decision and will not generate a new task until the previous task is executed.

B. Communication Latency

We use $Pos_i(t) = [x_i(t), y_i(t)]$ and Pos_Θ to denote the position of CV v_i and the fog node Θ at timestamp t, respectively. The distance between Θ and the vehicle v_i can be expressed as:

$$d_\Theta^{v_i}(t) = \|Pos_i(t) - Pos_\Theta\|. \tag{3}$$

We assume that Θ have a certain coverage area $D(\Theta)$, within which the CV can offload the task to the fog node:

$$d_\Theta^{v_i}(t) \leq D(\Theta). \tag{4}$$

CVs can transmit their data directly to a fog node through one-hop communication. According to the Shannon theorem, the achievable data rate $r_{comm}(\Theta, v_i)$ between a CV v_i and fog node Θ can be expressed as:

$$r_{comm}(\Theta, v_i) = B \log_2(1 + \frac{P_{v_i} H_\Theta^{v_i}}{N_0}), \tag{5}$$

where B denotes the bandwidth of the communication link, P_{v_i} is the transmit power of CV v_i, $H_\Theta^{v_i}$ is the channel gain from CV v_i to fog node Θ, and N_0 is the power of background noise.

The total communication latency $l_{comm,i}$ for a task τ_i can be calculated by:

$$l_{comm,i} = \begin{cases} \dfrac{S_i}{r_{comm}(\Theta, v_i)}, & \text{if } x_i = 1, \\ 0, & \text{if } x_i = 0. \end{cases} \tag{6}$$

C. Computation Latency

1) Computation Latency of New Task: When a newly generated task τ_i is decided to be offloaded to fog node Θ, it will compete for resources with tasks already in execution $\{\tau_e\}$. We assume that all the onboard tasks will share the computing resources of the fog nodes. Therefore, the computational resources allocated to an individual task will decrease accordingly. We use $g(\cdot)$ to denote the impact of the resource competition on the new task and the executing tasks. Given the $g(\cdot)$, the fog node's computational resources allocated to the new task τ_i and executing tasks $\{\tau_e\}$ at timestamp t can be expressed as:

$$f_\Theta^i(t), f_\Theta^{\{e\}}(t) = g(x_i, C_i, \{\tau_e\}, F_\Theta), \tag{7}$$

where F_Θ denotes the initial computational capacity of the fog node Θ, measured in floating-point operations per second.

Given the $f_\Theta^i(t)$, the computation latency $l_{comp,i}$ for task τ_i can be calculated by the following equation:

$$l_{comp,i} = \begin{cases} \dfrac{C_i}{f_\Theta^i(t)}, & \text{if } x_i = 1, \\ \dfrac{C_i}{F_{v_i}}, & \text{if } x_i = 0, \end{cases} \tag{8}$$

where F_{v_i} refers to the local computational capacity of CV v_i, measured in floating-point operations per second.

2) Remaining Computation Latency of Executing Tasks: Before calculating the remaining computation latency of a specific executing task τ_e, we need to calculate how many floating-point operations the task τ_e has been already executed at timestamp t. We use ζ_e^t to denote the number of finished floating-point operations of task τ_e at timestamp t, which can be expressed as:

$$\zeta_e^t = \lfloor \int_{T_e}^{t} f_{\Theta}^{e}(\ell)\ d\ell \rfloor, \tag{9}$$

where $f_{\Theta}^{e}(\ell)$ represents the fog node's computational resources allocated to the executing task τ_e at time ℓ.

Given the computational resources allocated to a specific executing task τ_e at timestamp t, the remaining computation latency at timestamp t can be expressed as:

$$\Delta_e = \frac{C_e - \zeta_e^t}{f_{\Theta}^{e}(t)}. \tag{10}$$

D. Problem Formulation

As described above, the latency of a new task τ_i constitutes communication latency $l_{comm,i}$ and computation latency $l_{comp,i}$. Each vehicle aims to optimize its own revenue, which reduces the task's service latency l_i:

$$l_i = l_{\text{comm},i} + l_{\text{comp},i}. \tag{11}$$

However, from the perspective of the entire system, offloading task τ_i will squeeze resources in the fog nodes, thereby increasing the remaining computation latency of ongoing tasks. Therefore, when defining rewards, we need to consider not only the agent's own benefits but also the overall system's benefits. We use U_i to denote the reward for task τ_i's offloading decision x_i, which is expressed as:

$$U_i = -(\alpha l_i + \beta \sum_{1}^{N_e} \Delta_e), \tag{12}$$

where α and β are coefficients reflecting the relative importance of agent's benefit and system's benefit, respectively.

In the end, we formulate the optimization problem as follows:

$$\xi 1 : \max_{x_i} \sum_{1}^{\infty} U_i \tag{13}$$

s.t.

$$x_i \in \{0, 1\}, \tag{13a}$$

$$d_{\Theta}^{i}(t) \leq D(\Theta). \tag{13b}$$

Note that problem $\xi 1$ is closely related to the computational capabilities of fog nodes and the computational requirements of tasks, which are numerous and diverse. In practice, it is challenging to make the right offloading decision x_i for each task τ_i to achieve long-term revenue maximization. Instead of applying model-based algorithms, we employ DQN to obtain an effective offloading strategy, where an individual CV makes its own offloading decision based on experience.

V. FL and DQN-Integrated Approach

A. FL Based Concurrency Overhead Model Obtaining

To maximize overall system performance, modern operating systems dynamically adjust computational resource allocation based on the demands of tasks and the system's current state. Consequently, it becomes infeasible to calculate the exact amount of computational resources allocated to a specific task. Additionally, in vehicular environments, the arrival of new tasks often follows a Poisson distribution, complicating the calculation of the remaining computation latency of an executing task. To address these challenges, we employ a Convolutional Neural Network (CNN), denoted by weights $\boldsymbol{w}$, to characterize the computation latency $l_{\text{comp},i}$ of a new task τ_i and the remaining computation latency Δ_e of an executing task τ_e, as detailed in Section IV.

Each fog node is typically allocated a relatively small dataset to expedite task processing. However, this limited dataset may not be sufficient to train a robust model at each node individually. We adopt the FL approach to enhance the accuracy of task latency prediction and address this limitation. In Moscato, each fog node Θ_i trains a CNN using its local dataset. Subsequently, the model parameters from each node are aggregated using Federated Averaging (FedAvg) [27], resulting in a more generalizable and accurate model for predicting the remaining computation latency of executing tasks within the fog computing environment.

Specifically, each fog node Θ_i utilizes its local dataset D_i to train a local model $\boldsymbol{w}_i$ through gradient descent. The update rule for the local model is expressed as:

$$\boldsymbol{w}_i^{(t+1)} = \boldsymbol{w}_i^{(t)} - \eta \nabla \mathcal{L}(\boldsymbol{w}_i^{(t)}, D_i), \tag{14}$$

where η represents the learning rate, and $\nabla \mathcal{L}(\boldsymbol{w}_i^{(t)}, D_i)$ denotes the gradient of the loss function $\mathcal{L}$ with respect to the local dataset D_i.

Upon completion of local training, the models from all fog nodes are aggregated using the FedAvg algorithm to obtain a global model $\boldsymbol{w}$:

$$\boldsymbol{w} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{w}_i, \tag{15}$$

where N denotes the total number of participating fog nodes. This global model $\boldsymbol{w}$ is then used to predict task latency across the entire network, benefiting from the diverse data distributions present across different fog nodes. The collaborative learning approach thus produces a more robust model that generalizes well to varying task latencies, overcoming the limitations imposed by smaller datasets at individual fog nodes.

In the subsequent section, we will conduct load testing on a server and collect the test results. By training on historical records, the weights $\boldsymbol{w}$ of the CNN will be learned.

B. Markov Decision Process Formulation

The problem described in Equation (12) can be formulated as a Markov Decision Process (MDP), because given the

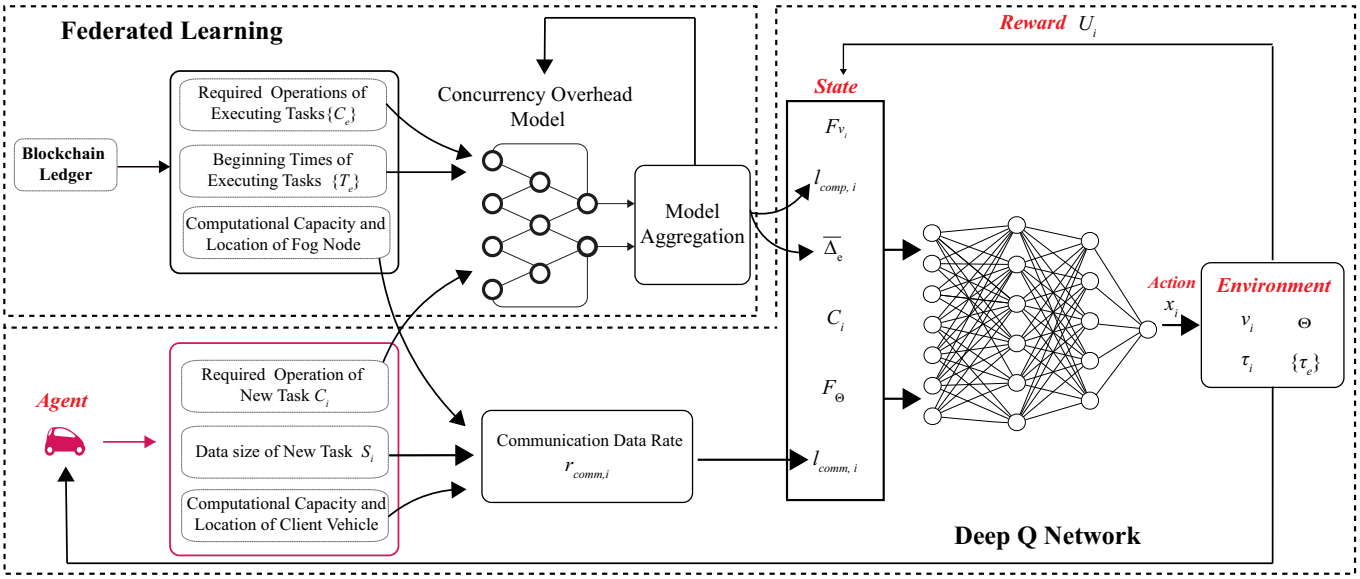


Fig. 2: FL and DQN-Integrated Framework

current state and action, the next state is independent of all the previous states and actions. The model-free MDP refers to an approach in which the agent does not directly learn the environment's transition probabilities or reward functions. Instead, the agent interacts with the environment to acquire a policy that dictates which action to take in a given state.

As shown in Fig. 2, the reward of the offloading policy for task τ_i is determined by the concurrency overhead and the task itself, therefore, in Moscato the transition probabilities are unknown, and we formulate the problem as a model-free MDP. For each vehicle v_i, we can define the state, action, and reward as follows.

- State: For a task hosted on a CV v_i, the vehicle observes the environment from blocks. We denote the state $s = \{\tau_i, Pos_i(t), N_e, \{\tau_e\}, f^i_\Theta(t)\}$.
- Action: Each CV decides whether to offload the task to the fog node or not. We denote the action $a = x_i$. $x_i \in \{0, 1\}$, $x_i = 1$ means the task would be offloaded to the fog node, $x_i = 0$ means the CV would execute the task locally.
- Reward: We use Equation (12) to denote the reward for a task τ_i under the decision $a = x_i$, and we denote reward $r = -(\alpha l_i + \beta \sum_1^{N_e} \Delta_e)$. We try to balance the latency of the new task and the executing tasks.

In our cases, although the exact probabilities are unknown, they are assumed to exist and are estimated through interaction with the environment. Consequently, we employ reinforcement learning to learn the optimal policy for the new task offloading. Within this framework, the CV dynamically assesses the state value or action value, thereby refining its policy through the aggregation of experiential data.

C. DQN for Task Offloading

Given the infinite nature of the state space described in Markov Decision Processes (MDPs), traditional table-based methods like Q-learning and Sarsa are inadequate for addressing this complexity. These methods rely on discrete state representations and become infeasible when faced with the vast or continuous state spaces typical in many real-world applications.

To overcome these limitations, we employ Deep Q Networks (DQN), which utilize neural networks to approximate the Q-value function. Unlike table-based approaches, DQN efficiently manages high-dimensional state spaces by learning directly from a continuous stream of data. This capability allows the agent to generalize across similar states, significantly enhancing its ability to navigate and make decisions in complex environments where the state space is infinite or exceedingly large.

1) Bellman Update Equation: At the core of the DQN algorithm lies the Bellman update equation, which is used to iteratively improve the Q-value function. The Bellman equation for the Q-function is given by:

$$Q(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s', a') \mid s, a\right], \tag{16}$$

where s' represents the next state resulting from taking action a in state s, and a' is the action that maximizes the Q-value in the next state s'. This maximization step is guiding the agent towards the optimal policy.

In the context of DQN, the Bellman update equation is modified to incorporate neural networks, where the Q-value function $Q(s, a; \theta)$ is approximated by a neural network $Q(s, a; \theta)$ with weights θ.

2) Target Netwrok in DQN: A critical aspect of DQN is the introduction of the target network, which plays a key role in stabilizing the training process. The target network $Q(s, a; \theta^-)$ is a copy of the primary Q-network $Q(s, a; \theta)$ (line 1 in Algorithm 1), but its parameters θ^- are updated less frequently—typically by periodically copying the weights from the primary network (line 18 in Algorithm 1). This approach decouples the target values from the rapidly changing Q-values

being learned by the primary network, thereby reducing the risk of divergence during training.

The Q-network network is trained by minimizing the loss function:

$$L(\theta) = \mathbb{E}\left[\left(y - Q(s, a; \theta)\right)^2\right], \tag{17}$$

where $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the target value calculated using the target network. By using the target network, DQN mitigates the issue of unstable Q-value updates, allowing the algorithm to converge more reliably.

In summary, DQN represents a significant advancement over traditional table-based methods, making it possible to tackle problems with large or continuous state spaces. The use of neural networks to approximate the Q-value function, combined with techniques like the Bellman update equation and target networks, empowers the agent to learn and make decisions in environments of unprecedented complexity.

Algorithm 1: Deep Q-Network (DQN) Training Algorithm

Input: Initialized Q-network with random weights θ
Input: Replay memory D of capacity N
Input: Exploration rate ϵ with decay factor ϵ_{decay} and minimum ϵ_{min}
Input: Learning rate α, discount factor γ
Output: Trained Q-network with optimized weights θ

Initialize target network with weights $\theta^- = \theta$;
for *each episode* $= 1$ *to* M **do**
  Initialize environment and get initial state s_0;
  for *each step* t *in episode* **do**
    if *random number* $< \epsilon$ **then**
      Select a random action a_t;
    else
      Select $a_t = \arg\max_a Q(s_t, a; \theta)$;
    Execute action a_t, observe reward r_t and next state s_{t+1};
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory D;
    Sample a random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D;
    Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^-)$;
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to network weights θ;
    Update state $s_t \leftarrow s_{t+1}$;
    if *done* **then**
      Break the loop;
  Periodically update the target network: $\theta^- \leftarrow \theta$;
  Decay exploration rate: $\epsilon \leftarrow \max(\epsilon \cdot \epsilon_{\text{decay}}, \epsilon_{\text{min}})$;
Save the trained model parameters θ;

3) FL and DQN-Integrated algorithm Design: After obtaining the FL-based concurrency overhead model and the DQN-based task offloading model, we can integrate these two models and formulate the FL-DQN algorithm to address the task offloading problem as shown in Fig. 2.

Algorithm 2 outlines the detailed process of the FL-DQN framework, which operates as follows:

- **Step 1: Preprocess Inputs**
  In the first step, the input data, including the required operation of executing tasks $\{C_e\}$, task beginning times $\{T_e\}$, computation capacity F_Θ, and task-specific information such as size S_i, is preprocessed to ensure compatibility with the FL model. This preprocessing stage standardizes and normalizes the input data, which improves the stability of the subsequent FL and DQN computations.
- **Step 2: Federated Learning Inference**
  Using the pre-trained FL model, each node evaluates the computation latency of the new task τ_i and the remaining computation latency of the currently executing task τ_e. The FL model, which has been collaboratively trained using FedAvg on data from multiple fog nodes, generalizes well across diverse task types, ensuring an accurate estimate of latency. The resulting latency values $l_{\text{comp},i}$ and Δ_e serve as crucial inputs to the DQN model, allowing it to assess the current load and forecast potential delays.
- **Step 3: DQN Inference**
  In this step, the combined information, including fog and vehicle computation capacities F_Θ and F_{v_i}, along with task-specific latencies from the FL model, is fed into the trained DQN network. This model processes the complete state information and estimates the Q-values for each possible action (offload or execute locally).
- **Step 4: Final Decision on Task Offloading**
  Based on the DQN output, a final decision is made regarding task offloading.

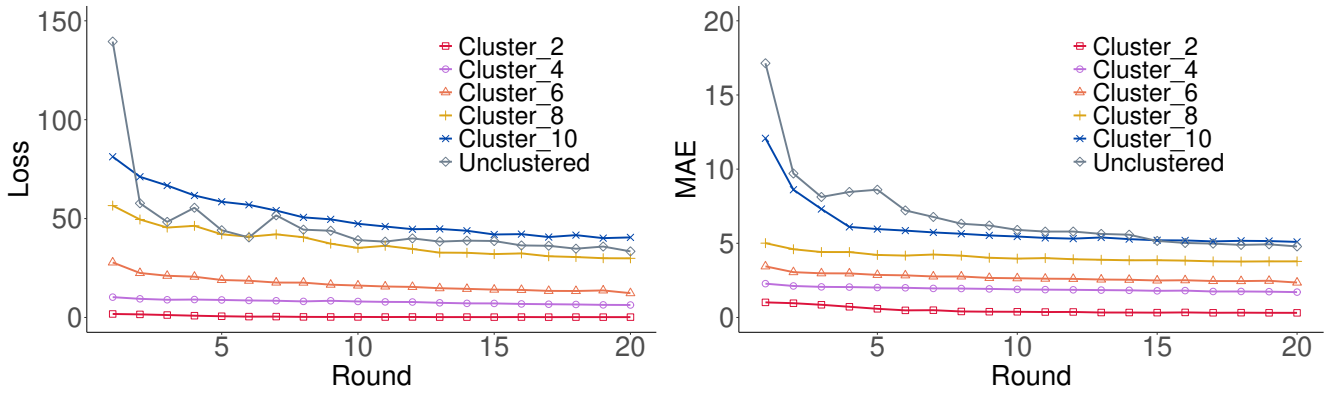
Through this integrated FL-DQN approach, the framework continuously refines its offloading strategy, benefiting from both the collaborative learning capabilities of FL and the decision-making strength of DQN. This integration enables adaptive task management within dynamic vehicular networks, optimizing system performance in complex, latency-sensitive environments.

VI. Simulation

A. Concurrency Overhead Data Collection

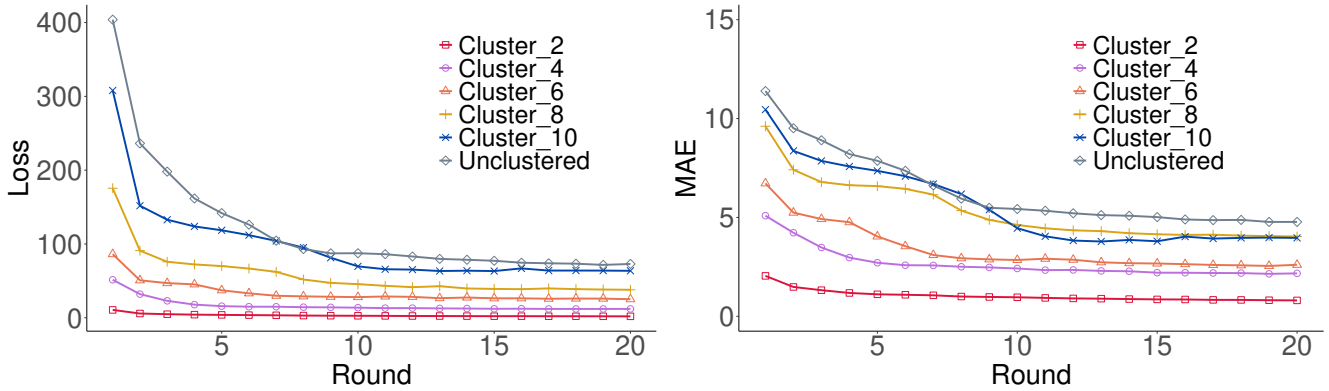
In the context of multi-client environments, fog nodes exhibit varying computational workloads. Collecting the processing latency of offloaded tasks is crucial for modeling concurrency overhead. To accurately represent the diverse operational conditions, we conducted experiments focusing on two types of tasks: image processing tasks and Fibonacci calculation tasks. These tasks were distributed across 5 different clusters classified by the number of tasks in parallel executing (i.e., 2, 4, 6, 8, 10), each representing a unique configuration of fog nodes with similar characteristics.

For the image processing tasks, we utilized a setup that employed the YOLOv5 model [28] to detect objects within a series of images. The data collection process involved running multiple concurrent tasks where each task processed a folder of images from the BDD100K actual roadway dataset [29].



(a) Loss (b) MAE

Fig. 3: Performance over Different Clusters for Image Processing Tasks



(a) Loss (b) MAE

Fig. 4: Performance over Different Clusters for Fibonacci Calculating Tasks

The key metrics captured during these tasks included the start and end times, task latency, CPU frequency, total data size, and total Floating Point Operations (FLOPs). These metrics provided a detailed insight into the computational overhead associated with each task, particularly under different levels of concurrency. Similarly, for Fibonacci calculation tasks, we designed a set of experiments where the size of the Fibonacci sequence to be calculated was randomly determined. Each task's computational load was measured and recorded, offering another dimension of data to be integrated into the overall concurrency overhead model.

The collected data was meticulously processed and organized into CSV files, with each file corresponding to a specific set of concurrent tasks. The processed data and organized in CSV files based on the number of concurrent tasks, have been uploaded to GitHub[1].

B. Concurrency Overhead Model Weight Optimization

In Moscato, we employ a three-layer CNN to train the concurrency overhead model for tasks such as image processing and Fibonacci calculation. In real-world scenarios, the number of tasks assigned to fog nodes within the same zone can vary significantly. For example, some fog nodes may be processing only one task, while others could be handling up to ten tasks concurrently. This variability in task count across fog nodes can lead to substantial differences in the remaining computation latency of executing tasks, which poses a significant challenge for efficient task management and resource allocation in fog computing environments.

To address this challenge, Moscato introduces an innovative approach that clusters fog nodes based on the number of tasks they are concurrently processing inspired by the Clustered Federated Learning (ClusteredFL) [11]. Specifically, we simulate fog nodes within the same zone and categorize them into five clusters. Each cluster corresponds to fog nodes managing 2, 4, 6, 8, 10 executing tasks. By doing so, we standardize the inputs to the CNN model, ensuring that each model instance receives a consistent, fixed-dimensional input. This clustering approach not only simplifies the input structure for the CNN but also allows the model to more effectively learn the impact of varying task loads on computation latency.

The effectiveness of this approach is demonstrated through our experimental results. Fig. 3 shows the performance of different clusters across 20 communication rounds in the context of image processing tasks. Specifically, Fig. 3a presents the loss values, while Fig. 3b shows the Mean Absolute

[1]https://github.com/xinlu99/FedRS-FL

Algorithm 2: FL-DQN Process

Input: Operation of executing tasks $\{C_e\}$, beginning time of executing tasks $\{T_e\}$, computation capacity F_Θ, and location of fog node Pos_Θ.
Input: Required operation of new task C_i, data size of new task S_i, computational capacity F_{v_i} and location Pos_{v_i} of CV.
Output: Offloding decision x_i

Step 1: Preprocess Inputs
Preprocess the input variables C_e, T_e, F_Θ, Pos_Θ, C_i, S_i as necessary for the federated learning model.
Step 2: Federated Learning Inference
Get computation latency of new task τ_i and remaining computation latency of executing task τ_e through FL model. $l_{comp,i}$, Δ_e = FLmodel(C_e, T_e, F_Θ, Pos_Θ, C_i, S_i)
Step 3: DQN Inference
The variables F_{v_i}, $l_{comp,i}$, Δ_e, C_i, F_Θ are input into the trained DQN model;
The DQN model outputs the Q-values for both $x_i = 0$ and $x_i = 1$;
The action with the highest Q-value is selected;
Step 4: Final Decision on Task Offloading
if $x_i == 1$ **then**
    Output: *Offload the task*;
else
    Output: *Processing locally*;

Error (MAE). The results indicate that clustered strategy significantly reduces both loss and MAE compared to the unclustered approach, particularly under fewer task loads, such as Cluster_2. Note that Cluster_2 means the cluster of these fog nodes with two executing tasks. This demonstrates that clustering fog nodes effectively mitigates task concurrency's impact on computation latency, thereby enhancing the accuracy of latency predictions.

For Fibonacci calculation tasks, a similar pattern illustrated in Fig. 4 is observed. Fig. 4a shows the loss values, and Fig. 4b presents the MAE over 20 communication rounds. The clustered approach consistently outperforms the unclustered method, especially in clusters with fewer tasks, reinforcing the robustness of the ClusteredFL across different types of computational tasks.

Furthermore, the detailed performance metrics for image processing tasks are summarized in Table III. Table IIIa focuses on the loss values, comparing the performance of different baselines across various clusters and communication rounds (5, 10, 15, 20). The table shows that the clustered approach achieves lower loss values than the client_max and client_avg baselines, particularly in clusters with fewer tasks. Note that client_max and client_avg refer to the maximum and average values, respectively, trained across fog nodes within the same cluster.

Table IIIb presents the corresponding MAE results for the image processing tasks. It confirms that the clustered approach reduces loss and significantly improves the MAE, especially in clusters with low concurrency tasks like Cluster_2.

Similarly, the performance of different baselines for Fibonacci calculation tasks is summarized in Table IV. Table IVa presents the loss values across clusters, while Table IVb provides the MAE results. The clustered approach consistently outperforms other methods in both loss and MAE, underscoring the effectiveness of the ClusteredFL in handling varied computational tasks in fog computing environments.

These findings suggest that by clustering fog nodes based on concurrency tasks, Moscato can optimize the performance of CNN models, leading to more accurate predictions of computation latency and better resource management in fog computing environments.

C. Performance Analysis

We evaluate the performance of the DQN model in the training process and compare Moscato with other algorithms.

1) Basic setting of model training: The training process employed a learning rate of $\alpha = 0.001$, an initial exploration rate of $\epsilon = 0.2$ with a decay rate $\epsilon_{\text{decay}} = 0.995$, and a minimum exploration rate of $\epsilon_{\text{min}} = 0.1$. The discount factor γ was deliberately set to a low value of $\gamma = 0.1$ to prioritize immediate rewards, as the reward is determined solely by the current state and action.

For simplicity, we set the weight parameter $\alpha + \beta = 1$, according to the Equation (13), the higher α means the higher weight on the new task τ_i, and the task service latency l_i has a bigger influence on the overall system benefits.

The training was conducted with parameters $\alpha = 0.2$, $\alpha = 0.5$, and $\alpha = 0.7$. The model was trained over 100 episodes, each consisting of 3,000 steps. The mean squared error (MSE) was used as the loss function to assess the model's performance.

2) Training Performance Analysis: Fig. 5 shows the convergence trend of the DQN training process. Fig. 5a illustrates the evolution of the Mean Loss over 100 episodes, and Fig. 5b displays the corresponding Reward progression.

Fig. 5a shows that all configurations of α demonstrate a rapid decrease in Mean Loss during the initial episodes, signifying efficient learning and quick adjustment of the DQN model's parameters. The loss values stabilize as training progresses, indicating that the model converges effectively for all the α. Among the three settings, $\alpha = 0.7$ converges to the lowest Mean Loss, demonstrating the most efficient learning process in minimizing the prediction error.

Fig. 5b provides insight into the Reward accumulation process, where we can observe that the training with $\alpha = 0.3$ results in a relatively slower increase in rewards compared to higher values of α. However, it eventually stabilizes around a reward of approximately -30.5. On the other hand, training with $\alpha = 0.7$ reaches a higher reward level more rapidly, stabilizing near -20.0, indicating a more favorable outcome. This trend suggests that a higher α not only facilitates faster convergence but also leads to better policy development, yielding higher rewards during training.

TABLE III: Performance over Different Algorithms for Image Processing Tasks

(a) Loss

Algorithm / Round / Cluster	5			10			15			20		
	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg
Cluster_2	0.9301	1.8364	1.076	0.2971	0.3741	0.321	0.2144	0.7137	0.3247	0.1891	0.4249	0.2533
Cluster_4	8.8552	12.8549	10.177	8.0786	12.0234	9.1256	7.1063	9.1335	7.7133	6.3093	7.8014	6.9737
Cluster_6	18.9544	24.1844	21.7709	16.1467	27.2259	19.1628	14.0702	22.5733	16.9997	12.2896	16.6686	13.5846
Cluster_8	42.047	75.0362	52.6577	35.1709	60.8498	42.5833	32.0477	44.51	37.0532	29.8945	40.0226	33.9443
Cluster_10	89.0458	146.0328	104.1231	72.1161	84.8495	79.4395	63.858	117.0226	76.8773	61.6226	70.5312	65.7743

(b) MAE

Algorithm / Round / Cluster	5			10			15			20		
	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg
Cluster_2	0.6952	1.0626	0.7505	0.4071	0.4596	0.4174	0.3546	0.6589	0.4255	0.3361	0.4961	0.3803
Cluster_4	2.0182	2.2802	2.1232	1.8926	2.211	2.001	1.7955	1.9567	1.8628	1.7092	1.8237	1.7769
Cluster_6	2.8688	3.2165	3.0832	2.6433	3.5253	2.8805	2.4929	3.2216	2.733	2.3531	2.7684	2.4659
Cluster_8	4.2159	5.8249	4.7337	3.969	5.1935	4.3116	3.8588	4.3539	4.0664	3.7814	4.2533	3.9558
Cluster_10	5.9593	8.0546	6.4306	5.4583	6.085	5.7266	5.2032	7.366	5.6606	5.0944	5.4417	5.2415

TABLE IV: Performance over Different Algorithms for Fibonacci Calculating Tasks

(a) Loss

Algorithm / Round / Cluster	5			10			15			20		
	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg
Cluster_2	3.8956	6.0555	4.6964	2.6965	4.2076	3.5242	2.1421	5.1734	3.0463	1.9488	3.7818	2.7911
Cluster_4	15.7605	23.9079	19.8047	13.6602	20.5382	16.9699	12.1641	16.9058	15.2997	11.9994	21.0481	15.4959
Cluster_6	37.2145	58.8623	43.6912	28.0041	45.9131	34.9085	26.3545	37.3943	30.4633	25.1432	48.2681	32.8248
Cluster_8	70.0654	83.5047	75.9681	45.5296	57.6301	50.4172	39.1002	54.2278	45.7333	37.9169	52.5926	44.4787
Cluster_10	118.5414	142.2743	128.4888	69.6441	106.2219	81.8247	63.2906	81.4797	73.1295	63.6254	82.4685	73.9758

(b) MAE

Algorithm / Round / Cluster	5			10			15			20		
	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg	Clustered_FL	Client_Max	Client_Avg
Cluster_2	1.1133	1.3382	1.2342	0.9639	1.1683	1.0755	0.8561	1.108	0.9705	0.8053	1.0198	0.9179
Cluster_4	2.7086	3.328	2.9537	2.4198	2.9296	2.6317	2.2071	2.5561	2.4352	2.1718	2.6771	2.411
Cluster_6	4.0386	5.2194	4.4158	2.8492	4.1611	3.4591	2.6708	3.3233	3.0721	2.6054	4.0936	3.2713
Cluster_8	6.5836	7.0154	6.7616	4.6193	4.9709	4.8009	4.1478	4.9828	4.4438	4.0354	4.5469	4.2849
Cluster_10	8.6537	9.5432	8.8883	5.243	6.5647	5.6418	4.4646	5.6984	5.0001	4.6646	5.7162	5.0487

3) Performance Comparison: We compare the performance of Moscato with other algorithms by analyzing the sum latency for different tasks as shown in Fig. 6. The two subplots present the results for two different scenarios: Image Processing and Fibonacci Calculation. The algorithms used are listed below.

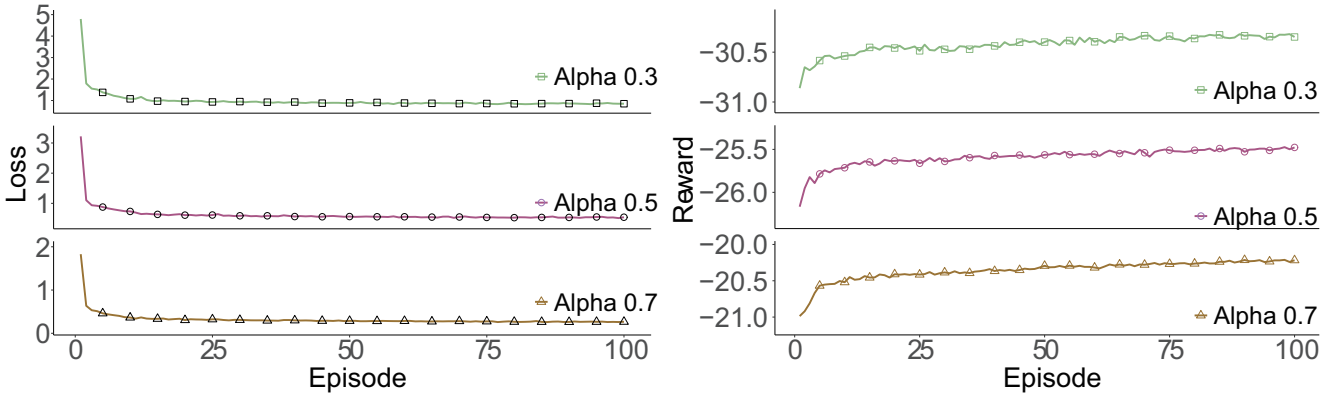
- **M-3:** Moscato with $\alpha = 0.3$.
- **M-5:** Moscato with $\alpha = 0.5$.
- **M-7:** Moscato with $\alpha = 0.7$.
- **Random:** This strategy involves randomly selecting an offloading action from $\{0, 1\}$.
- **Adaptive:** The adaptive strategy adjusts the offloading probability based on the number of tasks currently being processed on the fog node. For instance, if the maximum number of tasks that can be simultaneously processed on a fog node is 10 and the number of tasks run on the fog node is 1, then the probabilities that the client vehicle offloads the task and process locally are 90% and 10%, respectively [30].

In Fig. 6, we use the average latency to assess the algorithms' performance in minimizing the latency of the new task and executing tasks across two scenarios. Each scenario involves a varying number of parallel executing tasks (i.e., 2, 4, 6, 8, 10).

In Fig. 6 , the results for M-3, M-5, and M-7 demonstrate a clear trade-off between new task latency and executing task latency controlled by the alpha parameter. As α increases, the latency of new tasks decreases while the latency of executing tasks increases. This occurs because a larger α places a higher priority on new tasks, as indicated by Equation (13). Prioritizing the completion of new tasks sooner results in a higher system reward.

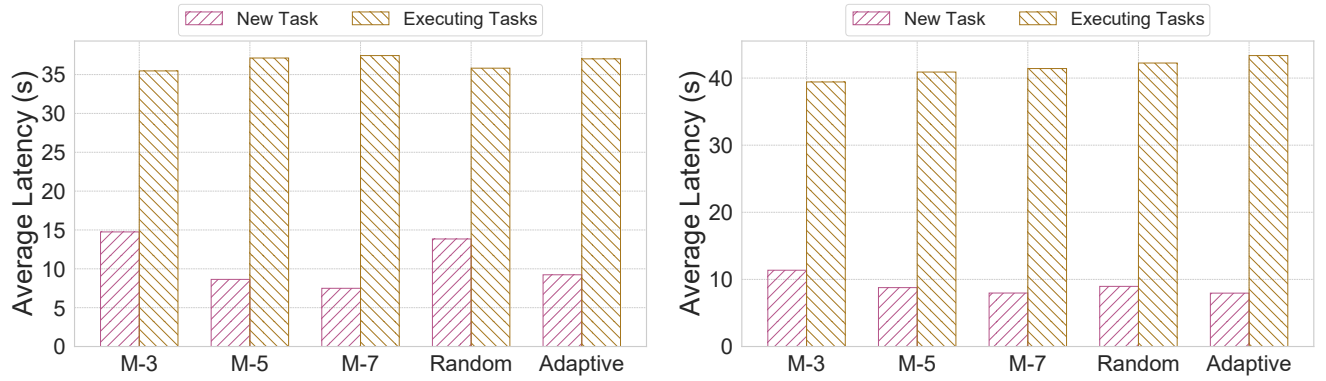
Fig. 6a illustrates that M-3 and the Random strategy perform poorly in managing new tasks, while M-7 demonstrates the best performance in minimizing new task latency. Specifically, M-7 achieves the most significant reduction in new task latency, outperforming the Random strategy by approximately 45.8% and the Adaptive strategy by 18.8%. The figure emphasizes offloading greatly reduces processing time for these resource-intensive tasks, which would otherwise face much longer delays on less powerful local devices.

Fig. 6b demonstrate that the M-7 and Adaptive strategies significantly reduce new task latency, highlighting their effectiveness in handling tasks that require immediate attention. Specifically, M-7 achieves an 11.0% reduction in new task



(a) Loss (b) Reward

Fig. 5: Convergence over Different α for DQN Training Process



(a) Image Processing (b) Fibonacci Calculation

Fig. 6: Comparison Between Different Algorithms.

latency compared to the Random strategy, while also achieving a 1.9% decrease in executing task latency. That comparison clearly illustrates the advantages of the Moscato approach, particularly when optimized with appropriate α values to meet varying prioritization needs.

VII. Conclusion

In this paper, we propose Moscato, a distributed blockchain-based vehicular task offloading strategy designed to optimize the balance between the latency of new offloading tasks and that of executing tasks. By integrating DQN into the task offloading decision-making process and utilizing FL to expand the training dataset, Moscato dynamically selects the optimal offloading decisions, adapting to varying traffic conditions and dynamic task profiles. Our extensive experiments, conducted using a real-world dataset of task processing latency and server workload, demonstrate that Moscato significantly decreases the service latency of both new and executing tasks compared to other methods.

References

[1] F. Yang, S. Wang, J. Li, Z. Liu, and Q. Sun, "An overview of internet of vehicles," *China communications*, vol. 11, no. 10, pp. 1–15, 2014.

[2] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez, "Internet of vehicles: architecture, protocols, and security," *IEEE internet of things Journal*, vol. 5, no. 5, pp. 3701–3709, 2017.

[3] C. Zhu, G. Pastor, Y. Xiao, and A. Ylajaaski, "Vehicular fog computing for video crowdsourcing: Applications, feasibility, and challenges," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 58–63, 2018.

[4] Y. Xiao and C. Zhu, "Vehicular fog computing: Vision and challenges," in *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*. IEEE, 2017, pp. 6–9.

[5] C. Zhu, J. Tao, G. Pastor, Y. Xiao, Y. Ji, Q. Zhou, Y. Li, and A. Ylä-Jääski, "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4150–4161, 2019.

[6] C. Zhu, Y.-H. Chiang, Y. Xiao, and Y. Ji, "Flexsensing: A qoi and latency-aware task allocation scheme for vehicle-based visual crowdsourcing via deep q-network," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7625–7637, 2021.

[7] C. Zhu, X. Xie, R. Zhang, R. Li, B. Zhu, and X. Bu, "Multi-task communication resource allocation for mimo-based vehicular fog computing," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 1, pp. 1115–1128, 2024.

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Satoshi Nakamoto*, 2008.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[11] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy con-

straints," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 8, pp. 3710–3722, 2020.
[12] C. Chen, Y. Zeng, H. Li, Y. Liu, and S. Wan, "A multihop task offloading decision model in mec-enabled internet of vehicles," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3215–3230, 2023.
[13] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 2169–2182, 2023.
[14] M. Cui, S. Zhong, B. Li, X. Chen, and K. Huang, "Offloading autonomous driving services via edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 535–10 547, 2020.
[15] Z. Xiao, J. Shu, H. Jiang, G. Min, H. Chen, and Z. Han, "Perception task offloading with collaborative computation for autonomous driving," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 2, pp. 457–473, 2023.
[16] B. Yang *et al.*, "Edge intelligence for autonomous driving in 6g wireless system: Design challenges and solutions," *IEEE Wireless Communications*, vol. 28, no. 2, pp. 40–47, 2021.
[17] Y. Sun *et al.*, "Adaptive learning-based task offloading for vehicular edge computing systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3061–3074, 2019.
[18] S. Jain *et al.*, "Blockchain and autonomous vehicles: Recent advances and future directions," *IEEE Access*, vol. 9, pp. 130 264–130 328, 2021.
[19] H. N. Abishu, A. M. Seid, Y. H. Yacob, T. Ayall, G. Sun, and G. Liu, "Consensus mechanism for blockchain-enabled vehicle-to-vehicle energy trading in the internet of electric vehicles," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 946–960, 2022.
[20] R. Jabbar, N. Fetais, M. Kharbeche, M. Krichen, K. Barkaoui, and M. Shinoy, "Blockchain for the internet of vehicles: How to use blockchain to secure vehicle-to-everything (v2x) communication and payment?" *IEEE Sensors Journal*, vol. 21, no. 14, pp. 15 807–15 823, 2021.
[21] C. Li, Y. Fu, F. R. Yu, T. H. Luan, and Y. Zhang, "Vehicle position correction: A vehicular blockchain networks-based gps error sharing framework," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 898–912, 2021.
[22] H. Xiao, J. Zhao, Q. Pei, J. Feng, L. Liu, and W. Shi, "Vehicle selection and resource optimization for federated learning in vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 073–11 087, 2022.
[23] J. H. Chen, M. R. Chen, G. Q. Zeng, and J. S. Weng, "Bdfl: A byzantine-fault-tolerance decentralized federated learning method for autonomous vehicle," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 8639–8652, 2021.
[24] X. Li, L. Lu, W. Ni, A. Jamalipour, D. Zhang, and H. Du, "Federated multi-agent deep reinforcement learning for resource allocation of vehicle-to-vehicle communications," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 8, pp. 8810–8824, 2022.
[25] H. Chai, S. Leng, Y. Chen, and K. Zhang, "A hierarchical blockchain-enabled federated learning algorithm for knowledge sharing in internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 3975–3986, 2021.
[26] S. D. Okegbile, J. Cai, H. Zheng, J. Chen, and C. Yi, "Differentially private federated multi-task learning framework for enhancing human-to-virtual connectivity in human digital twin," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, pp. 3533–3547, 2023.
[27] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
[28] G. Jocher, "ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements," https://github.com/ultralytics/yolov5, Oct. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4154370
[29] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF CVPR*, 2020, pp. 2636–2645.
[30] C.-L. Huang, Y. P. Fallah, R. Sengupta, and H. Krishnan, "Adaptive intervehicle communication control for cooperative safety systems," *IEEE network*, vol. 24, no. 1, pp. 6–13, 2010.