# Fast Adaptive Task Offloading and Resource Allocation via Multiagent Reinforcement Learning in Heterogeneous Vehicular Fog Computing

Zhen Gao, Lei Yang, and Yu Dai

***Abstract*—In vehicular fog computing, task offloading enables mobile vehicles (MVs) to offer ultralow latency services for computation-intensive tasks. Nevertheless, the edge server (ES) may have a high load when a large number of MVs offload their tasks to it, causing many tasks either experience long processing times or being dropped, particularly for latency-sensitive tasks. Moreover, most existing methods are largely limited to training a model from scratch for new environments. This is because they focus more on model structures with fixed input and output sizes, impeding the transfer of trained models across different environments. To solve these problems, we propose a decentralized task offloading method based on transformer and policy decoupling-based multiagent actor–critic (TPDMAAC). We first introduce a transformer-based long sequence forecasting network (TLSFN) for predicting the current and future queuing delay of ESs to solve uncertain load. Second, we redesign the actor-network using transformer-based temporal feature extraction network (TTFEN) and policy decoupling network (PDN). TTFEN can adapt to various input sizes through a transformer that accepts different tokens we build from the raw input. PDN provides a mapping between the transformer-based embedding features and offloading policies utilizing self-attention mechanism to address various output dimensions. Finally, the experiments on two real-world data sets show that TPDMAAC can quickly adapt to a new environment. And compared to existing algorithms, TPDMAAC reduces the system cost by 11.01%–12.03% as well as improves task completion rates by 10.45%–13.56%.**

***Index Terms*—Mobile-edge computing (MEC), multiagent reinforcement learning (MARL), task offloading, vehicular fog computing (VFC).**

## I. Introduction

WITH the growth of the Internet of Things (IoT), intelligent in-vehicle applications, such as multimedia entertainment and driving behavior monitoring, have been extensively applied in intelligent vehicles [1]. Nonetheless, many applications, especially real-time navigation and road-sensing, require considerable computational resources to satisfy ultralow task execution delay. For vehicles with constrained resources, this is a huge challenge. To address this challenge, cloud offloading has been proposed, where computing tasks are handled in cloud centers to get sufficient computational resources. Nevertheless, cloud offloading adds additional task processing delay and energy consumption due to long-distance transmission [2]. To satisfy the low-latency demands in task offloading, multiaccess edge computing (MEC) (or mobile-edge computing) [3], [4] is introduced in vehicular networks, i.e., vehicular fog computing (VFC) [5], [6]. In VFC systems environment, the latency and energy consumption of task processing can be greatly reduced by offloading the tasks to the adjacent edge servers (ESs) for processing. VFC is considered as a promising solution to handle computationally intensive tasks in vehicular networks [7], [8], [9].

Recently, some task offloading algorithms have been proposed in VFC system environment. For example, Zhao et al. [10] proposed a game theory-based task offloading approach to optimize system costs, including execution time, energy consumption, and ES lease prices. Kazmi et al. [11] proposed a task offloading solution based on reinforcement learning (RL) that enables mobile vehicles (MVs) to offload their tasks to either the ESs or the nearby MVs and minimizes energy consumption and latency of task execution. In these works, the computing resources that each MV gets from the ESs are irrelevant to the amount of tasks offloaded to the ESs, or it directly assumes that each ES has sufficient computational resources. Nevertheless, in practice, the computing resources of ES may be constrained [12], [13]. And the computing resources MV gains from ES are closely related to the load of ES (i.e., the number of offloaded tasks that the ES is processing). Specifically, when the traffic flow is dense (e.g., morning rush or evening rush), a large number of MVs offload their tasks to ES for execution. Many ESs accept a large number of offloaded tasks and need to process them simultaneously. Therefore, the load on the ES may be high. And many offloaded tasks may experience particularly long queuing delays, or even be dropped when the task's maximum tolerated delay expires. Without knowing the load on the ESs, it is a challenge for MVs to make the optimal offloading policy.

In addition, energy conservation is becoming critical due to the large-scale adoption of new resources and energy-consuming services in the intelligent vehicles [10], [11], [14]. For example, augmented reality (AR) and autonomous driving in intelligent vehicles have significantly increased the energy

Manuscript received 28 June 2022; revised 25 September 2022, 23 October 2022, and 16 November 2022; accepted 2 December 2022. Date of publication 9 December 2022; date of current version 7 April 2023. This work was supported by the National Key Research and Development Plan of China under Grant 2021YFF0901205. *(Corresponding author: Lei Yang.)*

Zhen Gao and Lei Yang are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110000, China (e-mail: gaozhen610351@163.com; yanglei@mail.neu.edu.cn).

Yu Dai is with the College of Software, Northeastern University, Shenyang 110000, China (e-mail: daiy@swc.neu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2022.3228246

and resource demand of vehicles. Furthermore, in a dynamic VFC system environment, this will require more resources and energy to perform the task due to the movement of the vehicle and rapid changes in the environment. One potential scheme to solve these problems is to offload resource-hungry (i.e., computational resources and energy) tasks to the ES for execution.

Recently, RL has demonstrated significant potential for optimizing decision-making problems in dynamic environments, such as traffic light control and robotics. Through repeated interaction of the agent (i.e., traffic light and robotics) with the environment, the agent learns the policy of how to maximize the cumulative reward. Similarly, task offloading and resource allocation are also decision-making problems. This is because we also need to find the best task offloading and the resource allocation policies to satisfy system cost minimization under the task deadline and resource constraints. For task offloading in VFC, ES or MV is usually considered as the agent responsible for making task offloading and resource allocation policies. The agent's state/observation typically contains information about the task, wireless channel information, the MVs, the ESs, etc. The optimization goal is to minimize the system cost; thus the reward is usually set to be inversely proportional to the system cost.

For example, some centralized task offloading algorithms based on RL are proposed. Zhang et al. [15] proposed a software-defined networking (SDN)-based load-balancing task offloading scheme to minimize the processing delay of the tasks. Zhao et al. [16] proposed an SDN-enabled unmanned aerial vehicles (UAVs)-assisted vehicular task offloading method to minimize the system cost. Shi et al. [8] developed a V2V partial computing offloading scheme and evaluated the service availability of adjacent MVs based on free computing resources and MV mobility. Shi et al. [17] proposed an RL-based vehicle-to-vehicle task offloading method in blockchain-enabled vehicular networks to minimize the energy consumption and time delay in performing the tasks. These centralized offloading schemes typically model the entire VFC system to learn the joint task offloading and resource allocation (JTORA) policies for all MVs. The input to these algorithms is the state of the entire VFC system and the output is the joint actions of all MVs. However, these centralized approaches typically require a central network (e.g., SDN) to collect state information for the entire VFC system, leading to increased communication burden. Further, a failure of the centralized network will lead to the failure of the entire VFC system.

Other decentralized task offloading algorithms have been proposed taking into account the load of ESs. However, designing these approaches is challenging since the available resources of the ESs, the load of the ESs, the states information of other MVs, and offloading decisions of other MVs cannot be perceived from the MV's own perspective. Therefore, it is difficult for MV to make the best offloading policy in a decentralized manner by knowing only part of the information of the MEC system. This is because the offloading decision of the MV is closely related to the available resource and load of the ES, as well as the states information and offloading decisions of other MVs.

To solve this challenge, some task offloading algorithms have been proposed. Zhu et al. [18] studied vehicular computation offloading problem in IoT and proposed a multiagent RL (MARL)-based decentralized computation offloading to minimize the task execution latency. This approach assumes that the offloading decision of MV is not affected by the load of ESs. However, when the load of ES is high, this usually leads to a lot of latency-sensitive tasks being abandoned. Jia et al. [19] proposed an RL-based queuing delay-aware decentralized task offloading solution in collaborative vehicular networks. Hazarika et al. [14] proposed a priority-sensitive RL-based decentralized task offloading and resource allocation solution in the vehicles network to maximize the average utility. These methods assume that the ESs or other MVs have sufficient computation, transmission resources, and energy. However, the ES may have limited resources when a large number of MVs offload their tasks to it, causing many tasks either experience long processing times or being dropped. Thus, to cope with the task offloading problem, it is important to develop an efficient offloading solution by considering the limited resources (i.e., computational resources, transmission resources, and energy). Zhu et al. [20] considered the uncertain communication resources in ES and proposed a decentralized power allocation method for vehicular computation offloading. This method assumes that MV can make task offloading decisions based on the channel state of the previous time slot. However, in RL, the agent makes decisions based on the current state. This approach may result in a suboptimal task offloading solution.

Moreover, these approaches [5], [8], [10], [11], [14], [15], [17], [18], [19], [20] are greatly restricted to training a network model from scratch for each new VFC system environment. This is because these methods focus more on model structures with fixed input and output sizes, impeding the transfer of trained agents across different environments (e.g., 1000 ESs and 500 MVs or 800 ESs and 300 MVs multi-MVs multi-ES VFC environment). Specifically, when the trained model faces a new VFC system environment, the number of MVs, the number of tasks, the number of ESs, and the network conditions may change [21], [22]. For example, the change of the number of ESs will cause the change of the offloading policy dimension (i.e., the output of the model). The changes in the number of MVs and the number of ESs will lead to changes in the number of agents (i.e., the MV) and the states/observations space dimension of the agents (i.e., the input of the model). Thus, we have to collect new data to retrain the proposed model to adapt it to the new VFC system environment, which implies a restriction on its transfer capability [23].

To solve this problem, some methods have been proposed in MEC system environments. For instance, Zhang et al. [21] presented a meta-RL-based task offloading scheme to improve the transfer ability of the whole model. Qu et al. [23] proposed a meta-RL-based offloading algorithm to quickly adapt to the new environment and can flexibly achieve the optimal offloading policy in MEC system environment. These algorithms are typically deployed to ESs and are responsible for handling task offloading requests and task scheduling for all mobile devices (MDs). However, with VFC networks becoming more complex and the increasing number of MVs, these approaches will suffer from a huge space of states and actions, resulting in the difficulty in convergence.

## A. Motivations and Contributions

In this article, a decentralized task offloading method based on transformer and policy decoupling-based multiagent actor–critic (TPDMAAC) is proposed. This method solves the unknown load of the ESs and fast adaptation in VFC system environments.

First, although the MV can provide ultralow latency services for latency-sensitive tasks via task offloading, the ES may have long task queuing delays when the traffic is heavy. This situation will cause many tasks either experience long processing times or being dropped. Therefore, without knowing the current queuing delays on the ESs, it will be a challenge for MVs to make the optimal offloading policy in a decentralized approach. Moreover, most existing task offloading methods, such as [19], [24], and [25], employ queues to simulate task execution and scheduling. However, these methods usually result in future potential processing competition among the continuously arriving tasks. The MVs make offloading policy without considering potential future competition, often resulting in additional delays or even task processing failures since the imbalanced distribution of task offloading requests. And it is a nontrivial work to incorporate the future competition since the MVs cannot know the precise amount of forthcoming tasks in ESs.

To solve the above challenges, we introduce a transformer-based long sequence forecasting network (TLSFN) based on the lightweight transformer [26], [27] and its excellent performance in long sequence prediction [28], [29]. TLSFN can predict the current and future tasks queuing delay of the ESs (e.g., the task queuing delay for the next 30–50 time slots in ESs) by employing long-period historical data to manage the future task processing competition information. Thus, this enables MV agents to learn task offloading policies to maximize long-term returns with future knowledge. Moreover, some works [24], [30] assume that the MDs are randomly distributed around the ESs as well as the MDs and ESs are always connected. However, in practice, it is possible for MDs to leave the service area of ES and disconnect due to the mobility of MDs. These methods usually do not fit the actual scenario. Different from these works, we consider the impact of MV mobility based on the trajectory of the real vehicle. And we use a long-sequence prediction model to predict the current and future period load of the ESs. This method can not only integrate the impact of current the ESs load on task execution but also integrate the impact of the ESs load on task execution in the future, greatly improving the accuracy of the offloading strategy.

Second, to enable the proposed method to quickly adapt to the new VFC environments, we attempt to design a general MARL framework for tasks with a variety of model input and output requirements. We redesign the actor-network using transformer-based temporal feature extraction network (TTFEN) and policy decoupling network (PDN). TTFEN can adapt to various input sizes through a transformer that accepts different tokens we build from the raw input. This is because the transformer architecture can accept various tokens by converting variable-length inputs into fixed dimensions through the self-attention mechanism similar to these works [31], [32]. "raw input" indicates the state/observation of the agent (i.e., the MV). "build" indicates an embedding operation, i.e., the observation is embedded into a token, which is a vector. PDN provides a mapping between the transformer-based embedding features and offloading policies utilizing self-attention mechanism to handle different model outputs. In this way, the proposed model can adapt to different VFC system environments and the trained model can quickly adapt to the new VFC system environment through fine tuning. For instance, when new MVs enter the VFC system environments (e.g., in the northeastern part of Chengdu city [30.653°N $\sim$ 30.705°N and 104.042°E $\sim$ 104.122°E)], the trained actor-network in ES can be downloaded to the new MVs, enabling them to quickly adapt to the new environment after training with a small number of samples. Moreover, TPDMAAC also implements the transfer of learned agents over tasks across diverse VFC system scenarios (e.g., the transfer between different districts of the city [e.g., Jinjiang, Qingyang, and Jinniu districts in Chengdu)]. The trained TPDMAAC agent can quickly adapt to new districts of the city. The following are the major contributions made by this article.

1) *The Dynamic Joint Task Offloading and Resource Allocation in VFC System:* The dynamic JTORA optimization problem is modeled as a decentralized hybrid cooperative–competitive multi-MV agent behavior coordination problem with the objective of selecting the optimal computing node (i.e., local MV or ES) and allocating the corresponding resources (i.e., transmission power, transmission channels, and computing resources) for the arrival task. The focus of this article is to address the dynamically unknown load of ESs, as well as to quickly adapt to new VFC system environments while reducing task dropout rates and average system costs.
2) *TPDMAAC-Based Task Offloading Algorithm:* As a variant of the MADDPG [33] algorithm, the TPDMAAC agent has two advantages in the VFC system. Specifically, we first introduce a TLSFN to solve the uncertain load problem of ES by predicting the current and future queuing delays of ES. Second, we redesign the actor-network using TTFEN and PDN. TTFEN can adapt to various input sizes through a transformer that accepts different tokens we build from the raw input. PDN provides a mapping between the transformer-based embedding features and offloading policies utilizing self-attention mechanism to address various output dimensions.
3) *Performance Evaluation:* In TPDMAAC, we integrate the centralized training and distributed execution (CTDE) algorithm. Specifically, we consider that Cloud Computing Center first trains the TPDMAAC model in a centralized manner. Then, each MV agent conducts distributed decisions about JTORA policies using the

trained TPDMAAC model. Finally, we use two real-world data sets to verify our experiments. Compared to the existing algorithms, extensive experiments show that TPDMAAC can quickly adapt to new VFC system environments and reduce task drop rates while reducing average system costs.

### B. Organization

The remaining portion of this article is structured as follows. In Section II, the related works are reviewed. In Section III, the system model is presented. In Section IV, the TPDMAAC-based task offloading algorithm is introduced. The experimental results are discussed in Section V, and Section VI summarizes this article.

## II. Related Works

In this section, we survey the literature works related to dynamic task offloading and resource allocation methods in VFC. These methods can be divided into traditional optimization-based and RL-based algorithms.

### A. Traditional Optimization Algorithm-Based Solutions

To reduce energy consumption, several works have been proposed that typically map the task offloading and resource allocation problem to a mixed-integer nonlinear programming problem. For example, Liu et al. [34] investigated the energy-aware task allocation problem in vehicular fog networks and presented task allocation solutions to minimize system cost. Feng et al. [35] formulated the joint optimization of task partitioning ratios and user association as a mixed-integer programming problem to minimize the average latency of all users. However, these solutions often require numerous mathematical operations, resulting in poor convergence. Moreover, they depend significantly on accurate mathematical modeling and expertise, which is inappropriate for a dynamic VFC system environment.

Moreover, several works studied task assignment in VFC system environment to minimize the delay of task offloading and employed heuristic algorithms to solve the task offloading and resource allocation optimization problem. Such as, Zhu et al. [36] investigated the task assignment problem with the aim of minimizing the average task execution delay and reducing quality loss. Chen et al. [37] formulated the task execution delay optimization model as min–max problem and proposed a particle swarm optimization-based method. However, heuristic algorithm-based solutions are typically inefficient in terms of the time required to find a best result, which is difficult to fulfill the requirements of in-vehicle applications for the ultralow delay.

### B. RL Algorithm-Based Solutions

The RL-based solution is a potential way to solve the dynamic task offloading and resource allocation problem in VFC system environment. These methods learn the best strategies and generate them quickly without depending on the specialized expertise or precise mathematical representations of the VFC system model. These solutions are divided into single-agent RL-based and multiagent RL-based schemes. In terms of algorithm principle, we further divide them into value-based and policy gradient-based algorithms.

*1) Q-Value-Based Single-Agent RLs:* Luo et al. [38] studied the task scheduling problem in VFC and proposed a deep $Q$ network (DQN)-based task scheduling scheme. In order to maximize users' Quality of Experience (QoE), Ning et al. [39] employed the RL algorithm to handle the combined optimization problem of job scheduling and resource allocation in VFC. The above RL-based solutions perform well and do not require prior knowledge of the VFC system model. Specifically, these value-based approaches typically first learn the value function in each state and then choose the action according to the greatest $Q$-value as the optimal strategy. However, the *Argmax* operation cannot be achieved if one is faced with a continuous action (i.e., there are an infinite number of actions [e.g., resource allocation or price determination)]. Therefore, these methods are not suitable for dynamic resource allocation.

*2) Policy Gradient-Based Single-Agent RLs:* Ke et al. [40] developed a task offloading algorithm in a heterogeneous VEC to find the best balance between energy usage and data transfer latency. In the VFC environment, Zhan et al. [41] researched the task offloading scheduling problem and presented an RL-based solution to deal with the dynamic environment and vast state space. While these solutions are well suited for dynamic task offloading and resource allocation problems, there is a tacit assumption that neural networks can automatically decouple observations of directly concatenated MV agents to obtain the optimal mapping between the entire observation and offloading policies. This assumption ignores the discrepancy of feature information that typically leads to many ineffective policies.

The above approaches [8], [25], [38], [39], [40], [41] achieve some advantages on the task offloading and resource allocation problem. These approaches formulate the task offloading and resource allocation problem with the single-agent setting. However, they have difficulty in modeling the complex interactions between MVs and between MVs and ESs, and thus oversimplify the dynamic task offloading requests and processing, which do not match realistic VFC system scenarios.

*3) Q-Value-Based Multiagent RLs:* Alam and Jamalipour [42] proposed an MARL-based algorithm for task offloading in MEC Internet of Vehicles (IoV). Zhang et al. [43] investigated joint communication and computation resource allocation in VFC system environments. These approaches first learn the value function $Q$ for each state and then obtain the best offloading policies according to the greatest $Q$ value. However, the *Argmax* operation cannot be achieved if one is faced with a continuous action. In practice, most scenarios involve a mixture of continuous–discrete action spaces. Therefore, these methods are not suitable for dynamic resource allocation.

*4) Policy Gradient-Based Multiagent RLs:* Several recent works attempt to apply MARL to the problem of task offloading and resource allocation in VFC, e.g., MADDPG.

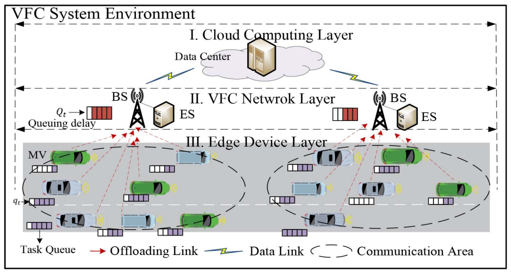

Fig. 1. VFC network model.

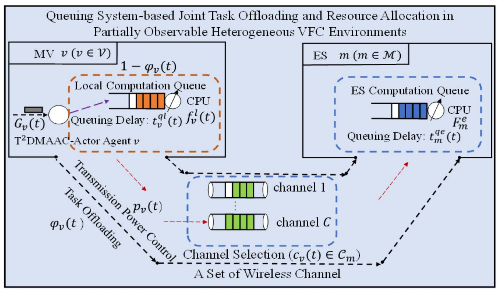
Queuing System-based Joint Task Offloading and Resource Allocation in Partially Observable Heterogeneous VFC Environments
MV $v$ ($v \in \mathcal{V}$)
ES $m$ ($m \in \mathcal{M}$)
$1-\varphi_v(t)$
Local Computation Queue
CPU
Queuing Delay: $t_v^{ql}(t)$ $f_v^l(t)$
$G_v(t)$
T²DMAAC-Actor Agent $v$
ES Computation Queue
CPU $F_m^e$
Queuing Delay: $t_m^{qe}(t)$
Transmission Power Control
Task Offloading
$p_v(t)$
$\varphi_v(t)$
channel 1
channel $C$
Channel Selection ($c_v(t) \in \mathcal{C}_m$)
A Set of Wireless Channel

Fig. 2. Task queuing system of VFC with an MV $v \in \mathcal{V}$, an ES $m \in \mathcal{M}$, and a set of wireless channels $c_v(t) \in \mathcal{C}_m$.

For example, Peng and Shen [44] studied task offloading and resource management in MEC and UAV-assisted vehicle networks and proposed an MADDPG-based approach to achieve higher latency/QoE satisfaction rates. Although the approach can model cooperative–competitive relationships between MVs well and are suited for dynamic task offloading and resource allocation problems, this method is substantially constrained to train a model from scratch for each new multi-MVs multi-ESs VFC environment, resulting in significant exploration costs.

## III. System Model

### A. Network Model

As shown in Fig. 1, the VFC system environment includes several MVs, ESs, and a Cloud Computing Center (i.e., the Data Center). Further, the VFC network is divided into three layers. First, at the Cloud Computing Center layer, we assume that the Data Center is with strong computing and transmission capability. This layer is mainly responsible for model training. Second, the VFC Network Layer exists with a number of base stations (BSs) and ESs that are responsible for handling offloading requests from MVs. Similar to these works [24], [25], we assume that the ES has limited computational power. The ESs are represented as $\mathcal{M} = \{1, 2, \ldots, M\}$. Finally, the Edge Device layer contains many different MVs moving around the city roads with different transmission and computing resources. As mentioned in these works [5], [10], [17], we assume that once the MVs enter the BSs' service area, they will agree to participate in the VFC system. The MVs are represented as $\mathcal{V} = \{1, 2, \ldots, V\}$.

### B. Policy-Making Description

In the VFC environment, the system time is divided into a number of episodes $\mathcal{T} = \{1, \ldots, T\}$ and each episode is divided into many time slots with a duration of $\delta$ seconds. The system state is regarded as unchanged in each time slot, yet it changes throughout the episode. Further, similar to these works [18], [41], we assume that each MV generates only one task in each time slot and that its task size follows a Poisson distribution. In addition, we assume that every arrival task is placed in the task queue of the corresponding MV. In each policy-making time slot, the offloading policy for the tasks in each MV queue is only determined by the individual observation $\boldsymbol{o}_v(t)$ of each MV $v$. The observation information includes the following details on the tasks, the MVs, the ESs, and wireless channels.

First, the generated tasks of each MV $v$ are denoted as $G_v(t) = \{d_v(t), \lambda_v(t), \ell_v(t)\}$. $d_v(t)$ is the data size of the generated task. $\lambda_v(t)$ is the computation size which represents CPU cycles required to complete the task. $\ell_v(t)$ denotes the task deadline. Similar to these works [25], [30], we consider that the task will be finished by its deadline in the proposed algorithm; otherwise, there will be a penalty. Second, as illustrated in Fig. 2, for arriving tasks to be executed on the MVs or offloaded to the ESs, we adopt a queue to simulate the execution process of the tasks. Each MV and ES maintains a task queue. If the previous task is taking up computational resources, we assume that the subsequent task has to wait in the task queue. Further, we define $t_v^{ql}(t)$ as the task queuing delay in the MV $v$ and $t_m^{qe}(t)$ as the queue waiting latency in the computational queue of ES $m$. Let $F_v^{l^{\max}}$ denote the maximum computation capacities of MV $v$, $f_v^l(t)$ indicate computing resources allocated by MV $v$ in the time slot $t$, which is the amount of CPU cycles every second on MV for handling tasks. Meanwhile, $F_m^e$ indicate the computing capacity of the ES $m$. Moreover, if some tasks need to be offloaded to ESs, the MVs need to assign certain transmission resources (i.e., uplink transmission power). Therefore, the maximum uplink transmission power of MV $v$ is denoted by $P_v^{l^{\max}}$. $p_v(t)$ indicate transmission power allocated by MV $v$ in the time slot $t$.

Finally, similar to these works [18], [45], we assume that a GPS positioning system is applied in VFC system that can obtain the longitude and latitude coordinates of the MV in time slot $t$, i.e., $I_v(t) = \{x_v(t), y_v(t)\}$. Specifically, in the experimental part, we simulate the mobility of the MV using the historical trajectory of the real cab. In this work, we consider the dynamically changing VFC system environment, i.e., since MVs are always in motion, the network transmission rate between them and each ES is dynamically changing. And the number of MVs in the VFC network is also dynamically changing. Specifically, MVs may actively exit the network because they leave the service area of the ES or complete the offloading request; or they may access to the MEC network because they are in the service range of the ES or have the task offloading request service enabled in the service range of the ES. Thus, we consider two wireless communication connection solutions by using the solutions set as $L(t) = \{0, 1\}$, where $l_{v,m}(t) = 0$ denotes that the MV $v$ and ES $m$ cannot build a wireless communication link, so the tasks that arrive can only be performed locally; $l_{v,m}(t) = 1$ indicates that MV $v$ and ES $m$ are able to build a wireless communication link and offload the arrival tasks to ES to execute or local processing.

Further, the channel gain and background noise power between MV $v$ and ES $m$ are defined as $g_{v,m}(t)$ and $\eta_{v,m}(t)$, $(v \in \mathcal{V}, m \in \mathcal{M})$, respectively. Similar to [5], [20], and [30], we assume that the channel state information (CSI) is to be known ahead of time, i.e., $\mathcal{W}(t) = \{(g_{v,m}(t), \eta_{v,m}(t)), v \in \mathcal{V}, m \in \mathcal{M}\}$. Accordingly, we define the local state $\boldsymbol{o}_v(t)$ of the MV $v$ by the following:

$$\boldsymbol{o}_v(t) = \left\{G_v(t), I_v(t), F_v^{l^{\max}}, P_v^{l^{\max}}, t_v^{ql}(t), \mathcal{W}(t), \boldsymbol{\Gamma}(t)\right\} \quad (1)$$

where $\boldsymbol{\Gamma}(t)$ denotes the matrix representing the task queueing delay history from time slot $t$-$T_{\text{seq\_len}}$ to time slot $t$-1 in the ESs computation queue. We consider that each MV's local state contain the task queueing delay history of the ES. This is because we treat each MV as an agent making offloading decisions independently, the current queueing delay on the ESs is not perceived from each MV's perspective, which is a typical partially observable Markov decision process (POMDP) problem. To alleviate the partial observability of MVs to ESs and to obtain the long-term maximum returns in the VFC system, we adopt the historical task queueing delay of the ESs to react to the load of the ESs. And we introduce a long sequence prediction network (e.g., a neural network that can predict the task queuing delay of the next 30–50 time slots in the ES) to predict the current and future queuing delay of tasks on the ESs computation queues.

Specifically, according to the current task queuing delay $t_m^{qe}(t)$ of the computational queue of ES $m$, we further define $\boldsymbol{\Gamma}(t)$ to denote the historic queueing delay between the time slot $t - T_{\text{seq\_len}}$ and time slot $t$-1 in the computational queue of ES $m$ (i.e., the load situation in the ES). It is a matrix with size $T_{\text{seq\_len}} \times M$. Let $\{\boldsymbol{\Gamma}(t)\}_{i,j}$ denote the element $(i, j)$ of $\boldsymbol{\Gamma}(t)$, representing the current task queuing delay of ES $y$ computation queue in the $i$th time slot starting from $t$-$T_{\text{seq\_len}}$. Further, we define $\{\boldsymbol{\Gamma}(t)\}_{i,j}$ by

$$\{\boldsymbol{\Gamma}(t)\}_{i,j} = t_j^{qe}\big(t - T_{\text{seq\_len}} + i - 1\big). \quad (2)$$

To get $\boldsymbol{\Gamma}(t)$, we suppose that at the end of each time slot, each ES broadcasts its task queueing delay. According to the above description, the VFC system contains $M$ ES, i.e., $M$ computation queues. Even if signals are transmitted for $M$ queues simultaneously, their number can be expressed in $\lfloor log_2 M \rfloor$ bit, such as, with 1000 MVs, the maximum number of bits needed is 10 bits. Therefore, only a small communication overhead is generated by broadcasting ESs task queueing delay signals. Since our offloading scheme involves a small communication overhead, similar to these works [46], [47], we consider data privacy through encryption.

After getting the individual current observation information $\boldsymbol{o}_v(t)$ of the MV $v$, a joint action $\boldsymbol{a}_v(t)$ for MV $v$ is made according to the local state in time slot $t$. Accordingly, the action $\boldsymbol{a}_v(t)$ of the MV $v$ is defined by the following:

$$\boldsymbol{a}_v(t) = \left\{\varphi_v(t), p_v(t), c_v(t), f_v^l(t)\right\} \quad (3)$$

where $\varphi_v(t)$ is a positive integer, $\varphi_v(t) \in [0, M]$, which indicates that the MV $v$ agent determines which computing node (e.g., local MV and the ESs) handles the task $G_v(t)$. Specifically, if $\varphi_v(t) = 0$, it means that MV decides to execute the arrival task locally, meanwhile, MV allocates the appropriate computing resources $f_v^l(t)$ for the task execution. Otherwise, it means that MV decides to offload the arrival task to the $\varphi_v(t)$th ES for execution, meanwhile, the MV assigns the corresponding transmission power $p_v(t)$ and selects the optimal transmission channel $c_v(t)$ for the offloaded task. The transmission channel $c_v(t)$ belongs to the target ES. Table I summarizes the major notations used in this article.

TABLE I
SUMMARY OF IMPORTANT SYMBOLS

| Symbols | Descriptions |
|---|---|
| $V$ | The number of MVs. |
| $M$ | The number of ESs. |
| $\mathcal{C}_m$ | Transmission channel of ES $m$. |
| $\delta$ | Duration of the time slot. |
| $d_v(t)$ | The task's data size. |
| $\lambda_v(t)$ | The task's needed CPU cycles. |
| $\ell_v(t)$ | The task's maximum tolerated delay. |
| $t_v^{ql}(t)$ | The MV's current local queueing delay. |
| $t_m^{qe}(t)$ | The current queuing latency of ES $m$. |
| $F_m^e$ | The maximum computation capacities of ES $m$. |
| $F_v^{l^{\max}}$ | The maximum computation capacities of MV $v$. |
| $f_v^l(t)$ | The computational resources assigned by MV $v$. |
| $P_v^{l^{\max}}$ | The MV's maximum uplink transmission power |
| $p_v(t)$ | The transmission power allocated by MV $v$. |
| $I_v(t)$ | The coordinates of MV $v$. |
| $r_{v,m}^{tran}(t)$ | Transmission rate between MV $v$ and $m$. |
| $\varphi_v(t)$ | The task offloading indicator of the MV $v$. |
| $g_{v,m}(t)$ | The channel gain between MV $v$ and ES $m$. |
| $L(t)$ | Wireless communication link solutions set. |
| $\eta_{v,m}(t)$ | The background noise power. |
| $T_{seq\_len}$ | The length of task queuing delay of ESs. |
| $\boldsymbol{\Gamma}(t)$ | The task queuing delay matrix of ESs. |
| $\tau_v^l(t)$ | The local processing delay in MV $v$. |
| $e_v^l(t)$ | The local processing energy consumption in MV $v$. |
| $C_v^l(t)$ | The local processing cost in MV $v$. |
| $\tau_{v,m}^{tran}(t)$ | The task upload delay in ES $m$. |
| $\tau_v^m(t)$ | The task processing delay in ES $m$. |
| $\tau_{v,m}^{tot}(t)$ | The total latency of computing offloading. |
| $e_v^m(t)$ | Transmission energy consumption of offloading. |
| $C_v^m(t)$ | Total cost of computing offloading. |

## C. Communication Model

Based on the wireless communication connection solution set $L(t)$, i.e., $l_{v,m}(t) = 1$, we define the transmission rate $r_{v,m}^{\text{tran}}(t)$ of the MV $v$ offloading task $G_v(t)$ to the target ES $m$ via wireless communication by the following:

$$r_{v,m}^{\text{tran}}(t) = W \log_2\left(1 + \frac{p_v(t) g_{v,m}(t)}{\eta_{v,m}(t) + \sum_{i \in \mathcal{V}, i \neq v} p_v(t) g_{i,m}(t)}\right) \quad (4)$$

where $W$ denotes the bandwidth of the VFC system [30]. For the wireless connection, $p_v(t)$ is the uplink transmission power allocated in the time slot $t$, and $\eta_{v,m}(t)$ is the background noise power. $g_{v,m}(t) = D_{v,m}^{-l}$ is the channel gain between MV $v$ and the ES $m$, where $-l$ is the path-loss exponent and $D_{v,m}$ denote the distance between MV $v$ and the ES $m$. $\sum_{i \in \mathcal{V}, i \neq v} p_v(t) g_{i,m}(t)$ denotes that the MVs (in addition to the MV $v$) transmit task data over the wireless network to the same selected ES at the same time.

## *D. Computation Model*

In the VFC system, we consider that the MV is not equipped with sufficient computing and transmission resources. Thus, when MV generates a computationally intensive and latency-sensitive task that needs to be processed, MV will consider offloading to ES for processing or local computation. In the following, we present the computation model.

*1) Local Computing:* When $\varphi_v(t) = 0$, the task $G_v(t)$ is handled locally by the MV $v$, thus there is no delay in task uplink transmission. We assume that the dynamic voltage and frequency scaling (DVFS) square [48] can be flexibly controlled by the chip voltage variation. Specifically, the computing resources $f_m^l(t)$ allocated by MV can be adjusted according to the needs of the task, but are limited by the computing capacity of the MV $f_v^l(t) \leq F_v^{l^{\max}}$. Accordingly, the delay of task processing on the local MV $v$ for the task $G_v(t)$, $\tau_v^l(t)$, is defined by the following:

$$\tau_v^l(t) = \frac{\lambda_v}{f_v^l(t)} + t_v^{ql}(t). \tag{5}$$

Moreover, the update of $t_v^{ql}(t+1)$ can be defined as

$$t_v^{ql}(t+1) = \max\left\{0, \tau_v^l(t) - \delta\right\}. \tag{6}$$

As mentioned in these works [10], [11], the locally computed energy consumption is defined by the following:

$$e_v^l(t) = \lambda_v(t) \cdot \chi_v \tag{7}$$

where $\chi_v$ denotes the mean energy cost every CPU cycle in MV $v$, which is determined by computing device parameters. We adopt real measurements in [49] and define $\chi_v = 10^{-27} \cdot (f_v^l(t))^2$.

Additionally, for many tasks emerging from VFC in-vehicle applications, such as smart driving, extremely short completion delays are required. Similar to these works [30], [49], the proposed algorithm also hopes that the task's execution delay is not greater than the task deadline $\ell_v(t)$; otherwise, we apply a punishment $\Psi$ that is often considerably greater than the weighted total of delay and energy consumption. Accordingly, the cost of local processing for MV $v$, $C_v^l(t)$, is defined by the following:

$$C_v^l(t) = \begin{cases} c_1\tau_v^l(t) + c_2 e_v^l(t), & \text{if } \tau_v^l(t) \leq \ell_v(t) \\ \Psi, & \text{otherwise} \end{cases} \tag{8}$$

where $\Psi$, $c_1$, and $c_2$ are constants. Similar to these works [13], [49], $c_1$ and $c_2$ are the preference of task $G_v(t)(v \in \mathcal{V}, t \in \mathcal{T})$ on latency and energy consumption, where they add up to 1. $\Psi$ is the failure punishment for local execution in the MV $v$.

*2) Computation Offloading:* When $\varphi_v(t)>0$, the task $G_v(t)$ is offloaded to the target computing node ES $m$ for processing. To perform tasks in the ES, the MV $v$ first requires to upload the task to the target ES $m$. As is the case with many research, the returned data is often considerably smaller than the uploaded data. As a result, we disregard the transmission time of the returned data. Therefore, task latency comprises data upload time and execution time in the ES. Accordingly, we define the data upload time by the following:

$$\tau_{v,m}^{\text{tran}}(t) = \frac{d_v(t)}{r_{v,m}^{\text{tran}}(t)}. \tag{9}$$

In addition, we define the execution time in ES by the following:

$$\tau_v^m(t) = \frac{\lambda_v(t)}{F_m^e(t)/N_m(t)} + t_m^{qe}(t) \tag{10}$$

where $N_m(t)$ indicates the number of tasks in the computational queue of ES $m$ during the time slot $t$. Moreover, the update of $t_m^{qe}(t+1)$ can be defined as

$$t_m^{qe}(t+1) = \max\left\{0, \tau_{\max}^m(t) - \delta\right\} \tag{11}$$

where $\tau_{\max}^m(t)$ denotes the maximum latency to process all tasks on ES $m$ during the time slot $t$. Further, according to (9) and (10), we define the total task processing latency $\tau_{v,m}^{\text{tot}}(t)$ that comprises data upload time and execution time by the following:

$$\tau_{v,m}^{\text{tot}}(t) = \tau_{v,m}^{\text{tran}}(t) + \tau_v^m(t). \tag{12}$$

Similar to these works [10], [11], we define the energy consumption $e_v^m(t)$ of the task $G_v(t)$ data transmission by the following:

$$e_v^m(t) = \tau_{v,m}^{\text{tran}}(t) \cdot p_v(t). \tag{13}$$

Finally, similar to local task execution, the task $G_v(t)$ offloaded to the ES $m$ should be completed before task deadline $\ell_v(t)$. For task $G_v(t)$, we define the total cost $C_v^m(t)$ of a task being offloaded to the ES $m$ by the following:

$$C_v^m(t) = \begin{cases} c_5\tau_v^{\text{tot}}(t) + c_6 e_v^m(t), & \text{if } \tau_{v,m}^{\text{tot}}(t) \leq \ell_v(t) \\ \Upsilon, & \text{otherwise} \end{cases} \tag{14}$$

where $c_5$ and $c_6$ are constants. Similar to local processing, $c_5$ and $c_6$ are the preference of task $G_v(t)(v \in \mathcal{V}, t \in \mathcal{T})$ on latency and energy consumption, where they add up to 1. $\Upsilon$ is the fail penalty for tasks to be executed on ES $m$.

## *E. Problem Formulation*

According to the above description, our VFC system considers a large number of MV scenarios with a hybrid cooperative–competitive relationship, in which the MVs compete for computational and transmission resources (i.e., transmission channel, transmission power, and CPU) to accomplish their delay-sensitive tasks. Without knowing the offloading decisions of other MVs and the resource allocation of ESs, each MV minimizes the system cost through optimal offloading policies. Further, the sum of all MV costs is defined to as the system cost. The cost of each MV, $C_v(t)$, is defined by the following:

$$C_v(t) = (1 - \varphi_v(t))C_v^l(t) + \varphi_v(t)C_v^m(t) \tag{15}$$

$$\min_{\{\varphi_v(t), p_v(t), c_v(t), f_v^l(t)\}} \frac{1}{|\mathcal{T}|}\sum_{t\in T}\sum_{v\in\mathcal{V}}\{C_v(t)\} \tag{16}$$

$$\text{s.t. } C_1 : \varphi_v(t) \in [0, M] \tag{16a}$$

$$C_2 : 0 < p_v(t) \leq P_v^{l^{\max}} \tag{16b}$$

$$C_3 : c_v(t) \in \mathcal{C}_m \tag{16c}$$

$$C_4 : 0 < f_v^l(t) \leq F_v^{l^{\max}} \tag{16d}$$

$$C_5 : \tau_v^l(t) \leq \ell_v(t) \tag{16e}$$

$$C_6 : \tau_{v,m}^{\text{tot}}(t) \leq \ell_v(t). \tag{16f}$$

where constraint (16a) implies that each task in the MV can be processed locally or offloaded to the ES for execution. Constraint (16b) means that the uplink power allocated by the MV cannot exceed the maximum power of the MV. Constraint (16c) shows that the MV agent $v$ needs to select the appropriate channel in the target ES $m$. Constraint (16d) denotes that the computational resources allocated by MV for local computing cannot be larger than the maximum computational capacity of MV. Constraints (16e) and (16f) ensure that each arrival task can be completed with task deadline.

## IV. MARL-Based Task Offloading Algorithm

The optimization problem (16) is an extremely difficult problem and has been shown to be NP-hard based on [18] and [44]. This is because we are also exploring the best combination of the task offloading and the resource allocation policies to satisfy system cost minimization under the task deadline and resource constraints. To solve this problem, we first define the JTORA problem, and our goal is to get a long-term return by determining the offloading policies for MVs. Second, each MV is considered as an agent and utilizes the actions made by the MV based on local states to interact with the VFC system environment. Finally, we employ an improved MARL algorithm to solve this problem. Next, we introduce the observations/states, actions, and rewards involved in the MARL algorithm.

### *A. MARL-Based Problem Reformulation*

Similar to these works [12], [30], we formulate the JTORA problem as a Markov decision process (MDP). Based on the proposed system model, each MV in the VFC network is treated as an agent. Theoretically, we represent MDP as a tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ defined as follows.

*1) Local State Space $\mathcal{O}$ of MV $v(v \in \mathcal{V})$:* $\mathcal{S}$ indicates that the system state space contains local state information $\mathcal{O}$ of all MVs and the state information of ESs. According to the proposed system model, we define the local state of the MV agent as $\boldsymbol{o}_v(t) = \{G_v(t), I_v(t), F_v^{l^{\max}}, P_v^{l^{\max}}, t_v^{ql}(t), \mathcal{W}(t), \boldsymbol{\Gamma}(t)\}$, $(\boldsymbol{o}_v(t) \in \mathcal{O})$. Moreover, the observation/state of the MV contains the local state information of the $k$ neighboring MVs. $G_v(t)$ contains task information. $I_v(t)$ denotes the longitude and latitude coordinates of the MV in time slot $t$. $F_v^{l^{\max}}$ denotes the maximum computation capacities of MV $v$. $P_v^{l^{\max}}$ denotes the maximum uplink transmission power of MV $v$. $t_v^{ql}(t)$ denotes the queuing delay of the current task in the local queue of MV $v$. $\mathcal{W}(t)$ denotes the CSI. $\boldsymbol{\Gamma}(t)$ is a matrix representing the task queueing delay history from time slot $t - T_{\text{seq\_len}}$ to time slot $t-1$ in the ESs computation queue.

*2) Action Space $\mathcal{A}$ of MV $v(v \in \mathcal{V})$:* According to the proposed system model, we define the action of the MV agent as $\boldsymbol{a}_v(t) = \{\varphi_v(t), p_v(t), c_v(t), f_v^l(t)\}$, $(\boldsymbol{a}_v(t) \in \mathcal{A})$. $\varphi_v(t)$ indicates the task offloading policy. $p_v(t)$ indicates the transmission power control policy. $c_v(t)$ denotes the transmission channel selection policy. $f_v^l(t)$ indicates the local computing resource allocation policy.

*3) Transition Probability $\mathcal{P}$:* $\mathcal{P}$ denotes the probability that the MV $v(v \in \mathcal{V})$ agent transfers from state $\boldsymbol{o}_v(t)$ to the next state $\boldsymbol{o}_v(t+1)$ at time $t$ based on action $\boldsymbol{a}_v(t)$.

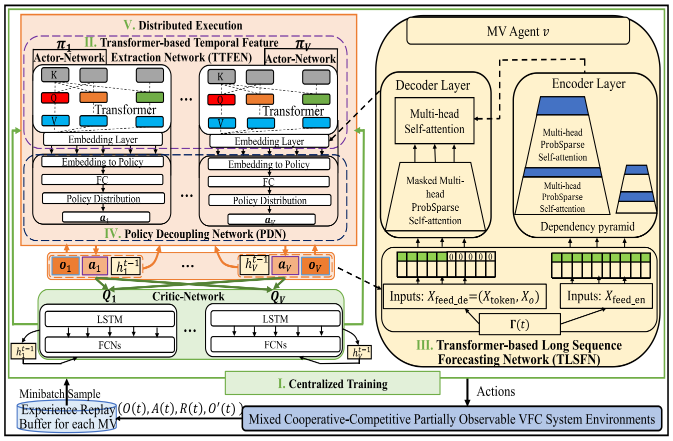
V. Distributed Execution
II. Transformer-based Temporal Feature Extraction Network (TTFEN)
$\pi_1$
$\pi_V$
Actor-Network
Actor-Network
K
Q
V
Transformer
Embedding Layer
Embedding to Policy
FC
Policy Distribution
$a_1$
$a_V$
IV. Policy Decoupling Network (PDN)
$o_1$
$a_1$
$h_1^{t-1}$
$h_V^{t-1}$
$a_V$
$o_V$
$Q_1$
Critic-Network
$Q_V$
LSTM
FCNs
MV Agent $v$
Decoder Layer
Multi-head Self-attention
Masked Multi-head ProbSparse Self-attention
Encoder Layer
Multi-head ProbSparse Self-attention
Dependency pyramid
Inputs: $X_{\text{feed\_de}}=(X_{\text{token}}, X_o)$
Inputs: $X_{\text{feed\_en}}$
$\Gamma(t)$
III. Transformer-based Long Sequence Forecasting Network (TLSFN)
I. Centralized Training
Minibatch Sample
Experience Replay Buffer for each MV
$(O(t), A(t), R(t), O'(t))$
Actions
Mixed Cooperative-Competitive Partially Observable VFC System Environments

Fig. 3. CTDE algorithm-based TPDMAAC model. The left side contains TTFEN and PDN and on the right side is TLSFN.

*4) Reward $r_v(t)$ of MV $v(v \in \mathcal{V})$:* Based on the observed information $\boldsymbol{o}_v(t)$, the MV agent $v$ uses offloading policies $\boldsymbol{a}_v(t)$ interacting with the VFC system, getting a reward. Accordingly, we define the reward of the MV $v$, $R_v(t)$, by the following:

$$R_v(t) = \sum_{t=1}^{T} \gamma^t \cdot r(\boldsymbol{o}_v(t), \boldsymbol{a}_v(t)) \tag{17}$$

where $r(\boldsymbol{o}_v(t), \boldsymbol{a}_v(t)) = (c_7/[c_8 + \varphi_v(t) \cdot C_v(t)])$. $C_v(t)$ denotes the cost of task processing. $\varphi_v(t)$ indicates the offloading decision for the task $G_v(t)$. $\gamma$ denotes the discount factor. $c_7$, $c_8$ is a constant. The optimization goal is to minimize the system cost; thus the reward is inversely proportional to the system cost.

### *B. T$^2$ DMAAC Agent Training and Execution*

As illustrated in Fig. 3, we first incorporate CTDE algorithms into TPDMAAC, i.e., I. The Centralized Training and V. The Distributed Execution. Specifically, to reduce the computational load on MVs, we first allow the Data Center to help MVs in centrally training network model. In the centralized training phase, MV agents can be trained using local observations and actions of other MV agents. Using this training method can solve the nonstationarity problem of multiagent training environments. Then, in the execution phase, based on local observations, each MV agent makes independent distributed decisions on task offloading and resource allocation policies using the trained TPDMAAC-agent network. And the MV agent does not need any information from other MV agents. Second, we integrate TLSFN, TTFEN, and PDN into the actor–critic algorithm to solve the unknown load in ES and the transfer of the model. Next, we further present the internetwork relationships of TPDMAAC.

In Fig. 4, the TPDMAAC adopts a multiagent actor–critic algorithm. On the one hand, the actor-network consists of the TTFEN and PDN, i.e., II and IV Part in Fig. 3. We take the MV observations information $G_v(t)$, $I_v(t)$, $F_v^{l^{\max}}$, $P_v^{l^{\max}}$, $t_v^{ql}(t)$, and $\mathcal{W}(t)$ as the inputs to the actor-network. First, TTFEN will perform feature extraction and feature embedding on this

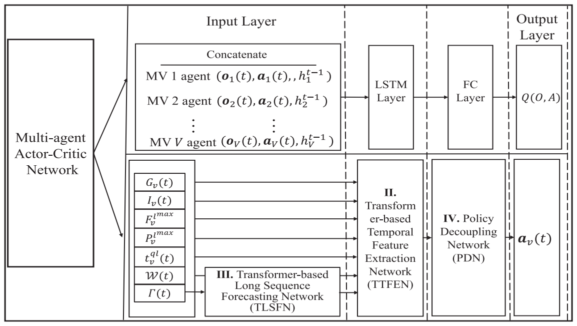
Input Layer
Output Layer
Concatenate
MV 1 agent $(\boldsymbol{o}_1(t), \boldsymbol{a}_1(t),, h_1^{t-1})$
MV 2 agent $(\boldsymbol{o}_2(t), \boldsymbol{a}_2(t), h_2^{t-1})$
MV $V$ agent $(\boldsymbol{o}_V(t), \boldsymbol{a}_V(t), h_V^{t-1})$
LSTM Layer
FC Layer
$Q(O, A)$
Multi-agent Actor-Critic Network
$G_v(t)$
$I_v(t)$
$F_v^{l^{max}}$
$P_v^{l^{max}}$
$t_v^{ql}(t)$
$\mathcal{W}(t)$
$\Gamma(t)$
III. Transformer-based Long Sequence Forecasting Network (TLSFN)
II. Transformer-based Temporal Feature Extraction Network (TTFEN)
IV. Policy Decoupling Network (PDN)
$\boldsymbol{a}_v(t)$

Fig. 4. Illustration of internetwork relationships in TPDMAAC.

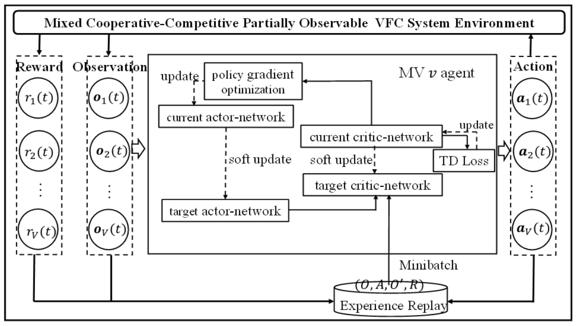

Fig. 5. Overall training architecture for TPDMAAC.

information to obtain a set of feature maps. Meanwhile, the historical task queuing delay of ESs $\boldsymbol{\Gamma}(t)$ will be delivered to TLSFN for the ESs task queueing delay prediction. Second, the feature vectors generated by both TTFEN and TLSFN will be concatenated passed to the PDN for further learning. Finally, PDN generate the actions $\boldsymbol{a}_v(t)$ of the MV agent $v$. On the other hand, the centralized critic-network takes the concatenate observations and the corresponding actions of all MV agents as input to get the corresponding $Q$-values [i.e., $Q(\boldsymbol{o}_v(t), \boldsymbol{a}_v(t))$], and evaluates the quality of the task offloading and resource allocation policies by the $Q$-values.

Each MV $v$ is deployed with an intelligent agent module (TPDMAAC). Each MV $v$ agent consists of actor-network and critic-network. First, the critic-network consists of the current critic-network $Q_v$ and the target critic-network $Q'_v$. As the upper side of Fig. 5, the critic-network takes the local observations and actions as the input and outputs $Q$ value. The critic-network estimates the average expected return from the current state to obtain an optimal policy. Second, similar to the critic-network, the actor-network consists of the current actor-network and the target actor-network. As the lower side of Fig. 5, the actor-network takes the local observations as the input and outputs the task offloading policies.

The MV agent acts the action $\boldsymbol{a}_v(t)$ on the VFC system and obtains the corresponding reward $r_v(t)$. At the same time, the local observation $\boldsymbol{o}_v(t)$ of the MV agent is changed under the action and the state is transferred to the next state $\boldsymbol{o}'_v(t)$. We use $O(t) = \{\boldsymbol{o}_v(t), v \in \mathcal{V}\}$, $A(t) = \{\boldsymbol{a}_v(t), v \in \mathcal{V}\}$, $R(t) = \{r_v(t), v \in \mathcal{V}\}$, and $O'(t) = \{\boldsymbol{o}'_v(t), v \in \mathcal{V}\}$ to denote the joint state, joint action, joint reward, and joint next state of all the MV agents, respectively. We store these experiences $\{(\boldsymbol{o}_v(t), \boldsymbol{a}_v(t), r_v(t), \boldsymbol{o}'_v(t)) | v \in \mathcal{V}\}$ generated by the MV agent interacting with the VFC system environment in an experience pool for each MV. These experience pools are usually stored in the Data Center for model training. Finally, we use these experiences and the gradient descent algorithm to train our model. In the following, we will elaborate on TTFEN, TLSFN, and PDN.

*1) TLSFN Module:* Based on the description of Section I, to solve the uncertain load of the ESs and the partial observability of the MV agent to the system model, we introduce a TLSFN by predicting the current and future queuing delay of ESs. Different from these works [30], [50], we replace RNN or LSTM with Transformer. This is because in recurrent neural networks, it is difficult for the long sequence prediction to obtain long-term dependencies by gradient descent because the backpropagation process of gradients usually leads to gradient vanishing or gradient explosion. Specifically, as shown on the left side of Fig. 3, TLSFN Module includes the Input Layer, Encoder Layer, and Decoder Layer. In the following, we will further describe these modules and their fundamentals.

*The Input Layer:* The input part of the TLSFN module consists of inputs for the encoder and decoder layers. As shown at the right of the TLSFN Module in Fig. 3, the Encoder layer takes a large number of long sequence inputs (green series) (i.e., the $t$-th sequence input $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}} \times M}$). Typically, in the long sequence time-series forecasting task, the capability to catch long-term independence needs global timestamps information to memorize the position information of the input sequence. Therefore, we initially retain the sequential relationship of sequences by employing a fixed-position embedding, which can be defined by

$$\begin{aligned} \text{PE}_{(p,2I)} &= \sin\Big(p/\big(2T_{\text{seq\_len}}\big)^{2I/d_{\text{TLSFN}}}\Big) \\ \text{PE}_{(p,2I+1)} &= \cos\Big(p/\big(2T_{\text{seq\_len}}\big)^{2I/d_{\text{TLSFN}}}\Big) \end{aligned} \tag{18}$$

where PE is positional embedding matrix. $p \in [0, T_{max\_seq\_len}]$ indicates a specific location in the historical experience sequence. $T_{max\_seq\_len}$ denotes the maximum length of task queuing delay history sequence in the ESs. $I \in [0, d_{\text{TLSFN}}/2]$ indicates a specific dimension in TLSFN model dimension ($d_{\text{TLSFN}}$).

According to the above description, the input of the Encoder $X_{\text{feed\_en}}$ layer mainly contains the timestamp-based positional embedding matrix PE and the length of task queuing delay history sequence in the ESs $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}} \times M}$. Similarly, the input of the Decoder $X_{\text{feed\_de}}$ layer contains positional embedding token $X_{\text{token}}$ and a placeholder for the task queuing delay prediction sequence $X_o$.

*The Encoder Layer:* As shown at the right of the TLSFN Module in Fig. 3, $X_{\text{feed\_en}}$ represents the input of the encoder, whose length is determined by the parameter $T_{\text{seq\_len}}$. The Encoder layer is devised to capture the strong long-term dependence of the long period of task queuing delay history sequence in the ESs. After that, we further conduct the concatenation operation on $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}} \times M}$ and PE, the $t$th sequence input $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}} \times M}$ has been shaped into a matrix $X^t_{\text{feed\_en}} \in \mathbb{R}^{T_{\text{seq\_len}} \times d_{\text{TLSFN}}}$. In addition, we replace the canonical self-attention mechanism [51] with a probsparse self-attention mechanism [52] in the Encoder layer. In comparison to the canonical self-attention mechanism, the probsparse self-attention mechanism offers the following three advantages.

First, the canonical self-attention mechanism's atomic operation, i.e., the original dot product, leads to a time complexity and memory usage of $\mathcal{O}(L^2)$ per layer. However, the probsparse self-attention mechanism by adding the maximum pooling layer (stride = 2) as well as downsampling $X^t_{\text{feed\_en}}$ into its half slice after stacking one layer, reducing the overall memory utilization to $\mathcal{O}(L\log L)$. Second, the multilayer stacking of Encoder/Decoder (e.g., the $J$ layers) results in a total memory utilization of $\mathcal{O}(JL^2)$, which restricts the scalability of the model when receiving long sequence inputs. The probsparse self-attention mechanism employs a self-attention distillation technique for prioritize dominant attention scores in $J$-stacking layers, drastically lowering overall space complexity to ($L\log L$) and facilitating the reception of long sequence input. Third, the dynamic decoding mechanism of the canonical self-attention mechanism reasoning as sluggish as the network model based on RNN when predicting long outputs. The probsparse self-attention mechanism employs a generative decoder that only needs one forward step to obtain a long sequence output, while preventing the spread of accumulated errors in the inference stage.

Finally, to improve the stability of the distillation operations, we built copies of the primary stack with halved inputs as well as gradually reduced the amount of self-attentive distillation layers by lowering one layer at a time, similar to a pyramid as shown at the right of TLSFN in Fig. 3. Eventually, we concatenate all the outputs to the last hidden representations of the Encoder layer.

*The Decoder Layer:* As shown at the left of the TLSFN Module in Fig. 3, $X_{feed\_de}$ consists of two parts, where $X_{\text{token}}$ is a historical sequence with a length of predicted sequence, and $X_0$ is an all-zero sequence with the same length as the predicted sequence. For the Decoder layer, it is similar to the Encoder layer in general, except that there is an additional multihead attention mechanism for interacting with the Encoder layer output. Specifically, we employ the following vectors as inputs to the decoder, which can be defined in

$$X^t_{\text{de}} = \text{Cat}\left(X^t_{\text{token}}, X^t_0\right) \in \mathbb{R}^{(L_{\text{token}}+L_y)\times d_{\text{TLSFN}}} \tag{19}$$

where Cat indicates the feature concatenation operation. $X^t_{\text{token}} \in \mathbb{R}^{L_{\text{token}}\times d_{\text{TLSFN}}}$ is the start token, and $X^t_0 \in \mathbb{R}^{L_y\times d_{\text{TLSFN}}}$ is a placeholder for the target sequence $L_y$ (set scalar as 0). After that, these inputs are passed through the masked multihead probsparse self-attention and multihead self-attention networks to obtain the final feature map. We define the final feature map by

$$\boldsymbol{X}_{\text{de}} = \text{Cat}\Big(\text{Attention}\Big(\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}}\times M}, X_{\text{feed\_en}}, X^t_{\text{de}}\Big)\Big) \tag{20}$$

where Attention denotes the masked multihead probsparse self-attention or multihead self-attention. These feature maps are concatenated sequentially in order to track the serial changes from $\{\boldsymbol{\Gamma}(t)\}_0$ to $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}}}$, which can illuminate the changes in the tasks queuing delay of the ESs over time slot. Similarly, Cat indicates the feature concatenation operation.

Eventually, the TLSFN module outputs a feature map representing the dynamics of the tasks queuing delay of the ESs in the future in the last multihead self-attention layer, where the output will be concatenated to the other observation information in the next layer for further learning.

*2) TTFEN and PDN-Based Actor-Network:* To adapt our proposed method to the new VFC environment quickly, we redesign the actor-network using TTFEN and PDN. In the following, we will further describe these modules and their fundamentals.

*TTFEN Module:* There are $V$ MV agents in the VFC system environment. We deploy the TPDMAAC-actor to each MV. We consider the policy parameters of these MV agents represented by $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_V\}$ and let $\boldsymbol{\pi} = \{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_V\}$ represent the set of policies of all MV agents. The parameters $\theta_v$ abbreviated as $\boldsymbol{\mu}_v$. To adapt to different observation/state space, we embed the observation of the MV agent into a token (semantic embedding), which is a vector, i.e., TTFEN can handle the various observation/state space through a transformer that accepts different token we build from the observation. This is because the transformer architecture can accept various tokens by converting variable-length inputs into fixed dimensions through the self-attention mechanism. Specifically, $X_{\text{de}}$ in (20), the local state of the MV agent $v$, and the $k$ neighboring MVs of the MV $v$ are embedded through an embedding layer $E$ in time slot $t$. Accordingly, we express this embedding by the following:

$$e_v(t) = \Big\{E(\boldsymbol{X}_{\text{de}}, \boldsymbol{o}_v(t)), E\big(\boldsymbol{o}_{v,1}(t)\big), \ldots, E\big(\boldsymbol{o}_{v,k}(t)\big)\Big\}. \tag{21}$$

Moreover, similar to MADDPG, we express the action-value function of the MV agent $v$ in the evaluation of each action step by the following:

$$q_v(t) = Q_v(t)\Big(h^{t-1}_v, e_v(t), A(t-1)\Big) \tag{22}$$

where $h^{t-1}_v$ denotes the temporal hidden state of the last time slot $t-1$. By using this the historical information, it is beneficial for MV agents to reach the optimal policy in a partially observable environment. $e_v(t)$ denotes the observation information embedding. In the process of centralized training, when the number of MVs increases to a certain number, the execution latency of the centralized critic-network will impact the execution of tasks. To solve this problem, similar to these works [37], [53], we use the action of the last time slots [i.e., $A(t-1)$] as input to eliminate the effect of the delay generated by the centralized critic-network on task completion. This is because the states, actions, and generated tasks of the MEC system in adjacent time slots have spatiotemporal properties and are therefore similar [54].

Furthermore, we implement the action-value function $Q_v(h^{t-1}_v, e_v(t), A(t-1))$ for the MV agent $v$ using the self-attention mechanism. Similar to the attention mechanism, for the identical matrix, query, key, and value are further represented $\mathcal{R}^l_v = K_v = Q_v = V_v$, where $l \in \{0, \ldots, L\}$ indicates the layer in the transformer. Therefore, the transformer can be re-expressed in

$$\mathcal{R}_v^l = \left\{h_v^{t-1}, e_v(t)\right\}$$
$$Q_v^l, K_v^l, V_v^l = \text{Line}_{Q,K,V}\left(\mathcal{R}_v^l\right)$$
$$\mathcal{R}^L = \text{Attention}\left(Q_v^l, K_v^l, V_v^l\right) \tag{23}$$

where $\text{Line}_{Q,K,V}$ denotes the linear functions used to calculate the parameters [i.e., $(Q, K, V)$] in the attention mechanism. Furthermore, we implement the features of the last transformer layer of the MV agent $v$ to the output space of the value function through a linear function, which can be further defined in

$$Q_v\left(h_v^{t-1}, e_v(t), A(t-1)\right) = \digamma_v\left(\mathcal{R}_v^L, \boldsymbol{a}_v(t)\right) \tag{24}$$

which $\digamma_v$ denotes a linear function.

*PDN Module:* A multiagent actor–critic framework with a self-attention mechanism based on the TTFEN module is not yet able to process the different policy dimensions requirements. Mapping functions $\digamma_v$ in (24) are required to process diverse input and output dimensions and to offer powerful representation capabilities. Furthermore, we devise PDN modules with correlations between the inputs and outputs. Specifically, first, according to the above description, we apply the transformer function to process all the MV agents' observations called "observation-entities" [27] and acquire the last layer of features. In the following, we partition the action space into action-groups (i.e., discrete task offloading and continuous resource allocation policies), and correspond both observation-entities and action-groups. By employing self-attention to learn the mapping relationships between observation-entities and action groups, i.e., the PDN module optimizes policies at the observation-entities to get optimal offloading policies. Specifically, for the continuous action, the actor parameters are updated by the gradient ascent based on MADDPG in

$$\nabla_{\theta_v} J^c\left(\boldsymbol{\mu}_v\right) = \mathbb{E}_{O,\boldsymbol{a}\sim\mathbf{E}}\Bigg[\nabla_{\theta_v}\boldsymbol{\mu}_v(\boldsymbol{a}_v(t)|\boldsymbol{o}_v(t))\nabla_{\boldsymbol{a}_v(t)} \times Q_v^{\boldsymbol{\mu}}\left(O(t), A(t-1), h_v^{t-1}\right)|_{\boldsymbol{a}_v(t)=\boldsymbol{\mu}_v(\boldsymbol{o}_v(t))}\Bigg] \tag{25}$$

where $\mathbf{E}$ denotes the experience replay pool. $\boldsymbol{a}_c$ is a vector consisting of the two continuous action components in $\boldsymbol{\mu}_v(\boldsymbol{a}_v(t)|\boldsymbol{o}_v(t)) = \{p_v(t), f_v^l(t)\}$.

Similarly, for the discrete action (i.e., task offloading policy probability $\boldsymbol{p}_v$ a constant here), we compute the gradient of the cross-entropy loss weighed by $Q_v^{\boldsymbol{\mu}}(O(t), A(t-1), h_v^{t-1})|_{\boldsymbol{a}_v(t)=\boldsymbol{\mu}_v(\boldsymbol{o}_v(t))}$ in

$$\nabla_{\theta_v} J^d\left(\boldsymbol{\mu}_v\right) = -Q_v^{\boldsymbol{\mu}}\left(O(t), A(t-1), h_v^{t-1}\right)|_{\boldsymbol{a}_v(t)=\boldsymbol{\mu}_v(\boldsymbol{o}_v(t))} \cdot \nabla_{\theta_v}\left(\boldsymbol{o}_v(t)\ \ln \boldsymbol{p}_v + \left(1-\boldsymbol{o}_v\ \ln\left(1-\boldsymbol{p}_v\right)\right)\right). \tag{26}$$

Finally, the loss function of the actor for MV agent $v$ is formulated in

$$\mathcal{L}(\theta_v) \approx \nabla_{\theta_v} J^c\left(\boldsymbol{\mu}_v\right) + \nabla_{\theta_v} J^d\left(\boldsymbol{\mu}_v\right). \tag{27}$$

**Algorithm 1** Training Process of the TPDMAAC Agent

```
1: Input: Historical vehicle trajectories in ChengDu City.
2: Randomization of critic networks $Q_\phi$ and actor network $\pi_\theta$ with random parameters initialization $\phi, \theta$.
3: Randomization of target-network initialization $\phi' \leftarrow \phi, \theta' \leftarrow \theta$.
4: Randomization of the temporal hidden state of the actor-network and critic-network with random parameters initialization $h$.
5: Randomization of replay buffe initialization $\mathbf{E}$.
6: Initialize the learning rate with random parameter $\varpi$.
7: for episode = 1 to $|\mathcal{T}|$ do
8:     Initialize a random process $\mathcal{N}$ for action exploration
9:     Initialize the state $O$
10:    for time_slot = 1 to max_episode_length do
11:        for each MV agent $v$, select action $\boldsymbol{a}_v = \boldsymbol{\mu}_{\theta_v}(\boldsymbol{o}_v) + \mathcal{N}_t$ w.r.t. the current policy and exploration
12:        Execute actions $A = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_V)$ and observe reward $r$ and new state $O'$
13:        Store $(O, A, R, O')$ in replay buffer $\mathbf{E}$
14:        Update $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}} \times M}$ matrix
15:        $O \leftarrow O'$
16:        for MV agent $v = 1$to$V$ do
17:            Sample a random minibatch of $S$ samples $(O^j, A^j, r^j, O'^j)$ from $\mathbf{E}$.
18:            By changing the shape of $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}} \times M}$ into the input of the TLSFN through Eq. (18) and Eq. (19), further, TLSFN outputs the last layer of feature maps $\boldsymbol{X}_{de}$ and connects them with other observations (i.e., except for $\{\boldsymbol{\Gamma}(t)\}_{T_{\text{seq\_len}} \times M}$) for further learning.
19:            By embedding the observation $O$ into a semantic embedding to fit into a different observation dimension:
20:            $e_v(t) = \{E(\boldsymbol{X}_{\text{de}}, \boldsymbol{o}_v(t)), E(\boldsymbol{o}_{v,1}(t)), \ldots, E(\boldsymbol{o}_{v,k}(t))\}$
21:            $q_v(t) = Q_v(t)(h_v^{t-1}, e_v(t), A(t-1))$
22:            $\mathcal{R}_v^l = \{h_v^{t-1}, e_v(t)\}$
23:            $Q_v^l, Kv^l, V_v^l = \text{Line}_{Q,K,V}(\mathcal{R}_v^l)$
24:            $\mathcal{R}^L = \text{Attention}(Q_v^l, K_v^l, V_v^l)$
25:            $Q_v(h_v^{t-1}, e_v(t), A(t-1)) = \digamma_v(\mathcal{R}_v^L, \boldsymbol{a}_v(t))$
26:            Set $y = r_v + \gamma\, Q_v^{\boldsymbol{\mu}'}(O', A', h_v^t)|_{\boldsymbol{a}'_v=\boldsymbol{\mu}'_v(\boldsymbol{o}_v)}$
27:            Update critic by minimizing the loss: $\mathcal{L}(\phi_v)=\mathbb{E}_{O,A,R,O'}[(Q_v^{\boldsymbol{\mu}}(O, A, h_v^{t-1}) - y)^2]$
28:            Update actor using the sampled policy gradient:
               $\mathcal{L}(\theta_v) \approx \nabla_{\theta_v} J^c(\boldsymbol{\mu}_v) + \nabla_{\theta_v} J^d(\mu_v)$(Based on Eq. (25) and (26)
29:        end for
30:        The target network parameters for each MV agent $v$ are updated by the soft update:
           $\theta'_v \leftarrow \varpi\theta_v + (1-\varpi)\theta'_v$
           $\phi'_v \leftarrow \varpi\phi_v + (1-\varpi)\phi'_v$
31:    end for
32: end for
```

*3) Centralized Critic-Network:* In Fig. 3, the critic-network is composed of LSTM and fully connected networks (FCNs). Based on MADDPG, the centralized action-value function $Q_v^{\boldsymbol{\mu}}$ is updated in

$$\mathcal{L}(\phi_v) = \mathbb{E}_{O,A,R,O'}\left[\left(Q_v^{\boldsymbol{\mu}}\left(O, A, h_v^{t-1}\right) - y\right)^2\right]$$
$$y = r_v + \gamma\, Q_v^{\boldsymbol{\mu}'}\left(O', A', h_v^{t-1}\right)|_{\boldsymbol{a}'_v=\boldsymbol{\mu}'_v(\boldsymbol{o}_v)} \tag{28}$$

where $O'$ denotes the next states of all MV agents. $A'$ denotes the next actions of all MV agents.

*4) Training Process:* Based on the above description, there are $V$ TPDMAAC agents, each operating on an MV. Through interaction with the local MV, each agent conducts their own offloading policies, which are actions performed in the MV. Furthermore, in Algorithm 1, we show the training process of TPDMAAC agent on MV $v, v \in \mathcal{V}$. The TLSFN module is described in detail in steps 14–20. Steps 20–30 provide a description of the details associated with the TTFEN and

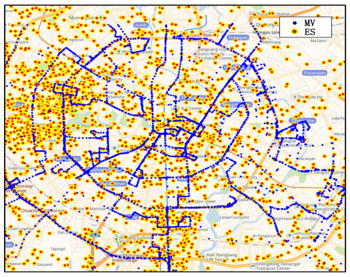

Fig. 6. Distribution of partial MVs and BSs in Chengdu city.

TABLE II
SIMULATION EXPERIMENT PARAMETER SETTING

| Parameters | Range of Values |
|---|---|
| $V$ | 622 |
| $M$ | 2087 |
| $F_v^{l^{max}}, v \in \mathcal{V}$ | 2.5 $\sim$ 5.5 GHz |
| $P_v^{l^{\max}}, v \in \mathcal{V}$ | 2 W |
| $F_m^e, m \in \mathcal{M}$ | 11.8$\sim$21.8 GHz |
| $W$ | 5$\sim$ 15Mbps |
| $d_v(t), t \in T$ | [500, 10000] KB |
| $\lambda_v(t), t \in T$ | [49, 1123] Megacycles |
| The length of each episode | 7200 |
| Episodes | 9000 |

PDN modules. Note that for the consideration of scalability, we share the parameters of actor and critic networks by all agents.

## V. Simulation Experiment

### A. Experimental Settings

*1) Data Set Description:* We evaluate the performance of the TPDMAAC model based on two real data sets. First, the real taxi data set[1] includes the trajectories of Didi taxi in Chengdu city, the time ranges from 3 February 2018 to 28 February 2018. Each trajectory consists of a set of GPS points with time points. The GPS points consist of the taxi ID, time point, taxi location, and the presence of passengers. There are 13 605 taxis in this data set. Second, the locations information of the BSs data set[2] includes the coordinates of 52 090 BSs collected by CMCC in Chengdu in December 2018. The real geographical distributions of some MVs and BSs on Chengdu city is shown in Fig. 6, where blue dots and red dots indicate MVs and BSs, respectively. We randomly select a district to evaluate our model, e.g., Qingyang district (30.653°N $\sim$ 30.705°N and 104.042°E $\sim$ 104.122°E). Moreover, we randomly selected 2087 BSs and 622 MVs driving trajectories in this district. And the chosen MVs are frequently driven near these BSs. We use the first 14 days of continuous data as the training set, the next three days of data as the validation set, and the remaining nine days of data for testing.

*2) System Setup:* Similar to this work [30], the whole experiments carried out on the server configured with Xeon Silver 4210R CPU @ 4.80 GHz, and 4×NVIDIA RTX 12G GPU. The experimental parameters and associated values are shown in Table II, which is identical to this work [18], [25].

*3) Training Setup:* The TPDMAAC model contains TLSFN, TTFEN, and PDN modules. First, for the TLSFN module, we implement it employing the Encoder–Decoder framework as shown at the right of Fig. 3. The Encoder component mainly contains the probsparse self-attention block (e.g., Multihead ProbSparse Attention, Add, LayerNorm, and Dropout layers) and distilling operations [e.g., conv 1d and max pooling (stride = 2)]. Moreover, as for the size of the vector in the model, we set to 512. The number of multiheads in the multihead attention mechanism is set 8. The dimension of the vector in the full connection layer is set to 2048. The dropout rate is set to 0.1. Similar to the Encoder layer, we adopt the same network components to build Decoder. In the following, we set the input sequence length of the TLSFN Encoder to 48 (i.e., $T_{\text{seq\_len}} = 48$). We set the starting token length of the Decoder of TLSFN to 48 and the prediction sequence length to 24. Second, for the TTFEN module, we adopt transformer to implement it as shown at the left of Fig. 3. Similar to the TLSFN module, we adopt probsparse self-attention instead of the original self-attention to reduce the model runtime delay. Finally, we implement the PDN module using self-attention and the corresponding parameters of PDN are set similar to those of the TTFEN module. Moreover, the discount factor is set to 0.99. The replay buffer size and batch size are set to 10 000 and 256, respectively.

*4) Benchmark Approaches:* We have chosen the following benchmark method to compare with TPDMAAC. First, to verify that task offloading decisions by the MV agent can be better facilitated by predicting ES task queuing delay, we compare the recent related decentralized task offloading algorithms, [20] (abbreviated as DE-VTO) and ARMAAC [24]. These two methods are improved versions of MADDPG and are decentralized task offloading schemes. And both task offloading solutions are based on improved versions of MADDPG. Second, to verify that the TPDMAAC can quickly adapt to the new environment, we compared the recent meta learning-based centralized task offloading algorithms, MR-DRO algorithm [21] and DMRO algorithm [23]. And similar to this work [13], we adapt to multiagent setting using a centralized training and centralized execution architecture. In order to enable fast convergence of the model, we use a network parameter sharing approach for training. Finally, to verify the scalability of TPDMAAC, we compared the centralized task offloading algorithm [44] (abbreviated as CMATO). This task offloading solution is based on an improved version of MADDPG. Moreover, we compared the task offloading scheme based on the traditional POMDP algorithm [3] (abbreviated as OFF_UNC).

### B. Analysis of Experimental Results

In this section, we first show the average system reward of the TPDMAAC, algorithm convergence, and migration capacity across episodes. Additionally, we compare TPDMAAC's performance to the baselines in terms of several system environmental variables.

[1]https://github.com/Phil610351/Chengdu_Taxi_Track
[2]https://github.com/Phil610351/Chengdu_BSs

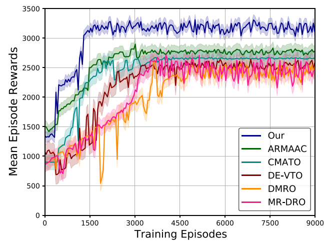

Fig. 7. Convergence and mean episode rewards of different algorithms under training episodes.

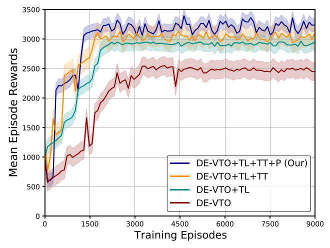

Fig. 8. Convergence and mean episode rewards of TPDMAAC under different optimization networks.

As demonstrated in Fig. 7, we show the mean episode rewards and convergence speed of the various task offloading solutions after 9000 episodes of training. From the mean episode rewards of various algorithms, we observe that the MADRL-based task offloading schemes (i.e., TPDMAAC, ARMAAC, and CMATO) are larger than the single-agent-based task offloading schemes (i.e., DE-VTO, MR-DRO, and DMRO), e.g., after various algorithms converge, the MADRL-based task offloading scheme improves the mean episode rewards by 8.26%–17.63%. This is because task offloading schemes based on a single-agent RL cannot model the complex interactions between MVs and the VFC system environment. However, we adopt MARL to simulate a multi-MVs multi-ESs task offloading environment, which allows MV agents to better capture the real interaction features among MVs and between MVs and the ESs for better collaboration and thus get higher rewards. Moreover, we introduce a forecasting network by predicting the current and future queuing delay of ESs to solve uncertain load and get higher reward. Finally, we discover that the average episode reward of the ARMAAC, CMATO, DE-VTO, DMRO, MR-DRO, and TPDMAAC-based task offloading schemes climbed with increasing episodes until convergence was reached, e.g., these algorithms reach convergence at 2800, 3700, 3200, 4533, 4622, and 1500 episodes, respectively. Clearly, this also proves that the decentralized model (i.e., TPDMAAC, ARMAAC, and DE-VTO) require less training episodes compared to the centralized model (i.e., CMATO, DMRO, and MR-DRO). This is because, with VFC networks becoming more complex and the increasing number of MVs, these centralized approaches will suffer from a huge space of states and actions, leading to difficult convergence of model training.

To verify the efficiency of these optimization modules, we undertake the corresponding ablation experiments with the

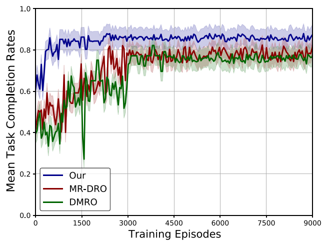

Fig. 9. Transfer between districts in Chengdu city under different algorithms.

DE-VTO algorithm. TLSFN, TTFEN, and PDN are abbreviated as TL, TT, and P, respectively. First, based on the DE-VTO algorithm, we introduce a TLSFN module. As shown in Fig. 8, the DE-VTO+TL can converge quicker and yield greater mean episode rewards compared to the DE-VTO, e.g., the DE-VTO+TL algorithm converges in the 2250 episodes and the mean episode rewards are increased by 11.62%. This is because the DE-VTO+TL can better alleviate the partial observability of the MV agents for the ESs as well as better capture the current and future load of the ESs, thus enhancing the collaboration among MV agents to improve the system reward. Second, based on the DE-VTO+TL algorithm, we further introduce the TTFEN module. As shown in Fig. 8, the DE-VTO+TL+TT can converge more quickly and achieve greater mean episode rewards compared to the DE-VTO, e.g., the DE-VTO+TL+TT converges in the 2022 episodes and the mean episode rewards are increased by 13.42%. This is because the actor-network in TPDMAAC s based on the TTFEN module using local and historical states with a special focus on capturing rich temporal features. This model not only facilitates a faster training process but also improves its stability. Finally, we further introduce a PDN module. As shown in Fig. 8, the DE-VTO+TL+TT+P algorithm method can converge faster and receive greater mean episode rewards compared to the DE-VTO algorithm, e.g., the DE-VTO+TL+TT+P algorithm converges in the 2240 episodes and the mean episode rewards are increased by 7.38%. This is because, TPDMAAC agent captures the correlation between input and output through the PDN module (i.e., self-attention mechanism) with attention weights that are more conducive to the generation of different types of policies (i.e., discrete and continuous policies), thus enhancing the accuracy of policy generation.

To verify that TPDMAAC can be quickly adapted to the new VFC system environment, we transfer the task offloading algorithm trained in Qingyang district to three other randomly selected districts (e.g., Jinjiang, Wuhou, and Jinniu). As shown in Fig. 9, TPDMAAC, MR-DRO, and DMRO reach convergence at 700, 2500, and 3080 training episodes, respectively. Obviously, TPDMAAC has stronger generalization capability for the new VFC system environment. Moreover, we can also observe that when these three algorithms reach convergence, the mean task completion rate of these algorithms is 0.8712, 0.7842, and 0.7611, respectively. Compared to these two algorithms, the TPDMAAC improves by 9.12%–11.04%. This is because the TPDMAAC can handle VFC scenarios with different inputs and outputs, thus learning a universal traffic pattern

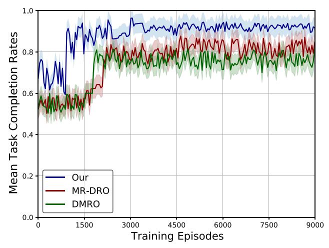

Fig. 10. Adaptability of new MVs to the VFC environment under different methods.

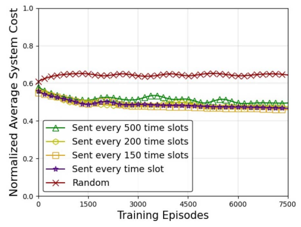

Fig. 11. Average system cost versus various network update time intervals.

in the interaction experience of different traffic forms of the MVs and the real traffic environments. The universal traffic pattern helps the MVs adapt to new environments faster, thus improving MVs task completion rates. Moreover, we consider another scenario, i.e., whether a new MV can quickly adapt to a new VFC system environment by loading a pretrained model and using only a few training. We randomly select three new MVs and generate some MV trajectories with random manner. We have done experiments under different algorithms. As shown in Fig. 10, we can observe that TPDMAAC, MR-DRO, and DMRO reach convergence at 782, 2000, and 1800 training episodes, respectively. Obviously, compared to training from scratch, the number of training episodes required to reach convergence is significantly reduced for these three algorithms, for example, these three algorithms reduce 720, 1800, and 2000, respectively. Obviously, TPDMAAC has stronger generalization capability for the new MV. Furthermore, it is worth noting that TPDMAAC still maintains a huge advantage in term of mean task completion rate.

Based on the CTDE algorithm, we consider that each MV sends parameter update requests to the Cloud Computing Center at a fixed time interval to update the parameters of the agent network deployed on them. We further explore the impact of various network update time intervals on the average system cost. As illustrated in Fig. 11, we find that updating the network parameters every 200 time slots does not have a substantial influence on the average system cost compared to updating the network parameters every time slot. This is because the data used for model training comes from the experience replay pool, not the data just generated. This indicates that updates to the network parameters can be delayed for a certain amount of time. Therefore, to reduce the communication overhead, we can reduce the frequency of network parameter updates. As shown in Fig. 12, we further explore the effect of various penalty values on the proposed algorithm. "L" denotes the penalty value for local execution in the MVs.

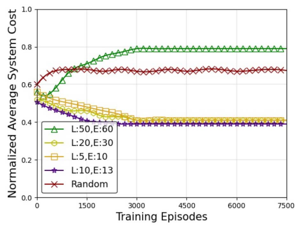

Fig. 12. Convergence of TPDMAAC under various penalty values.

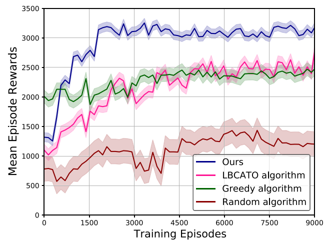

Fig. 13. Convergence and mean episode rewards of the channel selection algorithm.

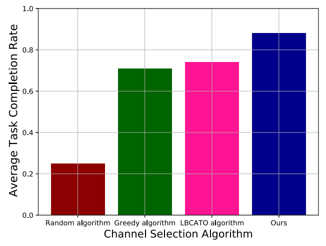

Fig. 14. Average task completion rate in channel selection algorithm.

"E" denotes the penalty value for tasks to be executed on ESs. As shown in Fig. 12, when the penalty value is set to L:10, E:13, the proposed algorithm has faster convergence and small system cost. When the penalty value is set small (i.e., L:5, E:10), the convergence rate is slow. When the penalty value is set large (i.e., L:20, E:30, L:50, E:60), the system cost increases. And the Random policy consistently maintains a high system cost.

As illustrated in Figs. 13 and 14, we further investigate the influence of various channel selection algorithms on the TPDMAAC model. First, similar to these works [49], [55], we consider that each ES contains $K (K \in [5, 12])$ communication channels. Then, we use channel selection algorithms, e.g., Random algorithm, Greedy algorithm, and the algorithm proposed in [56] (abbreviated as LBCATO) instead of the channel selection policy of the proposed algorithm. The results of the simulation experiment are shown in Fig. 13. It is evident that our proposed algorithm is the fastest converging algorithm and achieves the highest mean episode rewards, e.g., compared with the LBCATO policy and Greedy policy, the mean episode rewards of the proposed algorithm have increased by 31.81% and 36.36%, respectively. Due to the Greedy and LBCATO algorithms cannot learn cooperative strategies between MVs in channel selection, they always optimize their own strategies. This usually leads to a large number of MVs occupying the same transmission channel, resulting in higher

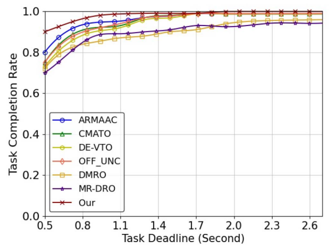

Fig. 15. Task completion rate in different task deadlines.

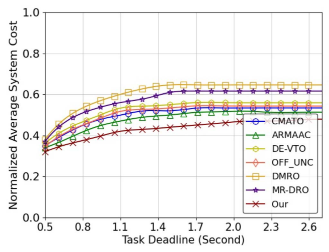

Fig. 16. Average system cost in different task deadlines.

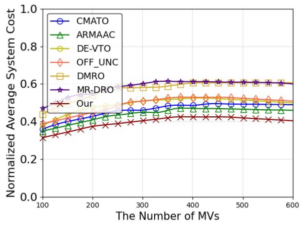

Fig. 17. Average system cost versus the number of MVs.

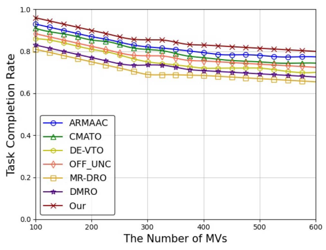

Fig. 18. Task completion rates versus the number of MVs.

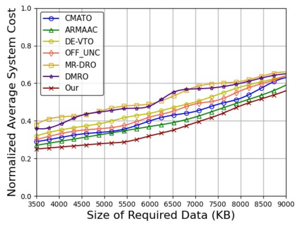

Fig. 19. Average system cost versus the size of required data.

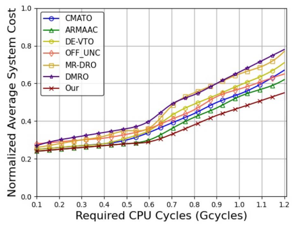

Fig. 20. Average system cost versus required CPU cycles.

task transmission latency. The proposed algorithm learns the policy of mutual cooperation among MVs, reduces the possibility of channel congestion, and improves the mean episode rewards. Moreover, as illustrated in Fig. 14, the proposed algorithm gets the highest average task completion rate. This is because the proposed algorithm provides a good channel selection strategy which optimizes channel interference between MVs that greatly improves the channel transmission rate, thus enabling more tasks to be completed within the task deadline.

As shown in Fig. 15, we further explored the impact of task deadlines on task completion rates. We can find that TPDMAAC consistently maintains a high task completion rate. For example, TPDMAAC increases task completion rate by 10.11%–19.12% compared to other approaches (e.g., ARMAAC, CMATO, and OFF_UNC) when we set the task deadline to 0.5 s. This is mainly because we introduce a long sequence forecasting network by predicting the current and future queuing delay of ESs to solve uncertain load at the ESs and thus reduce task drop rate. As shown in Fig. 16, we further explored the impact of task deadlines on average system cost. As we gradually increase the deadlines of the tasks, the average system cost of all algorithms (e.g., ARMAAC, CMATO, and OFF_UNC) will likewise rise. This is due to the fact that a task's deadline increases, most of the tasks are processed to completion, and thus the average system cost becomes larger. However, when the task deadline is very large, there is no restriction on the completion of the task and the average system cost remains constant.

As shown in Fig. 17, we can find that the average system cost grows with the number of MVs when the number of MVs is small ($V < 400$), since the VFC system (i.e., the ESs and MVs) can offer adequate transmission and processing resources to execute all offloaded or local tasks from the MVs. Nevertheless, after certain thresholds are exceeded (e.g., $V = 400$ MVs), the average system cost drops since the greater the number of MVs offloaded, the heavier the competition for CPU utilization. It is worth mentioning that, when compared to other methods, TPDMAAC consistently preserves a relatively low the average system cost, e.g., compared to other algorithms (e.g., ARMAAC, CMATO, and OFF_UNC), for the average system cost, the TPDMAAC reduces 15.34%–22.78%. This is because the TPDMAAC can better capture the competition and collaboration between MVs for resources, which can obtain a more collaborative model, thus reducing the average system cost. When compared to other algorithms, TPDMAAC consistently maintains a high task completion rate as shown in Fig. 18, especially when the number of MVs is large, e.g., compared to other algorithms (e.g., ARMAAC, CMATO, and OFF_UNC), for the task completion rate, the TPDMAAC improves 12.22%–24.65%. This is because by predicting the current and future task queuing delays of ES, TPDMAAC can effectively solve the uncertain load at the ESs. This promotes optimal task offloading policies for MV agents to improve task completion rates.

As shown in Fig. 19, we set data sizes ranging from 3500–9000 kB to simulate the generation of application tasks in each

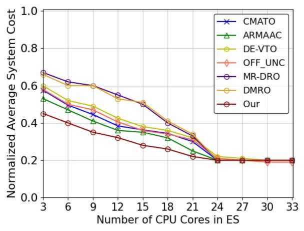

Fig. 21. Average system cost versus ES computing capability.

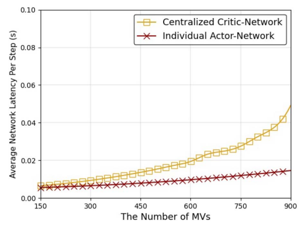

Fig. 22. Average critic and actor network latency per step.

MV. Apparently, we discover that higher task sizes leads to greater the average system cost compared to other algorithms (e.g., CMATO and OFF_UNC). This is because the ESs are more effective than the local MVs in decreasing the computational cost when dealing with large-scale tasks because of the high ESs capability. Further, we show the change in the average system cost of each algorithm as the number of CPU cycles required increases in Fig. 20. We can find that the average system cost of all algorithms increases as the required CPU cycles become larger, which is due to the increase in both energy consumption and latency of task execution. In particular, when the required CPU cycles are 0.6–1.2 Gcycles, the system cost of all offloading solutions rises quickly, which indicates that the load of each ES is high. It is noteworthy that the TPDMAAC keeps a low average system cost compared to other algorithms (e.g., ARMAAC and OFF_UNC), which indicates that the TPDMAAC is able to allocate resources efficiently.

As shown in Fig. 21, we further investigate the impact of the computational power of the ESs on the system cost of these offloading algorithms. In Fig. 21, when the number of CPU cores rises, the average system cost reduces. This is because high computing capacity can mitigate the contradiction between decreasing energy consumption and saving execution time. Moreover, TPDMAAC has always maintained a lower the average system cost compared to other algorithms (e.g., CMATO and OFF_UNC). This indicates that TPDMAAC can improve resource utilization in the face of resource-constrained ESs. This is mainly because TPDMAAC is able to predict the future load of ESs, which greatly improves the accuracy of MV agent decisions.

## *C. TPDMAAC's Complexity Analysis (Algorithm 1)*

Based on the description of Section IV, we further analyze the complexity of the TPDMAAC model (i.e., ProbSparse self-attention, LSTM, and FCN). And this is a complexity analysis of Algorithm 1 training process. First, according to [51] and [57], we can obtain the time and space complexity of ProbSparse self-attention in the training process as $\mathcal{O}(T_{\mathrm{seq\_len}} \log T_{\mathrm{seq\_len}})$ and $\mathcal{O}(T_{\mathrm{seq\_len}} \log T_{\mathrm{seq\_len}})$, respectively. $T_{\mathrm{seq\_len}}$ denotes the length of the historical ES task queuing delay. It is worth noting that the complexity of ProbSparse self-attention is $\mathcal{O}(1)$ in the testing phase. Since TLSFN mainly consists of ProbSparse self-attention, thus TLSFN has the same complexity. Second, we assume that the length of the input sequence is $S_n$ and the dimension of each element is $S_d$. According to [51], the complexity of LSTM is $\mathcal{O}((S_n)^2 \cdot S_d)$. And the complexity of LSTM is $\mathcal{O}(S_n)$ in the testing phase. In a centralized critic-network, $S_n$ is proportional to the number of MVs since the information of all MV agents is fused together in the centralized training phase. Finally, according to [24], we can obtain the complexity of FCN as $\mathcal{O}((S_n)^2 \cdot S_d^2)$. Based on the above analysis and the components of each module, we can get the complexity of each module as shown in Table III.

TABLE III
MODEL COMPLEXITY ANALYSIS

| Module | Complexity |
|---|---|
| ProbSparse Self-attention Module | $\mathcal{O}(T_{seq\_len}\ log\ T_{seq\_len})$ |
| TLSFN Module | $\mathcal{O}(T_{seq\_len}\ log\ T_{seq\_len})$ |
| LSTM Module | $\mathcal{O}(S_n \cdot S_d^2)$ |
| TTFEN Module | $\mathcal{O}(S_n log S_n)$ |
| PDN Module | $\mathcal{O}(S_n log S_n)$ |
| FCN Module | $\mathcal{O}((S_n)^2 \cdot S_d^2)$ |
| Centralized Critic-Network | $\mathcal{O}((S_n)^2 \cdot S_d^2)$ |
| Individual Actor-Network | $\mathcal{O}(1^2 \cdot S_d^2)$ |
| Training Networks | $\mathcal{O}((S_n)^2 \cdot S_d^2)$ |
| Test Networks | $\mathcal{O}(1^2 \cdot S_d^2)$ |

According to the above description, we find that $S_n$ is proportional to the number of MVs. We further explored the effect of the number of MVs on the average network execution delay per step. As for the training and execution of TPDMAAC, we consider that the data center first trains the actor-network and critic-network in a centralized way. Then, the MVs can execute task offloading and resource allocation decisions in a distributed way. As shown in Fig. 22, we can find that the per-step network execution delay rises for these two networks as the number of MVs increases. It is noteworthy that the execution delay of the Centralized Critic-Network rises faster. Nevertheless, there was no significant change in the execution delay of the Individual Actor-Network. This is due to the Centralized Critic-Network is proportional to the quadratic side of the number of MVs according to the complexity of the network, however, the Individual Actor-Network is not affected by the number of MVs. When the number of MVs is 600, the execution delay of the Centralized Critic-Network is 0.02 s per step. This delay is 40–60 times away from task deadline 0.8–1.2 s and thus this execution delay can be considered as a constant and neglected. When the number of MVs reaches to a certain number (e.g., 0.1-s network execution latency per step), the execution delay of the centralized critic-network will impact the execution of tasks. To solve this problem, we use the action of the last time slots [i.e., $A(t-1)$] as input to eliminate the effect of the delay generated by the Centralized Critic-Network on task completion similar to these works [37], [53]. This is because the states, actions and generated tasks of the MEC system in adjacent time slots have spatiotemporal properties and are therefore similar [54], [58]. Based on the above analysis, the trained the Individual Actor-Network can be used for online decision making for real-time applications.

## VI. Conclusion

We investigated the task offloading and resource allocation problem in a partially observable heterogeneous VFC system and proposed a TPDMAAC-based decentralized task offloading scheme. Each MV agent can select the optimal computing node and allocate the corresponding resources for the arrival task. Moreover, distinguished from the existing works, TPDMAAC has two significant advantages. First, TPDMAAC can handle uncertain load in the ESs. Second, TPDMAAC can quickly adapt to the new VFC environment by fine-tuning.

In our future work, we will continue to explore task offloading solutions in the following three aspects. First, graph neural networks (GNNs) have been widely used in various domains, the ESs and MDs exist in a natural form of graph structure. Next, we explore the schemes based on the joint GNNs and MARL. Second, incentivizing MVs to actively participate in the VFC system is also an important research topic. This is because the number of participating MVs has a significant impact on both system costs and task completion rates In future work, we will further explore efficient incentive mechanisms, e.g., Auction algorithms and RL-based pricing algorithms.

## References

[1] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.

[2] J. Zhang, J. Du, Y. Shen, and J. Wang, "Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9303–9317, Oct. 2020.

[3] P. A. Apostolopoulos, G. Fragkos, E. E. Tsiropoulou, and S. Papavassiliou, "Data offloading in UAV-assisted multi-access edge computing systems under resource uncertainty," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 175–190, Jan. 2021.

[4] J. Li et al., "Budget-aware user satisfaction maximization on service provisioning in mobile edge computing," *IEEE Trans. Mobile Comput.*, early access, Sep. 9, 2022, doi: 10.1109/TMC.2022.3205427.

[5] J. Shi, J. Du, J. Wang, J. Wang, and J. Yuan, "Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 16067–16081, Dec. 2020.

[6] J. Shi, J. Du, J. Wang, and J. Yuan, "Federated deep reinforcement learning-based task allocation in vehicular fog computing," in *Proc. IEEE 95th Veh. Technol. Conf.*, 2022, pp. 1–6.

[7] Q. Wu, H. Liu, R. Wang, P. Fan, Q. Fan, and Z. Li, "Delay-sensitive task offloading in the 802.11 p-based vehicular fog computing systems," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 773–785, Jan. 2020.

[8] J. Shi, J. Du, J. Wang, and J. Yuan, "Deep reinforcement learning-based V2V partial computation offloading in vehicular fog computing," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2021, pp. 1–6.

[9] J. Shi, J. Du, J. Wang, and J. Yuan, "Distributed V2V computation offloading based on dynamic pricing using deep reinforcement learning," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2021, pp. 1–6.

[10] L. Zhao et al., "Vehicular computation offloading for industrial mobile edge computing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7871–7881, Nov. 2021.

[11] S. A. Kazmi et al., "Computing on wheels: A deep reinforcement learning-based approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 11, pp. 22535–22548, Nov. 2022.

[12] T. M. Ho and K.-K. Nguyen, "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2421–2435, Jul. 2022.

[13] H. Li, K. D. Assis, S. Yan, and D. Simeonidou, "DRL-based long-term resource planning for task offloading policies in multi-server edge computing networks," *IEEE Trans. Netw. Service Manag.*, early access, Jul. 18, 2022, doi: 10.1109/TNSM.2022.3191748.

[14] B. Hazarika, K. Singh, S. Biswas, and C.-P. Li, "DRL-based resource allocation for computation offloading in IoV networks," *IEEE Trans. Ind. Informat.*, vol. 18, no. 11, pp. 8027–8038, Nov. 2022.

[15] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.

[16] L. Zhao, K. Yang, Z. Tan, X. Li, S. Sharma, and Z. Liu, "A novel cost optimization strategy for SDN-enabled UAV-assisted vehicular computation offloading," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3664–3674, Jun. 2021.

[17] J. Shi, J. Du, Y. Shen, J. Wang, J. Yuan, and Z. Han, "DRL-based V2V computation offloading for blockchain-enabled vehicular networks," *IEEE Trans. Mobile Comput.*, early access, Feb. 23, 2022, doi: 10.1109/TMC.2022.3153346.

[18] X. Zhu, Y. Luo, A. Liu, M. Z. A. Bhuiyan, and S. Zhang, "Multiagent deep reinforcement learning for vehicular computation offloading in IoT," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9763–9773, Jun. 2020.

[19] Z. Jia, Z. Zhou, X. Wang, and S. Mumtaz, "Learning-based queuing delay-aware task offloading in collaborative vehicular networks," in *Proc. IEEE Int. Conf. Commun.*, 2021, pp. 1–6.

[20] H. Zhu, Q. Wu, X.-J. Wu, Q. Fan, P. Fan, and J. Wang, "Decentralized power allocation for MIMO-NOMA vehicular edge computing based on deep reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 14, pp. 12770–12782, Jul. 2022.

[21] Z. Zhang, N. Wang, H. Wu, C. Tang, and R. Li, "MR-DRO: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments," *IEEE Internet Things J.*, early access, Nov. 8, 2021, doi: 10.1109/JIOT.2021.3126101.

[22] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[23] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, Sep. 2021.

[24] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mobile Comput.*, early access, Jan. 7, 2022, doi: 10.1109/TMC.2022.3141080.

[25] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[26] S. Mehta, M. Ghazvininejad, S. Iyer, L. Zettlemoyer, and H. Hajishirzi, "Delight: Deep and light-weight transformer," 2021, *arXiv:2008.00623*.

[27] B. Li et al., "Learning light-weight translation models from deep transformer," in *Proc. AAAI*, 2021, pp. 1–9.

[28] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," 2020, *arXiv:2001.04451*.

[29] J. Xu, H. Wu, J. Wang, and M. Long, "Anomaly transformer: Time series anomaly detection with association discrepancy," 2021, *arXiv:2110.02642*.

[30] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A DRL agent for jointly Optimizing computation offloading and resource allocation in MEC," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17508–17524, Dec. 2021.

[31] M. Fayyaz et al., "Adaptive token sampling for efficient vision transformers," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 396–414.

[32] Y. Xue, J. Chen, Y. Zhang, C. Yu, H. Ma, and H. Ma, "3D human mesh reconstruction by learning to sample joint adaptive tokens for transformers," in *Proc. 30th ACM Int. Conf. Multimedia*, 2022, pp. 6765–6773.

[33] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor–critic for mixed cooperative–competitive environments," 2017, *arXiv:1706.02275*.

[34] T. Liu, J. Li, F. Shu, and Z. Han, "Optimal task allocation in vehicular fog networks requiring URLLC: An energy-aware perspective," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 3, pp. 1879–1890, Jul./Sep. 2020.

[35] M. Feng, M. Krunz, and W. Zhang, "Joint task partitioning and user association for latency minimization in mobile edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 8108–8121, Aug. 2021.

[36] C. Zhu et al., "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4150–4161, Jun. 2019.

[37] C. Chen et al., "Delay-optimized V2V-based computation offloading in urban vehicular edge computing and networks," *IEEE Access*, vol. 8, pp. 18863–18873, 2020.

[38] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Collaborative data scheduling for vehicular edge computing via deep reinforcement learning," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9637–9650, Oct. 2020.

[39] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, pp. 1–24, 2020.

[40] H. Ke, J. Wang, L. Deng, Y. Ge, and H. Wang, "Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916–7929, Jul. 2020.

[41] W. Zhan et al., "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5449–5465, Jun. 2020.

[42] M. Z. Alam and A. Jamalipour, "Multi-agent DRL-based Hungarian algorithm (MADRLHA) for task offloading in multi-access edge computing Internet of Vehicles (IoVs)," *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 7641–7652, Sep. 2022.

[43] X. Zhang, M. Peng, S. Yan, and Y. Sun, "Joint communication and computation resource allocation in fog-based vehicular networks," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13195–13208, Aug. 2022.

[44] H. Peng and X. Shen, "Multi-agent reinforcement learning based resource management in MEC-and UAV-assisted vehicular networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 131–141, Jan. 2021.

[45] J. Zhang, S. Chen, X. Wang, and Y. Zhu, "DeepReserve: Dynamic edge server reservation for connected vehicles with deep reinforcement learning," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[46] F. Wang, Y. Xu, L. Zhu, X. Du, and M. Guizani, "LAMANCO: A lightweight anonymous mutual authentication scheme for $N$-times computing offloading in IoT," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4462–4471, Jun. 2019.

[47] S. Chen, X. Zhu, H. Zhang, C. Zhao, G. Yang, and K. Wang, "Efficient privacy preserving data collection and computation offloading for fog-assisted IoT," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 4, pp. 526–540, Oct.–Dec. 2020.

[48] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[49] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1085–1101, May 2021.

[50] H. Liu and G. Cao, "Deep reinforcement learning-based server selection for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13351–13363, Dec. 2021.

[51] A. Vaswani et al., "Attention is all you need," 2017, *arXiv:1706.03762*.

[52] J. Li, X. Hui, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," 2021, *arXiv:2012.07436*.

[53] B. Chen, M. Xu, L. Li, and D. Zhao, "Delay-aware model-based reinforcement learning for continuous control," *Neurocomputing*, vol. 450, pp. 119–128, Aug. 2021.

[54] W. Zhang et al., "Intelligent electric vehicle charging recommendation based on multi-agent reinforcement learning," in *Proc. Web Conf.*, 2021, pp. 1856–1867.

[55] Z. Cao, P. Zhou, R. Li, S. Huang, and D. Wu, "Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6201–6213, Jul. 2020.

[56] C. Yang, B. Liu, H. Li, B. Li, K. Xie, and S. Xie, "Learning based channel allocation and task offloading in temporary UAV-assisted vehicular edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 9, pp. 9884–9895, Sep. 2022.

[57] H. Zhou et al., "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. AAAI*, 2021, p. 8.

[58] Y. Wang, T. Xu, X. Niu, C. Tan, E. Chen, and H. Xiong, "STMARL: A spatio-temporal multi-agent reinforcement learning approach for cooperative traffic light control," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 2228–2242, Jun. 2022.

**Zhen Gao** received the M.S. degree from the School of Computer Science and Engineering, Northeastern University, Shenyang, China, in June 2020.

His research interests include mobile-edge computing, task offloading, and multiagent reinforcement learning.

**Lei Yang** was born in Liaoning, China, in 1974. He received the Ph.D. degree in computer application and technology from Northeastern University, Shenyang, China, in 2008.

He is an Associate Professor with the School of Computer Science and Engineering, Northeastern University. His current research interests include computer networks and mobile-edge computing.

**Yu Dai** was born in Liaoning, China, in 1980. She received the Ph.D. degree in computer application and technology from Northeastern University, Shenyang, China, in 2008.

She is an Associate Professor with the College of Software, Northeastern University. Her current research interests include computer networks and mobile-edge computing.