Zhu, Chao; Chiang, Yi-Han; Xiao, Yu; Ji, Yusheng

FlexSensing

Published in:
IEEE Internet of Things Journal

DOI:
10.1109/JIOT.2020.3040615

Published: 01/05/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Zhu, C., Chiang, Y.-H., Xiao, Y., & Ji, Y. (2021). FlexSensing: A QoI and Latency Aware Task Allocation Scheme for Vehicle-based Visual Crowdsourcing via Deep Q-Network. *IEEE Internet of Things Journal*, *8*(9), 7625-7637. https://doi.org/10.1109/JIOT.2020.3040615

FlexSensing: A QoI and Latency Aware Task Allocation Scheme for Vehicle-based Visual Crowdsourcing via Deep Q-Network

Chao Zhu, Yi-Han Chiang, Yu Xiao* and Yusheng Ji

***Abstract*—Vehicle-based visual crowdsourcing is an emerging paradigm where the visual data collected from dash cameras are analyzed with the aim of measuring phenomena of common interest. To ensure the efficiency in vehicle-based visual crowdsourcing, there remain at least two technical challenges. First, to maximize the quality of information (QoI), which measures the amount of information extracted from the collected data, the context of data collection (e.g., camera position and orientation) must be taken into account in the process of task allocation. Second, intensive data collection from dense measurement points is key to ensure timely and accurate sensing of the targets of interest, whereas there exists a trade-off between the amount and rate of data collection and the computing and communication resources required to fulfill the latency constraint. To solve these challenges, we propose gathering and processing the collected data at the edge of the network and design a context-aware task allocation scheme, called FlexSensing, to jointly optimize the QoI and processing latency. We target application scenarios where commercial vehicles are turned into vehicular fog nodes (VFNs). These nodes gather and process the visual data collected from other vehicles within their coverage areas. The key idea of FlexSensing is to determine the rate of data collection for each sensing vehicle in the targeted area and to assign processing tasks to VFNs based on the estimated QoI and the workload of the VFNs. Given the excessive computational complexity of task allocation in this context, we formulate task allocation as a Markov decision process and apply a deep Q-network (DQN) to learn the optimized task allocation strategies for increasing the QoI of collected data while reducing the processing latency. To evaluate the effectiveness of FlexSensing, we simulate the mobility of different vehicles involved in the scenario at different times of the day based on real-world traffic data collected from the city of Helsinki and select a real-time object detection application for a case study. As compared with the existing task allocation strategies, the DQN-based task allocation strategies reduce the average processing latency by up to 51% and increase the QoI of the collected data by up to 34%.**

***Index Terms*—Vehicular Fog Computing (VFC), Vehicle-based Visual Crowdsourcing, Deep Q-network (DQN).**

C. Zhu and Y. Xiao are with the Department of Communications and Networking, Aalto University, Espoo, Finland (e-mail: {chao.1.zhu,yu.xiao}@aalto.fi).

Y.-H. Chiang is with the Department of Electrical and Information Systems, Osaka Prefecture University, Osaka, Japan (email: chiang@eis.osakafu-u.ac.jp)

Y. Ji is with the Information Systems Architecture Science Research Division, National Institute of Informatics, Tokyo, Japan (e-mail: kei@nii.ac.jp).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 815191, Academy of Finland under grant number 317432 and 318937, JSPS KAKENHI under grant number JP18KK0279, JP20H00592, and JP20K19794.
*The corresponding author is Yu Xiao.

I. Introduction

Vehicle-based visual crowdsourcing is an emerging computing paradigm that utilizes the images/video collected from vehicles to measure phenomena of common interest, such as real-time traffic information and high-definition maps for autonomous driving [1], [2]. To accurately sense the targets of interest in a timely way, a widely adopted approach is to frequently collect high-quality visual data from dense measurement points. However, collecting and processing a large amount of visual data from moving vehicles require a tremendous number of communications and computing resources. In addition, latency constraints on time-sensitive information extraction require moving the computing resources closer to where the data are generated.

This paper proposes FlexSensing, a task allocation scheme that jointly optimizes the quality of information (QoI) and the processing latency in the case of vehicle-based visual crowdsourcing. Here, the QoI measures the amount of information extracted from the collected data [3]–[5]. It depends on the context of the data collection, such as the position, orientation, and moving speed of the cameras. In this paper, we take object detection as an example of a visual-crowdsourcing-based application and calculate the QoI as the number of pixels covering the targeted objects in each image.

To provide low-latency processing for the collected visual data, we propose applying the concept of vehicular fog computing (VFC) [6]. More specifically, we propose turning commercial vehicles into *vehicular fog nodes* (VFNs) equipped with CPU/GPU and V2X modules and to utilize these nodes for processing the visual data collected from other vehicles within the range of one-hop communication. We select commercial vehicles such as buses as carriers of VFNs due to their large dimensions and sufficient power supply.

The core idea of FlexSensing is to minimize the data collection rate for each sensing vehicle in the target area and to optimize the task distribution among VFNs to reduce the processing latency without degrading the QoI. We formulate this optimization problem as a constraint-aware Markov decision process (MDP) and leverage the advanced deep Q-learning network (DQN) to solve the problem as described below.

First, we evenly divide an urban area into service zones with a cellular base station in the center. The base station is configured as the coordinator of the service zone and runs a DQN agent. Second, we simulate the workload of VFNs based on the spatiotemporal distribution of vehicular traffic,

with the assumption that the VFN workload is proportional to the density of vehicle traffic on the road. In the initial stage, the DQN agents assign processing tasks to VFNs and select the data collection rate for each sensing vehicle (i.e., data collector) in a random manner. Gradually, based on the learned variation pattern in the workload of VFNs and the observed results of past decisions, the DQN agents learn and update the task allocation strategies (i.e., the frame rate of data collection for each sensing vehicle and the computing tasks assigned to each VFN) toward specific application demands. Notably, the learning process is purely based on experience, without any predefined rules.

To evaluate the effectiveness of FlexSensing in terms of QoI and processing latency, we simulate the scenario of vehicle-based visual crowdsourcing using real-world vehicular traffic data. We take an area of $1km^2$ in the center of Helsinki for the case study. Compared with the existing solutions, including adaptive [7] and MUEECA [8] task allocation strategies, the task allocation strategies provided by FlexSensing reduce the processing latency by up to 51% and increase the QoI by up to 34%. The contributions of this paper can be summarized as follows.

- We develop FlexSensing, a DQN-based task allocation scheme that jointly optimizes the QoI and processing latency in vehicle-based visual crowdsourcing, taking into account the spatiotemporal variation in the VFN workload.
- We prove the effectiveness of FlexSensing through large-scale simulation, using the profiles of a real-world object recognition application and traffic data as the input.

The rest of the paper is organized as follows. Section II discusses the related works. Section III describes the system. The system assumptions and problem formulation are presented in Section IV. The methodology for seeking the optimal task allocation strategies is described in Section V. We discuss the evaluation configuration and the results in Section VI and finally conclude this paper in Section VIII.

II. Related Work

A. Vehicle-based Visual Crowdsourcing

With the wide adoption of dash cameras, several works have been devoted to the study of collecting and processing images/video from vehicles to measure common interests, such as parking space detection [1], [9]–[16] and road surface monitoring [17], [18]. Coric *et al.* [11] utilized ultrasonic sensors and web cameras preinstalled on vehicles to identify legal street parking spaces. Shi *et al.* [12] proposed a logistic-regression-based method to evaluate the reliability of crowd-sourced knowledge for real-time parking space information. Grassi *et al.* [13] proposed parked-car localization algorithms that fuse information from the camera, GPS and inertial sensors of on-board mobile phones. Zhu *et al.* [14] analyzed the feasibility of vehicle-based visual crowdsourcing based on the exploration of the variation in parking space availability and vehicular traffic in urban areas.

In addition to parking space detection, Qiu [15] *et al.* proposed an augmented-reality-based system called AVR to broaden a vehicle's visual horizon by gathering and sharing visual information from neighboring vehicles. Hara *et al.* [16] identified street-level accessibility problems by combining visual information from vehicle-based visual crowdsourcing and Google Street View. Omer *et al.* [19] proposed a road surface monitoring system based on GPS-tagged images for distinguishing three types of snow coverage conditions: bare, wheel track bare, and fully snow covered. Qian *et al.* [20] presented a system for the classification of road conditions using still frames taken from uncalibrated dashboard cameras.

Several works proposed applying vehicular fog computing (VFC) to the real-time analytics of crowdsourced dash camera video. Zhu *et al.* [9] analyzed the feasibility of real-time video crowdsourcing under the vehicular fog computing architecture and tested VFC-based video crowdsourcing in real-world vehicular traffic network scenarios in terms of the network latency, packet loss ratio, and throughput. Ni *et al.* [1] examined the architecture of fog-based vehicular crowdsourcing with consideration of the security, privacy, and fairness.

B. Task Allocation in VFC

Several research works have investigated task allocation algorithms in fog/edge computing in terms of the minimization of latency [21]–[26]. Qiao *et al.* [24] proposed utilizing the graph-theory-based maximum weight independent set (MWIS) to remove redundant tasks to guarantee low computational and communication latency. Sun *et al.* [25] developed a multi-armed bandit (MAB)-based task offloading framework to minimize the average offloading delay, in which vehicles are enabled to learn the potential task offloading performance of their neighboring vehicles. Ning *et al.* [26] designed an Edmonds-Karp-based algorithm to minimize the response delay for traffic management by load balancing among the cloudlet and fog nodes. Huang *et al.* [27] proposed an SDN-based vehicle-to-vehicle (V2V) offloading method to solve computational offloading problems in the highway environment, in which the offloading performance, including offloading fraction, network throughput and average lifetime, can be improved. To the best of our knowledge, FlexSensing is the first joint optimization method to provide QoI and data processing latency aware task allocation strategies in vehicle-based visual crowdsourcing, with consideration of the variation in fog node workload and vehicular mobility.

III. System Description

In this section, the related terms are defined, and an overview of the process of FlexSensing is presented.

A. Related Terms

VFNs: The computing nodes carried by moving vehicles (e.g., buses) with vehicle-to-everything (V2X) capacities. In FlexSensing, the VFNs are responsible for visual data processing.

Service Zones: Currently, urban areas in modern cities are completely covered by cellular networks. Similar to [28],

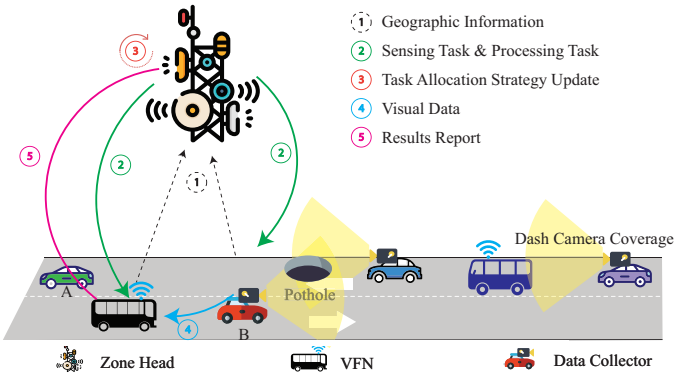


Fig. 1: Overview of FlexSensing.

[29], we divide an urban area into service zones of the same size based on the locations of the base stations [30]. The base station located in the center of each zone is selected to coordinate all the VFNs within the zone. We call the coordinator the ***zone head***. With the existing cellular registration mechanisms, vehicles always inform the zone head when they enter or leave the zone. In addition, VFNs periodically report their moving directions, locations, and available computing and communication resources to the zone head. Note that the locations of targets of interest are known beforehand and that the locations and dynamics of other vehicles are monitored by the system.

Data Collectors: In FlexSensing, we assume that all vehicles are equipped with dash cameras and V2X modules. Phenomena of common interest, such as the condition of the road surface, can be measured by collecting and analyzing the visual data captured by cameras installed on the vehicles. We consider any vehicle whose view covers a specific target of interest to be a *data collector*. For illustration, we choose crowdsourced road surface monitoring as an example. As shown in Fig.1, the pothole on the road surface is in the field of view of the dash camera installed on red vehicle B. Thus, red vehicle B becomes a data collector.

Customer Vehicles: A vehicle is defined as a *customer vehicle* of a VFN within its communication range. Customer vehicles running vehicle applications (e.g., real-time lane changing assistance) can request computing services from VFNs. Note that a vehicle can be a data collector and a customer vehicle at the same time. As shown in Fig.1, both green vehicle A and red vehicle B are customer vehicles of the VFN on the left.

Crowdsourcing Tasks: We define two types of crowdsourcing tasks in FlexSensing: visual data collection and real-time data processing. As shown in Fig.1, to monitor the condition of a road surface, the zone head periodically assigns crowdsourcing tasks to VFNs within the service zone. More specifically, the visual data captured by data collectors within the service zone are transferred to selected VFNs through V2V connections and processed there in real time.

VFN Workload: For simplicity, we assume that all the customer vehicles run the same visual processing tasks and generate service demand at the same rate. Thus, the workload of a VFN can be estimated by the number of customer vehicles within the communication range, which is closely related to the density of vehicle traffic on the road.

Processing Latency: The larger the amount and the higher the rate of data collection are, the greater the number of computing resources required to process the data. We assume that multiple processes can be run in parallel on each VFN and that the number of parallel processes depends on the number of tasks to be processed simultaneously. Furthermore, the parallel processes are allowed to share CPUs and other system resources. Therefore, with a limited number of computing resources, the larger the number of processes running on the VFN is, the smaller the number of computing resources allocated to each process and the longer the executing time for each process. In FlexSensing, we use *processing latency* to denote the average executing time of processes running on the VFN.

Data Collection Rate and QoI of Data: To accurately sense the targets of interest (e.g., road surface or parking spaces) in a timely manner, a widely adopted approach is to frequently collect high-quality visual data from dense measurement points. In FlexSensing, we evaluate the frame rate selected for transferring the visual data as a metric of the data collection rate.

Depending on the moving speed, orientation and position of the vehicle in question, the QoI, which depends on the quality and content of the collected visual data, varies. The definition of QoI is application-specific. In this work, we choose object detection as an example processing task. In this case, we propose measuring the QoI of crowdsourced visual data by using the number of pixels covering the targets of interest in each image frame. In practice, when the camera is closer to the target, or when the resolution is higher, the number of pixels covering the targets of interest is expected to increase.

System Reward: In FlexSensing, the QoI is highly likely to increase when more data are collected, whereas the processing latency will also increase with the data collection rate. To address the balance, we define a system reward, which decays along with the processing latency and increases with the QoI. In FlexSensing, the system reward unifies the processing latency and the QoI and can be tuned according to application-specific requirements. For example, if an application running on VFNs is more latency sensitive, the system reward will be more heavily weighted in favor of the processing latency, and vice versa.

B. Process of Vehicle-based Visual Crowdsourcing

Fig.III illustrates the vehicle-based visual crowdsourcing process in FlexSensing. The whole process consists of four operations.

- *Information Collection.* Once a crowdsourcing task arrives, the zone head first collects the information of the data collectors and VFNs within the service zone. The geographic information of the involved vehicles, such as the location, speed and driving direction, are collected and gathered at the zone head. The VFNs are also required to report their workload information.
- *Task Allocation Strategy Update.* Based on the collected information, the zone head learns and updates the *task*

allocation strategy, in which each data collector in the service zone is asked to transmit the camera-captured visual data to a selected VFN with a proper frame rate. Although desirable, it is challenging to achieve an optimal task allocation strategy due to the massive scale of data collectors, diversified variation of the VFN workload, and uncertain movement of vehicles. To this end, we make effective use of the advanced DQN to tackle the key challenges therein. The zone head maintains a DQN agent, which has no prior knowledge in the beginning but progressively learns to optimize the task allocation strategy based on the experienced observed performance.

- *Visual Data Collection and Processing.* Once the task allocation strategy is confirmed, each data collector tries to build a V2V connection with the selected VFN and starts to transmit the visual data at a specific frame rate.

  Once a frame is completely transferred, the VFNs process it in real time using object detection technologies (e.g., CNN [31]). For example, parked vehicles are recognized and calculated to detect free parking spots, and faces of suspicious persons are featured for crime scene reconstruction.
- *Results Reporting.* Due to the mobility of vehicles, VFNs may leave the current service zone and enter another one before a frame is completely processed. In this case, the VFNs report the results to the new zone head, who then hands over the results to the previous one through the X2 interface in LTE/LTE-A [32].

IV. System Assumption and Problem Formulation

In this section, we first introduce the system assumptions. Then, we model the FlexSensing system and present the optimization framework. The notations and definitions are summarized in Table I.

TABLE I: Notations and definitions

Notations	Definitions
j	decision epoch
$w, \mathcal{W}^j$	data collector at decision epoch j, set
$v, \mathcal{V}^j$	VFN at decision epoch j, set
$t, \mathcal{T}^j$	target of interest at decision epoch j, set
$h, \mathcal{H}$	frame rate level, set
c_w	VFN responsible for collecting the visual data from data collector w
h_w	frame rate level selection for data collector w
$(c_w, h_w), \mathcal{Y}^j$	joint command for data collector w, set
$(x_w, y_w), \theta_w, s_w$	coordinates, driving direction and speed of the data collector w
$(x_v, y_v), \theta_v, s_v$	coordinates, driving direction and speed of the VFN v
$\mathcal{R}$	effective transmission range between the VFN and data collector
(x_t, y_t)	coordinates of target of interest t
g_{wt}	effective travel distance of the data collector w on the target of interest t
$\mathbb{D}_{wv}$	connected duration between the data collector w and the VFN v
$\mathbb{D}_{wt}$	effective coverage duration between the data collector w and the target of interest t
r, η	effective range, field-of-view of the data collector dash camera
$f, \mathcal{F}_w$	effective frame transferred by the data collector w, set
l_f	depth of f^{th} frame transferred by the data collector w
θ_f	orientation of f^{th} frame transferred by the data collector w
$\mathbb{P}_w$	total number of pixels on a target of interest in all frames collected from the data collector w
P_v^j	total number of processes executed by VFN v
C_v^j	number of customer vehicles surrounding VFN v at decision epoch j
$N_v(h_w)$	number of processes occupied by the crowdsourcing task when the frame rate level is h_w
$\zeta_v(P_v^j)$	processing latency when P_v^j processes are being executed on VFN v
$P(l_f, \theta_f)$	number of pixels on a specific target of interest in the f^{th} frame
$\bar{v}$	average distance between vehicles surrounding the target of interest
$B(\bar{v})$	block probability when the average distance between vehicles surrounding the target of interest is $\bar{v}$

A. System Assumptions

1) Crowdsourcing Task and Task Allocation Strategy: In this paper, we assume that the crowdsourcing tasks arrive at the zone head one by one and at a fixed rate. We discretize a day (24 hours) into decision epochs, each of which is of equal duration δ (in seconds) and is indexed by an integer $j \in \mathbb{Z}_+$.

During a decision epoch j, we consider a service zone including a set $\mathcal{V}^j = \{1, ..., V\}$ of VFNs and a set $\mathcal{W}^j = \{1, ..., W\}$ of data collectors. A VFN is assigned to each data collector. Data collectors transfer the collected visual data to the assigned VFN, where the data are processed in real time. Moreover, we assume that the frame rate for data collection can be adjusted and include optional frame rates in a set $\mathcal{H} = \{1, ..., H\}$ and $h \in \mathbb{Z}_+$. Without loss of generality, we assume that the greater $h \in \mathcal{H}$ is, the higher the frame rate for data transfer.

As illustrated in Section III, at the beginning of decision epoch j, the zone head collects the relevant information of vehicles and sends a joint control command (c_w, h_w) to each data collector, where $c_w \in \mathcal{V}^j$ denotes the VFN responsible for collecting visual data from data collector w and $h_w \in \mathcal{H}$ is the selected frame rate level. Specifically, the data collector w transfers its captured visual data to VFN c_w at a frame rate h_w.

2) Connected Duration: We assume that all vehicles in the service zone have their clocks synchronized (e.g., by using the Network Time Protocol or their GPS clocks). Therefore, if the motion parameters of a specific data collector and a VFN (such as the speed and direction) are known, we can determine the period of time during which the two vehicles remain connected. Assume the data collector w and the VFN v are within the transmission range $\mathcal{R}$ of the other. At the beginning of decision epoch j, let (x_w, y_w) be the GPS coordinates of the data collector w and (x_v, y_v) be that of the VFN v. Additionally, let s_w and s_v be the speeds and θ_w and θ_v

$(0 < \theta_v, \theta_w < 2\pi)$ be the moving directions of w and v, respectively. Then, during decision epoch j, the amount of time in seconds that the data collector and VFN stay connected, $\mathbb{D}_{wv}$, is calculated by:

$$\mathbb{D}_{wv} = \frac{-(ab+cd) + \sqrt{(a^2+c^2)\mathcal{R}^2 - (ad-bc)^2}}{a^2+b^2}, \quad (1)$$

where

$$a = s_w \cos\theta_w - s_v \cos\theta_v,$$
$$b = x_w - x_v,$$
$$c = s_w \sin\theta_w - s_v \sin\theta_v,$$
$$d = y_w - y_v.$$

Note that when $s_w = s_v$ and $\theta_w = \theta_v$, $\mathbb{D}_{wv}$ becomes ∞.

3) Effective Coverage Duration: We use r to denote the effective range of the dash cameras and η to denote the field-of-view (FoV, represented as an angle) of the camera lens. Then, as illustrated in Fig.2, the tuple $\{(x_w, y_w), r, \eta, \theta_w\}$ defines the *effective coverage* in meters of the dash camera installed on data collector w. To simplify the notation, we use $\mathring{t} = (x_t, y_t)$ and $\mathring{w} = (x_w, y_w)$ to denote the location of the target of interest t and the data collector w, respectively. Therefore, $\overrightarrow{\mathring{w}\mathring{t}}$ denotes the view direction from the data collector to the target of interested.

The target of interest t is said to reside in the FoV of the dash camera installed on data collector w when two criteria are satisfied. First, the distance between t and w should not exceed the effective range of the dash camera:

$$|\overrightarrow{\mathring{w}\mathring{t}}| \le r. \quad (2)$$

Second, the angle between $\overrightarrow{\mathring{w}\mathring{t}}$ and the vehicle driving direction should be smaller than the FoV of the camera lens:

$$\angle(\overrightarrow{\mathring{w}\mathring{t}}, \overrightarrow{\theta_w}) \le \eta, \quad (3)$$

where $\overrightarrow{\theta_w}$ is the unit vector in the direction θ_w.

We use $\mathcal{T}^j = \{1, ..., T\}$ to denote the set of targets of interest at decision epoch j. After knowing the location, speed and direction of a specific data collector and the location of a target of interest, we can estimate the *effective travel distance* of the data collector toward the target of interest. Within the effective travel distance, the target of interest should always be included in the sight of the data collector. According to the *sine rule*, the effective travel distance g_{wt} of the data collector w on interested target t can be calculated as:

$$g_{wt} = \frac{|\overrightarrow{\mathring{t}\mathring{w}}| \cdot \sin(\eta - \angle(\overrightarrow{\mathring{t}\mathring{w}}, \overrightarrow{\theta_w}))}{\sin\theta_w}. \quad (4)$$

To simplify the system model, we suppose that data collectors will not change their driving direction and speed during a specific decision epoch. Thus, the effective coverage duration of data collector w for target of interest t can be calculated as:

$$\mathbb{D}_{wt} = \frac{g_{wt}}{s_w}. \quad (5)$$

Notably, in a real-world environment, more than one target of interest may simultaneously reside in the FoV of a dash camera installed on a data collector. In this case, we estimate the effective coverage duration of the data collector based on the location of the target of interest with the shortest distance and neglect the others.

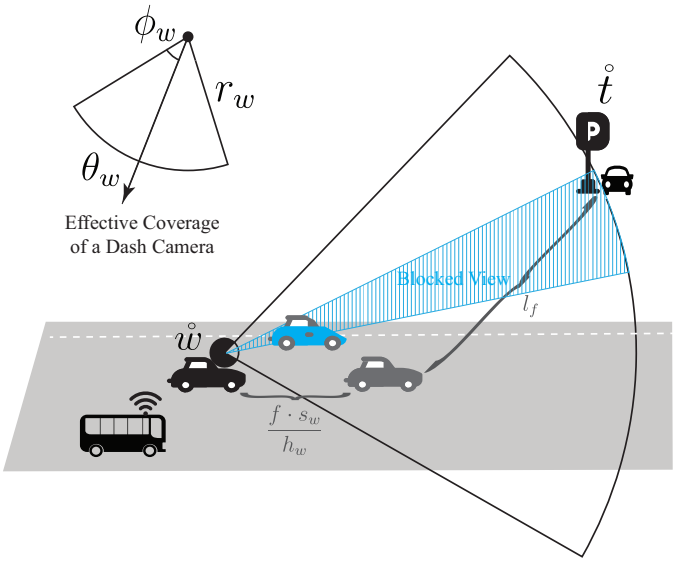


Fig. 2: Effective Coverage Duration.

B. Problem Formulation

1) QoI: In FlexSensing, a data collector keeps sending the captured visual data to the selected VFN at a required frame rate since the connectivity with the VFN is established. The transmission stops when the connection drops or when the target of interest is out of the effective coverage of the data collector's dash camera. We use $\mathcal{F}_w$ to denote the set of effective frames transferred by the data collector w in a certain data collection task. After the effective coverage duration $\mathbb{D}_{wt}$, the connected duration $\mathbb{D}_{wv}$ and the selected frame rate h_w are determined, the number of transferred frames $|\mathcal{F}_w|$ from a data collector w can be calculated as:

$$|\mathcal{F}_w| = \min\{\mathbb{D}_{wv}, \mathbb{D}_{wt}\} \cdot h_w. \quad (6)$$

We use $f \in \{1, ..., |\mathcal{F}_w|\}$ to denote the sequence of frames from the first frame captured at the beginning of the decision epoch to the last one captured before the end of the effective coverage duration or the end of the connected duration. The data collector keeps collecting data when it is approaching the target of interest defined in the data collection task. The index f increases as the data collector approaches the target, which also means that the number of pixels covering the target in the image frames increases with f.

To calculate the number of pixels covering the target of interest in a specific frame, we need to estimate the frame *depth*, which is the distance between the data collector and the target of interest at the time when the frame is captured. As illustrated in Fig.2, once the data collector speed s_w and the selected frame rate h_w are known, the depth l_f of the f^{th} frame can be calculated as:

$$l_f = \sqrt{|\overrightarrow{\mathring{t}\mathring{w}}|^2 + (\frac{f \cdot s_w}{h_w})^2 - 2|\overrightarrow{\mathring{t}\mathring{w}}|\frac{f \cdot s_w}{h_w}\cos(\angle(\overrightarrow{\mathring{t}\mathring{w}}, \overrightarrow{\theta_w}))}. \quad (7)$$

In addition to the captured distance, we can also obtain the f^{th} frame's *orientation* θ_f, which is the angle between the data collector and the target of interest at the time when the frame is captured:

$$\theta_f = \sin^{-1}\left(\frac{|\overrightarrow{\mathring{t}\mathring{w}}|\sin(\angle(\overrightarrow{\mathring{t}\mathring{w}}, \overrightarrow{\theta_w}))}{l_f}\right). \quad (8)$$

We assume that there is only one type of target of interest with a similar size (e.g., parking lots) and use $P(l_f, \theta_f)$ to denote the number of pixels on a specific target of interest in the f^{th} frame. Furthermore, in a real-world road scenario, the FoV of the dash camera installed on a data collector may be blocked by the vehicles in front, which reduces the number of pixels covering the target of interest. As shown in Fig.2, a part of the FoV of the black vehicle is blocked by the blue vehicle. We define the *block probability* $B(\bar{v}) \in [0, 1]$, where $\bar{v}$ is the average distance between vehicles surrounding the target of interest. Then, the number of pixels on the target of interest $\mathbb{P}_w$ in all the transferred frames from data collector w can be calculated as:

$$\mathbb{P}_w = (1 - B(\bar{v})) \sum_{f=1}^{|\mathcal{F}_w|} P(l_f, \theta_f). \quad (9)$$

We explore $B(\bar{v})$ and $P(l_f, \theta_f)$ by profiling the characteristics of a dash camera in Section VI.

2) Processing Latency: The workload of VFNs increases with the density of customer vehicles as well as the frame rate of data collection. With a higher workload, more processes can be created and run simultaneously on a selected VFN, resulting in longer processing latency. We use C_v^j to denote the number of customer vehicles surrounding the VFN v at decision epoch j and assume that the VFN will create one process for every P_N customer vehicles. Furthermore, we use $N_v(h_w)$ to denote the number of processes created for processing the visual data collected from data collector w. Therefore, the total number of processes P_v^j running on VFN v is:

$$P_v^j = C_v^j / P_N + \sum_{w \in \mathcal{W}^j} N_v(h_w) \cdot \mathbb{1}_{\{c_w = v\}}, \quad (10)$$

where $\mathbb{1}_{\{c_w=v\}} = 1$ if the data collector w is assigned to the VFN v and 0 otherwise.

Finally, we use $\zeta_v(P_v^j)$ to denote the processing latency when P_v^j processes are simultaneously executed on VFN v. We explore $N_v(h_w)$ and $\zeta_v(P_v^j)$ by profiling a visual object recognition application in Section VI.

3) Optimization Objectives: In FlexSensing, we aim at maximizing the QoI (defined as the number of pixels covering targets of interest in the case of object detection) while reducing the processing latency on VFNs. We use $\mathcal{Y}^j$ to denote the set of joint control commands for all data collectors at decision epoch j. Thus, we formulate the optimization problem as follows:

$$\xi 1: \max_{(c_w, h_w) \in \mathcal{Y}^j} \phi \sum_{w \in \mathcal{W}^j} \mathbb{P}_w - (1 - \phi) \frac{\sum_{v \in \mathcal{V}^j} \zeta_v(P_v^j)}{|\mathcal{V}^j|}, [1] \quad (11)$$

[1]By decoupling the maximization of QoI and the minimization of processing latency, E.q. 11 becomes a bi-objective optimization problem and then a Pareto-optimal solution is desirable.

where $\phi \in [0, 1]$ is a scalar weight.

Due to the massive number of participants, the time-varying traffic situation and the diversified geographical information of vehicles, it is difficult to solve the optimization problem defined in Eq. 11 by model-driven approaches. Instead, we prefer a data-driven solution, in which an optimized task allocation strategy can be learned from experience.

V. Constraint-Aware MDP and DQN Approach

To maximize the QoI and reduce the processing latency on VFNs, we adopt the constraint-aware Markov decision process (MDP) framework proposed in [33]. In this section, we first introduce the formulation of the constraint-aware MDP, including its basic elements and the recursion function. Then, we illustrate the methodology for seeking the optimal task allocation strategy via the DQN.

A. MDP Formulation

1) State Space: We consider the practical online scenario where data collectors and VFNs enter and leave a specific service zone dynamically. Within a specific decision epoch j, we assume that the numbers of vehicles involved (i.e., $|\mathcal{W}^j|$ and $|\mathcal{V}^j|$) remain the same.

Recall the system assumptions in Section IV. The state of an individual data collector w includes the geographical information (e.g., location, speed and direction) and the configuration of its dash camera (e.g., effective range and FoV) and is denoted by a vector $\boldsymbol{b_w} = \{(x_w, y_w), \theta_w, \vartheta_w, r, \phi\}$. Similarly, the state of a VFN v is presented by $\boldsymbol{b_v} = \{(x_v, y_v), \theta_v, \vartheta_v, C_v^j\}$, where the geographical information and the number of neighboring customer vehicles are included. For a specific target of interest t, we use $\boldsymbol{b_t} = \{(x_t, y_t)\}$ to denote its location state. Then, the state information at decision epoch j is included in the set $\{\{\boldsymbol{b_w}\}, \{\boldsymbol{b_v}\}, \{\boldsymbol{b_t}\}\}$, where $w \in \mathcal{W}^j, v \in \mathcal{V}^j$ and $t \in \mathcal{T}^j$, respectively.

If we allow the system to assign $|\mathcal{W}^j|$ data collectors simultaneously, the action space becomes $|\mathcal{W}^j|^{(|\mathcal{V}^j| * |\mathcal{H}|)}$ and increases exponentially with the increase of the data collectors. It is extremely difficult for the model to converge during training when $|\mathcal{W}^j|$ and $|\mathcal{V}^j|$ are large. Hence, we disjunct the state information set $\{\{\boldsymbol{b_w}\}, \{\boldsymbol{b_v}\}, \{\boldsymbol{b_t}\}\}$ by splitting the data collector set $\{\boldsymbol{b_w}\}$. Specifically, we divide the state at a decision epoch into substates $s_w = \{\boldsymbol{b_w}, \{\boldsymbol{b_v}\}, \{\boldsymbol{b_t}\}\}$, where one and only one data collector's information is included in a substate. Therefore, the state at a decision epoch j can be presented as $\mathcal{X}^j = \{s_w\}$, and the set $\mathbb{X} = \{\mathcal{X}^j\}, j \in [1, +\infty)$ is defined as the state space, which includes all possible states.

2) Action Space: As mentioned in Section IV, the zone head makes a joint control command (c_w, h_w) for the data collector w at decision epoch j, in which $c_w = v$ $(v \in \mathcal{V}^j)$ indicates that the data collector w transfers its visual data to VFN v under a frame rate h_w. With a higher frame rate, the number of pixels covering the targets of interest in the collected images will increase, and the computational cost will grow. By training the substates one by one, we can obtain a set of joint control commands $\mathcal{Y}^j = \{(c_w, h_w)\}$ for all the data collectors at the decision epoch j. We use the set

$\mathbb{Y} = \{\mathcal{Y}^j\}, j \in [1, +\infty)$ to denote the action space, which includes all possible joint control commands.

3) Task Allocation Strategy: A task allocation strategy Φ is defined as a mapping: $\Phi : \mathcal{X}^j \Rightarrow \mathcal{Y}^j$. More specifically, the zone head determines a joint control command $\Phi(\mathcal{X}^j) = (\Phi_{(c_w)}(\mathcal{X}^j), \Phi_{(h_w)}(\mathcal{X}^j)) = (c_w, h_w) \in \mathcal{Y}^j$ for a data collector w according to Φ after observing the network state $\mathcal{X}^j$ at the beginning of each decision epoch j, where $\Phi = (\Phi_{(c_w)}, \Phi_{(h_w)})$, with $\Phi_{(c_w)}$ and $\Phi_{(h_w)}$ being, respectively, the crowdsourcing task assignment and the frame rate selection.

4) System Reward: When applying joint actions $\mathcal{Y}^j$ to the state $\mathcal{X}^j$, an immediate system reward $u(\mathcal{X}^j, \mathcal{Y}^j)$ at epoch j is received to quantify the task allocation experience for the system. Considering the optimization problem $\xi 1$ in Section IV, we have two objectives, i.e., maximizing the QoI (in terms of the total number of pixels covering the targets of interest during object detection) while reducing the average processing latency.

We define the immediate system reward of a substate as follows:

$$u(s_w, (c_w, h_w)) = \begin{cases} A^{Wr}, & \text{if } c_w > |\mathcal{V}^j|, \\ \phi \mathbb{P}_w - (1-\phi) \sum\limits_{v \in \mathcal{V}^j} \zeta_v(P_v^j), & \text{otherwise.} \end{cases} \tag{12}$$

Notably, during the training of MDP models, the dimensions of the input states need to be fixed. However, due to the mobility of VFNs, the dimensions of the states vary over time. To address this problem, we set a constant number $B = \max\{\mathcal{V}^j\}, j \in \mathbb{Z}_+$ as the number of VFNs in the input at each training step. Once a VFN is invalid ($c_w > |\mathcal{V}^j|$), a punishment A^{Wr} is added, and this decision epoch is terminated.

The reward of each decision epoch can be defined as the summation of the rewards of the involved substates:

$$u(\mathcal{X}^j, \mathcal{Y}^j) = \sum_{w \in \mathcal{W}^j} (s_w, (c_w, h_w)). \tag{13}$$

Taking the expectation with respect to the per-epoch immediate reward $\{u(\mathcal{X}^j, \Phi(\mathcal{X}^j)) : j \in \mathbb{Z}_+\}$ over the sequence of states $\{\mathcal{X}^j : j \in \mathbb{Z}_+\}$, the expected long-term utility on an initial state $\mathcal{X}^1$ can be expressed as:

$$\mathcal{U}(\mathcal{X}, \Phi) = E_\Phi \left[(1-\gamma) \cdot \sum_{j=1}^{\infty} \gamma^{j-1} \cdot u(\mathcal{X}^j, \Phi(\mathcal{X}^j)) | \mathcal{X}^1 = \mathcal{X} \right], \tag{14}$$

where $\mathcal{X} \in \mathbb{X}$, $\gamma \in [0, 1)$ is the discount factor, and γ^{j-1} denotes the discount factor to the $(j-1)^{th}$ power.

B. DQN Approach

According to Bellman's optimality equation [34], the expected long-term reward can be obtained by solving the following equation:

$$\forall \mathcal{X} \in \mathbb{X}, \mathcal{U}(\mathcal{X}) = \max_{\mathcal{Y}} \Big\{ (1-\gamma) \cdot u(\mathcal{X}, \mathcal{Y}) + \gamma \cdot \sum_{\mathcal{X}'} Pr\{\mathcal{X}' | \mathcal{X}, \mathcal{Y}\} \cdot \mathcal{U}(\mathcal{X}') \Big\}, \tag{15}$$

where $\gamma \in [0, 1)$ is the discount factor, $u(\mathcal{X}, \mathcal{Y})$ is the immediate reward when a set of joint control actions $\mathcal{Y} \in \mathbb{Y}$ are performed under the state $\mathcal{X}$, and $\mathcal{X}' \in \mathbb{X}$ is the subsequent state of $\mathcal{X}$.

The traditional solutions for Eq. 15 are based on the value iteration or the policy iteration, which need the complete knowledge of the state transition probability of the data collectors, VFNs and targets of interest. However, due to the complexity of vehicular application scenarios (e.g., high mobility and large scale), it is impractical to obtain the complete knowledge of the involved vehicles and targets of interest. Therefore, we adopt an offline algorithm called *Q-learning*[2], which can learn a task allocation strategy purely based on experience without any predefined rules. Different from Bellman equation, Q-learning extends the definition of state-value function to state-action pairs, defining a value for each state-action pair, which is called the action-value function.

We define the right-hand side of Eq. 15 as:

$$Q(\mathcal{X}, \mathcal{Y}) = (1-\gamma) \cdot u(\mathcal{X}, \mathcal{Y}) + \gamma \cdot \sum_{\mathcal{X}'} Pr\{\mathcal{X}' | \mathcal{X}, \mathcal{Y}\} \cdot \mathcal{U}(\mathcal{X}'). \tag{16}$$

Accordingly, Eq. 15 can be converted to Eq. 17:

$$\mathcal{U}(\mathcal{X}) = \max_{\mathcal{Y}} Q(\mathcal{X}, \mathcal{Y}). \tag{17}$$

By substituting Eq. 17 into Eq. 16, we obtain

$$Q(\mathcal{X}, \mathcal{Y}) = (1-\gamma) \cdot u(\mathcal{X}, \mathcal{Y}) + \gamma \cdot \sum_{\mathcal{X}'} Pr\{\mathcal{X}' | \mathcal{X}, \mathcal{Y}\} \cdot \max_{\mathcal{Y}'} Q(\mathcal{X}', \mathcal{Y}'), \tag{18}$$

where $\mathcal{Y}' \in \mathbb{Y}$ is a joint control action performed under the state $\mathcal{X}'$.

With the Q-learning approach, the zone head learns task allocation strategies $Q(\mathcal{X}, \mathcal{Y})$ in a recursive way according to Eq. 19 [36].

$$Q^{j+1}(\mathcal{X}, \mathcal{Y}) = Q^j(\mathcal{X}, \mathcal{Y}) + \beta^j \cdot \Big((1-\gamma) \cdot u(\mathcal{X}, \mathcal{Y}) + \gamma \cdot \max_{\mathcal{Y}'} Q^j(\mathcal{X}', \mathcal{Y}') - Q^j(\mathcal{X}, \mathcal{Y}) \Big), \tag{19}$$

In decision epoch j, the agent in the zone head first observes its current state $\mathcal{X}$, and selects and performs a set of joint actions $\mathcal{Y}$. Then, the agent would observe the subsequent state $\mathcal{X}'$ and receives an immediately utility $u(\mathcal{X}, \mathcal{Y})$. The value of $Q^{j+1}(\mathcal{X}, \mathcal{Y})$ would be adjusted using $\beta^j \in [0, 1)$, which is a time-varying learning rate.

The standard Q-learning rule suffers from poor scalability and is not applicable to high-dimensional scenarios with extremely large state or action spaces. In our scenario, given continuous values of the geographical information of vehicles (e.g., location, driving speed and direction) and VFN computing resource usage, there is an infinite number of $\{\mathcal{X}, \mathcal{Y}\}$ pairs; thus, we cannot store them in tabular form and solve

[2]Note that, policy iteration based reinforcement approaches, such as SARSA, also can be used. However, Q-learning has no delayed reward, which tends to facilitate an earlier convergence than SARSA [35].

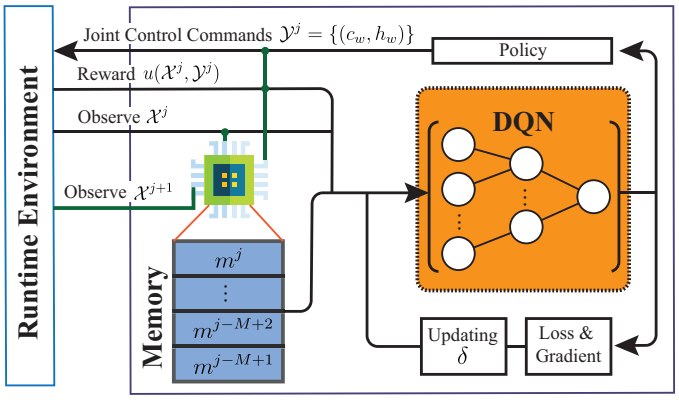


Fig. 3: DQN Approach.

the problem using the standard Q-learning method. To address the scalability problem, we apply an advanced algorithm, the DQN, to approach the optimal task allocation strategy, where the adjustable parameters of the neural network are referred to as the task allocation strategy parameters δ. Specifically, the Q-function, expressed as in Eq. 16, is approximated by $Q(\mathcal{X}, \mathcal{Y}) \approx Q(\mathcal{X}, \mathcal{Y}; \delta)$, where $(\mathcal{X}, \mathcal{Y}) \in \mathbb{X} \times \mathbb{Y}$ and δ denotes a vector of parameters associated with the DQN. Instead of finding the optimal Q-function, the DQN parameters can be learned iteratively.

To store the experience state, we assume that each zone head is equipped with a replay memory of finite size M. As illustrated in Fig.3, we use $m^j = (\mathcal{X}^j, \mathcal{Y}^j, u(\mathcal{X}^j, \mathcal{Y}^j), \mathcal{X}^{j+1})$ to denote the transition between two adjacent decision epochs j and $j+1$, and the set $\mathcal{M}^j = \{m^{j-M+1}, ..., m^j\}$ denotes the experience pool at decision epoch j during the learning process. A *policy DQN* $Q(\mathcal{X}, \mathcal{Y}; \delta^j)$ and a *target DQN* $Q(\mathcal{X}, \mathcal{Y}; \hat{\delta^j})$ are maintained in the zone head, where δ^j represents the parameters at decision epoch j and $\hat{\delta^j}$ represents the parameters at the previous epochs before j.

At each decision epoch, according to the experience replay technique [37], the zone head randomly samples a batch $\widehat{\mathcal{M}^j} \subseteq \mathcal{M}^j$ from the experience pool to train the DQN. In the training process, a loss function is defined in Eq. 20. By derivating the loss function with respect to δ^j, the parameters δ^j in the policy DQN are updated toward minimizing the mean-squared measure of equation error as shown in Eq. 20 at decision epoch j by replacing $Q^j(\mathcal{X}, \mathcal{Y})$ and its corresponding target $(1-\gamma) \cdot u(\mathcal{X}, \mathcal{Y}) + \gamma \cdot \max_{\mathcal{Y}'} Q^j(\mathcal{X}', \mathcal{Y}')$ with $Q(\mathcal{X}, \mathcal{Y}; \delta^j)$ and $(1-\gamma) \cdot u(\mathcal{X}, \mathcal{Y}) + \gamma \cdot Q(\mathcal{X}', \arg\max_{\mathcal{Y}'} Q(\mathcal{X}', \mathcal{Y}'; \delta^j); \hat{\delta^j})$, respectively. The agent in the zone head regularly resets the target DQN parameters with the updated parameters in the policy DQN.

C. Time Complexity Analysis

In training phase, the time complexity is governed by both of the forward and backward propagation of the DQN. Suppose the DQN has L layers and there are d_i neurons in the i-th layer. We denote the number of multiplications and the number of activation function performed in the forward propagation as N_{mul} and N_ρ, respectively. According to [38], $N_{mul} = d_0 d_1 + \sum_{i=2}^{L} (d_i d_{i-1} d_{i-2})$ and $N_\rho = \sum_{i=1}^{L} d_i$, and the time spent in the forward propagation is given by

$$\begin{aligned} O(T_{fwd}) &= O(N_{mul} + N_\rho) \\ &= O\Big(d_0 d_1 + \sum_{i=2}^{L} (d_i d_{i-1} d_{i-2}) + \sum_{i=1}^{L} d_i\Big). \end{aligned} \tag{21}$$

In the backward propagation, the gradient operation at the i-th layer requires $O(1)$. We denote the time complexity of delta error in the backward propagation as $O(T_\delta)$. According to [38], $O(T_\delta) = O(L(L-1) + \sum_{i=2}^{L} (d_i d_{i-1} d_{i-2}))$. As a result, the time complexity for updating all weights in backward propagation for one iteration can be obtained as

$$\begin{aligned} O(T_{bwd}) &= O(T_\delta + N_{mul}) \\ &= O\Big(L(L-1) + d_0 d_1 + \sum_{i=2}^{L} (d_i d_{i-1} d_{i-2})\Big). \end{aligned} \tag{22}$$

Suppose that we operate τ iterations for training. Then, the time complexities in the training phase can be expressed as

$$O(T_{train}) = O\Big(\tau (T_{fwd} + T_{bwd})\Big). \tag{23}$$

VI. Evaluation

In this section, we evaluate the effectiveness of FlexSensing. We first investigate the impact of the data collection configuration on the QoI in terms of the number of pixels covering a specific target of interest. Moreover, we profile the resource usage of a visual object recognition application and measure the processing latency with different numbers of processes running in parallel. Second, we configure a microscopic traffic simulator, SUMO, to generate routes of the vehicles in question, based on the profiles of real-world vehicular traffic data [39]. Finally, we compare the performance of the DQN approach with that of several reference approaches [40].

A. Application Profiling

1) Characteristics of Visual Crowdsensing: The camera view of one vehicle may be blocked by another vehicle in front. The proportion of the camera view blocked depends on the distance between the vehicles, given a fixed FoV. To quantify the effect, we ran the following experiment with two vehicles in an outdoor open space. The parked vehicle was considered an obstacle blocking the view of the other vehicle, which was equipped with a Garmin 55 dash camera with a 122 degree FoV and drove away from the parked vehicle at a constant speed. Images were captured periodically, with 2.5 meters between every two sequential measurement points. We used YOLO [31] for real-time object detection and calculated the number of pixels covering the detected vehicle in front in each image. As shown in Fig.4c, we obtained the following approximate formulation by fitting the recorded data into a polynomial regression model:

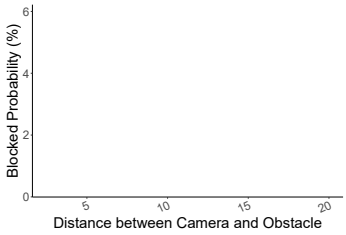
$$B(\bar{v}) = 0.101 - 2.104 \times 10^{-2} \bar{v} + 1.524 \times 10^{-3} \bar{v}^2 - 3.607 \times 10^{-5} \bar{v}^3, \tag{24}$$

where $B(\bar{v})$ is the block probability, indicating the percentage of the view that is blocked by the vehicle in front, and $\bar{v}$ is

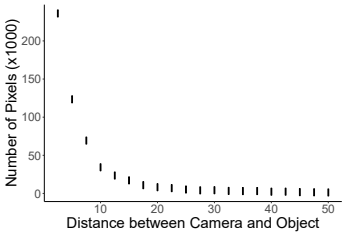
$$L(\delta^j) = E_{(\mathcal{X},\mathcal{Y},u(\mathcal{X},\mathcal{Y}),\mathcal{X}')\in\widehat{\mathcal{M}^j}}\left[\left((1-\gamma)\cdot u(\mathcal{X},\mathcal{Y}) + \gamma\cdot Q(\mathcal{X}', \arg\max_{\mathcal{Y}} Q(\mathcal{X}',\mathcal{Y}';\delta^j);\hat{\delta}^j) - Q(\mathcal{X},\mathcal{Y};\delta^k)\right)^2\right] \quad (20)$$

(a) Occupation of The Vehicle in Front in Photos Taken from Different Distances Away

(b) Occupation of The Target of Interest in Photos Taken from Different Distances Away

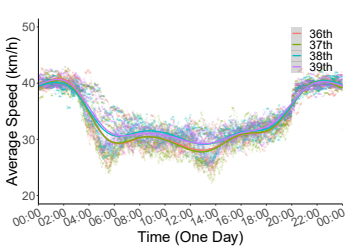


(c) Blocked Probability

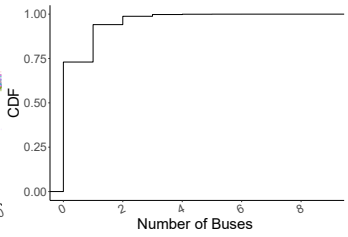


(d) Number of Pixels per Image

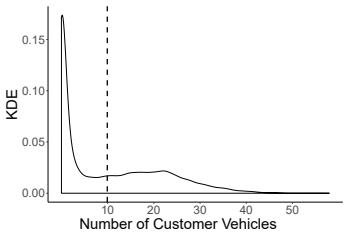
Fig. 4: Vehicle-based Visual Crowdsourcing Application Profiles.



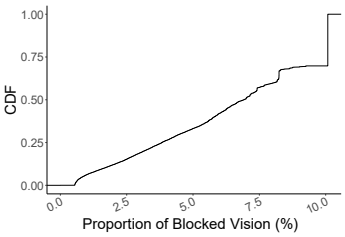
(a) Average Speed over Time



(b) Number of Buses within the Communication Range



(c) Number of Customer Vehicles within the Communication Range of a VFN



(d) Proportion of the Camera View Blocked by the Vehicle in Front

Fig. 5: Traffic Flow Variation.

TABLE II: Processing Latency vs. Number of Processes

Number of Processes	1	2	3	4	5
Processing Latency (ms)	39	45	50	61	74

the average distance between vehicles surrounding the target of interest.

We ran another experiment to figure out the relationship between the QoI (i.e., the number of pixels covering a specific target of interest) and the frame depth (i.e., the distance between the data collector and the target of interest at the time when the frame is captured). To simplify the system model, we consider that the angle between the data collector and the target of interest is equal to the data collector's moving orientation. As shown in Fig.4b, we turn the parked vehicle into a target of interest and the other vehicle into a data collector. We saved the image taken by the dash camera every 5 m while the data collector was driving away from the target of interest. As shown in Fig.4d, we obtain the following the approximate relationship between the number of pixels on the target of interest and the frame depth:

$$P(l_f, 0) = 200395 - 19754 * l_f + 617 * l_f^2 - 6 * l_f^3, \quad (25)$$

where l_f is the depth of the f^{th} frame.

2) Visual Object Recognition Application Profiling: Assume that the data collected from vehicles can be processed in parallel on VFNs. The processing latency depends on the number of processes executed simultaneously, since the underlying resources, such as the CPU, GPU and memory, are shared between processes. We took YOLO as an example to measure the processing latency and ran it on a GPU server (NVIDIA Corporation GV100, 16 GB of memory) for image processing. The test dataset includes 1000 720p dash camera images.

Because of the memory limitations, up to five YOLO processes can be simultaneously executed on the GPU server. Table II lists the average processing latency per process when different numbers of processes were executed simultaneously. In FlexSensing, we assume that the processing capacity of one process is 10 fps and that the input from each customer vehicle is 1 fps (P_N = 10). Furthermore, we consider three optional frame rates for visual crowdsourcing: 10 fps, 20 fps and 30 fps. In addition, based on the measurements listed in Table II, we assume that up to five processes can be simultaneously executed on a VFN.

B. Simulation Setup

1) Datasets: Concerning the vehicle mobility, we simulated the routes of different vehicles, including data collectors, customer vehicles and VFNs, using the following real-world traffic datasets.

(i) Traffic flow dataset. We selected the city of Helsinki as the test area and collected the traffic message channel

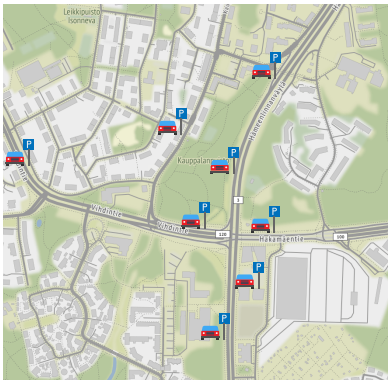
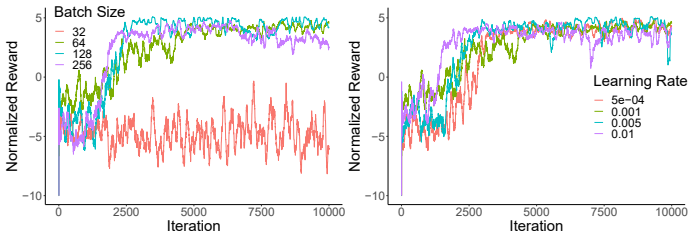


Fig. 6: Locations of Targets of Interest.



(a) Batch Size vs. Convergence (b) Learning Rate vs. Convergence

Fig. 7: Batch Size and Learning Rate.

(TMC) data using HERE APIs [41] from September 3 to October 6, 2018. The TMC data contain the traffic flow information, including the driving directions, average speeds, and identities of the road segments traversed by the vehicles.

The TMC data generally describe the congestion level of city traffic, where the detailed routes of individual vehicles cannot be obtained. To address this problem, we utilize the microscopic road traffic simulator SUMO [39] to generate routes, including the time stamp, speed, direction and location, of each data collector and customer vehicle. Notably, the number of these vehicles is estimated based on the average speed of traffic flows following the approach proposed in [42], and the moving patterns of these vehicles are generated following the method used in [43].

(ii) Bus trajectories dataset. Through the open high-rate positioning (HFP) API provided by *HSL* [44], we collect the trajectories of buses running in the same region and during the same period as that of the traffic flow dataset. By exploring this dataset, we can identify the location, driving direction and speed of any bus operated in the Helsinki region. In the simulation, these buses are turned into VFNs.

We selected an area of 1 km^2 in Helsinki, with the latitude ranging between 24°53′16″ and 24°54′25″ and the longitude ranging between 60°12′16″ and 60°12′49″. As shown in Fig.5a, the temporal variation of the average speed exhibits a repetitive pattern on a weekly basis. Furthermore, as illustrated in Fig.6, we place 8 targets of interest, which are located along the roads in that area.

Fig.5b shows the cumulative distribution function (CDF) of the number of VFNs surrounding the data collectors in a typical week (36th week, September 3 ∼ 9, 2018). We can see that most of the data collectors (more than 70%) are not located within the communication range of any VFN. In addition, less than 10% of the data collectors are reachable by more than 3 VFNs at any given time.

Fig.5c shows the kernel density estimation (KDE) of the number of customer vehicles located within the communication range of at least one VFN on a typical working day (September 3, 2018). On average, 10 customer vehicles are covered by the communication range of a VFN wherever one exists.

Based on Eq. 4d, we calculate the blocked view based on the distance between vehicles. As shown in Fig.5d, more than 50% of the data collectors have more than 7.5% of their camera views blocked by vehicles in front of them.

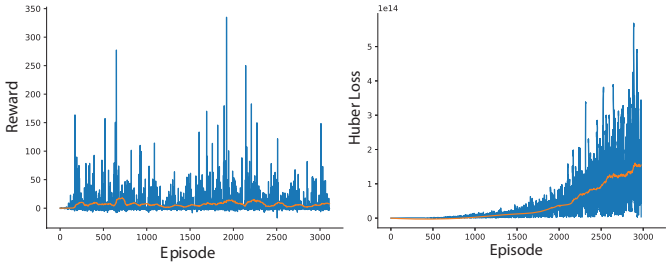
2) Learning Episodes and Training Setup: According to the application profiles shown in Fig.4d, we set the dash camera effective coverage r to 50 m. Furthermore, according to the measurements in [45], we set the effective transmission range between a VFN and data collector $\mathcal{R}$ to 50 m.

We implement the DQN learning network using PyTorch. The default neuron numbers in the hidden layer are 4096 plus 2048 [46]. We consider one minute as the length of a decision epoch, and trajectories of the participants involved in this decision epoch are put into the model and trained as an episode. We set the default parameters of the rewards as $A^{Wr} = -10$.

To determine the value of batch size and learning rate, we set $\phi = 1$ and train the DQN network with 10000 iterations. As shown in Fig.7, we can see DQN would converge after 3000 iterations when the batch size is larger than 64, whereas DQN would not converge when the batch size is 32. This is because updating the weights based on a small batch will be more noisy. However, sometimes the noise can help the DQN jerk out of local optima. On the other hand, using a smaller value of learning rate can help the stability of learning processes in DQNs, but selecting an adequate learning rate is heuristic and application dependent. Fig.7b demonstrates that using learning rate 0.001 can achieve the most stable performance compared with other values. Therefore, we set the batch size and learning rate in the actor-critic training phase as $|\mathcal{M}^j| = 64$ and $\gamma = 0.999$, respectively.

C. Evaluation Results and Analysis

1) System Reward and Huber Loss: For comparison, we train the policy DQN using the traffic data captured from September 3 to September 9, 2018 (i.e., the 36th week of 2018) and train the target DQN with three weeks of traffic data from September 3 to September 23, 2018 (i.e., 36th∼38th week, 2018). Fig.8a shows the system reward variation of each episode during the training of the target DQN. From the figure, we can see that the system reward shares a similar variation pattern with the vehicle average speed, as shown in Fig.5a. This occurs because the higher the density of the traffic is, the



(a) Episode vs. System Reward (b) Episode vs. Huber Loss

Fig. 8: System Reward and Huber Loss.

higher the number of data collectors and the larger the amount of information regarding the targets of interest available in the collected data. Fig.8b shows the computed Huber loss variation during the DQN training. The Huber loss acts similar to the mean squared error when the error is small but similar to the mean absolute error when the error is large. From the figure, we can see that the computed Huber loss remains stable after approximately 2500 episodes.

2) Policy Network vs. Target Network: According to Fig.5a, we select two time periods to represent the times of high and low vehicular traffic density:

* *Time Period I*: 11 : 00 ~ 13 : 05, September 3, 2018,
* *Time Period II*: 19 : 00 ~ 21 : 05, September 3, 2018.

We then evaluate the performance of the policy and target DQN with the traffic data collected from these two periods.

Fig.9 compares the distribution of the selected frame rates for each data collector when the workload of the VFN (i.e., the number of the processes available for serving customer vehicles) varies. Fig.9a shows the selected frame rate distribution in Time Period I. We can see that 18.5% of the data collectors select an invalid VFN ($c_w > |\mathcal{V}^j|$) according to the task allocation strategy learned from the policy DQN, while all the data collectors select a valid VFN according to that learned from the target DQN. This outcome occurs because the policy DQN has not converged, unlike the target DQN, due to the limited number of training episodes.

From Fig.9a, we can see that the task allocation strategy learned from the target DQN prefers to select a higher frame rate when the VFN workload is lower, and vice versa. For example, 42% of the data collectors select 30 fps for data transmission when the VFN workload is 1, while only 28% of the data collectors select the highest frame rate when the VFN workload is 4. This result occurs because the density of traffic is higher in Time Period I and because the processing latency will increase with the amount of data to be processed.

Fig.9b illustrates the distribution of the selected frame rates in Time Period II. Compared with the policy DQN, the task allocation strategy learned from the target DQN tends to select a higher frame rate because more resources become available for collecting and processing visual data to obtain more information about the targets of interest when the density of traffic is lower.

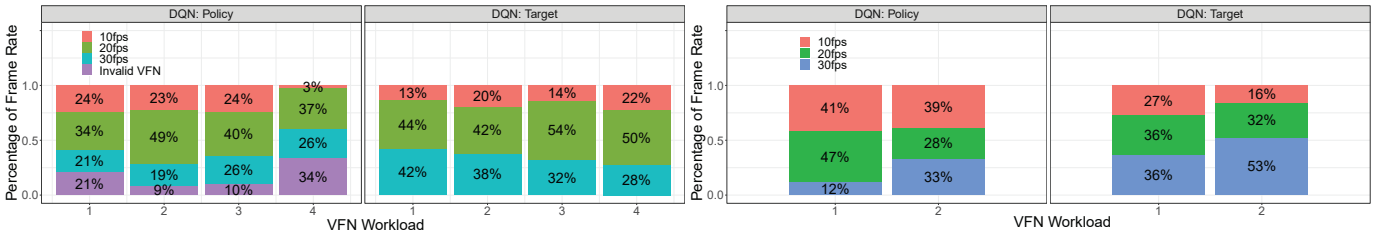
3) Processing Latency vs. Number of Pixels: We define 3 variants of task allocation strategies that are learned from the target DQN based on the value of the scalar weight ϕ mentioned in Section IV.

* *DQN-T*: Processing Latency Sensitive with $\phi = 0.1$.
* *DQN-Q*: QoI Sensitive with $\phi = 0.9$.
* *DQN-B*: Balanced with $\phi = 0.5$.

For comparison, two other task allocation strategies, i.e., MUEECA and Adaptive, are implemented. The MUEECA task allocation strategy has been investigated in the recent publication [8]. According to the MUEECA task allocation strategy, the data collector selects a VFN with the maximum QoI and meanwhile satisfies the latency constraints. If the processing latency exceeds the constraint, the data collector would choose the minimum QoI for task offloading. Here, we set the latency constraint as $50ms$. The Adaptive task allocation strategy uses a probability distribution method to scale the frame rate selection linearly with the VFN workload [7]. In our case, the data collector transmits data at a frame rate of 30 fps when the workload of the VFN (i.e., the number of the processes available to serve customer vehicles) is 1 and at a frame rate of 10 fps when the workload of the VFN is 4. Otherwise, the data collector transfers data at a frame rate of 20 fps.

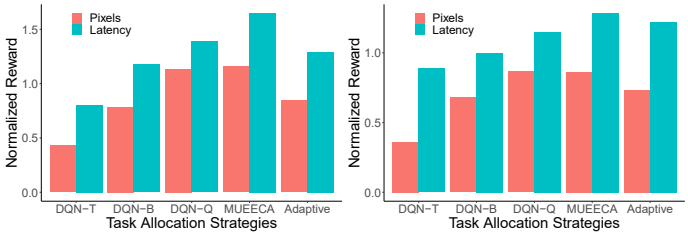
To unify the metrics, we normalize the number of pixels covering the targets of interest. As shown in Fig.4d, we select 200000 as the maximum number of collected pixels (i.e., when the distance between the data collector and the target of interest is 0). By dividing the achieved QoI by the maximum number of collected pixels, we can obtain the normalized QoI of each episode. Furthermore, to unify the magnitude scale, we divide the processing latency by 50 ms. Fig.10a illustrates the average normalized reward and the average scaled processing latency for the different task allocation strategies in use during Time Period I. The system achieves the highest QoI with MUEECA. Compared with MUEECA, DQN-Q only reduces the QoI by 2% whereas shortens the processing latency by 18%. Moreover, DQN-Q increases the QoI by 34% compared with the Adaptive task allocation strategy. In DQN-B, the average processing latency is shortened by 10%, but the QoI is also reduced by 4% compared with the Adaptive task allocation strategy. In DQN-T, the visual data are transferred with greater emphasis on reducing the processing latency. In this case, the average processing latency on VFNs is shortened by 51% and 37% compared with the MUEECA and Adaptive task allocation strategies, respectively. In addition, the QoI decreases by 62% and 49%. These results indicate that the task allocation strategies learned from the target DQN can reduce the processing latency or increase the QoI depending on the specific demand of the application.

As shown in Fig.10b, in Time Period II, DQN-Q increases the QoI by 18% and DQN-T shortens the average processing latency by 27% compared with the Adaptive task allocation strategy. The differences are smaller than in Time Period I. This outcome indicates that the task allocation strategies learned from the target DQN are more effective in reducing the processing latency and increasing the QoI when the vehicular traffic is denser. Moreover, the DQN-Q task allocation strategy improves both the average processing latency and the QoI.



(a) Time Period I (b) Time Period II

Fig. 9: Distribution of Selected Frame Rates.



(a) Time Period I (b) Time Period II

Fig. 10: Comparison of QoI and Processing Latency between Task Allocation Strategies.

TABLE III: Data collection frequency requirements

Applications	Data collection rate
Driving assistance	High
Local map generation	High
Parking navigation	Moderate
Construction detection	Moderate
Improvement recommendation	Low

Specifically, it reduces the average processing latency by 1% and 6%, and increases the QoI by 11%and 18% compared with the MUEECA and Adaptive task allocation strategies, respectively.

In summary, DQN-based task allocation strategies moderate the frame rate for visual data transmission, taking into account the variation in fog node workload. Specifically, DQN-T reduces the average processing latency by up to 51%, and DQN-Q increases the QoI by up to 34%. FlexSensing could reduce the data processing latency or increase the QoI according to the specific demands of individual vehicle-based visual crowdsourcing applications.

VII. Limitation and Future Work

In this section, we discuss the limitation of our work and present the future plan. In FlexSensing, the data collectors do not have the ability to detect objectives. Therefore, we only take into account the targets of interest with fixed locations. The locations of targets would be known by the zone head in advance, and data collectors can obtain the location information either from an off-line map or from messages sent by the zone head.

We focus the scheduling of task allocation in a single service zone and place only one DQN agent in the zone head. However, in the real-world mobile edge computing (MEC) scenarios, there may exist several edge servers in a service zone. Therefore, performing learning process in a distributed manner (e.g., federated and parallelized learning) over multiple edge servers connected via wireless networks would help reduce the training time and the need for centralized parameter server.

Moreover, we design task allocation schemes based on the assumption that the learning problem can be cast as Markov Decision Process (MDP). However, in reality, the task allocation problem resists being treated as a MDP because it is impractical to obtain the system state in real time. In the future, we would like to deprive the learner of perfect information about the state of the environment and replace the MDP with the mathematic models that could address task allocation under uncertainty, such as partially observable Markov decision processes (POMDPs) and hidden Markov models (HMMs).

From the communication perspectives, the data processors in our system are assumed to be moving buses which commute on specified trajectories. As an alternative solution particularly in situations when there are no regular buses available in the vicinity of data collectors, the moving drones can act as relay nodes which establish the communication between the data collectors and the data processors [47]. In order to ensure the autonomous and continuous task offloading service, the system design for drones should take into account the limitation of high energy consumption.

In our simulation, we consider one minute as the length of a decision epoch. However, the length of decision epoch of different crowdsourcing applications can be determined on various update cycles, such as daily, hourly, or continuous basis. We have listed the requirements of the data collection rate of different vehicular crowdsourcing applications in Table III. For example, applications that relate to driving assistance (e.g., real-time situational awareness) involve data-intensive and latency-sensitive computing tasks, and have an extremely strict requirement for the validity period of crowdsourced video. On the other hand, applications that relate to city traffic regulation, such as the change in speed limitation on freeways, allow longer update cycles (e.g., hourly or daily). Furthermore, the requirements of QoI may vary from application to application. For example, crowdsourcing applications that only consider targets of interest with unified appearance, such as

traffic signs, have a lower QoI requirement than that involving targets of interest with various appearances.

VIII. Conclusions

In this paper, we propose FlexSensing, a QoI- and processing-latency-aware task allocation scheme for vehicle-based visual crowdsourcing. It aims at increasing the QoI while reducing the processing latency of crowdsourced visual data, taking into account the variation in the VFN workload and vehicle mobility. We address the problem of seeking the optimal task allocation strategy via the formulation of an MDP and solve it through the DQN. Compared with previous works, our solution reduces the average processing latency by up to 51% and increases the QoI by up to 34%.

References

[1] J. Ni, A. Zhang, X. Lin, and X. S. Shen, "Security, Privacy, and Fairness in Fog-Based Vehicular Crowdsensing," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 146–152, 2017.

[2] B. Guo, Q. Han, H. Chen, L. Shangguan, Z. Zhou, and Z. Yu, "The Emergence of Visual Crowdsensing: Challenges and Opportunities," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2526–2543, 2017, conference Name: IEEE Communications Surveys Tutorials.

[3] C. H. Liu, J. Fan, P. Hui, J. Crowcroft, and G. Ding, "QoI-Aware Energy-Efficient Participatory Crowdsourcing," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3742–3753, Oct. 2013, conference Name: IEEE Sensors Journal.

[4] M. Noreikis, Y. Xiao, J. Hu, and Y. Chen, "SnapTask: Towards Efficient Visual Crowdsourcing for Indoor Mapping," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2018, pp. 578–588, iSSN: 2575-8411.

[5] D. Pal, V. Vanijja, C. Arpnikanondt, X. Zhang, and B. Papasratorn, "A Quantitative Approach for Evaluating the Quality of Experience of Smart-Wearables From the Quality of Data and Quality of Information: An End User Perspective," *IEEE Access*, vol. 7, pp. 64 266–64 278, 2019, conference Name: IEEE Access.

[6] Y. Xiao and C. Zhu, "Vehicular fog computing: Vision and challenges," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Mar. 2017, pp. 6–9.

[7] C. Huang, Y. P. Fallah, R. Sengupta, and H. Krishnan, "Adaptive intervehicle communication control for cooperative safety systems," *IEEE Network*, vol. 24, no. 1, pp. 6–13, Jan. 2010.

[8] H. Liu, L. Cao, T. Pei, Q. Deng, and J. Zhu, "A fast algorithm for energy-saving offloading with reliability and latency requirements in multi-access edge computing," *IEEE Access*, vol. 8, pp. 151–161, 2019.

[9] C. Zhu, G. Pastor, Y. Xiao, and A. Ylajaaski, "Vehicular Fog Computing for Video Crowdsourcing: Applications, Feasibility, and Challenges," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 58–63, Oct. 2018.

[10] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi, "Mobeyes: smart mobs for urban monitoring with a vehicular sensor network," *IEEE Wireless Communications*, vol. 13, no. 5, pp. 52–57, Oct. 2006.

[11] V. Coric and M. Gruteser, "Crowdsensing Maps of On-street Parking Spaces," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, May 2013, pp. 115–122.

[12] F. Shi, D. Wu, D. I. Arkhipov, Q. Liu, A. C. Regan, and J. A. McCann, "ParkCrowd: Reliable Crowdsensing for Aggregation and Dissemination of Parking Space Information," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2018.

[13] G. Grassi, K. Jamieson, P. Bahl, and G. Pau, "Parkmaster: An In-vehicle, Edge-based Video Analytics Service for Detecting Open Parking Spaces in Urban Environments," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 16:1–16:14, event-place: San Jose, California. [Online]. Available: http://doi.acm.org/10.1145/3132211.3134452

[14] C. Zhu, A. Mehrabi, Y. Xiao, and Y. Wen, "CrowdParking: Crowdsourcing Based Parking Navigation in Autonomous Driving Era," in *2019 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, Sep. 2019, pp. 1401–1405.

[15] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan, "AVR: Augmented Vehicular Reality," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '18. New York, NY, USA: ACM, 2018, pp. 81–95. [Online]. Available: http://doi.acm.org/10.1145/3210240.3210319

[16] K. Hara, V. Le, and J. Froehlich, "Combining crowdsourcing and google street view to identify street-level accessibility problems," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*. Paris, France: ACM Press, 2013, p. 631. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2470654.2470744

[17] S. Sattar, S. Li, and M. Chapman, "Road Surface Monitoring Using Smartphone Sensors: A Review," *Sensors*, vol. 18, no. 11, p. 3845, Nov. 2018, number: 11 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/18/11/3845

[18] K. Yagi, "A Measuring Method of Road Surface Longitudinal Profile from Sprung Acceleration, and Verification with Road Profiler," *Journal of Japan Society of Civil Engineers, Ser. E1 (Pavement Engineering)*, vol. 69, no. 3, pp. I_1–I_7, 2013.

[19] R. Omer and L. Fu, "An automatic image recognition system for winter road surface condition classification," in *13th International IEEE Conference on Intelligent Transportation Systems*, Sep. 2010, pp. 1375–1379, iSSN: 2153-0017.

[20] Y. Qian, E. J. Almazan, and J. H. Elder, "Evaluating features and classifiers for road weather condition analysis," in *2016 IEEE International Conference on Image Processing (ICIP)*, Sep. 2016, pp. 4403–4407, iSSN: 2381-8549.

[21] J. Wang, C. Jiang, K. Zhang, T. Q. Quek, Y. Ren, and L. Hanzo, "Vehicular sensing networks in a smart city: Principles, technologies and applications," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 122–132, 2017.

[22] Y. Wang, C. Xu, Z. Zhou, H. Pervaiz, and S. Mumtaz, "Contract-Based Resource Allocation for Low-Latency Vehicular Fog Computing," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2018, pp. 812–816, iSSN: 2166-9589.

[23] K. Zhang, J. Wang, C. Jiang, T. Q. Quek, and Y. Ren, "Content aided clustering and cluster head selection algorithms in vehicular networks," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2017, pp. 1–6.

[24] G. Qiao, S. Leng, K. Zhang, and Y. He, "Collaborative Task Offloading in Vehicular Edge Multi-Access Networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 48–54, Aug. 2018, conference Name: IEEE Communications Magazine.

[25] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-Based Task Offloading for Vehicular Cloud Computing Systems," *arXiv:1804.00785 [cs, math]*, Apr. 2018, arXiv: 1804.00785. [Online]. Available: http://arxiv.org/abs/1804.00785

[26] Z. Ning, J. Huang, and X. Wang, "Vehicular Fog Computing: Enabling Real-Time Traffic Management for Smart Cities," *IEEE Wireless Communications*, vol. 26, no. 1, pp. 87–93, Feb. 2019, conference Name: IEEE Wireless Communications.

[27] X. Huang, R. Yu, J. Kang, and Y. Zhang, "Distributed Reputation Management for Secure and Efficient Vehicular Edge Computing and Networks," *IEEE Access*, vol. 5, pp. 25 408–25 420, 2017, conference Name: IEEE Access.

[28] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, "Fog Following Me: Latency and Quality Balanced Task Allocation in Vehicular Fog Computing," in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Jun. 2018, pp. 1–9.

[29] C. Zhu, J. Tao, G. Pastor, Y. Xiao, Y. Ji, Q. Zhou, Y. Li, and A. Ylä-Jääski, "Folo: Latency and Quality Optimized Task Allocation in Vehicular Fog Computing," *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[30] F. Richter, A. J. Fehske, and G. P. Fettweis, "Energy Efficiency Aspects of Base Station Deployment Strategies for Cellular Networks," in *2009 IEEE 70th Vehicular Technology Conference Fall*, Sep. 2009, pp. 1–5, iSSN: 1090-3038.

[31] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arXiv:1612.08242 [cs]*, Dec. 2016, arXiv: 1612.08242. [Online]. Available: http://arxiv.org/abs/1612.08242

[32] K. Alexandris, N. Nikaein, R. Knopp, and C. Bonnet, "Analyzing X2 handover in LTE/LTE-A," in *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2016, pp. 1–7.

[33] P. Geibel and F. Wysotzki, "Risk-Sensitive Reinforcement Learning Applied to Control under Constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, Jul. 2005, arXiv: 1109.2147. [Online]. Available: http://arxiv.org/abs/1109.2147

[34] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34–37, Jul. 1966, publisher: American Association for the Advancement of Science Section: Articles. [Online]. Available: https://science.sciencemag.org/content/153/3731/34

[35] R. S. Sutton, A. G. Barto *et al.*, "Introduction to reinforcement learning. vol. 135," 1998.

[36] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[37] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, no. 2-3, pp. 311–365, 1992.

[38] Y.-H. Chiang, T.-W. Chiang, T. Zhang, and Y. Ji, "Deep dual learning-based cotask processing in multi-access edge computing systems," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9383–9398, 2020.

[39] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent Development and Applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, Dec. 2012.

[40] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[41] "Here API." [Online]. Available: https://developer.here.com/documentation/traffic/ [Accessed Sep. 01, 2018]

[42] R. Sen, A. Cross, A. Vashistha, V. N. Padmanabhan, E. Cutrell, and W. Thies, "Accurate Speed and Density Measurement for Road Traffic in India," in *Proceedings of the 3rd ACM Symposium on Computing for Development*, ser. ACM DEV '13. New York, NY, USA: ACM, 2013, pp. 14:1–14:10. [Online]. Available: http://doi.acm.org/10.1145/2442882.2442901

[43] L. Codeca, R. Frank, and T. Engel, "Luxembourg SUMO Traffic (LuST) Scenario: 24 hours of mobility for vehicular networking research," in *2015 IEEE Vehicular Networking Conference (VNC)*, Dec. 2015, pp. 1–8.

[44] "HSL API." [Online]. Available: https://digitransit.fi/en/developers/apis/4-realtime-api/vehicle-positions/ [Accessed Sep. 01, 2018]

[45] C. Zhu, Y. Chiang, A. Mehrabi, Y. Xiao, A. Yla-Jaaski, and Y. Ji, "Chameleon: Latency and Resolution Aware Task Offloading for Visual-based Assisted Driving," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2019.

[46] F. Wang, C. Zhang, F. wang, J. Liu, Y. Zhu, H. Pang, and L. Sun, "Intelligent Edge-Assisted Crowdcast with Deep Reinforcement Learning for Personalized QoE," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. Paris, France: IEEE, Apr. 2019, pp. 910–918. [Online]. Available: https://ieeexplore.ieee.org/document/8737456/

[47] J. Wang, C. Jiang, Z. Han, Y. Ren, R. G. Maunder, and L. Hanzo, "Taking drones to the next level: Cooperative distributed unmanned-aerial-vehicular networks for small and mini drones," *IEEE Vehicular Technology Magazine*, vol. 12, no. 3, pp. 73–82, 2017.