

LiveMap: Real-Time Dynamic Map in Automotive Edge Computing

Qiang Liu, Tao Han, Jiang (Linda) Xie

The University of North Carolina at Charlotte
{qliu12, tao.han, linda.xie}@uncc.edu

BaekGyu Kim

Toyota Motor North America R&D InfoTech Labs
baekgyu.kim@toyota.com

Abstract—Autonomous driving needs various line-of-sight sensors to perceive surroundings that could be impaired under diverse environment uncertainties such as visual occlusion and extreme weather. To improve driving safety, we explore to wirelessly share perception information among connected vehicles within automotive edge computing networks. Sharing massive perception data in real time, however, is challenging under dynamic networking conditions and varying computation workloads. In this paper, we propose *LiveMap*, a real-time dynamic map, that detects, matches, and tracks objects on the road with crowdsourcing data from connected vehicles in sub-second. We develop the data plane of *LiveMap* that efficiently processes individual vehicle data with object detection, projection, feature extraction, object matching, and effectively integrates objects from multiple vehicles with object combination. We design the control plane of *LiveMap* that allows adaptive offloading of vehicle computations, and develop an intelligent vehicle scheduling and offloading algorithm to reduce the offloading latency of vehicles based on deep reinforcement learning (DRL) techniques. We implement *LiveMap* on a small-scale testbed and develop a large-scale network simulator. We evaluate the performance of *LiveMap* with both experiments and simulations, and the results show *LiveMap* reduces 34.1% average latency than the baseline solution.

Index Terms—Dynamic Map, CrowdSourcing, Computation Offloading, Automotive Edge Computing

I. INTRODUCTION

As the development of contemporary artificial intelligence and parallel computing hardware, autonomous driving and advanced driving assistance system (ADAS) are becoming a reality more than ever [1]. Vehicles rely on sensors such as stereo camera, LiDAR and radar, to perceive surrounding environments, and depend on advanced onboard processors to process the massive volume perception data in real time. By understanding the environment, e.g., high-accurate localization, lane detection and pedestrian recognition, alongside the accurate high-definition (HD) map, intelligent control algorithms can correctly react to most of the environmental situations by controlling the vehicle, e.g., lane changing and passing vehicles.

It is, however, very difficult to realize high reliable and safe driving under extremely diverse environmental uncertainties such as extreme weather, lighting conditions, visual occlusion, and sensor failures [2]. For example, vehicle sensors are primarily range-limited and line-of-sight, which means they are incapable of perceiving information from occluded areas [3]. Consider a single-lane road, a car is following a big truck that occludes the car's sensor perception, passing the truck without sufficient information about the opposite lane is absolutely unsafe. Besides, the perception range and fidelity of sensors

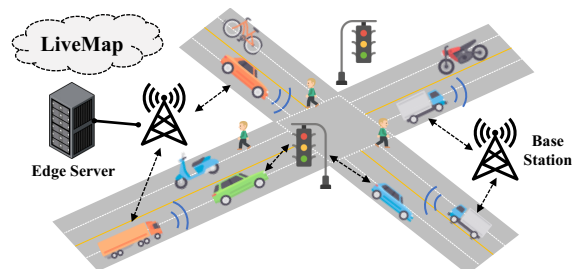


Fig. 1: An illustration of automotive edge computing.

might be further impaired by extreme weather, e.g., rain, snow, and dust. Thus, relying on sensors in a single vehicle alone may not be sufficient to fulfill high-safety driving.

Connected vehicle is the key building block of the Internet of Vehicles (IoV) [4], which connects vehicles with wireless technologies, e.g., cellular networks and dedicated short-range communications (DSRC). It allows the communication of vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-network (V2N), and could substantially improve driving safety by effectively sharing information among vehicles. For example, if the truck shares its perception data to the car, e.g., a detected vehicle on the opposite lane, the car can decide not to pass the truck even if the vehicle is unobserved by the car's sensors.

However, sharing perception data among connected vehicles in automotive edge computing networks is challenging. For example, in a dense urban scenario, allowing all vehicles to share could lead to redundant information exchanging since their sensing coverages are heavily overlapped. Besides, letting vehicles to share their raw data, e.g., point clouds or RGB-D images, requires tremendous wireless bandwidth and might result in network congestion [5]. Furthermore, edge servers need to support hundreds or thousands of vehicles simultaneously, thus its workloads vary from time to time. Optimizing the time to share for vehicles needs intelligence under the complex networking and computation in automotive edge computing networks.

In this paper, we propose *LiveMap*, a real-time dynamic map in automotive edge computing illustrated in Fig. 1, that detects, matches and tracks objects on the road in sub-second based on the crowdsourcing data from connected vehicles. We design the data plane of *LiveMap* that consists of object detection, projection, feature extraction and object matching for processing individual vehicle data efficiently, and object combination for combining objects from multiple vehicles effectively. Specifically, we reduce the detection time with neural network pruning techniques in the object detection,

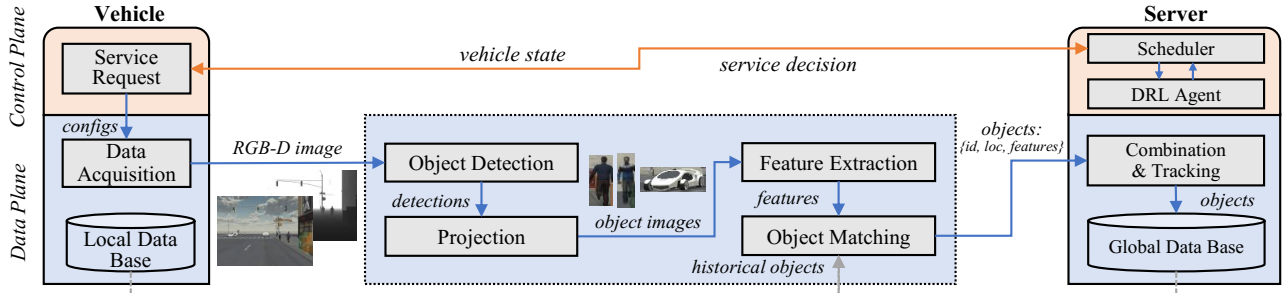


Fig. 2: The *LiveMap* system

decrease the object feature size with variational autoencoder in the feature extraction, and improve the matching accuracy with a novel location-aware distance function. To efficiently exploit available networking and computation resources, we design the control plane of *LiveMap* by allowing adaptive offloading of vehicle computations, and developing an intelligent vehicle scheduling and offloading algorithm to decrease the offloading latency of vehicles while maintaining the map coverage based on deep reinforcement learning approaches.

The contributions of this paper are summarized as follows:

- We design *LiveMap* that realizes real-time dynamic map with crowdsourcing vehicles in automotive edge computing networks.
- We develop the data plane of *LiveMap* with an efficient processing pipeline for processing individual vehicle data and an effective object combination method for combining objects from multiple vehicles.
- We design an intelligent vehicle scheduling and offloading algorithm that significantly reduces the offloading latency of vehicles in the control plane of *LiveMap*.
- We implement *LiveMap* on a small-scale testbed and develop a large-scale network simulator.
- We evaluate *LiveMap* with both experiments and simulations, and the results validate *LiveMap* substantially outperforms existing solutions.

II. *LiveMap* OVERVIEW

The control plane and data plane of *LiveMap* are shown in Fig. 2. The control plane is composed of a service request module on individual vehicle, a scheduler and a DRL agent on the edge server. When a vehicle attempts to start the offloading service, it sends a service request with current vehicle states such as computation capability and wireless quality to the scheduler. The scheduler determines if this vehicle is scheduled based on current map coverage, and the DRL agent will be engaged to optimize the adaptive offloading decision under the current vehicle and system state if the vehicle is scheduled. Once the service decision is received by the vehicle, it starts the data plane accordingly if scheduled, otherwise, it waits for a backoff time before next requesting.

The data plane consists of multiple sequential processing modules for vehicle raw data processing, i.e., data acquisition, object detection, projection, feature extraction, object matching and combination. Depending on the adaptive offloading decision of the vehicle, these modules are executed

on either vehicle or server, where the data acquisition and combination module can only be run on the vehicle and server, respectively. The data acquisition module captures RGB-D images from onboard sensors and obtains the localization of the vehicle. The object detection module detects objects in the RGB image with a state-of-the-art object detection framework, i.e., YOLOv3 [6]. These detected objects that are with 2D bounding boxes, are projected to 3D world locations with the depth image and camera-to-world matrix obtained in the data acquisition module. The feature extraction module extracts the features of detected objects from cropped images based on a novel variational autoencoder. Then, these objects are matched within either local or global database depending on where to execute the matching module, based on their features and locations. The combination module on the server integrates duplicated objects from multiple vehicles by combining the detection confidence, geo-location, and features of objects, and also predicts their mobility. Finally, the global database is updated, whose new updates, e.g., new objects or updated locations, are broadcasted to all connected vehicles to update their local databases.

III. DATA PLANE DESIGN

The design of data plane is to accomplish the data processing of vehicle raw data and to optimize the performance in terms of accuracy and processing delay.

A. Data Acquisition

The data acquisition module on the vehicle is used for acquiring the raw vehicle data, i.e., RGB-D images, from various sensors, e.g., front and rear cameras or LiDARs. In addition, it obtains the current accurate 3D world location of the vehicle based on either advanced relocalization algorithms, e.g., ORB SLAM2 [7], or high-accuracy GPS, which could achieve centimeter-level estimation errors of vehicle locations. Since we need to combine the detected objects from crowdsourcing vehicles on the edge server, the accurate estimation of vehicle locations are desired to improve the system performance.

B. Object Detection

The object detection module detects objects in the RGB image, e.g., pedestrians, cars and trucks, where detection results are composed of object names, confidences and 2D bounding boxes. Existing object detection algorithms, e.g., YOLO [6], Fast-RCNN [8], SSD [9], are primarily designed for detecting generic categories which include hundreds of

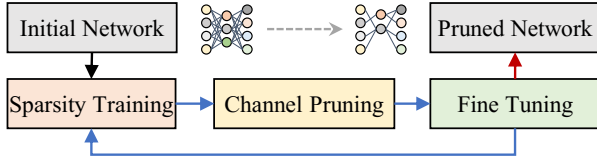


Fig. 3: The flow-chart of neural network pruning.

object classes, e.g., person, book, boat, table and kite. As a result, applying these algorithms directly to embedded platforms such as electronic control unit (ECU) in vehicles, leads to long detection time. In general, a larger neural network is required to achieve similar detection accuracy, e.g., mean average precision (mAP), for a larger number of classes.

To address this issue, we propose a slim object detector with reduced categories specifically for automotive transportation system, by using neural network pruning techniques. The neural network pruning is to reduce the neural network size by removing unnecessary neurons without dramatically sacrificing detection accuracy. As shown in Fig. 3, we adopt a similar network pruning workflow in [10], which mainly consists of sparsity training, channel pruning, fine-tuning. Specifically, the initial network is re-trained by minimizing the loss function with a weighted L1 regulation on the scaling factors in batch normalization (BN) layers during the sparsity training phase. By decreasing these BN scaling factors, insignificant convolutional channels that are with nearly zeros scaling factors can be pruned during the channel pruning phase. Then, the neural network is fine tuned during the fine-tuning phase. Such train-prune-tune sequential processes can be repeated to seek the optimal trade-off between detection accuracy and network size.

As a result, we show the performance of detection network pruning in Table I. We apply the network pruning on the YOLOv3 tiny framework, where we decrease 80 classes to 10 classes that includes person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, and traffic light. We reduce the network size by 93.7% with the cost of 0.01 mAP degradation. Meanwhile, the detection time on Nvidia Jetson Nano is decreased by 19.4% or 18.7% if using TensorRT acceleration.

C. Projection

The projection module calculates the 3D world location of detected objects in the world coordinate system based on the detection results, i.e., 2D bounding boxes of objects, depth image, and camera specifications.

As shown in Fig. 4, the objects in real world are projected onto the image plane by the camera sensor. The calculation of the world location of an object is completed by two steps, i.e., from pixel coordinates to camera coordinates, and from camera coordinates to world coordinates. Denote (u_0, v_0) as the center of an object in pixel coordinates, the focal length of the camera as f , and the image resolution as (R_W, R_H) . The 3D location of the object (X, Y, Z) in the camera coordinate system can be written as

$$\begin{aligned} X &= -(d * (v_0 - 0.5 * R_H)) / f, \\ Y &= (d * (u_0 - 0.5 * R_W)) / f, \\ Z &= d, \end{aligned} \quad (1)$$

Detection Networks	mAP@0.5 640x	Num. of parameters	time(Nano) w/o TensorRT
YOLOv3 tiny	0.534	8.69e+06	191.9/37.4 ms
Pruned Network	0.524	0.54e+06	154.7/30.4 ms

TABLE I: Pruning results of detection network on our dataset

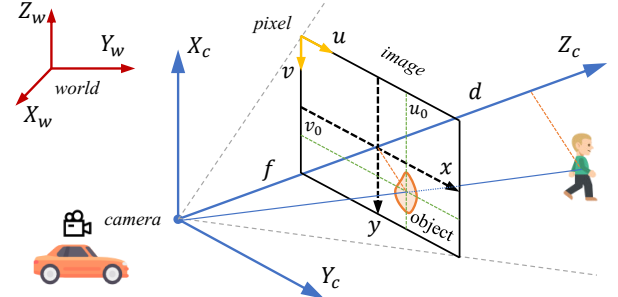


Fig. 4: The illustration of coordinate systems.

where d is the Z-axis depth of the object w.r.t. the camera. However, estimating the depth of an object is not easy since the object usually occupies an irregular 2D area in the depth image while its bounding box only gives the rectangle area. Given the bounding box of the object and depth image, we sample multiple small 5x5 squares in close proximity to the object center and calculate the average depth after removing the largest and smallest values.

Next, we convert the object location in camera coordinate to the world location (W_x, W_y, W_z) in world coordinate as

$$[W_x, W_y, W_z, 1]^T = M_{c2w} \times [X, Y, Z, 1]^T, \quad (2)$$

where M_{c2w} is the 4x4 camera-to-world conversion matrix obtained from the data acquisition module, and $[\cdot]^T$ is the operation of matrix transpose.

In addition, the projection module estimates the coverage of a vehicle by obtaining the default coverage of its cameras and calculating the visual occlusion incurred by objects. In *LiveMap*, we consider the area is occluded by an object if the object height is higher than that of the camera in the camera coordinate system, i.e., $X \geq 0$.

D. Feature Extraction

Although we obtain the world location of all detected objects by the detection and projection modules, we need to identify and match them within the database to track their mobility. The feature extraction module is used to extract features from cropped object images based on variational autoencoder framework. The conventional feature extraction algorithms, e.g., SIFT, SURF [11], and ORB [12], could generate keypoint features with similar data size as compared to that of the object image [13]. Consequently, the computation complexity of feature matching raises and the transmission delay of features offloading if applicable is increased accordingly. Besides, the detected objects are usually small, e.g., pedestrian images could be 50x50 out of 540p camera images, because they are tens of meters away from vehicles. In practice, we found that these algorithms either generate no features or trivial features from small objects, which results in low matching accuracy.

To solve this issue, we propose to use variational autoencoder [14], an unsupervised machine learning framework, to

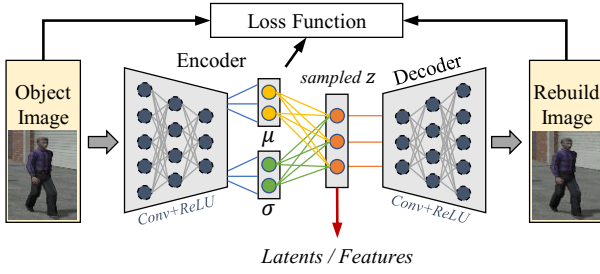


Fig. 5: The design of autoencoder as feature extraction.

extract lightweight features from object images, as shown in Fig. 5. The autoencoder primarily consists of an encoder that encodes the input image into condensed latent vectors and a decoder that rebuilds the image from the latents. Unlike conventional autoencoders, which are prone to generate irregular latent space [14], e.g., similar images may be encoded to distinct latent vectors, the variational autoencoder uses a unique neural network architecture and introduces a regularization in the loss function. Denote x, z as the object image and the sampled latent vector from the distribution $\mathcal{N}(\mu, \sigma^2)$, respectively. The training loss of variational autoencoder can be written as

$$Loss = -\mathbb{E}_{z \sim q(z|x)} [\log p(x|z)] + D_{KL} [q(z|x)|p(z)], \quad (3)$$

where $q(z|x)$ and $p(x|z)$ denote the encoder and decoder, respectively. And D_{KL} is the Kullback-Leibler divergence to evaluate the difference between two probability distributions, where $p(z) \sim \mathcal{N}(0, 1)$ is selected as a normal distribution. After the training of the variational autoencoder, the encoder is used to extract features from object images and the generated latent vectors are recognized as the features of objects.

E. Object Matching

The object matching module is designed to match the detected objects within the database based on their features and locations. Since the database in *LiveMap* could have hundreds or thousands of items such as pedestrians and cars, matching an object with all these items is compute-intensive and time-consuming. Meanwhile, matching objects merely based on the distance of features, i.e., latent vectors, could fail in a dynamic automotive environment [5].

To solve this issue, we propose a novel location-aware distance function for matching based on our estimated world locations of vehicles and objects. Specifically, we only match the database items in close proximity to the detected object, e.g., 100 meters for vehicles and 10 meters for pedestrians, which decreases the size of matching set and reduces the matching time accordingly. In *LiveMap*, we construct a mobility model for each object in the database based on its historical locations. The location of objects in the database are predicted when matching objects at the current time. Then, we introduce a novel location-aware distance function by considering not only the features distance but also the geographic distance between two objects. Denote g as the geo-location of an object in the world coordinate system, the distance between the i th and j th object is defined as

$$D_{i,j} = \min(\|z_{i,m} - z_{j,m}\|^2, \forall m \in \mathcal{M}) + w\|g_i - g_j\|^2, \quad (4)$$

object id	class	geo-location	confidence
speed	direction	update time	multi-view latents

TABLE II: Attributes of an object in the database

where w is a weighted factor, z are the latent features of objects, $\|\cdot\|^2$ is the L2-norm operation, and \mathcal{M} denotes the set of latent features associated with an item in the database. Since an object may be observed by multiple vehicles from different angles, these multi-view generated features are associated with the object in the database. Here, we use the minimum feature distance among these multi-view features to calculate the final distance between two objects.

F. Object Combination

The object combination module on the server integrates the detected objects from different vehicles and updates their information in the global database, e.g., locations and latent features. The global database is a collection of historical detected objects, where each object is represented by several attributes as shown in Table II. In *LiveMap*, we remove objects from the global database if their information is outdated, e.g., a pedestrian is deleted if not observed for more than 1 hour.

Due to the high diversity of vehicles, e.g., camera specs, view angles and lighting, an object captured and processed by different vehicles may generate slightly different results in terms of confidence, location and latents features. To effectively integrate these results together, we first consider the objects with the same object id as a unique object, and then propose a confidence weighted combination method that calculates the geo-location of the unique object as

$$g = \sum_{m \in \mathcal{M}} \frac{\mathcal{P}_m * g_m}{\sum_{m \in \mathcal{M}} \mathcal{P}_m}, \quad (5)$$

where \mathcal{P}_m is the confidence generated by the object detection module, and g_m is the geo-location estimated by the projection module. Here, we assign more weights to the results with higher confidence. Meanwhile, we consider each latent feature of a unique object is valid and associate it with the object into the database for better matching accuracy as shown in Eq. 4.

Finally, these unique objects are updated and stored in the global database. The new updates at the current time in the global database, e.g., newly detected objects, new location and latents of existing objects, are broadcasted to all connected vehicles in *LiveMap*.

IV. CONTROL PLANE DESIGN

In this section, we describe the system model, formulate the vehicle scheduling and offloading problem in *LiveMap*, and develop a novel algorithm to solve the problem efficiently.

A. System Model

We consider an automotive edge computing network with multiple vehicles denoted as \mathcal{I} , a cellular base stations (BS) and an edge computing server, where vehicles are wirelessly connected to the BS. Connected vehicles offload vehicle computations to the edge server asynchronously to build *LiveMap*. We consider vehicle computations, e.g., the data plane in

LiveMap, can be separated between computation modules¹, denoted $y_i \in \{0, 1, \dots, N\}, \forall i \in \mathcal{I}$, where N is the maximum separation scheme. For example, if the separation scheme is 1 in *LiveMap*, it means the object detection module is executed on the vehicle and the remaining modules, i.e., projection, feature extraction and object matching, are processed on the server. Meanwhile, the intermediate data generated by the object detection module, i.e., detection results and cropped object RGB-D images, are offloaded to the edge server for remaining processing.

Before the computation offloading, a connected vehicle sends an offloading request to the scheduler in the edge server with some vehicle information, e.g., wireless quality and computation capacity. Denote $x_i \in \{0, 1\}, \forall i \in \mathcal{I}$ as the binary schedule indicator of the i th vehicle, where $x_i = 1$ means the current request of the vehicle is scheduled, otherwise, the request is not scheduled. Denote $C_i^{(t)}$ as the geographic coverage area of the i th vehicle at the t time slot. Denote \mathcal{X}, \mathcal{Y} as the set of vehicle scheduling and offloading decision for all vehicles, respectively. Define the latency $L_i^{(t)}$ of the i th vehicle at the t time slot as the time between the vehicle gets the raw data by the data acquisition module and the vehicle receives the broadcasted database updates from the edge server.

B. Problem Formulation

On maintaining *LiveMap*, our objective is to provide information about the environment to all connected vehicles as fast as possible. Due to the high mobility of vehicles, the outdated information is less meaningful, e.g., a recorded location of a truck 10 seconds ago does not help on making controlling decisions in a highway scenario. Meanwhile, *LiveMap* should maintain sufficient coverage areas by scheduling more crowdsourcing vehicles where each vehicle covers a certain area on its current location. Here, we define the overall map coverage at the current time as $\bigcup_{i \in \mathcal{I}} C_i^{(t)}$.

Therefore, we formulate the vehicle scheduling and offloading problem as

$$\min_{\{\mathcal{X}, \mathcal{Y}\}} \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} L_i^{(t)} \quad (6)$$

$$s.t. \quad \bigcup_{i \in \mathcal{I}, x_i \neq 0} C_i^{(t)} \geq \beta \bigcup_{i \in \mathcal{I}} C_i^{(t)}, \forall t \in \mathcal{T}, \quad (7)$$

$$x_i^{(t)} \in \{0, 1\}, \forall i \in \mathcal{I}, t \in \mathcal{T}, \quad (8)$$

$$y_i^{(t)} \in \{0, 1, \dots, N\}, \forall i \in \mathcal{I}, t \in \mathcal{T}, \quad (9)$$

where \mathcal{T} is a given time period such as 1 hour, constraints in Eq. 7 guarantee the minimum requirement of overall map coverage, and $\beta \in [0, 1]$ is a factor.

The key difficulties in solving the above problem are highlighted. First, due to the heterogeneity of vehicles in terms of computation capability and varying wireless quality, alongside the complicated of networking and computation in *LiveMap*, the accurate modeling of vehicle latencies are impractical to be obtained in real systems. Second, with the asynchronous

offloading of crowdsourcing vehicles, their wireless transmissions and server computations are probably overlapped in time. As a result, the vehicle scheduling and offloading in *LiveMap* exhibits *Markovian* property on serving these connected vehicles, which further complicates the problem.

C. Algorithm Design

To effectively solve the problem, we develop a novel algorithm based on deep reinforcement learning. The DRL techniques have shown promising improvement in network management and control [16], [17] in terms of system performance, however, it is challenging to apply DRL in solving the aforementioned problem directly. On one hand, the number of connected vehicles in *LiveMap* is varying from time to time because the high-speed vehicles might come and leave the coverage of the BS. Most DRL solutions are designed to solve problems with fixed action space, and thus they are unable to handle the dynamic vehicle scheduling in *LiveMap*. On the other hand, existing DRL solutions are inefficient to optimize problems with multiple constraints², i.e., the requirement of map coverage in Eq. 7.

We address the problem by optimizing the vehicle scheduling and offloading decision in different time scales. This is based on the observation that the offloading of vehicles run in sub-second time scale, such as vehicle latencies are hundreds of milliseconds, whereas the scheduling of vehicles can operate at second time scale. Therefore, we design a two-layer vehicle scheduling and offloading decision (HEAD) algorithm (see Alg. 1) in *LiveMap*. In the upper layer, we schedule the minimum number of vehicles while maintaining the requirement of map coverage. Here, minimizing the number of scheduled vehicles corresponds to decreasing the number of offloading vehicles that share the common networking and computation resources in *LiveMap*. In the lower layer, we optimize the offloading decision for every single incoming vehicle with DRL techniques, where the action space becomes fixed.

1) *Vehicle Scheduling*: We build a complete graph (V, E) where V is the set of vertices that correspond to all vehicles, and E is the set of edges between vertices. Then, we define the overlapping ratio between the coverage of i th and j th vehicle as

$$o_{i,j} = \frac{C_i \cap C_j}{C_i \cup C_j}, \quad (10)$$

and assign $o_{i,j}$ to the edge value between the i th and j th vehicle, where $o_{i,j} = o_{j,i}$. Denote the average overlapping ratio of the i th vehicle as

$$O_i = \frac{1}{|\mathcal{I}|} \sum_{j \in \mathcal{I}, j \neq i} o_{i,j}. \quad (11)$$

Then, we greedily prune the graph (V, E) by removing the i th vehicle if it has the largest average overlapping ratio, i.e., $i = \arg \max_{k \in \mathcal{I}} O_k$. The basic idea behind this pruning is that we

¹The discrete separation model can be easily extended for different systems, such as partial neural network offloading in AR/VR system [15].

²Although there are some works [18], [19] target to solve constrained reinforcement learning problems, they are unable to guarantee these constraints are met at any time slots.

continuously remove a vehicle with the minimum decrease in the overall map coverage. The pruning processes stop until we reach the required map coverage by evaluating Eq. 7.

2) *Offloading Decision*: To determine the offloading decision of a vehicle, we resort to the deep reinforcement learning, e.g., deep Q network [20], that is capable of handling the complex offloading in *LiveMap*. Consider a generic reinforcement learning setting where an agent interacts with an environment in discrete decision epochs. At each decision epoch t , the agent observes a state \mathbf{s}_t , takes an action \mathbf{a}_t , i.e., offloading decision, based on its policy π_θ that parameterized by neural networks with parameters θ . Then, the agent receives a reward $r(\mathbf{s}_t, \mathbf{a}_t)$, and the environment transits to the next state \mathbf{s}_{t+1} according to the action taken by the agent. The objective is to seek a optimal policy π_θ^* that maximizes the discounted cumulative reward $R_0 = \sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)$. Here, $\gamma \in [0, 1]$ is a discounting factor and the transition $\tau = (\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$.

Then, we define the state space, action space and reward.

State Space: The state space determines what information can be observed from the system by the DRL agent. The design of state space is to represent the status of *LiveMap* completely and informatively. Thus, we build the state space $\mathbf{s}_t \triangleq [\mathbf{s}_t^v, \mathbf{s}_t^s, \mathbf{s}_t^w]$, where \mathbf{s}_t^v is *vehicle status*, \mathbf{s}_t^s is *server status*, and \mathbf{s}_t^w is *system workload*. The vehicle status provides useful information about the vehicle, including wireless quality (measured by received signal strength) and computation capability (represented by the number of CPUs, CPU frequency, memory size, GPU cores and GPU frequency). The server status includes the computation capability of the edge server and the wireless bandwidth. The system workload includes the number of total connected vehicles in *LiveMap* and the number of queuing vehicles on the edge server.

Action Space: Based on the observed state space, the DRL agent decides which offloading decision is applied to the current vehicle, which is $\mathbf{a}_t \triangleq [y]$.

Reward: When applying the offloading decision \mathbf{a}_t to the vehicle under the current state space \mathbf{s}_t , the DRL agent will receive a reward from *LiveMap*, which is defined as the negative latency of this vehicle, i.e., $r(\mathbf{s}_t, \mathbf{a}_t) \triangleq -L^{(t)}$. In the real system, the reward is delayed because the latency can only be obtained after the vehicle offloading is completed. During this time interval, requests from other vehicles may arrive for offloading decisions. We resolve the issue by allowing *LiveMap* to temporally store state-action pairs and report the state-action-reward pairs once available to the DRL agent.

Policy Training: We use Deep Q network (DQN) [21] with prioritized experience replay (PER) [20] to train the policy of DRL agent in *LiveMap*. Denote the value function $Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ as the expected discounted cumulative reward if the agent starts with the state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$ at decision epoch t and then acts according to the policy π . Thus, the value function can be expressed as $Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\tau \sim \pi} [R_t | \mathbf{s}_t, \mathbf{a}_t]$, where $R_t = \sum_{k=t}^T \gamma^{(k-t)} r(\mathbf{s}_k, \mathbf{a}_k)$. Based on the Bellman equation [22], the optimal value function $Q^*(\mathbf{s}_t, \mathbf{a}_t)$ is

$$Q^*(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}_{t+1}} Q^*(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}). \quad (12)$$

Algorithm 1: The HEAD Algorithm

Input: $\beta, \theta^*, \mathbf{s}_t^s, \mathbf{s}_t^w$

Output: x, y

```

1  $\mathbf{s}_t^v, i \leftarrow \text{vehicle}, /* \text{accept vehicle} */;$ 
2  $\mathbf{s}_t \leftarrow [\mathbf{s}_t^v, \mathbf{s}_t^s, \mathbf{s}_t^w], /* \text{build state} */;$ 
3 if time to schedule then
4    $x_k \leftarrow 1, \forall k \in \mathcal{I};$ 
5   while True do
6      $k \leftarrow \arg \max_{k \in \mathcal{I}, x_k \neq 0} O_k;$ 
7      $x_k \leftarrow 0;$ 
8     if  $\bigcup_{i \in \mathcal{I}, x_i \neq 0} C_i^{(t)} \leq \beta \bigcup_{i \in \mathcal{I}} C_i^{(t)}$  then
9        $x_k \leftarrow 1;$ 
10      break;
11 if  $x_i == 1$  (scheduled) then
12    $y \leftarrow \arg \max_{\mathbf{a}_t} Q^*(\mathbf{s}_t, \mathbf{a}_t | \theta^*), /* \text{get action} */;$ 
13 else
14    $y \leftarrow -1, /* \text{not scheduled} */;$ 
15 return  $x, y;$ 
```

To obtain the optimal policy, DQN is trained by minimizing the mean-squared Bellman error (MSBE) as follow

$$Loss(\theta^Q) = \mathbb{E}_{\tau \in \mathcal{D}} \left[(h_t - Q(\mathbf{s}_t, \mathbf{a}_t | \theta^Q))^2 \right], \quad (13)$$

where θ^Q are weights of the Q-network and \mathcal{D} is a replay buffer. h_t is the target value estimated by a target network

$$h_t = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}_{t+1}} Q(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1} | \theta^{\pi'}) | \theta^{Q'}), \quad (14)$$

where $\theta^{Q'}$ are weights of the target network. The target network has the same architecture with the Q-network and its weights $\theta^{Q'}$ are slowly updated to track that of Q-network.

In the DQN, experience transitions in replay buffer are uniformly sampled, regardless of the significance of experiences. Prioritized experience replay (PER) [20] improves the efficiency of DQN sampling by prioritizing experience transitions in the replay buffer. The importance of experience transitions are measured by the absolute TD error, that is

$$p \propto |h_t - Q(\mathbf{s}_t, \mathbf{a}_t | \theta^Q)|^\alpha, \quad (15)$$

where α is a hyper-parameter.

V. SYSTEM IMPLEMENTATION

In this section, we implement *LiveMap* on a small-scale testbed and develop a large-scale network simulator for automotive edge computing networks.

A. System Prototype

We prototype *LiveMap* system on a small-scale automotive edge computing testbed as shown in Fig 6, which is composed of four JetRacers, an 802.11ac 5GHz WiFi router with 20MHz wireless bandwidth and an Intel i7 edge server with Nvidia GTX 1070 GPU and CUDA 10.1 [23]. The JetRacer is a racecar equipped with an onboard Nvidia Jetson Nano embedded GPU. The dynamic wireless channel of vehicles are emulated by randomly configuring the transmit power of both the JetRacers and the WiFi router with Linux "iw" CLI

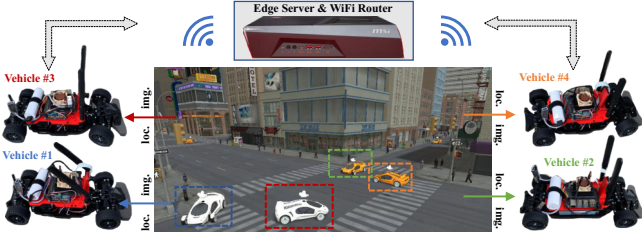


Fig. 6: The implementation of *LiveMap*.

configuration utility, i.e., from 1dBm to 22dBm. On the edge server, we develop a single queue to process all incoming offloading of vehicles. To reduce the transmission delay, we use LZ4 compression algorithm before socket communication.

We implement the DRL agent by using Python 3.7 and PyTorch 1.40. Specifically, we use a 2-layer fully-connected neural network, i.e., [256, 256], with Leaky Recifier activation function [24]. The learning rate of DQN is $5e-4$ with 512 batch size, and the discounted factor $\gamma = 0.9$. We add a decaying ϵ -greedy starts from probability 0.5 to 0.1 during the training phase for balancing the exploitation and exploration.

Due to the Markovian property of the vehicle scheduling and offloading problem in *LiveMap*, i.e., the current vehicle offloading decision could immediately affect the offloading of the next vehicle, it is ineffective to collect dataset and train the DRL agent offline, and apply the trained policy online. Hence, we online train the DRL agent by directly interacting with JetRacers in the real *LiveMap* system with 100k training steps. To accomplish the functionalities of control and data plane in *LiveMap*, e.g., object detector, autoencoder, scheduler, the DRL agent and network simulator, we finish more than 6000 line codes.

B. Traces DataSet

We build a Unity3d environment to generate traces for both testbed experiments and network simulations. The traces are composed of more than 1000 frames, where each frame includes the world location, RGB-D images, and camera-to-world matrix of vehicles, and the world location of pedestrians for calculating estimation errors. We create multiple transportation scenarios, e.g., intersection, highway and circle, based on the modern city package in Unity3d, where each scenario includes hundreds of pedestrians and vehicles. The movement of pedestrians and vehicles follow their predefined paths. Each vehicle is mounted a front RGB-D camera with focal length 50mm, field of view 54.04° , maximum sensing range 50m, which generates 741x540 images.

C. Network Simulator

We build a time-driven network simulator, which is composed of multiple onboard computation modules, a wireless transmission module, and a server computation module as depicted in Fig. 7. When the offloading decision of a vehicle is determined, a task is created on this vehicle's onboard computation module. The task describes the remaining on-board computation time, data size of uplink and broadcast transmission, and server computation time, where these data

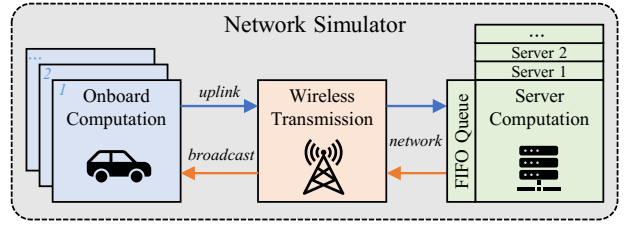


Fig. 7: The design of network simulator.

are sampled from experimental measurements (see Fig. 10). The onboard computation of a task is simulated by decreasing its remaining computation time for every simulation interval such as 1 ms. The simulation of server computation is similar, but based on a single FIFO (First-In-First-Out) queue and multiple parallel servers architecture.

The wireless transmission module is developed based on an open-source 5G simulator [25], where we use the urban micro (UMi - Street Canyon) channel model recommended in ETSI TR 138.901 [26], and consider all vehicles equally share the total bandwidth for the sake of simplicity. We use both 1MHz wireless bandwidth for uplink and downlink channels, and place the base station at the center of the Unity3d environment. Thus, the transmission of tasks are simulated by calculating their wireless data rates and decreasing their remaining uplink/downlink data sizes. A task is sent to the next simulation module only if it is completed in the last module, e.g., zero remaining computation time or transmission data size.

D. Comparison Algorithms

In the experiments, we compare *LiveMap* with the following algorithms: 1) Edge offloading (**EO**): EO offloads RGB-D images of vehicles and allows all the processing modules to be executed on the edge server. 2) Local process (**LP**): LP executes all the processing modules onboard, and sends the matched objects to the edge server. 3) Random offloading (**RO**): RO randomly selects the offloading decision for every vehicle. 4) Regression model (**RM**): We propose RM that identifies wireless data rate and the number of vehicles as two important factors when making the offloading decision. Thus, RM fits a multivariate polynomial regression model with *scikit-learn* tool based on an experimental dataset that includes different combinations of wireless data rate, number of vehicles, offloading decision and the resulted latency. To make the offloading decision, RM predicts the latency of different actions under the current network state, and takes the action with the minimum predicted latency. 5) **LiveMap-Lite**: LiveMap-Lite determines offloading decision as same as that of LiveMap, but it schedules all vehicle requests. Besides, these comparison algorithms schedule all vehicle requests.

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performance of *LiveMap* on both small-scale testbed and large-scale simulator. We aim to study: 1) what's the performance of *LiveMap* as compared to existing solutions; 2) how does *LiveMap* optimize offloading decision in complex automotive edge computing networks; 3)

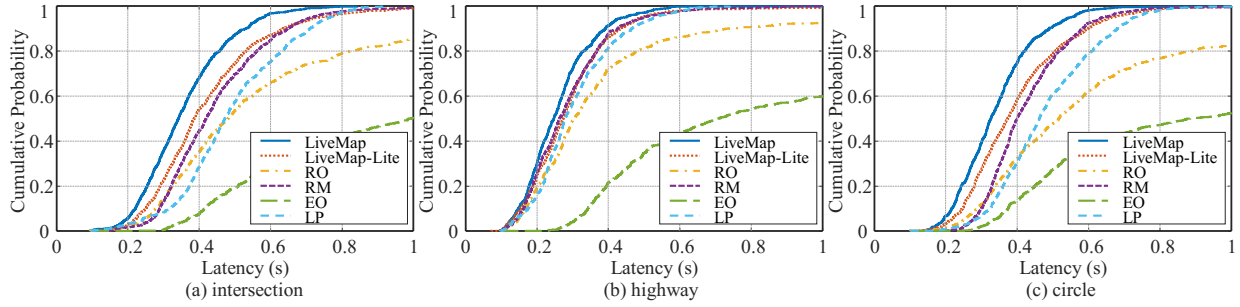


Fig. 8: The cumulative probability of latency by different algorithms under various scenarios.

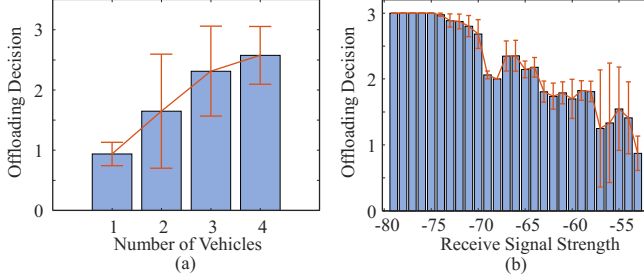


Fig. 9: The intelligent offloading decision in *LiveMap*.

how does the data plane of *LiveMap* perform over a baseline system; 4) whether *LiveMap* can effectively scale under the different number of vehicles. In the experiments, we consider the minimum requirement of overall map coverage is 80%, i.e., $\beta = 0.8$. The potential offloading decisions in *LiveMap* are $[0, 1, 2, 3, 4]$, which correspond to offloading after the data acquisition, object detection, projection, feature extraction, and object matching module, respectively.

A. Impact of Various Scenarios

Fig. 8 shows the latency performance of different algorithms under various scenarios. We observe that *LiveMap* achieves the lowest latency as compared to other algorithms. In the intersection scenario, *LiveMap* reduces 20.3% average latency as compared to RM, which validates that *LiveMap* can effectively schedule vehicles and intelligently determine offloading decisions, whereas model-based approach (RM) is ineffective in handling complex network system. Meanwhile, we see *LiveMap* outperforms *LiveMap-Lite* with 16.4% average latency reduction, which indicates that selectively scheduling vehicles could decrease the offloading traffic in the system and thus improve the latency performance. Furthermore, *LiveMap* obtains less significant performance improvement over the other algorithms under the highway scenario, as compared to that of other scenarios. This can be attributed to the less coverage overlap among vehicles in the highway scenario.

B. Intelligent Offloading Decision

We illustrate how *LiveMap* makes the offloading decision intelligently under varying system workloads and dynamic wireless qualities. In Fig. 9 (a), we show the statistics of offloading decisions when there are different number of vehicles in the system. We can see that *LiveMap* is prone to make larger offloading decisions, i.e., executing more modules onboard before offloading, when more vehicles are observed by the

DRL agent. In Fig. 9 (b), we show the correlations between offloading decisions and received signal strength of vehicles. *LiveMap* is likely to lower the offloading decision of a vehicle when better wireless quality, e.g., -60dBm, is observed. Under the worst wireless quality, e.g., -80dBm, *LiveMap* does not keep increasing the offloading decision, because the offloading data size after the feature extraction is similar to that of object matching. As a result, increasing the offloading decision at such conditions will only cost more onboard execution time without reducing significant transmission delay. These results indicate that the DRL agent can intelligently optimize the offloading decision of vehicles in *LiveMap*.

C. Design of Data Plane

We show the performance of *LiveMap* data plane as compared to a baseline system in terms of processing latency, offloading data size, and the number of successfully detected objects in Fig. 10. The baseline system is implemented with tiny YOLOv3 model [6], ORB feature extraction [12], and brutal-force feature matching algorithm, and the other modules are implemented as same as *LiveMap*. In Fig. 10 (a), We see that *LiveMap* has lower onboard execution latency than the baseline system on different processing modules. This is achieved by optimizing various modules in *LiveMap*, e.g., lower detection time with neural network pruning in object detection, lower extraction time with autoencoder for feature extraction, and lower matching time since total object features are smaller. Here, the object detection, including image preparation, detection network inference, and post-processing, consumes an average 72.4ms in *LiveMap* and 80.2ms in the baseline system. This is because the preparation, e.g., image reading and formatting, and the post-processing, e.g., non-maximal suppression (nms) in YOLOv3 and memory copying from GPU to CPU, account for considerable latency in the Jetson Nano embedded GPU platform. Besides, we observe the offloading data size of *LiveMap* is substantially smaller than that of the baseline system after feature extraction in Fig. 10 (b). This is attributed to the high compression ratio of autoencoder during the feature extraction, where the output latent features of an object image have only 25 values. Here, the large variations in projection and feature extraction, i.e., offloading decision 1 and 2, come from the varying number of objects detected from RGB images.

In Fig. 10 (c), we show the cumulative probability of latency obtained by *LiveMap*, the HEAD algorithm and LP algorithm

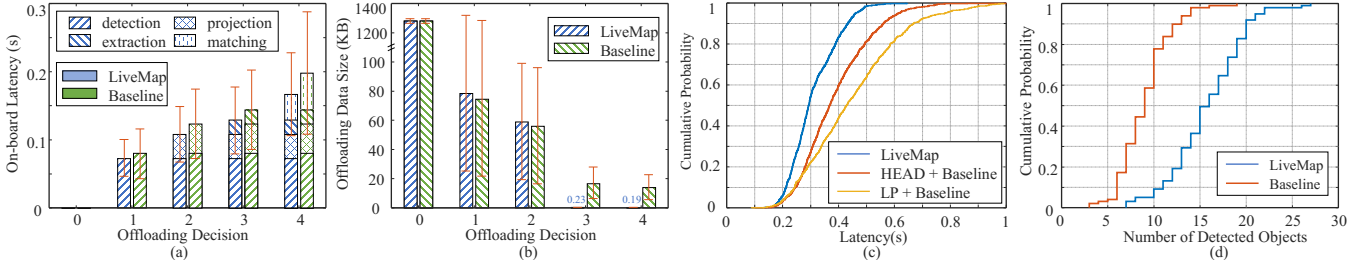


Fig. 10: The system comparison between *LiveMap* and baseline.

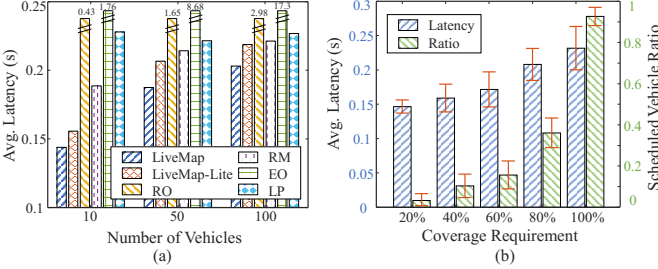


Fig. 11: The simulated results under different number of vehicles and coverage requirements.

in the baseline system, where *LiveMap* obtains 20.1% and 34.1% the average latency reduction than the HEAD and LP algorithm in the baseline system, respectively. This result validates the performance of data plane in *LiveMap* is considerably better than that of the baseline system. Furthermore, we evaluate the system performance of *LiveMap* and the baseline system in Fig. 10 (d), in terms of the number of successfully detected objects. In the experiments, we consider the object is successfully detected only if its object ID is matched correctly and the estimated geo-location error is less than 1 meter. We can see *LiveMap* substantially outperforms the baseline system with 74.9% improvement on the average number of detected objects. This is because the ORB feature extraction is ineffective in extracting meaningful features from small images. As a result, the baseline system produces more matching errors, either matching to incorrect objects in the database or identifying existing objects as newly detected. In addition, the location-aware distance function in the object matching module also contributes to the better performance of *LiveMap*. Therefore, we can conclude that the optimization of data plane in *LiveMap* significantly improves the system performances.

D. Scalability in Trace-driven Simulation

We further evaluate the performance of *LiveMap* in the large scale network simulator. In Fig. 11 (a), we show the average latency under various algorithms with the different number of vehicles. We observe that *LiveMap* outperforms other algorithms, e.g., it reduces 10.4% and 12.7% latency than LiveMap-Lite and RM respectively, when there are 50 vehicles in the system. Besides, we show the average latency and ratio of scheduled vehicles obtained by *LiveMap* under different coverage requirements in Fig. 11 (b). We can see that *LiveMap* schedules fewer vehicles with the decreasing of coverage requirement. By sacrificing more coverage performance, e.g.,

from 100% to 60%, the scheduled vehicle ratio is decreased from 100% to 15.6%, and thus the average latency is reduced from 231.7ms to 159.1ms. These results validate that *LiveMap* is scalable under the different number of vehicles.

VII. RELATED WORK

This work relates to ML-based resource management and vehicle sensing in automotive edge computing networks.

ML in Networking: Harmony [27] exploits deep learning based scheduler to optimize the average completion time of concurrent ML tasks in cloud computing clusters. EdgeS-lice [16] uses a decentralized DRL based approach to manage multiple network resources while meeting the service level agreement (SLA) of network slices. DeepCast [17] utilizes DRL techniques to learn the personalized quality of experience (QoE) of viewers and optimize edge servers assignment in crowdsourcing livecast. However, these works focus on unconstrained RL problems with fixed action space, e.g., fixed number of users, while *LiveMap* handles the varying number of vehicles under the requirement of map coverage.

Vehicle Sensing: Augmented vehicular reality (AVR) [3] extends vehicular vision with V2V visual sensor data sharing to improve driving safety, where only point clouds of dynamic objects are exchanged for efficient transmission. F-Cooper [28] fuses visual features from multiple vehicles to cooperatively perceive objects on the road, which allows the tradeoff between detection accuracy and wireless bandwidth requirements. CarMap [5] realizes near real-time updates on the feature-represented map by excluding transient information, e.g., parked cars and pedestrians, from map processing. However, these works focus on static information sharing, e.g., point clouds or features, where *LiveMap* allows dynamic adaptive offloading, e.g., images, features or labels, for crowdsourcing vehicles in automotive edge computing networks.

VIII. CONCLUSION

In this paper, we have presented *LiveMap*, a real-time dynamic map that allows efficient information sharing among connected vehicles in automotive edge computing networks. We have developed the data plane of *LiveMap* to detect, match, and track objects on the road based on the crowdsourcing data from connected vehicles in sub-second. We have designed the control plane of *LiveMap* to intelligently schedule vehicles and determine offloading decision according to the availability of networking and computation resources. We have demonstrated *LiveMap* has better performance than existing solutions with both experimental and simulation results.

REFERENCES

- [1] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.
- [2] I. Yaqoob, L. U. Khan, S. A. Kazmi, M. Imran, N. Guizani, and C. S. Hong, "Autonomous driving cars in smart cities: Recent advances, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 174–181, 2019.
- [3] 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*, 2018, pp. 81–95.
- [4] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez, "Internet of vehicles: architecture, protocols, and security," *IEEE internet of things Journal*, vol. 5, no. 5, pp. 3701–3709, 2017.
- [5] F. Ahmad, H. Qiu, R. Eells, F. Bai, and R. Govindan, "CarMap: Fast 3d feature map updates for automobiles," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 1063–1081.
- [6] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
- [7] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [8] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.
- [9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.
- [10] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
- [11] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.
- [12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.
- [13] W. Zhang, B. Han, and P. Hui, "Jaguar: Low latency mobile augmented reality with flexible tracking," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 355–363.
- [14] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.
- [15] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, 2017, pp. 42–47.
- [16] Q. Liu, T. Han, and E. Moges, "EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning," *arXiv preprint arXiv:2003.12911*, 2020.
- [17] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, L. Sun *et al.*, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 910–918.
- [18] J. Achiam, D. Held *et al.*, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 22–31.
- [19] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.
- [20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.
- [21] V. Mnih, K. Kavukcuoglu *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
- [22] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.
- [23] C. Nvidia, "Nvidia CUDA C programming guide," *Nvidia Corporation*, vol. 120, no. 18, p. 8, 2011.
- [24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [25] E. J. Oughton, K. Katsaros, F. Entezami, D. Kaleshi, and J. Crowcroft, "An open-source techno-economic assessment framework for 5G deployment," *IEEE Access*, vol. 7, pp. 155 930–155 940, 2019.
- [26] G. T. . R. 14, *Study on channel model for frequencies from 0.5 to 100 GHz*. 3GPP, 2018.
- [27] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 505–513.
- [28] Q. Chen, X. Ma, S. Tang, J. Guo, Q. Yang, and S. Fu, "F-Cooper: feature based cooperative perception for autonomous vehicle edge computing system using 3D point clouds," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 88–100.