MobileEdge: Enhancing On-board Vehicle Computing Units using Mobile Edges for CAVs

Lin Wang*, Qingyang Zhang*, Youhuizi Li†, Hong Zhong* and Weisong Shi‡
*School of Computer Science and Technology, Anhui University, Hefei, China, 230601
†Key Laboratory of Complex Systems Modeling and Simulation, Hangzhou Dianzi University, Hangzhou, China, 310018
‡Department of Computer Science, Wayne State University, Detroit, MI, USA., 48202
{linwang, qingyang}@thecarlab.org, huizi@hdu.edu.cn, zhongh@ahu.edu.cn, weisong@wayne.edu

***Abstract*—As the rapid growth of connected and autonomous vehicles (CAVs) and 5G intensifies, more third-party applications are increasingly being deployed on CAVs. They not only improve user experience but also provide more helpful services, for example, enhancing public safety by recognizing criminals in real-time videos. Current CAVs prefer to process collected data on the vehicle to avoid long transmission latency and extra network cost. However, due to the limitations of the on-board vehicle computing unit (VCU) and increasing use of computing-intensive in-vehicle applications, the burden of on-board VCU has sharply increased, which may affect driving safety. In particular, for existing vehicles on the road, adding more computing devices is a challenge if not impossible due to cost concerns. Inspired by edge computing, we propose a novel platform, MobileEdge, to enhance the computing capability of the unchangeable on-board VCU, which leverages mobile devices as edge nodes, *e.g.,* the passengers' smartphones, by offloading computing tasks to them for collaboratively computing. Moreover, MobileEdge provides the dynamic management of mobile devices, monitoring device status and interfaces for customizable task offloading strategies and eventually achieves optimal task scheduling. We build a prototype to demonstrate the designed platform and evaluate three task offloading strategies which were implemented based on the developed interfaces. The results show that MobileEdge significantly reduces the application response latency. Compared with the baseline which does not employ task offloading, the response latency is almost near real-time when more computing resources are available. In addition, the proposed shortest response latency strategy outperforms the best overall task scheduling among the three strategies.**

***Index Terms*—edge computing; vehicular data analysis; distributed computing; connected and autonomous vehicles.**

I. Introduction

As the connected and autonomous vehicles (CAVs) technology becomes more mature, drivers are gradually liberated from driving vehicles and have more idle time to do other things, which stimulates the development and application of third-party CAV applications. By utilizing various on-board sensors (*e.g.,* dash camera, lidar, *etc.*), these applications greatly improve the user experience and make the CAVs smarter [1]. However, according to a report from Intel [2], a CAV in 2020 will generate 4TB of data per day. Uploading such a large volume of data to the cloud leads to long transmission latency and extra cost of communication/bandwidth, which is insufferable and expensive for owners. For example, assuming the average uploading speed of the current 4G/LTE cellular network is 20 Mbps, it will take about 20 days to upload 4 TB data to the cloud. Thus, current CAVs (such as Baidu's Apollo [3] and Waymo [4]) chose to process data on their on-board vehicle computing units (VCUs). Aside from the demands of native applications, ever-increasing third-party applications will also consume a lot of vehicle computing resources, most of which leverage a variety of computation-intensive computer vision based or machine learning based algorithms [5] [6], increasing the burden on on-board VCUs. Hence, it is a big challenge for future CAVs to guarantee the driving safety and satisfy the requirements of both native applications and various third-party applications at the same time. Further, the capability of today's VCUs is not as powerful as expected, and adding other high-performance computing devices, such as NVIDIA Drive PX 2, is very expensive and impractical, especially for existing vehicles on the road.

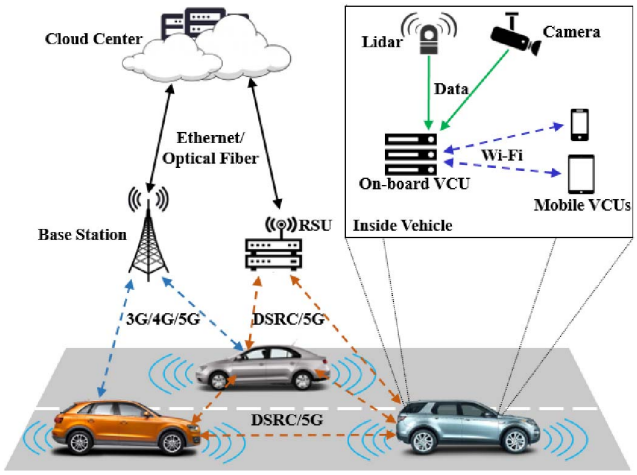

Fig. 1. The overview of edge-based solutions for future CAVs.

Inspired by the emergence of edge computing [7], [8] (also known as fog computing [9], cloudlet [10]), which enables the computation to be performed on the network edge devices/nodes in proximity to the data source so that applications' response latency and bandwidth cost can be reduced, the following two solutions have been developed to enhance the computing capability of VCUs. As shown in Figure 1, the first solution is envisioned by the Open Vehicular Data Analytics Platform (OpenVDAP) [1]. The on-board VCU

978-1-7281-2583-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPADS47876.2019.00073

can offload the computation to the base stations/roadside units and other vehicles using Vehicle-to-RoadSideUnit and Vehicle-to-Vehicle communication [11], such as DSRC or future 5G, respectively. In this solution, the base stations/roadside units and vehicles are performing the role of edge nodes. Note that this is also mentioned in the Vehicle Ad-hoc Network (VANET) [12]. The second solution is the *Inside Vehicle* part in Figure 1. The on-board VCU can also integrate all devices inside a vehicle, such as the mobile devices (*i.e.,* smartphones and tablets) carried by passengers, which are widely used and have an increasingly powerful computing capability. In this case, the mobile devices act as the edge nodes to provide extra computing resources for the on-board VCU. However, existing platforms do not support the collaborations of the on-board VCU and mobile devices very well. The widely-used container based or virtual machine based approaches have not considered the mobile device scenario in the design phase.

In this paper, we propose to employ mobile devices (*e.g.,* smartphones) as auxiliary computing units (*i.e.,* mobile VCUs) to efficiently support CAVs. We designed a collaborate platform inside CAVs, MobileEdge, which can enable the on-board VCU to flexibly offload part or all computation to the mobile VCUs based on a scheduling and offloading strategy. In this way, we can relieve the on-board VCU from the heavy workload as well as improve the response latency and user experience. MobileEdge is usually used in crowdsourcing scenarios to offload the additional computation of various third-party applications, while the vehicles' own VCUs are primarily used to handle the native applications, such as Advanced Driver-Assistance System (ADAS) [1]. The contributions are summarized as follows:

- We formulated the problem that the resource-limited on-board VCU is incapable of meeting the computation requirements of more and more latency-sensitive and computation-intensive CAV applications, including native driving applications and diverse third-party applications.
- We designed an edge-based platform, MobileEdge, which employs passengers' mobile devices to enhance the capability of on-board VCU. It splits the whole workload into a general-purpose computing part and an AI computing part, supports offloading at both the code level and data level, and provides task scheduling interfaces for developers. Additionally, to achieve optimal results, MobileEdge manages devices dynamically by monitoring devices' real-time status and then offloading tasks accordingly.
- We implemented a prototype system and three task scheduling strategies. Experimental results showed that our system can indeed improve VCU performance. Moreover, the lessons learned from this process are presented for further study.

The remainder of this paper is organized as follows. We first discuss the motivation and introduce the design of MobileEdge in Sections II and III, respectively. The implementation of a prototype is presented in Section IV, and the three task scheduling strategies are discussed in Section V. We present the experiments and results in Section VI, which is followed by a discussion in Section VII. Related works are reviewed in Section VIII. Finally, we conclude this paper in Section IX.

II. Motivation for MobileEdge

The current resource-limited on-board VCU may not be powerful enough to support extra computation-intensive and latency-sensitive third-party CAV applications since native CAV applications have consumed most vehicle computing resources. Fortunately, the development of mobile devices provides a way to enhance the capability of on-board VCU. In this section, we discuss the motivations of our work from the aspects of main algorithms and third-party applications as well as the capability of mobile devices.

A. Computation-intensive Algorithms

Generally, most CAV applications use computer vision based or deep learning based data processing algorithms to analyze a large amount of vehicle data collected by various on-board sensors (*e.g.,* dash camera and lidar), which requires a lot of computing resources and results in a high response time.

TABLE I
The performance of some CAV applications' core algorithms.

Name	Latency (ms)
Lane Detection (Computer Vision) [13]	41.11
Lane Detection (TensorFlow) [14]	1022.86
Vehicle Detection [15]	2437.43
Image Recognition [16]	101.903

To better understand the computing intensity in CAV applications, we take three types of algorithms as examples and evaluate their performance. We first choose two types of the most representative algorithms in ADAS: *Lane Detection* and *Vehicle Detection*. We adopt two different *Lane Detection* algorithms to detect the lane. One leverages various computer vision based image processing methods, and the other mainly is a deep learning algorithm implemented using TensorFlow. *Vehicle Detection* also employs a deep learning algorithm to detect vehicles. Additionally, an *Image Recognition* algorithm is also evaluated, which is commonly used in video-related CAV applications. The *Image Recognition* is mainly composed of two steps. The first step uses the histogram equalization method to enhance the contrast of the image, making darkening the image to provide a higher quality for recognition. The second step runs a convolutional neural network based deep learning model to identify objects and label the content of the image.

We have performed experiments with the above four algorithms on a laptop computer with an Intel i5-7300HQ CPU @ 2.50GHz, and the results are presented in Table I. It can be seen that the deep learning based algorithms have higher latency than the computer vision based algorithms, but they also achieve higher accuracy. For example, the deep learning based *Lane Detection* could still successfully detect all lane

lines when they are partially covered or erased. Thus, the deep learning based algorithms will increasingly be applied. In conclusion, the core algorithms of native applications and diverse third-party applications are generally computation-intensive and resource-hungry.

B. Latency-sensitive Third-party Applications

Recently, with the advancement of edge computing, many CAV applications have been proposed, which can greatly improve users' experience. In particular, video analytics on CAV could enhance public safety [17]. For example, Zhang *et al.* [18] proposed Amber Alert Assistant (A3), which leverages edge nodes to track a kidnapper's vehicle by recognizing the license plate number from the video data. It speeds up the search efficiency and improves the AMBER Alert system. Liu *et al.* [19] presented SafeShareRide, which analyzes the in-vehicle audio and video data as well as real-time driving behavior to guarantee the safety of drivers and passengers in ride-sharing services. Lee *et al.* [20] proposed Gremlin, an interactive scheduling system, which determines whether the driver can safely pay attention to other tasks by analyzing current driving conditions and interactive features. Grassi *et al.* [21] proposed ParkMaster, which utilizes an on-board camera to analyze the roadside parking situation and reports free parking spaces to the cloud server to guide other vehicles. Additionally, Raja *et al.* [22] designed WiBot, which detects distracted behavior and enables passenger-car interaction by analyzing human body movement in the in-vehicle video data to improve driving safety and users' experience. We can see that time is a key factor for CAV applications; they process real-time data and use the feedback to guide the next operation. Late results may be useless in many situations such as object/human tracking and collision prediction. Hence, CAV applications usually require low response latency to process a large amount of data. However, as these applications' core algorithms are computation-intensive, the resource-limited on-board VCU is incapable of meeting deadlines, especially when a lot of vehicle computing resources have been consumed by ADAS applications. Therefore, how to use resource-limited on-board VCU to achieve the computing requirements of these latency-sensitive applications is a big challenge.

C. High Capability Mobile Devices

As we discussed aboved, the current on-board VCU cannot satisfy the requirements of the rapid-growth of CAV applications. For existing vehicles on the road, adding more computing devices or replacing them with a high-performance computing device is challenging due to cost concerns. Fortunately, today's mobile devices are powerful with high-end chips. Thus, enhancing on-board VCUs by using mobile devices in a vehicle is a promissing approach.

Mobile devices, such as smartphones and tablets, are widely used-today, and their computing capability has experienced substantial growth. Modern smartphones contain not only multi-core architectured CPUs but also co-processors such as GPUs and AI chips, and both of them provide huge computing

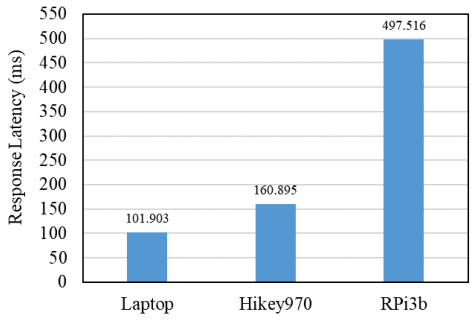

Fig. 2. The performance of image recognition algorithm on different mobile devices.

capability. The Huawei Mate10, for example, is equipped with a Hikey970 8-core CPU, a Mali-G72 12-core GPU and an AI chip, which is referred to as the Neural Processing Unit (NPU). The AI chip is designed to handle fast-growing AI computing tasks like *Vehicle Detection*. It can recognize 2000 pictures per minute, which is 25 times faster than CPU, and its energy consumption is one-fifth of the CPU [23]. Hence, the AI chip is the perfect computing unit to perform a large number of AI related tasks on the CAV platform. Moreover, Huawei unveiled the new Hikey980, which is equipped with a dual-core NPU AI chip that can recognize up to 4,500 images per minute [24]. Likewise, other chip manufacturers have also integrated some specialized chips such as the AI chip or DSP, to support complex AI functions. The products include the Apple A12 [25], the Qualcomm Snapdragon 855 [26], and so on. In addition, as these mobile devices become ever increasingly powerful, especially smartphones, there are many machine learning frameworks adapted to mobile devices. Task TensorFlow Lite [27] released by Google is an example; it is a lightweight deep learning framework for mobile and embedded devices that supports running various deep learning algorithms across platforms including Android and iOS. With TensorFlow Lite, mobile devices can perform deep learning based AI computing tasks, even leveraging the acceleration benefits of GPUs.

To further understand the high computing capability of these mobile devices, we use the *Image Recognition* algorithm described in the previous subsection to evaluate the following three different mobile devices: a laptop , a Hikey970 board and a Raspberry Pi 3b (RPi3b). The results, as presented in Figure 2, show that mobile devices could be fully exploited to greatly enhance the capability of the on-board VCU.

Up to this point, we have discussed the motivation of our work in detail. More latency-sensitive third-party CAV applications are increasingly being deployed, and part of them use computation-intensive core algorithms. As our preliminary experimental results show, it is impossible for a resource-limited VCU to satisfy the requirements of these applications. Modern mobile devices, such as smartphones, have an increasing computing capability and are widely used. Hence, we propose that mobile devices in the CAV can be exploited

to share the computing burden of the on-board VCU.

III. MobileEdge System Design

In this section, we will introduce the detailed design of MobileEdge, including terminology, design goals, and system architecture, to illustrate how MobileEdge enhances the on-board VCU computing capability.

A. Terminology

In MobileEdge, *code migration* and *data offloading* are two essential parts for computing task offloading.

- ***Code Migration***: Code includes the business logic and dependent libraries of an application [28]. Generally, the computing tasks of most CAV applications are composed of general-purpose computing such as image processing and AI computing such as running a deep learning model. Thus, in MobileEdge, an application's code consists of three parts: the general-purpose computing code, the AI computing code and the task profile, which defines the composition of the two types of computing. Code migration is critical for correctly executing the tasks on the target device. In addition, the same types of tasks can share the same code.
- ***Data Offloading***: In MobileEdge, data refers to the input data of the tasks used for executing the code. Therefore, in the process of offloading a task, data offloading is also critical for completing a task and obtaining the correct result on the target device.

B. Design Goals

MobileEdge is designed to reduce CAV applications' response latency by enhancing the computing capability of on-board VCU by utilizing passengers' mobile devices. Specifically, the design goals could be described as follows:

- ***Scalable distributed computing platform***: As mentioned above, mobile devices, such as passengers' smartphones, could be leveraged to enhance the on-board VCU. However, these mobile devices are distributed and have high mobility. It is necessary to build a scalable distributed computing platform to scale and orchestrate these mobile devices so that code/data can be migrated and offloaded to these devices for computing.
- ***Dynamic device management***: In MobileEdge, generally, these mobile devices are heterogeneous and possess diverse computing capabilities. In addition, the available computing resources of each mobile device are also changing dynamically due to the continuous use of these devices, leading to a varying workload. In addition, due to their high mobility and independence, these mobile devices may join or leave MobileEdge at any time. As an example of this variable environment, users can refuse to connect to the CAV, or they are unable to connect due to a lost or weak signal. Therefore, a dynamic device management service is required to maintain and manage these connected mobile devices as well as the on-board VCU.

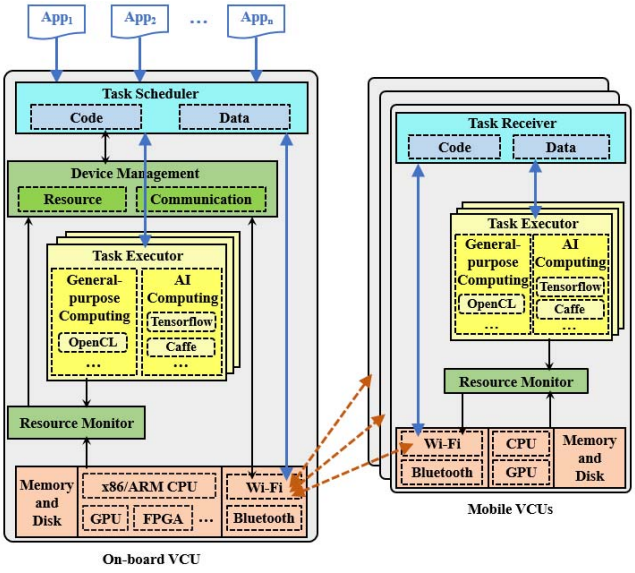

Fig. 3. The architecture of MobileEdge.

- ***Flexible task scheduling***: In MobileEdge, multiple mobile devices are connected to the on-board VCU. In order to make the best use of the available computing resources of each device and guarantee various CAV applications to be completed with acceptable latency, a flexible task scheduling service is required. It can offload the tasks to the optimal device (either the on-board VCU or the connected mobile device) according to a pre-defined scheduling strategy such as computing-capability-aware, workload-aware or energy-aware strategies.

C. System Architecture

To achieve the aforementioned goals, we designed a system architecture for MobileEdge, as presented in Figure 3. MobileEdge consists of two types of devices: an on-board VCU and a number of mobile VCUs. Each mobile VCU connects with the on-board VCU via a wireless local area network such as the in-vehicle Wi-Fi network. For a CAV application, the on-board VCU can first migrate the code to each mobile VCU, and then, for each task of this application, the on-board VCU decides, according to a task scheduling strategy, whether the task should be executed locally or offloaded to a connected mobile VCU by offloading the input data. Through offloading tasks, MobileEdge could significantly shorten the applications' response latency.

According to the device type, the entire system architecture can be divided into two parts: the on-board VCU and the mobile VCU.

On-board VCU: The on-board VCU is inherently installed in the vehicle to provide computing, storage and communication capabilities. In MobileEdge, the on-board VCU not only acts as the system central control unit but also is a task execution unit. It consists of four modules: *Resource Monitor*, *Device Management*, *Task Scheduler* and *Task Executor*. These

modules work collaboratively to enable the on-board VCU to manage all mobile VCUs and itself, schedule and offload and execute tasks. The details of the four modules are as follows:

- ***Resource Monitor***: This module is responsible for monitoring the usage status of a variety of resources, including computing and energy resources on a device. Because the available resources of a device may be changing all the time based on the system workload, this module needs to periodically update the profile about available resources and report to the *Device Management* module.
- ***Device Management***: This module takes charge of managing all devices, including the on-board VCU and all connected mobile VCUs, in terms of resources and communication. First, it accepts and manages the connections from the mobile VCUs and can authenticate and authorize the connected mobile VCUs to improve system security. Second, it also integrates and manages the profile of available resources reported by the *Resource Monitor* module on each device. In addition, this module also needs to record some feature information, such as task type, about computing tasks to support scheduling different types of tasks efficiently.
- ***Task Scheduler***: The primary responsibilities of this module include migrating the code to each mobile VCU, scheduling and offloading the computing tasks of CAV applications to an optimal destination, and gathering the computing results from each VCU. When a new mobile VCU is connected to MobileEdge or a connected mobile VCU is missing code , this module will migrate the code to it. For a computing task, this module analyzes the available resources on each VCU provided by the *Device Management* module to determine if the task should be performed locally on the on-board VCU or offloaded to a mobile VCU according to a task scheduling strategy.
- ***Task Executor***: This module only needs to execute the computing tasks using the input data based on the code. As a computing task generally has two types of computing, we design two sub-modules, namely *General-purpose Computing* and *AI Computing* as shown in Figure 3, to process the corresponding computing. The former can be supported by several frameworks such as Open Computing Language (OpenCL) [29], and the latter also can be supported by some frameworks such as TensorFlow or Caffe [30]. The two sub-modules create a running environment for computing tasks in accordance with the code. Then they process the corresponding computing according to the task profile which illustrates the execution process of the two types of computing.

Mobile VCU: Mobile VCUs refer to the mobile devices, *e.g.,* smartphones, which are brought on board vehicles by passengers. In MobileEdge, the mobile VCU mainly acts as a destination device for task offloading. It is composed of three modules: *Task Receiver*, *Task Executor* and *Resource Monitor*. These modules enable the mobile VCU to monitor its available resources and report to the on-board VCU. Additionally, the mobile VCU can receive and process tasks, and finally return the computing results. The last two modules have the same function as on the on-board VCU, so we only introduce the different module in detail as follows:

- ***Task Receiver***: This module receives the migrated code and the offloaded input data for computing tasks from the *Task Scheduler* module on the on-board VCU. Then, it assigns these tasks to its own *Task Executor* module for execution. After these tasks are completed, this module will return the computing result to the *Task Scheduler* module of the on-board VCU. In addition, when the mobile VCU is missing an application's code, the module needs to request the code from the on-board VCU.

IV. Prototype Implementation

To assess the feasibility and evaluate the system performance, we implemented a prototype system with communication support between the on-board VCU and the mobile VCUs. Futher, we leveraged two existing frameworks, OpenCL and TensorFlow Lite, to support various CAV applications. In addition, we also implemented the resource management service to handle diverse task scheduling strategies.

In MobileEdge, we implemented the on-board VCU as the system central control unit to accept the connection from all mobile VCUs, and we adopted a whitelist-based authentication mechanism, so only the authenticated mobile VCUs can be connected to MobileEdge. Each CAV application running in MobileEdge should have three types of code files: the OpenCL code files, the TensorFlow Lite model files and a task configuration script file, which corresponds to the three parts of the CAV applications' code. When a mobile VCU is successfully connected to MobileEdge or a connected mobile VCU is missing code, to achieve code migration, the on-bard VCU packages the corresponding code and profile files for each CAV application and migrates them to the mobile VCU.

As mentioned above, a computing task in MobileEdge has two types of computing: the general-purpose computing and the AI computing. Therefore, we use OpenCL to support the general-purpose computing. Specifically, we rely on OpenCL kernel programming to achieve the general-purpose computing task. As to the AI computing, we employ TensorFlow Lite to execute the deep learning models. Moreover, for the task profile, JavaScript Object Notation (JSON) structure, which is a lightweight data-interchange format, is used to describe the task configuration information, including the application name, the application code files and the task execution steps. Taking *Image Recognition* (described in Section II) as an example, we give its detailed task profile as shown in Listing 1, which is also used in our experiment. In the "code_files", it describes the OpenCL code file and the TensorFlow Lite model file. The "execution_steps" is the execution steps of two types of computing in a task. For the general-purpose computing, the "opencl_kernels" details each OpenCL kernel's function name and parameter settings. Furthermore, the kernel's execution device type can be specified in "device", including CPUs, GPUs and other acceleration processors. Regarding the AI

computing, it runs the deep learning model specified in "tensorflow_model_file", according to the input and output tensors' settings given in "input_tensors" and "output_tensors".

Listing 1. The task profile of *Image Recognition* in JSON format.

```
{ "application_name": "Image_Recognition",
  "code_files": {
    "opencl_code_file": "histogram_equalization.cl",
    "tensorflow_model_file": "mobilenet_v1.tflite" }
  "execution_steps": [
    { "computing_type": "General-purpose_Computing",
      "opencl_kernels": [
        { "kernel_name": "calculate_histogram",
          "device": "cpu",
          "parameters": [
            { "parameter_type": "cl_mem",
              "buffer_label": "input_buf",
              "buffer_size": [720, 1280, 3] },
            { "parameter_type": "cl_mem",
              "buffer_label": "mid_buf_1",
              "buffer_size": [3, 256, 4] } ],
          "global_work_size": [3] },
        { "kernel_name": "histogram_equalization",
          "device": "gpu",
          "parameters": [
            { "parameter_type": "cl_mem",
              "buffer_label": "input_buf" },
            { "parameter_type": "cl_mem",
              "buffer_label": "mid_buf_1",
              "buffer_size": [3, 256, 4] },
            { "parameter_type": "cl_mem",
              "buffer_label": "mid_buf_2",
              "buffer_size": [720, 1280, 3] } ],
          "global_work_size": [720, 1280, 3] },
        { "kernel_name": "resize_image",
          "device": "gpu",
          "parameters": [
            { "parameter_type": "cl_mem",
              "buffer_label": "mid_buf_2",
              "buffer_size": [720, 1280, 3]},
            { "parameter_type": "cl_mem",
              "buffer_label": "output_buf",
              "buffer_size": [224, 224, 3] } ],
          "global_work_size": [224, 224, 3] } ] },
    { "computing_type": "AI_Computing",
      "input_tensors": [{ "dim_size": [1, 224, 224, 3] } ],
      "output_tensors": [{ "dim_size": [1, 1001] } ] } ] }
```

To support a variety of task scheduling strategies, we also implemented a resource management service, which provides multiple different interfaces to developers to build efficient task scheduling algorithms based on available system computing resources. We mainly manage the resources of the system in the following four aspects: 1) the computing capability of each VCU, 2) the system resources usage (includes the CPU and the memory utilization), 3) the system workload on each VCU, and 4) the energy status of each VCU, especially the available residual battery energy for the mobile VCUs.

V. Scheduling Strategies

Task scheduling strategies are critical to the performance of MobileEdge, which guides the scheduler to assign the tasks to the optimal VCU to minimize the response latency. The response latency is defined as the time interval between the start time, when a task is generated, and the end time, when the on-board VCU obtains the result of the task. As mentioned in the previous section, MobileEdge provides interfaces for developers to build various task scheduling strategies, which may depend on multiple types of computing resources. In this paper, we take the following three task scheduling strategies as examples to show how they are built and how they work in MobileEdge; other strategies are also compatible. For simplicity, the three case strategies are used for the same type of task, while the scheduling of multiple different types of tasks can also be considered by developers.

Shortest Queue Length (SQL) strategy: The SQL strategy always assigns the tasks to the VCU that has the least number of tasks queued upon the time of scheduling. When a task is generated, the scheduler first calls the interfaces to query the current task queue length (*i.e.,* workload status) of all VCUs (including the on-board VCU and all connected mobile VCUs), summarized in the local *Device Management* module. Then, it assigns the task to the VCU that has the minimum workload.

Strongest Computing Capability (SCC) strategy: The SCC strategy prefers to assign the tasks to the most powerful VCU which is not saturated. Given the fact that there may be many diverse types of mobile devices (*i.e.,* smartphones, tablets, etc.) in the real scenario, we consider that the on-board VCU and all connected mobile VCUs are heterogeneous on the aspect of computing capability, and the VCU with a higher computing capability is able to complete the task faster. To this end, when a task is generated, the scheduler first calls the interfaces to acquire the computing capability and current workload status of each VCU, also summarized in the local *Device Management* module. Then it chooses the one with the strongest computing capability among VCUs with unsaturated workload to offload the task. Note that if all current VCUs are saturated, the scheduler will need to wait for an available unsaturated VCU to reschedule the task.

Shortest Response Latency (SRL) strategy: The SRL strategy tends to assign a task to the VCU that is estimated to have the shortest response latency. For the tasks offloaded to the mobile VCUs, the response latency mainly consists of three parts: network transmission time, computing queuing time and computing processing time. If the tasks are performed locally on the on-board VCU, then there is no transmission time cost. The transmission time can be estimated using $\frac{d}{r}$, in which d is the data size of a task and r is the network data transmission rate measured periodically. The queuing time, which represents the waiting latency in the task queue, can be approximately calculated by $\frac{n_t \times T_a}{n_e}$, in which T_a is the average processing time of all tasks of the same type on each VCU, n_t and n_e represent, respectively, the number of tasks queued (*i.e.,* the current workload status) and the number of tasks executed. For processing time, it can also be estimated by the average processing time T_a of all tasks completed so far on this VCU. Hence, to build the SRL strategy, the scheduler needs to know parameters d, r, n_t, n_e and T_a on all VCUs. When a task is generated, the scheduler first analyses task data size d, then calls the interfaces to obtain the value of other parameters in the local *Device Management* module, and finally uses them to estimate the response latency for each VCU based on the above method. Finally, the scheduler assigns the task to the VCU with the estimated shortest response latency.

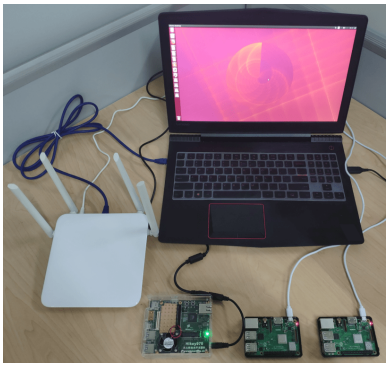

Fig. 4. The testbed.

VI. Performance Evaluation

In this section, we comprehensively evaluate the performance of MobileEdge. After briefly introducing the experimental setup, we compare and analyze the results of the three proposed strategies in detail.

A. Experimental Setup

We have built a testbed for our prototype system, which consists of a router, a laptop computer and three development boards as shown in Figure 4. In this testbed, the router acts as a wireless access point to provide a stable network. Similar to the Baidu Apollo platform, which uses Intel processors, we also use a laptop with an Intel i5-7300HQ CPU @ 2.50GHz as the on-board VCU, and it is connected to the router via a wire. Considering that mobile VCUs are usually heterogeneous in the real world, we leverage three different development boards: one Hikey970 development board (mobile VCU #1) and two RPi3b board (mobile VCU #2 and #3) to work as three mobile VCUs which are connected to the router using built-in 2.4 GHz Wi-Fi.

Since real-time video analytics is one of the most important services in CAV applications, we take it as our evaluation workload. Specifically, we use a 60 second video data, which is 25 frames per second with the resolution of 1280 × 720 pixels, as the input. The video stream is decoded locally to extract every frame image, and then the goal is to identify objects and label the content of the image. The *Image Recognition* algorithm (as described in Section II) is used, and one frame image is corresponded to one task. In addition, in our experiments, we don't use the TensorFlow Lite's gpu acceleration feature due to its own limitation on the development boards.

B. Experimental Results

We evaluate the performance of MobileEdge by conducting a set of controlled experiments on our testbed. First, as the baseline case, the on-board VCU executed all tasks locally without offloading any tasks to mobile VCUs. Then, we analyzed the performance of the three proposed task scheduling strategies (*i.e.,* the SQL, SCC and SRL strategies discussed in Section V) in MobileEdge.

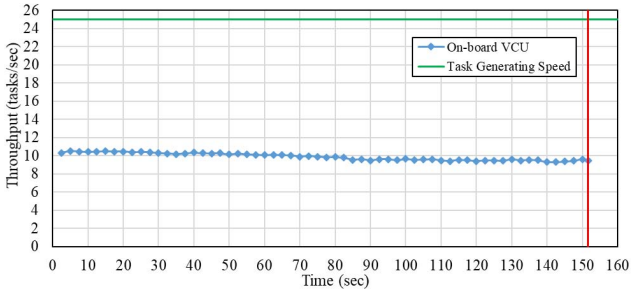

Fig. 5. The performance without task offloading.

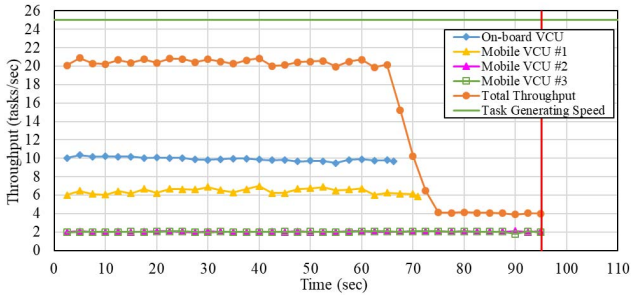

Fig. 6. The performance with using the SQL strategy.

Figure 5 illustrates the task throughput on the on-board VCU when all tasks are performed locally. There is no mobile VCU data shown in the figure since no task is offloaded to mobile VCUs and their throughput is zero all the time. It can be seen that the on-board VCU takes a very long time, more than twice the time of the video stream length, to complete all the tasks.

Figures 6, 7 and 8 present the task throughput on each VCU when the on-board VCU offloads part of the tasks according to the SQL, the SCC and the SRL task scheduling strategies, respectively. Figure 9 presents the number of tasks (*i.e.,* workload) assigned to each VCU based on the SQL, SCC and SRL strategies. Figure 10 shows the task response latency regarding different strategies. From these results, we can see that, compared to the baseline case (no task offloading), the workload of the on-board VCU is greatly reduced, and the final completion time and the response latency of all tasks is also significantly improved because of the offloading with MobileEdge. Nevertheless, there are still some differences between these three task scheduling strategies. To further explore, two performance metrics are analyzed: *final completion time* and *average response latency* for all tasks.

The final completion time: As shown in Figures 5, 6, 7 and 8, the final completion time of these four cases (*i.e.,* no task offloading as well as using the SQL, SCC, and SRL scheduling strategies) is indicated by the red lines. The SQL strategy has the longest time among the three strategies. That is because the SQL strategy always assigns tasks to the VCU that have the least workload without considering the computing capability, which leads to the situation of more tasks being assigned to

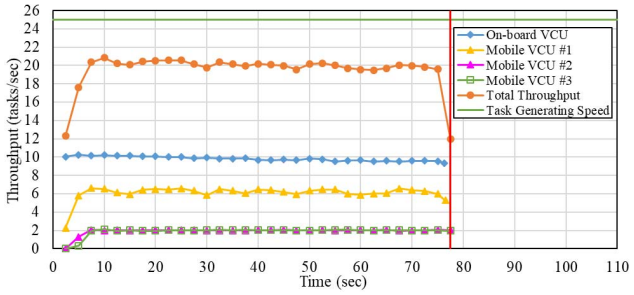

Fig. 7. The performance with using the SCC strategy.

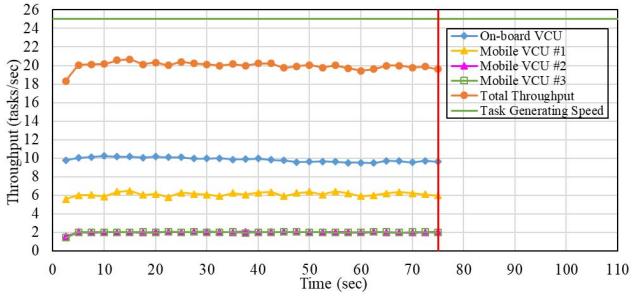

Fig. 8. The performance with using the SRL strategy.

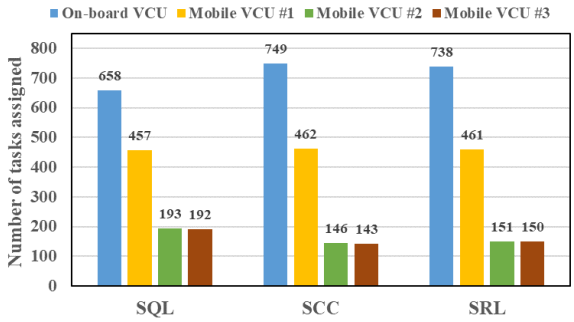

Fig. 9. The number of tasks assigned to each VCU.

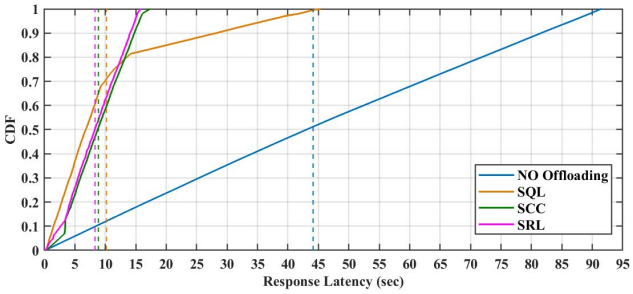

Fig. 10. Response latency regarding different task scheduling strategy.

the low power VCUs (*i.e.,* mobile VCU #2 and #3) while the VCUs (*i.e.,* the on-board VCU and the mobile VCU #1) that have higher computing capability only processed fewer tasks. Therefore, as shown in Figure 6, although the system's total task throughput is high from the beginning, it decreases sharply as the on-board VCU and the mobile VCU #1 quickly complete the assigned tasks and then stay low until all tasks are completed because mobile VCUs #2 and #3 still take more time to finish the over-assigned tasks. The SCC strategy works better than the SQL strategy while it is slightly worse than the SRL strategy. As presented in Figure 7, the system's total task throughput is low in the beginning since the SCC strategy tends to assign tasks according to the computing capability, resulting in most tasks being assigned to the on-board VCU while mobile VCUs #1, #2 and #3 have a low or even zero task throughput. Further, it declines in the end because the time taken by each VCU to complete the saturated workload varies slightly, in which the on-board VCU is fastest, followed by mobile VCU #1, and mobile VCUs #2 and #3 are the slowest. In addition, as shown in Figure 9, the SQL strategy significantly reduces the number of tasks assigned to mobile VCUs #2 and #3, while it increases the workload of the on-board VCU compared to the SRL strategy. Different from the SQL and SCC strategies, the SRL strategy takes into account the transmission time, the queuing time, and the processing time; thus, it results in an appropriate task distribution by predicting the response time of each task so that each VCU takes almost the same amount of time to complete the assigned tasks, enabling the system's total task throughput to remain high from beginning to end as shown in Figure 8. Additionally, compared to the SCC strategy, in the first few seconds, it improves the task throughput of mobile VCUs #1, #2 and #3 by assigning them more tasks. As a result, the SRL strategy achieves the best performance of the three strategies.

The average response latency: As shown in Figure 10, the four different colored dotted lines represent the average response latency of all tasks in the four cases, respectively. Compared with the baseline case (NO Offloading), the average response latency can be significantly reduced when the SQL, SCC, and SRL scheduling strategies are applied in MobileEdge. More specifically, among the three strategies, the SRL strategy has the lowest average response latency, followed by the SCC strategy, and the highest is the SQL strategy. Note that for the SQL strategy, fewer tasks are assigned to the on-board VCU with higher computing capability, and more tasks are assigned to the lowest computing capability mobile VCUs #2 and #3; thus, although the response latency of most tasks is less than the SRL strategy, there are still a small number of "tail tasks" whose response latency is so much higher, which leads to the high average response latency.

VII. Lessons Learned and Discussion

In this section, we discuss the lessons learned from the design and evaluation process and the potential future directions of system improvement. In addition, we believe that MobileEdge can also be used in other similar scenarios that involve a centralized node and some mobile devices (*i.e.,* smartphones).

To improve the performance, an effective way is offloading more tasks to more mobile VCUs. For example, if we continue to add a Hikey970 board, our preliminary experimental result shows that the response latency can be further reduced and even achieve near real-time video analytics. Note that, in MobileEdge, scalability is one of the crucial design goals, and we can successfully support a large number of connected VCUs. However, there is always a maximum limit for adding VCUs. With increasingly more computation-intensive and resource-hungry CAV applications installed, it is better to improve the system performance from the workload itself, such as algorithms and application design. For instance, model compression can be used to significantly reduce the computing intensity of deep learning algorithms [31]. Further, in video analytic applications, a motion detection module can first be used to detect the different areas between two frames to avoid repeated calculations [18].

MobileEdge also has some aspects that can be optimized in the future. First, the task execution in the current prototype does not support parallel computing, so we plan to modify the *Task Processor* module so that it can handle parallel computing and automatically expand or shrink tasks based on the devices' status. Second, security is very important in CAV, and it directly affects driving safety. However, the current version of our design has not taken security into account. Therefore, in the future, we will design a security module in such a distributed computing platform, regarding data security, identity security and system security. For example, we will consider how to guarantee the integrity of the offloaded data and how to authenticate the identity of mobile VCUs with a securer strategy. In addition, the offloaded code should also be authenticated. To solve these problems, we will first consider utilizing some trusted platforms, such as Intel Software Guard eXtensions (SGX) for on-board VCU and ARM TrustZone for mobile VCUs [32], to provide a hardware-based trusted execution environment and the proof of integrity for these offloaded code and transmitted data. In addition, there has been some research on the safety of our scenario, such as [33]. Finally, since the focus of this paper is to build MobileEdge and provide various scheduling interfaces for developers, the three scheduling strategies proposed are only applicable to a single type of task. In future work, we will continue to consider scheduling strategies for multi-type tasks.

VIII. Related Work

In recent years, edge computing has become popular in the CAVs research field due to its advantage of dramatically reducing response latency. There have been many studies on the edge computing system and architecture for CAVs. For examples, Hou *et al.* [34] presented a new architecture known as Vehicular Fog Computing, which aggregates available computing and communication resources from each vehicle to enhance the quality of vehicular services. It relies on a cluster composed of parked and slow-moving vehicles. Feng *et al.* [35] proposed a framework, named autonomous vehicular edge, which employs the idle computing resources of each vehicle to increase the computing capability of vehicles in a decentralized manner through the autonomous organization of vehicular edges. Li *et al.* [36] also proposed a framework called vehicular edge cloud computing, which offloads vehicular computing tasks to the edge cloud server on the base station. Each vehicle can flexibly request its temporarily individual computing resources in the edge cloud. Moreover, Zhang *et al.* [37] proposed a regional cooperative fog computing based architecture for dealing with big Internet of Vehicle (IoV) data in a smart city, which adopts a coordinator server to organize local fog servers on multiple base stations and roadside units to achieve low-delay for vehicular applications. Zhou *et al.* [38] presented a framework, called BEGIN, which utilizes big data to improve energy efficiency in vehicular edge computing. It mainly employs an edge computing layer which is composed of the vehicles, base stations and roadside units. However, all of these studies have focused on the collaborations between vehicles or a vehicle and edge/fog servers, which may not contribute these computing resources. In [1], Zhang *et al.* proposed the OpenVDAP, an open vehicular data analytics platform for CAVs, which is an edge-based full-stack platform. It not only supports security and privacy protection but also effectively improves the in-vehicle services' quality and user experiences. Different from the aforementioned frameworks, in addition to leveraging vehicles, base stations and roadside units, the OpenVDAP also tries to exploit other possible on-board computing devices, such as passengers' smartphones, but it did not give a specific architecture design.

In addition to native vehicle cases, several general-purpose edge computing platforms are also promising to adapt to CAVs [39]. Yi *et al.* [40] presented the LAVEA platform, which enables computation offloading to edge nodes to provide video analytics services. Zhang *et al.* [41] also proposed Firework, an edge computing based programming framework for data processing and sharing in hybrid edge-cloud analytics. However, they cannot support the code migration or do not mention how to manage the code in their platforms.

IX. Conclusion

With the development of CAVs, various native applications and more third-party applications will be installed in the future. It is a huge challenge for CAV systems due to the limited resources on vehicles. Hence, we proposed an edge computing based platform which employs mobile devices carried by passengers as the mobile edge nodes to enhance the on-board VCU's computing capability. With MobileEdge, the on-board VCU can offload part of or all of the computing tasks to mobile VCUs. Compared with traditional offloading methods, MobileEdge not only supports dynamic device management, flexible task scheduling and offloading strategies, but it can also efficiently process general-purpose computing and AI computing tasks on mobile devices. Moreover, we implemented a prototype and three task scheduling strategies. The experimental results show that MobileEdge can significantly reduce the response latency of in-vehicle applications, which helps effectively guarantee driving safety and user experience.

ACKNOWLEDGMENT

The corresponding author is Qingyang Zhang. This work is supported in part by the Key Technology R&D Program of Anhui Province (No. 1704d0802193), the National Natural Science Foundation of China (No. 61572001, 61802093) and the Natural Science Foundation of Zhejiang Province (NO. LQ18F020003).

REFERENCES

[1] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, and H. Zhong, "Openvdap: An open vehicular data analytics platform for cavs," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, July 2018, pp. 1310–1320.

[2] P. Nelson, "Just one autonomous car will use 4,000 gb of data/day," https://www.networkworld.com/article/3147892/internet/one-autonomous-car-will-use-4000-gb-of-dataday.html, 2016, accessed: 2018-12-01.

[3] Baidu, "Apollo," http://apollo.auto, 2018, accessed: 2018-12-10.

[4] Waymo, "Waymo," https://waymo.com/, 2018, accessed: 2018-12-10.

[5] Y. Wang, S. Liu, X. Wu, and W. Shi, "Cavbench: A benchmark suite for connected and autonomous vehicles," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2018, pp. 30–42.

[6] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Realtime computer vision with opencv," *Queue*, vol. 10, no. 4, pp. 40:40–40:56, Apr. 2012.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[8] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[11] S. Chen, J. Hu, Y. Shi, Y. Peng, J. Fang, R. Zhao, and L. Zhao, "Vehicle-to-everything (v2x) services supported by lte-based systems and 5g," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 70–76, 2017.

[12] S. Garg, A. Singh, K. Kaur, G. S. Aujla, S. Batra, N. Kumar, and M. S. Obaidat, "Edge computing-based security framework for big data analytics in vanets," *IEEE Network*, vol. 33, no. 2, pp. 72–81, March 2019.

[13] "Lane detection for self-driving car, using computer vision techniques," https://github.com/maunesh/advanced-lane-detection-for-self-driving-cars, 2017, accessed: 2019-09-23.

[14] "Implemention of real time lanenet model for lane detection using deep neural network model," https://github.com/MaybeShewill-CV/lanenet-lane-detection, 2018, accessed: 2019-09-23.

[15] "A car detection model implemented in tensorflow," https://github.com/MarvinTeichmann/KittiBox, 2017, accessed: 2019-09-23.

[16] "Image classification," https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/g3doc/models/image_classification/overview.md, 2018, accessed: 2019-09-23.

[17] Q. Zhang, H. Sun, X. Wu, and H. Zhong, "Edge video analytics for public safety: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1675–1696, Aug 2019.

[18] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Distributed collaborative execution on the edges and its application to amber alerts," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3580–3593, Oct 2018.

[19] L. Liu, X. Zhang, M. Qiao, and W. Shi, "Safeshareride: Edge-based attack detection in ridesharing services," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2018, pp. 17–29.

[20] K. Lee, J. Flinn, and B. D. Noble, "Gremlin: Scheduling interactions in vehicular computing," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 4:1–4:13.

[21] 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.

[22] M. Raja, V. Ghaderi, and S. Sigg, "Wibot! in-vehicle behaviour and gesture recognition using wireless network edge," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, July 2018, pp. 376–387.

[23] Huawei, "Huawei reveals the future of mobile ai at ifa 2017," https://consumer.huawei.com/en/press/news/2017/ifa2017-kirin970/, 2017, accessed: 2019-07-16.

[24] ——, "Huawei launches kirin 980, the world's first commercial 7nm soc," https://consumer.huawei.com/en/campaign/kirin980/, 2018, accessed: 2019-07-16.

[25] Apple, "A12 bionic: The smartest, most powerful chip in a smartphone," https://www.apple.com/iphone-xs/a12-bionic/, 2018, accessed: 2019-07-16.

[26] Qualcomm, "Qualcomm announces new flagship snapdragon 855 mobile platform - a new decade of 5g, ai, and xr," https://www.qualcomm.com/news/releases/2018/12/05/qualcomm-announces-new-flagship-snapdragon-855-mobile-platform-new-decade, 2018, accessed: 2019-07-16.

[27] Google, "Introduction to tensorflow lite," https://www.tensorflow.org/mobile/tflite/, 2017, accessed: 2018-02-17.

[28] L. Chao, X. Peng, Z. Xu, and L. Zhang, "Ecosystem of things: Hardware, software, and architecture," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1563–1583, Aug 2019.

[29] K. sGroup, "The opencl specification - version 2.1," https://www.khronos.org/registry/OpenCL/specs/opencl-2.1.pdf, 2015, accessed: 2019-09-23.

[30] BVLC, "Caffe: a fast open framework for deep learning." https://github.com/BVLC/caffe, 2014, accessed: 2019-09-23.

[31] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[32] Z. Ning, F. Zhang, W. Shi, and W. Shi, "Position paper: Challenges towards securing hardware-assisted execution environments," in *Proceedings of the Hardware and Architectural Support for Security and Privacy*, ser. HASP '17. New York, NY, USA: ACM, 2017, pp. 6:1–6:8.

[33] H. Zhong, L. Pan, Q. Zhang, and J. Cui, "A new message authentication scheme for multiple devices in intelligent connected vehicles based on edge computing," *IEEE Access*, vol. 7, pp. 108 211–108 222, 2019.

[34] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, June 2016.

[35] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Ave: Autonomous vehicular edge computing framework with aco-based scheduling," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 10 660–10 675, Dec 2017.

[36] X. Li, Y. Dang, and T. Chen, "Vehicular edge cloud computing: Depressurize the intelligent vehicles onboard computational power," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 3421–3426.

[37] W. Zhang, Z. Zhang, and H. Chao, "Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 60–67, Dec 2017.

[38] Z. Zhou, H. Yu, C. Xu, Z. Chang, S. Mumtaz, and J. Rodriguez, "Begin: Big data enabled energy-efficient vehicular edge computing," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 82–89, December 2018.

[39] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537–1562, Aug 2019.

[40] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 15:1–15:13.

[41] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2004–2017, Sep. 2018.