CAVBench: A Benchmark Suite for Connected and Autonomous Vehicles

Yifan Wang*†§, Shaoshan Liu‡, Xiaopei Wu†, Weisong Shi†
*SKL of Computer Architecture, Institute of Computing Technology, CAS, Beijing, China
†Department of Computer Science, Wayne State University, Michigan, USA
‡PerceptIn, California, USA
§University of Chinese Academy of Sciences, Beijing, China
wangyifan2014@ict.ac.cn, shaoshan.liu@perceptin.io, {xiaopei.wu, weisong}@wayne.edu

***Abstract*—Connected and autonomous vehicles (CAVs) have recently attracted a significant amount of attention both from researchers and industry fields. Numerous studies targeting algorithms, software frameworks, and applications on the CAVs scenario have emerged. Meanwhile, several pioneer efforts have focused on the edge computing system and architecture design for the CAVs scenario and provided various heterogeneous platform prototypes for CAVs. However, a standard and comprehensive application benchmark for CAVs is missing, hindering the study of these emerging computing systems. To address this challenging problem, we present CAVBench, the first benchmark suite for the edge computing system in the CAVs scenario. CAVBench is comprised of six typical applications covering four dominate CAVs scenarios and takes four datasets as standard input. CAVBench provides quantitative evaluation results via application and system perspective output metrics. We perform a series of experiments and acquire three systemic characteristics of the applications in CAVBench. First, the operation intensity of the applications is polarized, which explains why heterogeneous hardware is important for a CAVs computing system. Second, all applications in CAVBench consume high memory accesses bandwidth, so the system should be equipped with high bandwidth memory or leverage good memory bandwidth management to avoid the performance degradation caused by memory bandwidth competition. Third, some applications have worse data/instruction locality based on the cache miss observation, so the computing system targeting these applications should optimize the cache architecture. Last, we use the CAVBench to evaluate a typical edge computing platform and present the quantitative and qualitative analysis of the benchmarking results.**

I. Introduction

With the rapid development of computer vision, deep learning, mobile communication and sensor technology, the functions of vehicles are no longer limited to driving and transportation, but have gradually become an intelligent, connected, and autonomous system. We refer to these advanced vehicles as connected and autonomous vehicles (CAVs). The evolution of the vehicles has given rise to numerous new application scenarios, such as Advanced Driver Assistance Systems (ADAS) or Autonomous Driving (AD) [1], [2], Internet of Vehicles (IoV) [3] and Intelligent Transportation Systems (ITS) [4], etc. Especially for ADAS/AD scenarios, many industry leaders have recently published their own autonomous driving systems, such as Google Waymo [5], Tesla Autopilot [6], and Baidu Apollo [7].

Under these scenarios, the CAVs system becomes a typical edge computing system [8], [9]. The CAVs computing system collects sensors data via the CAN bus and feeds the data to on-vehicle applications. In addition, the CAVs system is not isolated in the network, so the CAVs will communicate with cloud servers [10], Roadside Unit (RSU) and other CAVs to perform some computing tasks collaboratively. Much research focusing on the edge computing on CAVs from the application aspect have emerged [11]–[13]. There have also been some pioneer studies about exploring the computing architecture and systems for CAVs. NVIDIA®DRIVE™PX2 is an AI platform for autonomous driving that equips two discrete GPUs [14]. Liu et al. proposed their computing architecture for CAVs, which fully used hybrid heterogeneous hardware (GPUs, FPGAs, and ASICs) [15]. Unlike other computing scenarios, edge computing is still a booming computing domain. However, to date, a complete, dedicated benchmark suite to evaluate the edge computing platforms designed for CAVs is missing, both in academic and industrial fields. This makes it difficult for developers to quantify the performance of platforms running different on-vehicle applications, as well as to systematically optimize the computing architecture on CAVs or on-vehicle applications. To address these challenges, we propose CAVBench, the first benchmark suite for edge computing systems on CAVs.

CAVBench is a benchmark suite for CAVs computing system performance. It takes six diverse real-world on-vehicle applications as evaluation workloads, covering four applications scenarios summarized in OpenVDAP: autonomous driving, real-time diagnostics, in-vehicle infotainment and third-party applications [16]. The six applications that we have chosen are Simultaneous localization and mapping (SLAM), object detection, object tracking, battery diagnostics, speech recognition and edge video analysis. We collect four real-world datasets for CAVBench as the standard input to the six applications, which include three types of data: image, audio, and text. CAVBench also has two categories of output metrics. One is application perspective metrics, which includes the execution time breakdown for each application, helping developers find the performance bottleneck in the application side. Another is system perspective metrics, which we called the quality of service-resource utilization curve (QoS-RU curve). The QoS-RU curve can be used to calculate the Matching Factor between the application and the computing platform on CAVs. The QoS-RU curve can be considered as a quantitative performance index of the computing platform that helps researchers and developers optimize on-vehicle applications and CAVs computing architecture. Furthermore, we analyze the characteristics of the applications in CAVBench. We observe the application information and conclude three basic features of the applications on CAVs. First, the CAVs applications types are diverse, and the real-time applications are dominated in CAVs scenarios. Second, the input data of CAVs applications is mostly unstructured. Third, deep learning applications in CAVs scenarios prefer end-to-end models. Then, we comprehensively characterize the six applications in CAVBench via several experiments. On a typical state-of-the-practice edge computing platform, we have the following conclusions:

- The operation intensity of applications in CAVBench is polarized. The deep learning applications have higher floating point operation intensity because the neural networks are their main workloads, which includes plenty of floating point multiplications and additions. As for computer vision applications, the algorithms rely more on mathematical models, which contains lower floating point operation intensity. Hence, the computing platform in CAVs should contain heterogeneous hardware for the different applications.
- Similar to the traditional computing scenarios, the applications in CAVBench need high memory access bandwidth. That will cause competition for memory bandwidth when multiple applications are running concurrently in real environments.
- On average, the CAVBench has a lower cache and TLB miss rate, which means the applications in CAVs scenarios have good data/instruction locality, but for specific workloads, some applications have one or two higher miss rates. Thus, the computing systems targeting these applications should value the optimization of the cache architecture and the data/instruction locality to improve the application performance.

Finally, we use the CAVBench to evaluate a typical edge computing platform and present the quantitative and qualitative evaluation results of this platform.

The remainder of this paper is organized as follows. In Section II, we discuss the related work. Section III summarizes the methodology for designing CAVBench and presents the overview and detailed components of CAVBench. In Section IV, we analyze the characteristics of the applications in CAVBench from different views. The experimental evaluation results of CAVBench are illustrated in Section V. Finally, we conclude our work in Section VI.

II. Related Work

CAVBench is designed for evaluating the performance of the computing architecture and system of connected and autonomous vehicles. In this section, we summarize the related work from two aspects: the CAVs computing architecture and system, and the benchmark suite related to CAVs.

A. Architecture and System for CAVs

Junior [17] was the first work to introduce a complete system of self-driving vehicles, which included type and location of sensors, as well as software architecture design [18]. Junior presented dedicated and comprehensive information about applications and a software flow diagram for autonomous driving. However, Junior provided less information about the computing system on their self-driving vehicles.

Liu et al. proposed a computer architecture for autonomous vehicles which fully used hybrid heterogeneous hardware [15]. In this work, the applications for autonomous driving were divided into three stages: sensing, perception, and decision-making. They compared the performance of different hardware running basic autonomous driving tasks and concluded some rules to perform different tasks for dedicated heterogeneous hardware. Lin et al. explored the architectural constraints and acceleration of autonomous driving system in [19]. They presented a detailed comparison of accelerating related algorithms using heterogeneous platforms including GPUs, FPGAs, and ASICs. The evaluation metrics included running latency and power, which will help developers build an end-to-end autonomous driving system that meets all design constraints. OpenVDAP [16] proposed a vehicle computing unit, which contained a tasks scheduling framework and heterogeneous computing platform. The framework scheduled the tasks to specific acceleration hardware, according to task computing characteristics and hardware utilization. In the industrial field, there are several state-of-the-practice computing platforms designed for CAVs, such as NVIDIA®DRIVE™PX2 [14] and Xilinx®Zynq®UltraScale+™ZCU106 [20].

These projects can be regarded as pioneering research in exploring the computing architecture and systems for connected and autonomous vehicles from different aspects. However, the evaluation method of these systems lacks uniform standards; all the research groups chose application type and implementation from their perspectives. Hence, it is challenging to evaluate and compare these systems fairly and comprehensively.

B. Benchmark Suite Related to CAVs

There are many classic benchmark suites in the traditional computing field, such as BigDataBench [21] for big data computing, Parsec [22] for parallel computing and HPCC [23] for high-performance computing etc. However, for the computing scenario in CAVs, the benchmark research work is still at the beginning stage and can be divided into two categories according to their contents: datasets and workloads.

KITTI [1], [24] was the first benchmark datasets related to autonomous driving. It comprised rich stereo image data and 2D/3D object annotated data. According to different data types, it also provided the dedicated method to generate the ground truth and calculate the evaluation metrics. KITTI was built for evaluating the performance of algorithms in the autonomous driving scenario, including but not limited to optical flow estimation, visual odometry, and object detection. There are some customized benchmark datasets for each algorithm, such as TUM RGB-D [25] for RGB-D SLAM, PASCAL3D [26] for 3D object detection and the MOTChallenge benchmark [27], [28] for multi-target tracking. These kinds of benchmark suites will help us choose the implementations and datesets of CAVBench.

Another class of related benchmark suites used a set of computer vision kernels and applications to benchmark novel hardware architectures. SD-VBS [29] and MEVBench [30] both are system performance benchmark suites based on computer vision workloads in diversified fields. SD-VBS assembled 9 high-level vision applications and decomposed them into 28 common computer vision kernels. It also provided single-threaded C and MATLAB implementations of these kernels. MEVBench focused on a set of workloads related with visual recognition applications including feature extraction, feature classication, object detection and tracking, etc. MEVBench provided single- and multi-threaded C++ implementations for some of the vision kernels. However, these two benchmarks are prior works in the field, so they are not targeted toward heterogeneous platforms such as GPUs. SLAMBench [31] concentrated on using a complete RGB-D SLAM application to evaluate novel heterogeneous hardware. It chose KinectFusion [32] as the implementation and provided C++, OpenMP, OpenCL and CUDA versions of key function kernels for different platforms. The RGB-D cameras are more suitable for indoor environments and the workload type in SLAMBench is single. These efforts are a step in the right direction, but we still need a comprehensive benchmark which contains diverse workloads that cover varied application scenarios of CAVs to evaluate the CAVs system as we mentioned above.

III. Benchmark Design

The objective of developing CAVBench is to help developers determine if a given computing platform is competent for all CAVs scenarios. This section presents the methodology, overview, and components of our CAVBench.

A. Methodology and Overview

Combined with the survey and analysis of the related works on CAVs architectures and systems and existing benchmark suites, we present our methodology for designing CAVBench as shown in Figure 1. The computing and application scenarios of connected and autonomous vehicles are much different from the traditional domain. First, we investigate the typical application scenarios of CAVs. It is well-known that Advanced Driver-Assistant Systems (ADAS) and Autonomous Driving (AD) have already become a dominant application scenario of CAVs [1], [18], [33], [34]. In addition to ADAS/AD, OpenVDAP summarizes three other scenarios which are Real-time Diagnostics (RD), In-Vehicle Infotainment (IVI) and Third-Party Application (TApp) [16]. Thus, we focus on the exemplary and key applications in each dominant scenario.

The tasks in the ADAS/AD scenario can be divided into three stages according to their functions: sensing, perception, and decision-making [15]. Sensing tasks manage and calibrate the various sensors around the CAVs and provide reliable sensing data to upper-level tasks. Perception tasks take the sensing data as the input and output the surrounding information to the decision-making tasks, which in turn generate a safe and efficient action plan in real time. It can be seen that perception is an important connecting link between sensing and decision-making. The three main perception tasks are simultaneous localization and mapping (SLAM), object detection, and object tracking, which are all visual-based applications. Many studies take them as the vital parts in the autonomous driving pipeline [1], [19], [35]. Hence, we chose these three applications in the ADAS/AD scenario.

Vehicle system fault diagnostics and prognosis is important for keeping vehicles stable and safe [36]. With the development and widespread use of electric and

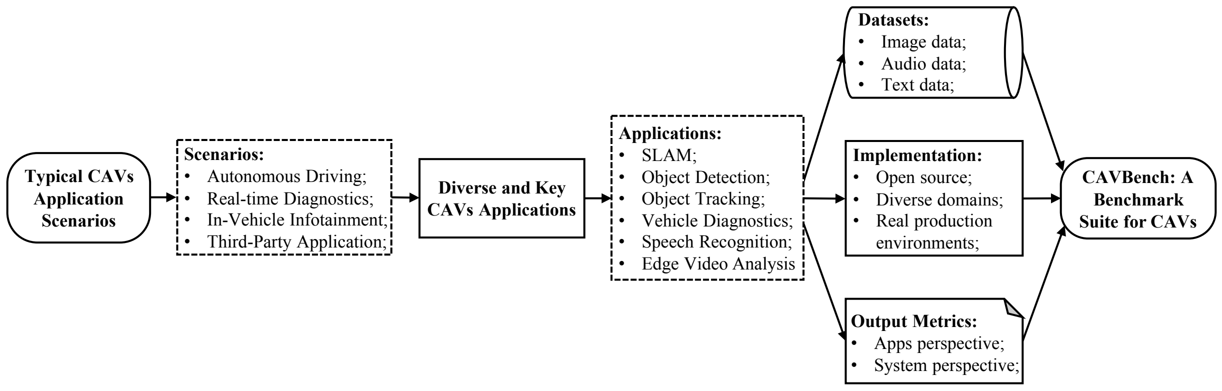


Fig. 1. CAVBench Methodology.

hybrids vehicles, the health monitoring and diagnostics of Li-ion batteries in these vehicles have received increasingly more attention [37]. It is extremely important to monitor and predict the battery status in a real-time fashion, including multiple parameters of each battery cell, e.g., voltage, current, temperature, and so on. Thus, battery diagnostics will be the application chosen in the RD scenario.

The In-Vehicle infotainment (IVI) scenario includes a wide range of applications that provide audio or video entertainment. Compared with manual interaction, the speech-based interaction method reduces the distraction of drivers and ensures driving safety [38], so motor companies have increasingly begun to develop their own IVI systems with speech recognition such as Ford® SYNC® [39]. We choose speech recognition applications for the IVI scenario.

There are some preliminary projects for the third-party application scenario. PreDriveID [40] is a driver identification application based on in-vehicle data. It can enhance vehicle safety by detecting whether the driver is registered or not through by analyzing how a driver operates a vehicle. A3 [41] is an edge video analysis application which uses a vehicle onboard camera to recognize targeted vehicles to enhance the AMBER alert system. It is easier to acquire the data from an onboard camera than the vehicle bus data, and edge video analysis could well be a killer application for edge computing [42], so we choose this kind of application for the TApp scenario.

After selecting the six applications, we pay attention to the implementation, data sets and output metrics for the applications. The implementation of each application should be state-of-the-art and representative, ensuring that it can be deployed in a real production environment. We provide real-world data sets for each application which are open source or collected by ourselves to let the applications have a standard input. To give the user a complete understanding of the benchmark results, the output metrics contain two categories: application perspective metrics and system perspective metrics. These three parts (implementation, data sets and output metrics) form the CAVBench and will be introduced in detail in the next three subsections, respectively.

B. Implementation

The implementation we chose in CAVBench is shown in Table I. The reasons for choosing these implementations are presented as follows.

1) SLAM: The simultaneous localization and mapping (SLAM) technique helps CAVs with real-time building a map of an unknown environment and localizing themselves in the map. ORB-SLAM2 provides a stereo SLAM method, which has been ranked as the top in KITTI benchmark datasets according to the accuracy and runtime. ORB-SLAM2 is more suitable for large environments than monocular [47] and RGB-D SLAM [48], so we chose it as the SLAM implementation. Figure 2 shows the ORB-SLAM2 pipeline. The stereo image stream is fed into the ORB extractor to detect feature points and generate descriptions of the extracted feature points. Then, the main thread attempts to match the current descriptions with the prior map point to localize and generate new keyframes. The local mapping thread manages keyframes to create new map point. The loop closing thread tries to detect and close trajectory loops via the last keyframe processed by the local mapping and creates the fourth thread to optimize a global map by the full Bundle Adjustment (BA).

2) Object Detection: The visual object category recognition and detection have always been a challenging problem in last decade [49]. In recent years, the series of algorithms based on Convolutional Neural Networks (CNNs) have become one of the mainstream techniques in the field of object detection [50], [51], especially in the autonomous driving scenario [52]. Single Shot multibox Detector (SSD) [43] and You Only Look Once (YOLO)

TABLE I
OVERVIEW OF IMPLEMENTATION IN CAVBENCH

Scenario	Application	App Type	Implementation	Main Workloads	Data Type	Data Source
ADAS/AD	SLAM	Real-Time	ORB-SLAM2 [40]	ORB Extractor and BA	Unstructured	Image (Stereo)
ADAS/AD	Object Detection	Real-Time	SSD [43]	CNNs	Unstructured	Image (Monocular)
ADAS/AD	Object Tracking	Real-Time	CIWT [44]	EKF and CRF Model	Unstructured	Image (Stereo)
RD	Battery Diagnostics	Offline	EVBattery	LSTM Networks	Semi-Structured	Text
IVI	Speech Recognition	Interactive	DeepSpeech [45]	RNNs	Unstructured	Audio
TApp	Edge Video Analysis	Interactive	OpenALPR [46]	LBP Feature Detector	Unstructured	Image (Monocular)

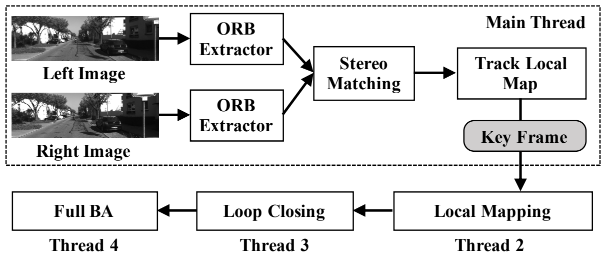


Fig. 2. Overview of the ORB-SLAM2 Pipeline.

[53] are kinds of end-to-end CNNs model. Compared with the R-CNN series [54], they do not hypothesize bounding boxes or resample pixels or features for these hypotheses, which improves the speed for detection and is as accurate as the R-CNN series. The network structure of SDD is shown in Figure 3. SSD uses multiple feature maps from the different stages of the network to perform detection at multiple scales, which is more accurate than the detection by one full connection layer in YOLO. In a word, SSD has higher accuracy and processing speed than other models; hence, we chose SSD as the implementation for object detection.

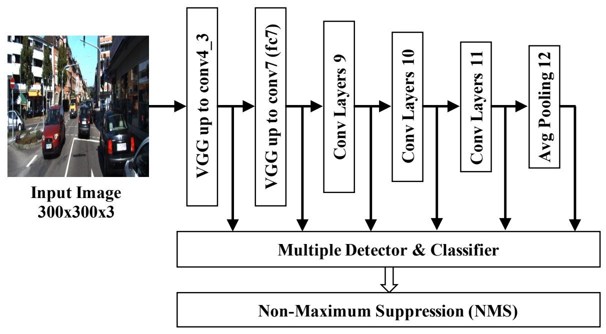


Fig. 3. Overview of the Single Shot MultiBox Detector Network Structure.

3) *Object Tracking:* The main goal of object tracking is to ensure that the vehicle does not collide with a moving object, whether a vehicle or a pedestrian crossing the road. We chose Combined Image- and World-Space Tracking (CIWT) [44] as the implementation for object tracking, which is an online multiple object tracking application dedicated to the autonomous driving scenario. In KITTI results, CIWT is not the most accurate, but it costs fewer computation resources and less processing time than some algorithms ranked ahead of it. It is more practical in the real-world environment than some offline [55] or single target tracking algorithms [56]. Figure 4 shows the overview of the CIWT pipeline. CIWT uses a stereo image stream to fuse the observation and estimate the egomotion. The observation fusion includes 2D detection and 3D proposals. The tracking process uses these results to generate tacking hypotheses through the Extended Kalman Filter (EKF) and uses the Conditional Random Field (CRF) model to select a high score tacking hypothesis.

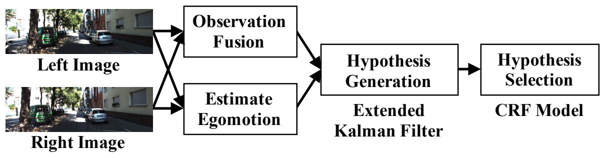


Fig. 4. Overview of the Combined Image- and World-Space Tracking pipeline.

4) *Battery Diagnostics:* The devices monitor and log data is a kind of temporal data. Recently, some works leverage Long-Short Term Memory (LSTM) networks to perform failure prediction according to log data for hard drives [57]. LSTM belongs to Recurrent Neural Networks (RNNs), but it has better performance on the long-term prediction task than RNNs. We use the similar method to process EV battery data. We call our implementation as EVBattery Diagnostics and the process is shown in Figure 5.

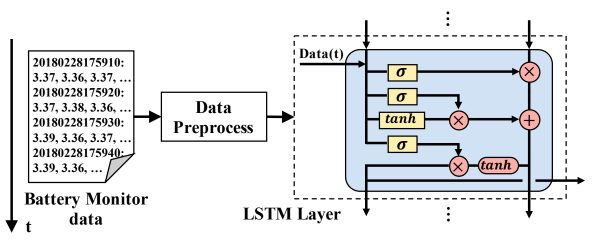


Fig. 5. Overview of the EVBattery Diagnostics.

5) Speech Recognition: Voice service generally consists of two steps: speech to text and text to intent, with the former being the process of speech recognition. DeepSpeech [45] is an end-to-end speech recognition algorithm based on RNNs. The deep learning method supersedes traditional processing stages in speech recognition systems, such as those that have been hand-engineered. Figure 6 shows the DeepSpeech network structure. The first three and the fifth layers in DeepSpeech have the basic NNs structure. The fourth layer is a bi-directional recurrent layer which is used to characterize the temporal correlation of the voice data. The evaluation results show that DeepSpeech has less latency and error rates than traditional methods based on Hidden Markov Model (HMM) [58], especially in noisy environments. Therefore, DeepSpeech is a suitable implementation for speech recognition.

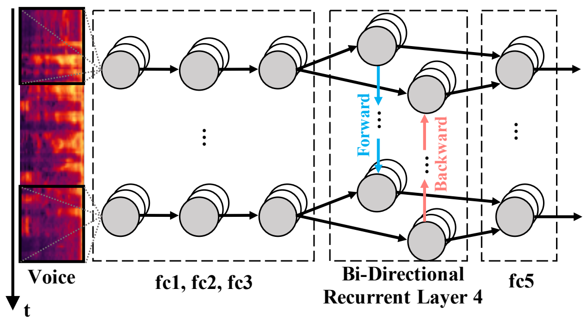


Fig. 6. Overview of the DeepSpeech Network Structure.

6) Edge Video Analysis: As we mentioned above, AMBER Alert Assistant (A3) is a typical edge video analysis application that takes OpenALPR [46] as the core workload to detect target vehicles in the video. The OpenALRP pipeline is shown in Figure 7. OpenALRP is a classic implementation of automatic license plate recognition, including several typical computer vision modules: LBP detector, image deskew and ORC. Therefore, we chose OpenALPR as the implementation of edge video analysis.

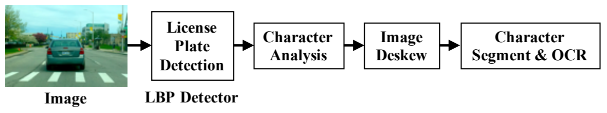


Fig. 7. Overview of the OpenALPR Pipeline.

C. Datasets

To guarantee that the evaluation results are similar to running in the real environment, we need to choose a real-world, not a synthetic, dataset for each application. Table II shows the basic information of the datasets we have chosen. It must be noted that we collected the datasets of Battery Diagnostics and Edge Video Analysis because of the lack of relative open source datasets or some datasets did not meet our real-world requirements.

TABLE II
THE SUMMARY OF DATASETS IN CAVBENCH

Application	Datasets	Data Size and Type
SLAM	KITTI VO/SLAM Datasets [24]	21 sequences stereo grayscale image
Object Detection	KITTI Object Detection Datasets [24]	7518 monocular color image
Object Tracking	KITTI Object Tracking Datasets [24]	28 sequences stereo color image
Battery Diagnostics	EV Battery Monitor Data	30 days battery monitor data, 160000 rows
Speech Recognition	Mozilla Corpus [59]	3995 valid human voice data
Edge Video Analysis	Images of Vehicles with License Plates	1000 monocular color image

1) KITTI Datasets [24]: As mentioned in Section II, KITTI provides rich, open source and real-world image data for different autonomous driving applications. The image characteristics varies, because each dataset focuses on one specific application. Each dataset contains various traffic scenes which can evaluate the application performance comprehensively.

2) EV Battery Monitor Data: There are few datasets that provide vehicle battery monitoring data. We collect the battery monitoring data of an electric vehicle in the real environment for one month. Each record of the data contains 60 items, such as voltage and temperature.

3) Mozilla Corpus [59]: The Mozilla corpus provides 3995 valid common voice files for testing. Each file is a record in which a person reads a common sentence, and records are collected by numerous people reading different sentences. It must be noted that the Mozilla corpus still has some limitations. It was collected in a daily environment, so it may not contain words that are used in the vehicular setting and may not have enough background noise that is likely to be very common in a vehicular environment.

4) Images of Vehicles with License Plates: The images in KITTI datasets could not meet the resolution requirement of performing license plate recognition, and some license plate datasets are not collected by a vehicle onboard camera. Thus, we use the Leopard® LI-USB30-AR023ZWDRB video camera [60] with a 25mm lens which was suggested by the Apollo project [7] to collect image data in real traffic scenes. Each image we provided contains at least one vehicle with its license plate.

D. *Output Metrics*

The Output metrics show quantitatively whether the given hardware platform can be used for CAVs scenarios or not. In CAVBench, the output metrics contain two parts: application perspective metrics and system perspective metrics.

1) Application Perspective Metric: Like some traditional benchmark suites, the application perspective metric shows the running time of each application. For computer vision applications (ORB-SALM2, CIWT, and OpenALPR), we output the average latency for each module in the applications, and we provide the average and tail latency for deep learning applications (SSD, EVBattery, and DeepSpeech). This metric helps developers optimize the platform in term of applications.

2) System Perspective Metric: For the system perspective metric, we call it the quality of service-resource utilization curve (QoS-RU curve). We evaluate the QoS of each application under different system resource allocations and draw the QoS-RU curve for each system resource (CPU utilization, memory footprint, and memory bandwidth, etc.). Figure 8 shows an example of the QoS-RU curve. We use the area under the curve of each system resource to calculate the Matching Factor between the application and the platform, indicating whether the platform is suitable for the CAVs application. Following is our approach to calculate the Matching Factor:

We denote the area under the curve of each system resource as A_i, and we take the weighted average of each area as the Matching Factor M, as Equation 1 shows.

$$M = \sum_{i=1}^{n}(w_i \cdot A_i) \tag{1}$$

The weight for each resource w_i can be calculated by Equation 2, in which the n is the number of system resources we considered.

$$w_i = (1 - A_i)/\sum_{i=1}^{n}(1 - A_i) \tag{2}$$

We notice that the w_i is the normalized $1 - A_i$. If the A_i is large, the resource i is relatively sufficient for the application. Similarly, if the A_i is small, the resource i has the potential to be the bottle-neck of the platform. Thus, the A_i and its weight have opposite values, which is why we chose the normalized $1 - A_i$ as the weight.

A more detailed explanation of the output metrics will be presented in Section V.

IV. Benchmark Characterization

In this section, we present the detailed description of the experiments of our CAVBench workload characterization analysis.

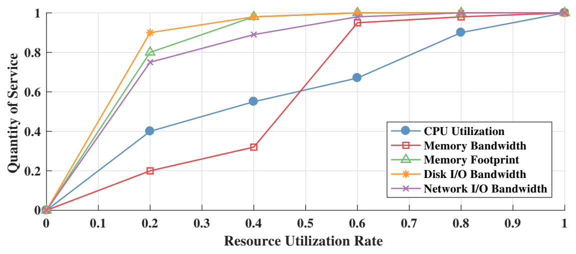


Fig. 8. An example of QoS-RU Curve.

A. *Overview*

Before we did the characterization analysis experiments, we first observed the information of the six applications presented in Table I to conclude some basic features of the applications in CAVs scenarios. The observations are described as follows:

1) Real-Time Applications are Dominant: The application types in the CAVs computing scenarios are diverse, including real-time, offline and interactive. This diversity corresponds to cloud computing and the big data computing domain [21]. In contrast to traditional computing fields, real-time applications are dominant in CAVs, and they all belong to the ADAS/AD scenario. Furthermore, the applications in other scenarios are offline and interactive. That explains why ADAS/AD applications always have the highest priority.

2) Unstructured Data Type: The input data type of CAVs applications is mostly unstructured data. As we mentioned above, CAVs is a typical embedded and edge computing system, and it deploys various sensors to collect information from the real physical world, and uses the information (data) to execute computation tasks. Therefore, the input data is generally unstructured, such as images and audios. Even for vehicle monitor data, it usually has little structural constraints, so we consider it as semi-structured data. As for cloud computing, there are still some classic workloads that use the structured data, for example, the relational query operation.

3) End-to-End Deep Learning Workloads: According to the classification of the main workloads in CAVBench, we find that the workloads in CAVs all belong to computer vision, machine learning, and the deep learning domain because the main functions of the applications in CAVs are detection, recognition, and prediction. In addition, due to the limitation of latency, deep learning workloads in CAVs choose end-to-end models, which means that with the exception of deep neural networks, there are no other processes between input and output; this improves the running speed while not decreasing the algorithm accuracy.

[Insights] These three observations can be regarded as the basic characteristics of the applications in the CAVs

TABLE III
EDGE COMPUTING PLATFORM CONFIGURATIONS

Platform	Intel Fog Node
CPU	Intel Xeon E3-1275 v5
Number of Sockets	1
Core(s) per Socket	4
Thread(s) per Core	2
Architecture	X86_64
L1d cache	4×32KB
L1i cache	4×32KB
L2 cache	4×256KB
L3 cache	8MB
Memory	32GB SODIMM 3122 MHz
Disk	256GB PCIe SSD

computing scenario. We can conclude some insights regarding the CAVs computing system and applications design. First, real-time applications have the highest priority in real production environments, so the CAVs system should contain a task scheduling framework to ensure that the real-time applications can be allocated with enough computing resources. Second, preprocessing the unstructured data consumes more time, so a hardware accelerator aimed at transforming the unstructured data to structured will be a benefit to the performance of the whole CAVs system. Third, the end-to-end deep learning algorithm reduces the number and frequency of data movements (main memory to GPUs memory), decreasing the processing latency. Thus, this kind of algorithm will be more suitable for CAVs applications.

B. *Experiments Configurations*

To obtain insights regarding application characteristics in CAVBench, we ran the six applications in a typical edge device, and use the Linux profiling tool Perf to collect the behaviors of the applications at the architecture level. We chose Intel® Fog Reference Design (FRD) as the experiment platform, which has one Xeon® E3-1275 v5 processor equipped with 32GB DDR4 memory and 256GB PCIe SSD. The processor has four physical cores, and hyper-threading is enabled. Other detailed information of the platform is shown in Table III. The operating system is Ubuntu 16.04 with Linux kernel 4.13.0. The deep learning applications are built on TensorFlow 1.5.0, and some visual modules are implemented on OpenCV 3.3.1. To acquire the pure and original characteristics of the applications, the platform is not equipped with heterogeneous devices. Each application executes for 500 seconds, sequentially processing different data.

C. *Operation Intensity*

First, we analyzed the operation intensity of CAVBench via the instruction breakdown of each application in CAVBench. As shown in Figure 9, the distribution of the instructions is diverse, even polarized. The difference is mainly due to floating point instruction (FL). The average proportion of floating point instruction in CAVBench is 20.77%, and the average ratio of integer instructions (Int) to FL is 24.44. However, for each application, the minimum and maximum FL proportion is 0.38% (CIWT) and 48.89% (SSD), and the minimum and maximum Int/FL ratio is 0.63 (SSD) and 79.59 (OpenALPR). In addition, the average FL proportion for BigDataBench, HPCC and Parsec are 2.12%, 24.11%, and 18.25%, respectively. The instruction distribution in CAVBench is similar to HPCC, Parsec according to the average values, but the distribution is also polarized when investigating specific applications in CAVBench. In contrast, the distribution of each workload in the traditional benchmark is comparable, such as the results presented in BigDataBench [21].

In order to characterize the computation behaviors, we calculated the ratio of computation to memory access for each application, which represents the integer and floating point operation intensity. As shown in Figure 10, the floating point operation (FLO) intensity of each application in CAVBench is still polarized, and the minimum and maximum values are 0.0079 (CIWT) and 5.46 (SSD),respectively, which differ by about three orders of magnitude. As for integer operation (IntO) intensity, the minimum and maximum values are 0.28 (CIWT) and 1.80 (SSD), and the IntO intensity for BigDataBench, HPCC and Parsec are 0.52, 0.43 and 1.50, respectively. Hence, the IntO intensity of CAVBench is almost in the same order of magnitude as those of the other benchmarks.

[Insights] According to operation intensity experiments, we can draw an important conclusion: the operation intensity of applications in the CAVs scenario is polarized. Deep learning applications, such as SSD and DeepSpeech, have higher floating point operation intensity, which is similar to the workloads in the high-performance computing domain. That is because the neural networks are the main workloads in deep learning applications, which includes a large number of matrix operations, causing plenty of point operation

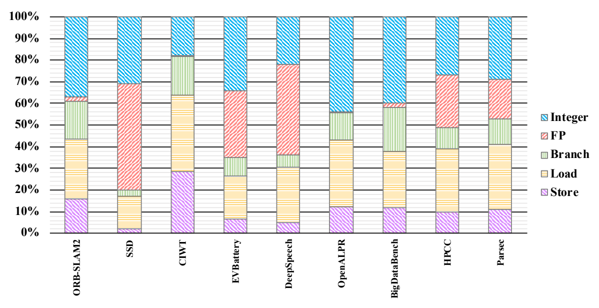


Fig. 9. Instruction Breakdown.

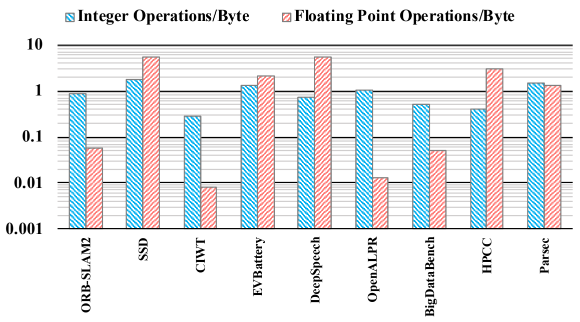


Fig. 10. Integer and Floating Point Operation Intensity.

multiplications and additions. As for computer vision applications, the algorithms rely more on mathematical modeling, and the computation is not so high; thus, they have lower floating point operation intensity. This phenomenon explains why modern CAVs computing platforms leverage heterogeneous hardware to accelerate some tasks. The state-of-the-practice CPUs provide SSE, AVX instructions for floating point operations, but the performance does not yet match that of the GPUs.

D. Memory Behavior

Memory is a very important part of the computer system; the memory wall problem exists in many computing domains. Therefore, we further investigated the memory behaviors of CAVBench. Our experiments included three parts: memory bandwidth, memory footprint, and cache behavior, which we discuss below:

Due to the restriction of the hardware function in the processor, we cannot monitor memory access bandwidth directly, so we use Perf tools to acquire the indirect memory access bandwidth. The measuring method will lead to some errors, but it can still help us analyze the memory access behaviors of each application in CAVBench.

The real-time memory access bandwidth of each application is shown in Figure 11. The black line is the memory access bandwidth upper limit of our experiment platform, whose average value is $10.98GB/s$. The applications in CAVBench sequentially process the data in the same size and type, so they all have stable memory access bandwidth, as shown in Figure 11. The minimum bandwidth is $1.01GB/s$ (EVBattery), which is 9.20% of the bandwidth upper limit, and the maximum value is $6.57GB/s$ (SSD), which is 59.84% of the upper limit. According to this observation, we can see that the applications in the CAVs scenario have a high memory access bandwidth, and the bandwidth may become the bottleneck of the performance in the real environment. When several CAVs applications run concurrently, they will compete for memory access bandwidth resources, leading to higher latency for each application. We observe that the memory access bandwidth of ORB-SLAM2 has four peak values during the running time, which reaches $7.21GB/s$ on average. This is due to the loop closing module performing full BA when a loop is detected in the trajectory. This kind of memory access burst will cause interference in the performance of other applications, especially the tail latency.

Furthermore, we investigated the resident memory (memory footprint) behavior of CAVBench. The experiment results are shown in Figure 12. Except for ORB-SLAM2, other applications have stable memory footprint. As we mentioned above, the experiment platform total memory is $32GB$. With the exception of ORB-SLAM2, the lowest memory footprint is $0.061GB$ (OpenALPR), which is 0.19% of total memory, and the highest memory footprint is $1.17GB$ (DeepSpeech), which is 3.66% of total memory. The reason for the continued increment of ORB-SLAM2 memory footprint is the application continues to generate new map points and the point data stores in the memory. The four jumps of the memory footprint is also caused by the full BA, which corresponds to the observation in memory access

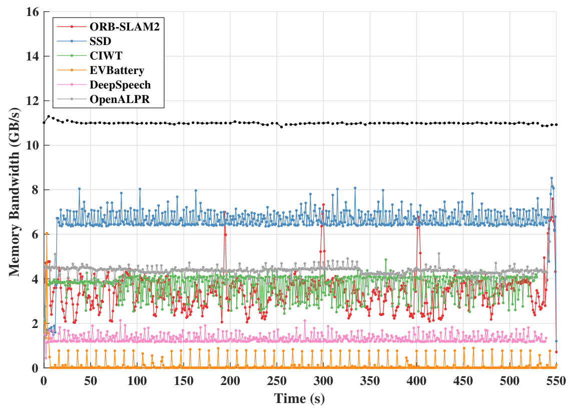


Fig. 11. Memory Access Bandwidth Behaviors.

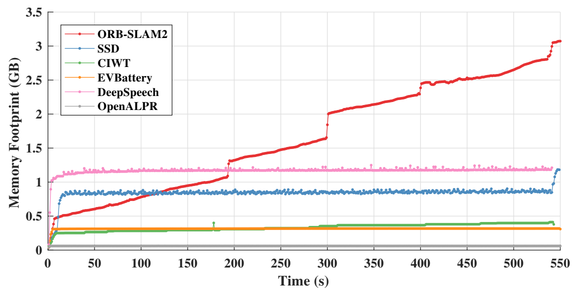
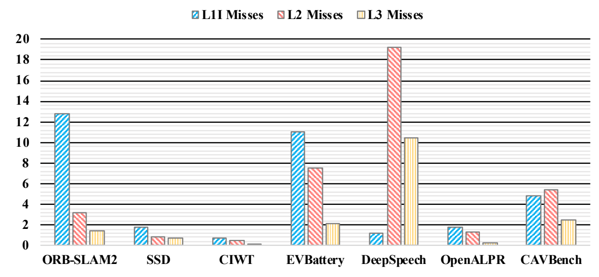
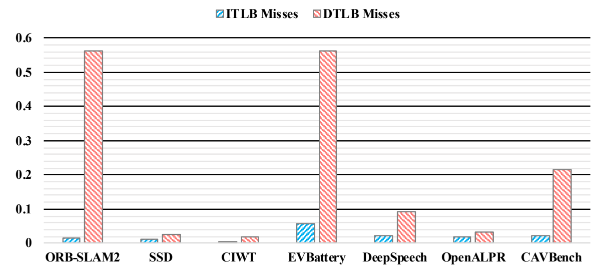


Fig. 12. Memory Footprint Behaviors.



(a) Cache Behaviors



(b) TLB Behaviors

Fig. 13. Cache and TLB Behaviors.

bandwidth experiment. The maximum memory footprint of ORB-SLAM2 is $3.07GB$ (9.59%). We can conclude that the applications in the CAVs scenario consume less memory footprint, and the large capacity memory is available for the edge computing platform, so the CAVs application performance will not be constrained by the memory footprint.

Finally, we investigated the cache behaviors of CAVBench to see that the memory hierarchy architecture in the state-of-the-practice edge platform is proper for CAVs applications. The cache behavior and TLB behavior of each application are shown in Figure 13. The CAVBench average L1i, L2 and L3 cache MPKI (Misses Per Kilo Instructions) are 4.86, 5.44, and 2.51, respectively, which are almost in the same order of magnitude as HPCC (0.41, 5.59, and 4.22) and Parsec (3.51, 7.25, and 3.37), respectively. The TLB behavior of CAVBench is also similar to HPCC and Parsec. According to this observation, we find that the applications in CAVBench have good data and instruction locality. These characteristics differ from the big data computing, which has a huge code size and deep software stack leading to higher MPKI.

Focusing on specific applications, ORB-SLAM2, EVBattery, and DeepSpeech have a higher MPKI than the other three. The ORB-SLAMS have high L1i MPKI and DTLB MPKI. We infer that this phenomenon is still caused by the periodic loop detection operation and irregular full BA operation. The loop closing module queries the local map database periodically to detect the potential trajectory loop. The DLTB has less capacity to store all the page tables of the local map database, causing the high DTLB miss rate, and the irregular full BA operation interferes with the instruction locality, increasing the L1i cache miss rate. As for EVBattery and DeepSpeech, we infer that the RNNs structure leads to a high cache miss rate. The convolution operations in CNNs make the data and instruction localized, which is why SSD does not have a high cache miss rate, but the RNNs do not have such convolution operations.

[Insights] According to the memory experiments, we can draw some important conclusions: First, applications in the CAVs scenario consume high memory access bandwidth, which will be a performance bottleneck when multiple applications run concurrently in real environments. Second, the memory footprint of each application takes a very low proportion of the total memory in the state-of-the-practice edge computing platform. Third, on average, the applications in the CAVs scenario have good data and instruction locality. This characteristic is similar to the workload in high-performance computing and the parallel computing domain. As for specific applications, the SLAM and RNNs model based applications have a higher probability to increase the cache and TLB miss rate. Therefore, with the CAVs computing system paying more attention to these applications, we should focus on the optimization of the cache architecture and the application data/instruction locality.

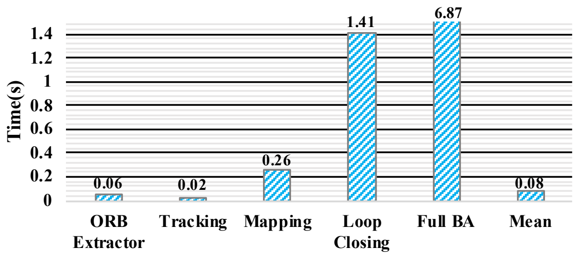
V. Evaluation

We use the CAVBench to evaluate our typical edge computing platform; its configurations are presented in Section IV. The evaluation results are presented in this section.

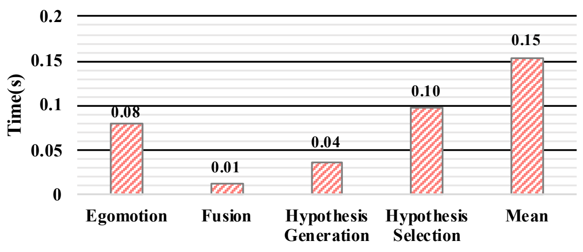
A. Latency Results

First, we present the application perspective metrics (latency) of the platform in Figure 14. According to the results, we find this platform has a good performance in terms of the computer vision applications; the average FPS (Frame Per Second) of ORB-SLAM2, CIWT and OpenALPR are 12.5, 6.67 and 8.83, respectively. The processing speed of these applications is near-acceptable in the real environment. Please note that the latency of Mapping, Loop Closing, and Full BA is quite high, but these modules are not executed for each frame, and they run in different threads, so the latency of these functions is negligible to the average latency.

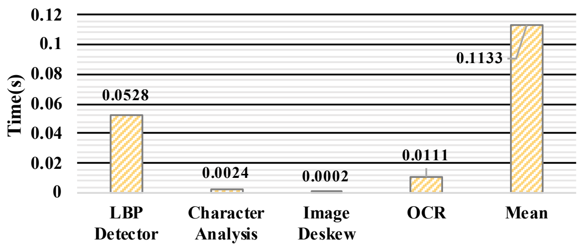
However, the performance of deep learning applications is not as good as computer vision applications. The FPS of SSD is only 2.55, which is unacceptable in the real autonomous driving scenario, and the average latency of EVBattery and DeepSpeech are $0.21s$ and $3.74s$, respectively. The latency of DeepSpeech is much



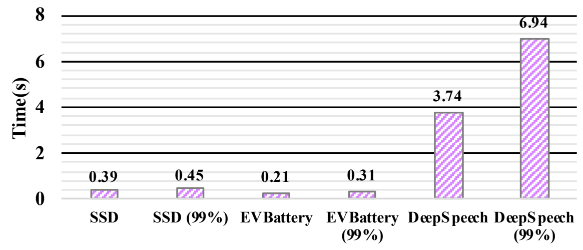
(a) ORB-SLAM2



(b) CIWT



(c) OpenALPR



(d) Deep Learning Applications

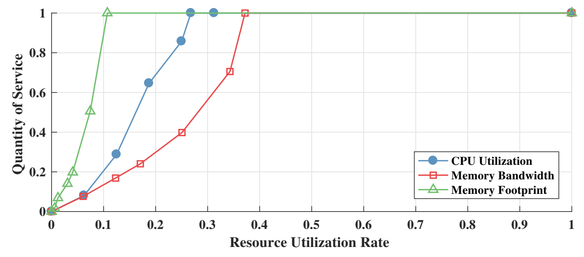
Fig. 14. Application Latency Results on Intel Fog Node.

longer than user-perceived QoS in an interactive system. Because the input data source of EVBattery is text, the data batch size is only one-hour monitor data, and the neural networks scale is smaller (one layer LSTM), so it has less running latency, which is enough for a real environment. As for SSD and DeepSpeech, they both have a deep and large-scale network structure, and the input data source is image and audio (Unstructured data). The platform is incompetent for these kinds of applications since it is not equipped with heterogeneous hardware to acceleration.

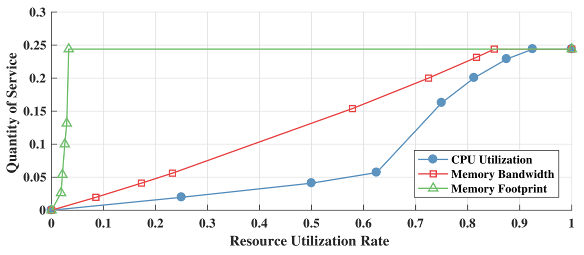
B. *QoS-RU Curve Case Study*

The application perspective metrics give the user an overview of the platform performance when running different applications. Furthermore, CAVBench provides the system perspective metrics (QoS-RU) to calculate a quantitative benchmarking result. We take ORB-SLAM2 and SSD as the case study. According to the above results, we set 10 FPS as the best QoS (QoS=1). The QoS-RU curves of ORB-SLAM2 and SSD are shown in Figure 15. As for ORB-SLAM2, the area under CPU utilization (CPU), memory bandwidth (MEMBAND) and memory footprint (MEMFOOT) curve are 0.84, 0.75, and 0.93, respectively. The Matching Factor (MF) is 0.80. In SSD case, the area under CPU, MEMBAND, and MEMFOOT curves are 0.09, 0.14, and 0.23, respectively, and the MF is 0.15. The results correspond to the application perspective metrics, but it is from the system view and quantitative.

No doubt, if an application cannot reach the acceptable QoS, the MF will be poor. Meanwhile, if the application needs more system resources to reach high QoS, the MF will decrease more. In the real production environments, the on-vehicle applications run concurrently in one system and compete for the system's resources. Hence, the high QoS application with less system resource consumption will be preferable to CAVs computing systems. That is why we consider system resource utilization when calculating the Match Factor.



(a) ORB-SLAM2



(b) SSD

Fig. 15. QoS-RU Curves Case Study of ORB-SLAM2 and SSD.

VI. Conclusion

CAVBench is the first benchmark suite for computing system and architectures designed for connected and autonomous vehicles targeting computational performance evaluation. We chose four typical and dominate application scenarios of CAVs, and summarized six applications in these scenarios as the evaluation applications. After that, we collected state-of-the-art implementation and standard input datasets for each application and determined the output metrics of the CAVBench. We got three basic features from CAVBench. First, the real-time applications are dominated in CAVs scenarios. Second, the input data is mostly unstructured. Third, the end-to-end deep learning algorithm is more preferable for CAVs computing system. Then, we ran a series experiments to explore the characteristics of CAVBench, and concluded three observations as follows. First, the operation intensity of the applications in CAVBench is polarized. Second, the applications in CAVBench all consume high memory access bandwidth. Third, CAVBench has a lower cache miss rate on average, but for specific applications, the optimization of the cache architecture and data/instruction locality is still important. According to these features and characteristics, we presented some insights and suggestions about the CAVs computing system design or CAVs application implementation. Finally, we used the CAVBench to evaluate a typical edge computing platform and presented the quantitative and qualitative analysis of the benchmarking results.

We hope this work will be helpful to researchers and developers who target the computing system or architecture design of connected and autonomous vehicles. According to the insights proposed in this paper, our future work will proceed from the following aspects. First, we will focus on providing the CUDA and OpenCL implementation of the CAVBench to support more heterogeneous platforms. Second, we will explore more methodologies to evaluate the computing system in the CAVs scenarios comprehensively, such as a benchmark dedicating system memory behaviors. Third, we will implement a full stack computing system for CAVs that will be competent for all CAVs applications.

Acknowledgment

The authors are very grateful to the reviewers and our shepherd Fan Bai for their constructive comments and suggestions. We also would like to thank our colleague Yongtao Yao, who provided the implementation of some applications in CAVBench. This work is supported in part by National Science Foundation (NSF) grant CNS-1741635.

References

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3354–3361.

[2] SAE International., "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles J3016," https://www.sae.org/standards/content/j3016_201609/, 2016.

[3] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 241–246.

[4] G. Dimitrakopoulos and P. Demestichas, "Intelligent transportation systems," *IEEE Vehicular Technology Magazine*, vol. 5, no. 1, pp. 77–84, 2010.

[5] Waymo, "Waymo Self-Driving Car." https://waymo.com, 2018.

[6] Tesla, "Tesla Autopilot: Full Self-Driving Hardware on All Cars." https://www.tesla.com/autopilot, 2018.

[7] Baidu, "Apollo Open Platform," http://apollo.auto/index.html, 2018.

[8] 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, 2016.

[9] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[10] S. Liu, J. Tang, C. Wang, Q. Wang, and J.-L. Gaudiot, "A unified cloud platform for autonomous driving," *Computer*, vol. 50, no. 12, pp. 42–49, 2017.

[11] G. Kar, S. Jain, M. Gruteser, F. Bai, and R. Govindan, "Real-time traffic estimation at vehicular edge nodes," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 3.

[12] 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*. ACM, 2017, p. 4.

[13] B. Qi, L. Kang, and S. Banerjee, "A vehicle-based edge computing platform for transit and human mobility analytics," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 1.

[14] NVIDIA Corporation, "NVIDIA DRIVE PX2: Scalable AI platform for Autonomous Driving," https://www.nvidia.com/en-us/self-driving-cars/drive-platform, 2018.

[15] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017.

[16] 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 *Distributed Computing Systems (ICDCS), 2018 IEEE 38th International Conference on*. IEEE, 2018.

[17] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The Stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[18] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 163–168.

[19] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 751–766.

[20] Xilinx Inc., "Xilinx Zynq Ultrascale+ MPSoC ZCU106 Evaluation Kit," https://www.xilinx.com/products/boards-and-kits/zcu106.html, 2018.

[21] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, "BigDataBench: A big data benchmark suite from internet services," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. IEEE, 2014, pp. 488–499.

[22] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[23] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC Challenge (HPCC) benchmark suite," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. Citeseer, 2006, p. 213.

[24] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[25] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 573–580.

[26] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond PASCAL: A benchmark for 3D object detection in the wild," in *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*. IEEE, 2014, pp. 75–82.

[27] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015: Towards a benchmark for multi-target tracking," *arXiv preprint arXiv:1504.01942*, 2015.

[28] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A benchmark for multi-object tracking," *arXiv preprint arXiv:1603.00831*, 2016.

[29] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "SD-VBS: The San Diego vision benchmark suite," in *Workload Characterization (IISWC), 2009 IEEE International Symposium on*. IEEE, 2009, pp. 55–64.

[30] J. Clemons, H. Zhu, S. Savarese, and T. Austin, "MEVBench: A mobile computer vision benchmarking suite," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 91–102.

[31] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. Kelly, A. J. Davison, M. Luján, M. F. O'Boyle, G. Riley *et al.*, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for slam," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5783–5790.

[32] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.

[33] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[34] C. Berger and B. Rumpe, "Autonomous driving-5 years after the urban challenge: The anticipatory vehicle as a cyber-physical system," *arXiv preprint arXiv:1409.0413*, 2014.

[35] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[36] Y. Zhang, G. W. Gantt, M. J. Rychlinski, R. M. Edwards, J. J. Correia, and C. E. Wolf, "Connected vehicle diagnostics and prognostics, concept, and initial practice," *IEEE Transactions on Reliability*, vol. 58, no. 2, pp. 286–294, 2009.

[37] J. Zhang and J. Lee, "A review on prognostics and health monitoring of li-ion battery," *Journal of Power Sources*, vol. 196, no. 15, pp. 6007–6014, 2011.

[38] J. Maciej and M. Vollrath, "Comparison of manual vs. speech-based interaction with in-vehicle information systems," *Accident Analysis & Prevention*, vol. 41, no. 5, pp. 924–930, 2009.

[39] Ford, "SYNC," https://www.ford.com/technology/sync/, 2018.

[40] G. Kar, S. Jain, M. Gruteser, J. Chen, F. Bai, and R. Govindan, "PredriveID: Pre-trip driver identification from in-vehicle data," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, pp. 2:1–2:12.

[41] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Enhancing AMBER alert using collaborative edges: poster," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 27.

[42] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[43] 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.

[44] A. Osep, W. Mehner, M. Mathias, and B. Leibe, "Combined image-and world-space tracking in traffic scenes," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1988–1995.

[45] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep Speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[46] OpenALPR Technology Inc., "OpenALPR," http://www.openalpr.com, 2018.

[47] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[48] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D mapping with an RGB-D camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[49] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[50] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[51] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *European Conference on Computer Vision*. Springer, 2016, pp. 354–370.

[52] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3D object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2147–2156.

[53] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[54] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[55] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun, "3d traffic scene understanding from movable platforms," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 5, pp. 1012–1025, 2014.

[56] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 749–765.

[57] F. D. dos Santos Lima, G. M. R. Amaral, L. G. de Moura Leite, J. P. P. Gomes, and J. de Castro Machado, "Predicting failures in hard drives with lstm networks," in *Intelligent Systems (BRACIS), 2017 Brazilian Conference on*. IEEE, 2017, pp. 222–227.

[58] A. L. Maas, A. Y. Hannun, C. T. Lengerich, P. Qi, D. Jurafsky, and A. Y. Ng, "Increasing deep neural network acoustic model size for large vocabulary continuous speech recognition," *arXiv preprint*, 2014.

[59] Mozilla, "Mozilla Corpus," https://voice.mozilla.org/en/data, 2018.

[60] Leopard Imaging Inc., "USB 3.0 Box Camera: AR023ZWDRB," https://leopardimaging.com/product/li-usb30-ar023zwdrb/, 2018.