SafeShareRide: Edge-based Attack Detection in Ridesharing Services

Liangkai Liu*, Xingzhou Zhang*†, Mu Qiao‡ and Weisong Shi*
*Department of Computer Science, Wayne State University, Detroit, MI, USA, 48202
†Institute of Computing Technology, Chinese Academy of Sciences, UCAS, Beijing, China, 100190
‡IBM Research - Almaden, San Jose, CA, USA, 95120
{liangkai, weisong}@wayne.edu, zhangxingzhou@ict.ac.cn, mqiao@us.ibm.com

***Abstract*—Ridesharing services, such as Uber and Didi, are enjoying great popularity; however, a big challenge remains in guaranteeing the safety of passenger and driver. State-of-the-art work has primarily adopted the cloud model, where data collected through end devices on vehicles are uploaded to and processed in the cloud. However, data such as video can be too large to be uploaded onto the cloud in real time. When a vehicle is moving, the network communication can become unstable, leading to high latency for data uploading. In addition, the cost of huge data transfer and storage is a big concern from a business point of view. As edge computing enables more powerful computing end devices, it is possible to design a latency-guaranteed framework to ensure in-vehicle safety. In this paper, we propose an edge-based attack detection in ridesharing services, namely *SafeShareRide*, which can detect dangerous events happening in the vehicle in near real time. *SafeShareRide* is implemented on both drivers' and passengers' smartphones. The detection of *SafeShareRide* consists of three stages: speech recognition, driving behavior detection, and video capture and analysis. Abnormal events detected during the stages of speech recognition or driving behavior detection will trigger the video capture and analysis in the third stage. The video data processing is also redesigned: video compression is conducted at the edge to save upload bandwidth while video analysis is conducted in the cloud. We implement the *SafeShareRide* system by leveraging open source algorithms. Our experiments include a performance comparison between *SafeShareRide* and other edge-based and cloud-based approaches, CPU usage and memory usage of each detection stage, and a performance comparison between stationary and moving scenarios. Finally, we summarize several insights into smartphone based edge computing systems.**

I. Introduction

As ridesharing services, such as Uber in the US and Didi in China, have become increasingly popular, in-vehicle safety has become a critical issue. Two kinds of attacks may happen in vehicles [1]–[3]: drivers being attacked by passengers or passengers being attacked by drivers. For example, the driver may be kidnapped by the passenger or the passenger may encounter a crazy driver threatening her life. As millions of rides take place through ride sharing services daily, it is important to guarantee their safety. To tackle this issue, Didi has applied facial recognition, itinerary sharing, SOS calling, privacy numbers, driver verification and a safe driving system to provide ride safety [4]–[6]. However, there are still several open problems. For example, when the passenger or driver is being attacked, it may be impossible for her to press the SOS button or share the itinerary. In addition, facial recognition and driver verification are usually only conducted once when a driver first registers the share ride vehicle. Furthermore, even though real-time GPS data can be used in a safe driving system to detect risky driving behavior, dangerous scenarios can still occur in vehicles with normal driving trajectories. Therefore, much more effort is required to ensure in-vehicle safety for ridesharing services.

In addition, the large scale of rides also poses significant technical challenges for cloud based infrastructure. State-of-the-art safe detection systems are usually implemented in the cloud [1], [5]. Take uploading in-vehicle video data to the cloud as an example. The total amount of data that needs to be sent to the cloud every day can easily reach PB scale. Therefore, it can be extremely difficult for a pure cloud-based approach to keep track of millions of rides and detect abnormal scenarios in real time. Additionally, as the vehicle is moving at fast speed, the wireless channel for applications on a vehicle may become unstable [7]. Therefore, it may take a much longer time to upload data to the cloud, let alone doing further analysis and performing detection. If the latency of the anomaly detection is high, it is unlikely to guarantee safety. Therefore, a pure cloud-based safe driving system is not sufficient.

With the rise of edge computing, end devices now enable more powerful computing [8]–[10]. Meanwhile, the computation of mobile devices like smartphones has become increasingly more powerful. It is feasible to design and implement an attack detection platform on mobile devices using edge computing. In this paper, we propose an edge-based three-stage attack detection framework, namely *SafeShareRide*, which aims to ensure the safety of share rides. The first stage uses speech recognition to detect keywords such as "help" or a loud quarrel during a ride. The second stage is driving

behavior detection. It collects driving data from an on-board diagnostics (OBD) adapter and smartphone sensors, and detects abnormal driving behaviors exhibited through speed, acceleration and the angular rate. The third stage is analyzing in-vehicle video recordings to determine whether there is an emergency. At the beginning of each detection period, the first two stages are running independently to capture in-vehicle danger. When the speech recognition recognizes a cry for help or the driving behavior detection discovers dangerous driving behaviors, video capture and analysis will be automatically activated to process the in-vehicle video of the current detection period. The detection results from the first two stages, and the extracted video will be sent to the cloud or edge server. Through this three-stage detection, *SafeShareRide* can provide highly accurate detection with very low bandwidth demand from video uploading. The contributions of this paper are as follows:

- An edge-based attack detection service, *SafeShareRide*, is proposed to address safety concerns in share rides. *SafeShareRide* leverages smartphones as the edge computing platform and consists of three stages: speech recognition, driving behavior detection, and video capture and analysis.
- *SafeShareRide* adopts an edge cloud collaborative approach which can be summarized as a general approach for edge enabled applications.
- We implemented *SafeShareRide* as an android based application and evaluated its CPU usage, memory usage, bandwidth and latency. The performance of the application under stationary and moving scenarios was compared. We obtained four insights for edge-based applications.

The remainder of this paper is organized as follows. We present the background and motivation in Section II. Section III introduces the design of *SafeShareRide*. We discuss preliminary experiments and observations in Section IV. The system implementation is discussed in Section V. We present the evaluation of *SafeShareRide* in Section VI. In Section VII, we discuss the limitations and future work of the paper. Related work is discussed in Section VIII. We finally conclude in Section IX.

II. Background and Motivation

Although ridesharing services have become increasingly popular, in-vehicle safety remains a big challenge for companies like Uber and Didi. There have been many news reports about attacks in ridesharing services. In this section, we will discuss the background and motivation of *SafeShareRide* in two aspects: 1) why we need to use edge computing, and 2) why we choose the smartphone as the edge computing platform.

A. Edge Computing

Cloud computing can provide powerful computational capability, but it requires data to be uploaded. Edge computing has been proposed to enable computation at the edge of the Internet, where the data is also produced.

There are two main reasons why we need to use edge computing rather than cloud computing. First, there is too much data to be uploaded in the cloud based approach. According to the statistics from [11], about 45,787 rides took place through Uber every minute in 2017. The size of a one-minute 720P video can be as large as 100MB. Assuming the average ride duration is about 20 minutes, the total amount of data that needs to be sent to the cloud every day is around 9.23 PB. Uploading such a huge amount of data bears a high cost. Second, the latency for uploading data in moving scenarios can be much greater than in stationary scenarios. According to the experiments in [10], [12], 4G/LTE may be unstable for moving vehicles. The data loss rate can increase dramatically as the vehicle's speed increases. The latency of uploading data onto the cloud can be affected by many factors such as speed, weather conditions, and the locations of base stations. Cloud computing may not be a robust approach in a moving scenario. Therefore, it is suitable to use edge computing for in-vehicle safety detection [13].

B. Smartphone as the Edge Computing Platform

In edge-based attack detection, the choice of the computing platform is important in the entire system design. There are many options, such as raspberry pi, smartphone, pad, on-board devices including Nvidia Jetson TX2 and Intel Fog. As most detections are designed to work at the edge, we need to consider availability, performance and cost of the edge device, when choosing the computing platform.

Our reason for choosing the smartphone as the edge computing platform is multifold. First, smartphones, as the most popular mobile devices, are accessible to both passengers and drivers. Second, ride sharing services or apps are widely available on smartphones. It can be easier to use a smartphone to examine the start and end of a ride. Third, there are many machine learning libraries that can run on the smartphone, like TensorFlow Lite [14] for Android phone and Core ML [15] for iPhone. The model inference time can be significantly reduced.

III. SafeShareRide Design

In this section, we first use an example to illustrate *SafeShareRide*, followed by a detailed discussion of each detection stage. We will discuss the algorithms and models of each detection stage and then introduce the application framework of *SafeShareRide*.

A. Example

Peter calls a ridesharing service through his smartphone. When he gets into the car, an app on his phone is launched to collect audio, driving and video data to detect abnormal events like kidnapping, fighting and quarreling. Meanwhile, services running on the driver's phone are also automatically activated to detect abnormal scenarios for the safety of the driver. When emergencies are detected, the related real-time video, the car information as well as the location will be sent to the cloud. In addition, a video link will be sent to law enforcement.

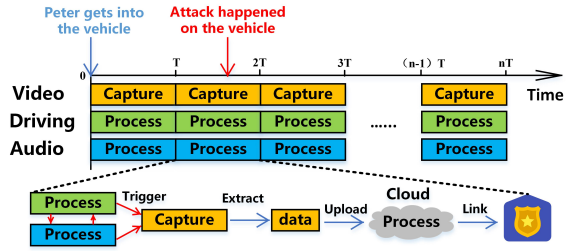

Fig. 1. The time series of *SafeShareRide*.

The time series of *SafeShareRide* is illustrated in Figure 1. When Peter gets into the car, all three stages of detection are launched on his smartphone as well as the driver's. For every detection period T, the program runs on both sides to collect the audio and driving data, and process them to determine unsafe events. In order to reduce the demand for computation and storage resources, a trigger mechanism is implemented between these stages. If no emergency is detected, the captured video data will be abandoned and no computational analysis is performed on that video. When certain unsafe events are detected by speech recognition or driving behavior detection, the video capture and analyzing process will be triggered to extract related video clips from the captured video. The compressed video data will then be sent to the cloud. Video analysis will be conducted in the cloud to further examine in-vehicle safety. The video clip, along with the contextual information, such as location, time, and vehicle information, will be automatically shared with the local police station.

B. Speech Recognition

Speech recognition is the first stage in *SafeShareRide*. It is designed to detect abnormal audio, such as keywords like "help" [16], [17], and abnormal high pitched sounds such as screaming and loud quarrelling. Specifically, we leverage an open source speech recognition project named CMUSphinx [18] for keyword detection.

The speech recognition model used in *SafeShareRide* is shown in Figure 2. The model is based on Hidden Markov Model (HMM) which has been widely used for

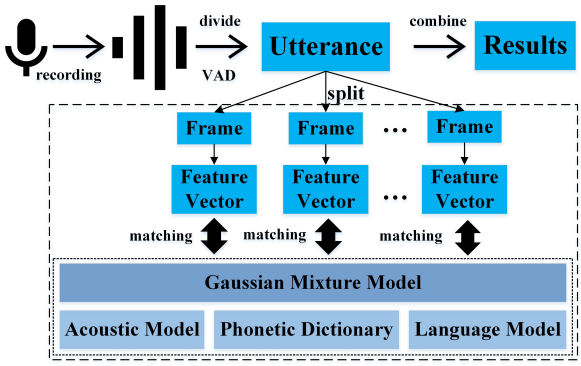

Fig. 2. The speech recognition in *SafeShareRide*.

speech decoding [19]. First, the microphone records the audio every T seconds(e.g., 20.0s) and gets a waveform. Voice Active Detection (VAD) is then used to delete the silence in the front and back of the waveform. The waveform is divided into utterances. Each utterance is further split into a multitude of speech frames. For each frame, typically 10 milliseconds in length, a feature vector is extracted to represent the speech frame. The feature vector is then used in speech frame matching and scream detection. Gaussian Mixture Model (GMM) is applied to detect screaming and loud quarrelling [20]. Three more models are used to match the speech to words. Specifically, the acoustic model calculates the acoustic properties for each frame. The phonetic dictionary provides the mapping between speech frames and words. The language model restricts the word search by giving the possibility of word sequences. The goal of matching process is to choose the best matching combination.

C. Driving Behavior Detection

Driving behavior detection is designed to detect dangerous driving behaviors. We define three dangerous driving behaviors in *SafeShareRide*: drunk driving, speeding and distracted driving [21]. They are also the three biggest causes of fatalities on the road [1].

The data used for detection is collected from the OBD adapter, sensors on the phone and GPS. *SafeShareRide* leverages the Bluetooth/WiFi communication between a smartphone and the OBD adapter to get the driving speed, longitude, and latitude, among others.

The driving behavior detection in *SafeShareRide* is shown in Figure 3. According to the evaluation results of the edge-based approach and the cloud-based approach [22], it is more efficient to train the model in the cloud, deploy the model and do inference at the edge. For the driving behavior detection in *SafeShareRide*, the inference on the data from OBD is conducted in near

[1]http://www.nsc.org/Pages/nsc-on-the-road.aspx

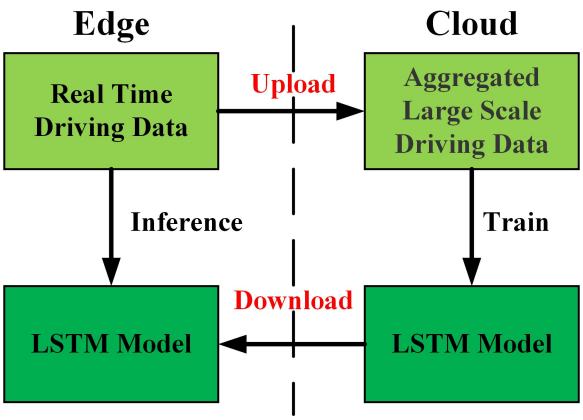

Fig. 3. The driving behavior detection in *SafeShareRide*.

real time, while the upload of large data set to the cloud and the download of the detection model to the edge only happens when idling.

To detect speeding, we compare the driving speed with the road speed limits obtained from the car's navigation system. The detection of distracted driving and drunk driving is based on the Long Short-Term Memory (LSTM) model [23]. In *SafeShareRide*, we collect driving data from the OBD adapter and then train an LSTM model on this data set to distinguish normal driving behavior from abnormal driving behavior [24]. The trained model is then deployed on smartphones. For each detection process, the collected driving data will be divided into overlapping sliding windows according to their timestamps, such as, 0.0-5.0s, 1.0-6.0s, and 2.0-7.0s. The sliding window driving data is used as the input of the LSTM model. For each sliding window, the LSTM model outputs the probability of being abnormal.

D. Video Capture and Analysis

Because video analysis consumes the largest amount of computing resources, video capture and analysis is designed as the last stage in *SafeShareRide*. It will only be activated when abnormal events are detected by speech recognition or driving behavior detection. In addition, in order to reduce the latency of video analysis, only the video compression is conducted on the phone.

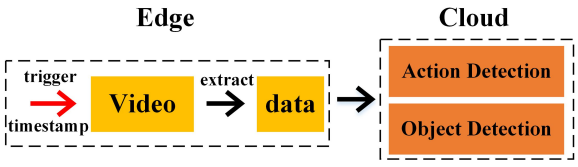

Fig. 4. The video capture and analysis in *SafeShareRide*.

As shown in Figure 4, video capture and analysis adopts an edge-cloud collaborative model. At the edge, based on the timestamp of the trigger signal, relevant clips will be extracted from the video and sent to the cloud. In the cloud, two kinds of detection are used for the video analysis. The first is action detection, which can detect excessive movements of the driver and passenger [25], [26]. For action detection, we first divide the uploaded video clip into many frames. Each frame needs to do background subtraction to get the outline of the human body [27]. We then compare the outline of every pair of continuous frames to estimate the range of movement of the human body. Finally, we compare this range with the normal moving space for passengers and drivers to determine whether the movement is abnormal. Here the normal moving space can be defined as the average moving space in normal cases. The second is object detection, such as detecting dangerous objects like guns and knives [28]. CNN can be applied to recognize such objects from frames [29]. When abnormal movements or dangerous objects are detected, the video will be shared with the law enforcement via a security link.

E. Application Framework

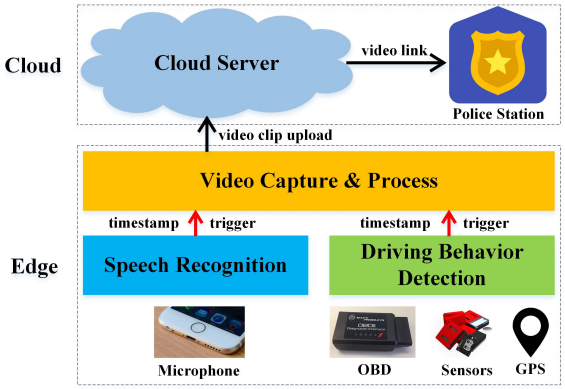

Fig. 5. The framework of *SafeShareRide*.

The overall framework of *SafeShareRide* is shown in Figure 5, which consists of two components: the component of edge or mobile devices, such as iPhone or Android phones, and the component of cloud, such as remote servers at the police station.

For the edge component, a three-stage detection model is deployed to detect the attacks happening on a vehicle. When a passenger gets into the car, all three stages of detection will be activated. The same detection model is deployed on both the driver's and passenger's smartphones. The detection frequency is set to be every 20 seconds. During each detection period, the speech recognition uses the audio information recorded by the smartphone to extract important key words. The driving safety detection utilizes the data from OBD and other sensors to determine whether the driving behavior is

normal, for example, whether the route is zigzag or the vehicle is speeding. During these two detection stages, any abnormal event will trigger the third stage detection, i.e., video capturing and analysis.

The application of *SafeShareRide* is an edge-cloud collaborative system, where the data collection and processing of audio and driving behavior data is conducted at the edge while the processing and analysis of video as well as the storage of related data are conducted in the cloud. This collaborative system is more efficient than pure edge or cloud based approaches. Because of low computation and storage requirements, edge devices can handle both speech recognition and driving behavior detection. In addition, edge devices compress videos to save the bandwidth for video uploading. The content of videos is analyzed in the cloud since the computation is intensive.

IV. Preliminary Experiments and Observations

In contrast to the cloud, edge-based services have more stable performance and save the bandwidth of data transmission. In order to evaluate the performance of each detection stage of *SafeShareRide*, we set up three demos corresponding to the three detection stages. We compared the proposed methods with cloud-based approaches in terms of latency, detection accuracy, bandwidth requirements, memory occupation and power consumption. In addition, we also conducted an energy consumption analysis by running the apps on the smartphone and comparing them with low energy consumption applications, such as *Gmail*.

All the edge-based experiments were conducted on a mobile phone (Huawei Nexus 6P). In order to monitor the overall system performance for the cloud-based approach, the cloud related experiments were conducted on the Intel Fog Node rather than a virtual machine. The bandwidth consumption was measured by an app called *Trepn Profiler* [30]. The latency was calculated based on the timestamp on the smartphone.The detection period was empirically set to be 20.0 seconds.

A. Speech Recognition

In order to evaluate the performance of speech recognition, we defined a list of key phrases or words such as "help", "help me", "help us", "rescue me", and "rescue us" [31]. We collected the audio data for each phrase from 10 volunteers as the test set. The CMUSphinx speech recognition and Google Cloud Speech were compared in terms of latency, accuracy and bandwidth. Accuracy is defined as the ratio between the number of correctly recognized phrases and the total size of the test set. The experiment results are shown in Table 1. We can see that Google Cloud Speech has higher accuracy and lower latency than that of CMUSphinx speech recognition. Since CMUSphinx works offline, the bandwidth consumption is 0.0 KB/s. The bandwidth of Google Cloud Speech is 23.0 KB/s. As we don't know how much computational resource the cloud has consumed, it may not be fair to compare the accuracy and latency. According to the results in [32], the accuracy of CMUSphinx can reach above 95% when used on a small number of vocabularies. The latency can also be reduced.

TABLE I
THE EXPERIMENT RESULTS OF SPEECH RECOGNITION.

Metric	CMUSphinx	Google Cloud Speech
Accuracy	73.6%	86.1%
Latency	2.279s	0.158s
Bandwidth	0KB/s	23KB/s
Energy	0.024J	0.055J

TABLE II
THE EXPERIMENT RESULTS OF DRIVING BEHAVIOR DETECTION.

Metric	Huawei Nexus 6P	Intel Fog Node
Latency	0.264s	0.036s
Bandwidth	0KB/s	35KB/s
Energy	0.38J	0.79J
Memory	84MB	170MB

Observation 1: Although edge computing is very promising, it still needs optimization to become more competitive, comparing with the cloud-based approach.

B. Driving Behavior Detection

For driving behavior detection, we trained the LSTM detection model in the cloud. We used *TensorFlow Lite* [14] as the deep learning framework to perform model inference at the edge. The length of the slide window was empirically set to be 5.0 seconds. The data collected by the OBD adapter included the timestamp, longitude, latitude, speed, altitude, bearing, gravity, etc. The collection frequency was 0.1 seconds. We empirically set the detection probability threshold at 0.8. That is, if the output probability of LSTM is larger than 0.8, we consider the driving behavior as abnormal. We measured the average latency, bandwidth assumption, and energy consumption for one detection process. The experiment results are shown in Table 2. As we can see, the latency of both approaches is below one second. The edge-based approach does not have bandwidth requirements, so it is more stable than the cloud-based approach. When the communication signal is weak, the edge-based approach will not be affected. The energy consumption of the edge-based approach is also lower than that of the cloud-based, so the edge-based driving behavior detection is more efficient.

TABLE III
THE EXPERIMENT RESULTS OF VIDEO ANALYSIS.

Metric	Huawei Nexus 6P	Intel Fog Node
Latency	0.675s	0.603s
Bandwidth	13KB/s	65KB/s
Energy	0.34J	0.81J

Observation 2: It is more effective to train machine learning models in the cloud and deploy the trained models on edge devices.

C. Video Capture and Analysis

As we only uploaded corresponding video clips based on the timestamps from the first two detection stages, the required bandwidth was significantly reduced compared to uploading all the video to the cloud. We set up a demo of video analysis to compare the performance of the edge-based approach and the cloud-based approach. In this demo, the video clip was transformed into 1280x720 (720P) and the number of frames per second was set to be 30.0. The video data was encoded in the H.264 [33] format with a baseline profile. One intra-frame (IFrame) is configured to be followed by fifty-nine predictive-frames (PFrames) without a bi-directional frame (BFrame), because we simulated a live video stream and could not compute the differences between the current frame and the next frame. The data was sent to the edge nodes using the real-time transport protocol (RTP) over UDP/IP network. We used Simple RTMP Server [34] to push and pull the video stream. The data was transmitted through the TCP/IP and HTTP protocol. For the video analysis, we used the object classification of OpenCV to analyze the video clips. The experiment results of video uploading and analysis are shown in Table 3. As we can see, the edge-based video analysis approach is both more bandwidth and energy efficient than the cloud-based approach. The latency of both approaches is close, smaller than one second.

D. Energy Consumption

As the battery power on smartphones is limited, the energy consumption of the applications is an important factor to consider. We calculated the total energy consumption by combining the consumption from each detection stage, and measured it at different detection periods. As a comparison, we also measured the energy consumption of *Gmail* when it was running as a background application in *Huawei Nexus 6P* . The energy consumption of 30 minutes is shown in Figure 6, where T in *SafeShareRide-T* indicates that the detection period is set to be T seconds.

From Figure 6, we can see that the energy consumption of *SafeShareRide* is lower than that of *Gmail* even when the detection period is set at 3.0 seconds. When the detection period is longer, the energy consumption can be further reduced.

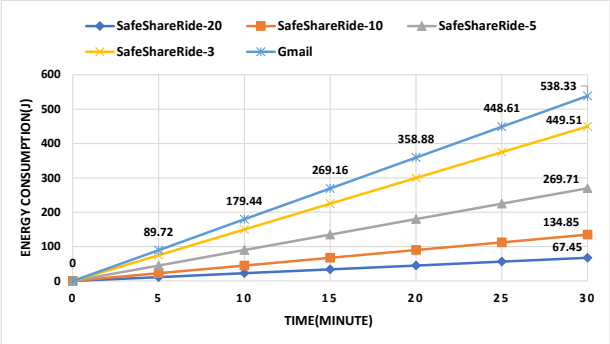

Fig. 6. The energy consumption of *SafeShareRide* and *Gmail*.

TABLE IV
THE HARDWARE CONFIGURATION COMPARISONS

Metric	Huawei Nexus 6P	Virtual Machine
CPU	4x1.55 GHz Cortex-A53 & 4x2.0 GHz Cortex-A57	4xIntel(R) Xeon(R) 2.40GHZ
GPU	Adreno 430	Microsoft Corporation Hyper-V
OS	Android 8.1.0	Ubuntu 16.04
MEMORY	3 GB RAM	13 GB RAM
BATTERY	3450 mAh	—

Observation 3: Energy consumption can be further reduced by enabling different detection frequency at different detection stage.

V. SYSTEM IMPLEMENTATION

Based on the preliminary experiments and the above observations, in this section, we introduce the specific system design of SafeShareRide on Huawei Nexus 6P and the Microsoft Azure virtual machine. We also discuss the insights gained through implementing the detection system.

A. Hardware Configuration

The edge-based *SafeShareRide* system is based on Huawei Nexus 6P and a virtual machine of Microsoft Azure. The virtual machine is responsible for storing and analyzing the compressed video sent from Huawei Nexus 6P. For the cloud-based approach, all three detection stages are conducted on the virtual machine. All the data, including audio from the microphone, driving data from OBD and video clips from a camera, are sent to the cloud for further analysis. The virtual machine on Microsoft Azure is the Standard_D3_v2 series. Table 4 shows the specific hardware configurations of the Huawei Nexus 6P and the virtual machine.

As we can see from Table 4, the virtual machine is much more powerful than the smartphone. In the design of *SafeShareRide*, we want to leverage the advantages

of both edge and cloud, therefore enhancing the entire system performance. Computation intensive and data intensive tasks, such as video analysis and model training, can be conducted on the virtual machine. In addition, as the battery power of the smartphone is limited, we need to run energy efficient tasks on the smartphone.

B. System Design

In this section we discuss the system design of *SafeShareRide* based on the observations that we gained in the preliminary demos. As shown in Figure 7, we developed the Android app as well as the web service on the virtual machine. The connection between speech recognition activity and the virtual machine and the connection between driving behavior activity and the virtual machine only exist in the cloud-based approach.

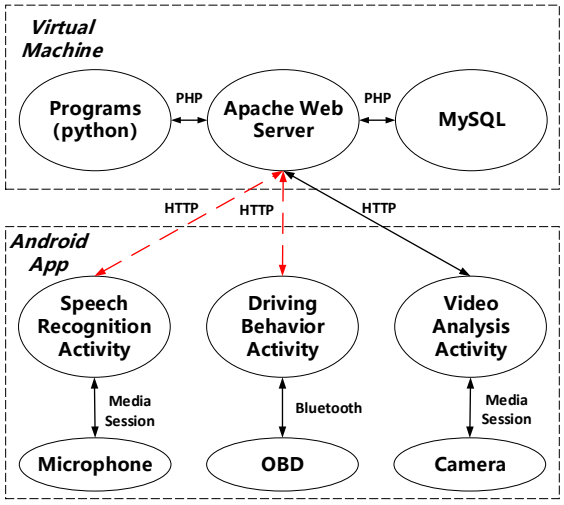

Fig. 7. The system design of *SafeShareRide*.

For speech recognition, the Android app reads data from the microphone through media session APIs and stores the audio data on the smartphone. The speech-to-text process is then conducted. In the cloud-based approach, the audio is sent to the cloud through the HTTP protocol. For driving behavior detection, we first set up a service to read the driving data from the OBD adapter. This service is based on the Bluetooth protocol. The detection model is trained in the cloud and then deployed on the smartphone. We perform model inference on the data collected from the OBD adapter. For video analysis, the Android app receives video from a camera through media session APIs. We extract the video clips, which correspond to the detection periods where speech recognition or driving behavior detection identifies abnormal events. These clips are uploaded to the virtual machine.

On the virtual machine, we deploy an Apache Web Server to respond to the HTTP request. PHP scripts are used to direct the HTTP requests. For HTTP requests to upload the data, they are directed to the MySQL database. For HTTP requests to run the inference of the driving behavior model and object classification model, they are directed to the corresponding python programs. The results are finally attached to the HTTP response. In addition, the training process of both the driving behavior detection and object classification are conducted on the virtual machine. The inference of object classification is also conducted on the virtual machine.

C. Detection Models

For the models at each stage, we leveraged open source libraries to do the implementation. Specifically, we leveraged the speech recognition toolkit, PocketSphinx, which is based on CMUSphinx [18] on Android phones. We designed the phonetic dictionary and language model for keywords extraction.

The driving behavior detection model is based on LSTM [21], [35], [36], which is widely used to model sequential data. We use one layer of LSTM, which has 128 neurons in each cell and is followed by a sigmoid classifier. The classifier outputs the probability of a sequence being abnormal or not. We set learning rate to 10^{-3}. The batch size is 10 and the number of epochs is 50.

The driving behavior data collected by OBD comes from a ride which lasted about 1.5 hours and the sampling frequency is 1.0s. We divided the time series into overlapping sliding windows. Each sliding window consist of driving data over five seconds, such as, 0.0-5.0s, 1.0-6.0s, and 2.0-7.0s. Since the data only included normal driving behaviors, we simulated abnormal driving data by randomly disordering the data of one dimension in the normal driving data. In this way, we generated the dataset consist of one half normal driving data and one half abnormal data. We use 80% for training and the remaining 20% for test.

The model of the video capture and analysis was based on CNN, while the algorithm of video analysis was based on OpenCV [37] and Inception-v3 [28]. We implemented the object classification on Android using TensorFlow Lite. We trained the CNN model with pictures containing knives and guns to identify whether there are such dangerous objects in each video frame. We built an image data set which contains many kinds of knives and guns. The dataset was used to train the inception-v3 model to make the latter perform better for our scenario.

VI. Performance Evaluation

In this section, we evaluate the performance of *SafeShareRide* on four aspects: workload setup, memory and CPU usage of three stages, edge and cloud performance comparison, and stationary and moving condition performance comparison. In order to evaluate the availability

TABLE V
THE TESTING DATASET FOR EXPERIMENTS

Data	Size	Time
Audio	579MB	8000s
OBD adapter	322KB	30min
Video	1400MB	600s

and efficiency of *SafeShareRide*, we chose four metrics, including CPU usage, memory usage, bandwidth requirements, and latency. We deployed the same algorithms on both Huawei Nexus 6P and the cloud virtual machine. For the edged-based approach, Android Profile is used to monitor the CPU usage, memory usage, network usage, and record the latency of every inference. In the cloud-based approach, as the virtual machine is based on Ubuntu, we used Linux commands such as *top*, *htop*, and *time* to collect the metrics. The biggest challenge is the instability of wireless communication. We conducted the experiments under various network conditions, including WiFi, 4G stationary, and 4G moving.

A. Setup

To begin with, we first discuss the setup of *SafeShareRide* as well as the cloud-based approach. For speech recognition in *SafeShareRide*, we implemented PocketSphinx-based app using Google's speech recognition toolkit on the smartphone. With Google's Android-based speech-to-text service, Android apps can do speech-to-text translation offline. We also implemented the module using the CMUSphinx speech recognition library and the Google Cloud Speech library on the virtual machine.

For edge-based speech-to-text translation, the speech recognition module puts the audio file into a recognizer class and uses the CMUSphinx models or Google's offline model to do the inference. For the cloud-based approach, we use the HTTP protocol to transmit the audio data to the virtual machine. The URL of HTTP request directs the data to an *audio.php* script on the virtual machine. This script then loads the data into a MySQL database and launches the inference of the speech recognition model. The final results are attached to the HTTP response.

The processes of cloud-based driving behavior detection and video capture and analysis are similar to those in speech recognition. They are all based on HTTP protocols and use PHP to direct the data to corresponding detection methods, which are implemented in python.

As workload is a significant factor influencing performance, here we discuss the workload in the experiments. The testing data sets used in the experiments are shown in Table V. We used the Mozilla Common Voice Data sets [38] as the voice experiment data set. In total, we chose 3994 voice files from Mozilla as the input to the speech recognition activity. We collected 10 hours of driving data from the OBD adapter as the testing data set for driving behavior detection. We also recorded in-vehicle videos with Huawei Nexus 6P for the object classification experiments.

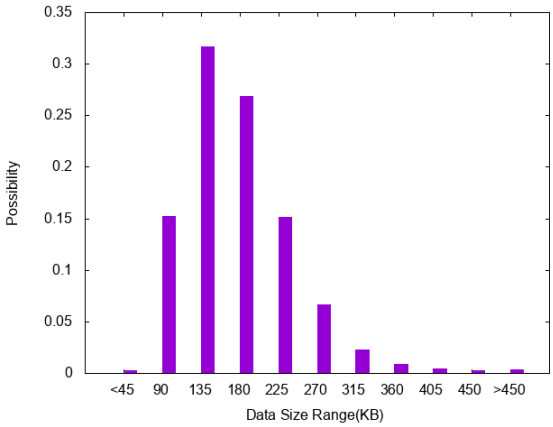

Fig. 8. The Distribution of Audio Data Size.

The distribution of audio data size is shown in Figure 8. As we can see, it roughly follows a normal distribution. The mean audio size is between 135 KB and 180 KB. For OBD data, the size of a file with 300 second driving data is about 53.8 KB. However, a 60 second video recorded by Huawei Nexus 6P can be around 140MB, which is much larger than the audio and driving data.

B. Memory Usage and CPU Usage of Three Stages

The three detection stages of *SafeShareRide* are based on computationally intensive algorithms and models, for example, Hidden Markov Model in Speech Recognition, LSTM in driving behavior detection, and CNN in object classification. Therefore, it is important to evaluate the memory and CPU usage. We implemented CMUSphinx, Google Cloud Sphinx, driving behavior detection model, and object classification model on the virtual machine. We collect the Resident Memory Usage (RES) and single core CPU usage of these algorithms during model inference. For object detection, we took one hundred frames as the input of model inference.

Figure 9 and Figure 10 show the RES and CPU usage of Sphinx and Google Cloud over time. From Figure 9, we can see that the CPU usage of Sphinx is constantly above 90%, while the memory usage varies. The peak RES is 130000 Kb. In contrast, the CPU usage of Google Cloud Speech is constantly below 5 %, and the peak RES is only 17200 Kb. The accuracy and latency of Sphinx is not as good as Google Cloud Speech. However, since Google Cloud Speech provides the service in the cloud, what we have measured is only the resource usage of

a single client. Therefore, it is unfair to compare these two approaches. The following is our first insight:

Insight 1: It is unfair to compare the edge-based approach with the industry commercial cloud-based approach because obtaining the complete resource consumption of the latter is almost impossible.

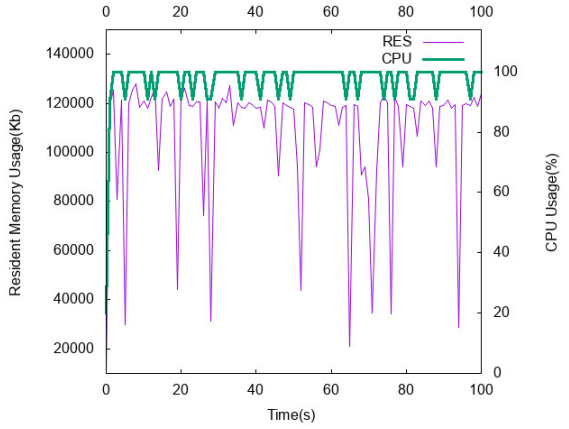

Fig. 9. The Resident Memory Usage and CPU Usage of Sphinx Speech Recognition.

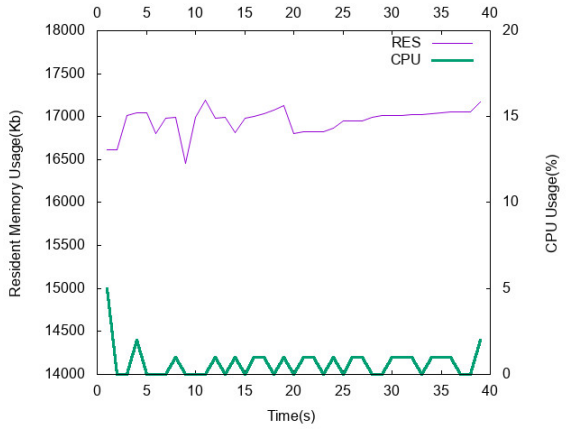

Fig. 10. The Resident Memory Usage and CPU Usage of Google Cloud Speech.

Figure 11 and Figure 12 show the RES and CPU usage of the driving behavior detection model and the object classification model, respectively. During the model inference, the first step is to load the trained model into the memory. Therefore, the RES and CPU will increase greatly at the beginning of the inference. The second step is to read data into the memory. When the inference is done, both the memory of the model and data will be released. For the driving behavior detection model, its RES peak is 350,000 Kb while the peak CPU usage is 110%. For the object classification model, its RES peak is 500,000 Kb while the peak CPU is 100%. Both are significantly larger than that of speech recognition. The time cost of driving behavior detection is nearly 7.0 seconds while the object classification model takes about 2.5 seconds. Therefore, the model inference is not computationally expensive. It is suitable to do model inference at the edge. We present our second insight below.

Insight 2: The model training and inference to be conducted in the cloud or at the edge should take specific applications into account.

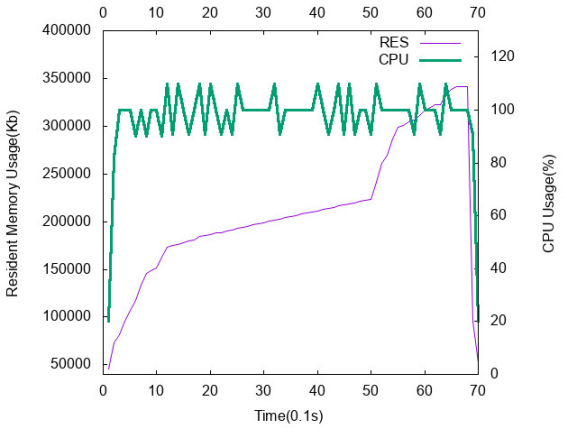

Fig. 11. The Resident Memory Usage and CPU Usage of Driving Behavior Model Inference

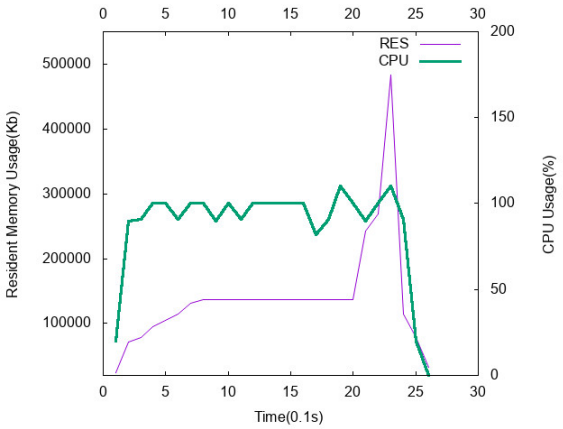

Fig. 12. The Resident Memory Usage and CPU Usage of Object Classification Model Inference

C. Edge and Cloud Performance Comparison

In order to evaluate the performance of *SafeShareRide*, we implemented the speech recognition, driving behavior detection, and object classification on the virtual machine using the same algorithms and models as those on the smartphone. Google Cloud Speech is also implemented on both the smartphone and the cloud. We collected metrics such as the model inference latency, network

bandwidth usage, average CPU usage and memory usage. The results are shown in Table VI, where offline means that the computation is conducted at the edge, online means that the data needs to be uploaded to the cloud, and the computation is also conducted in the cloud.

For speech recognition, Google Speech Recognition is the offline version while Google Cloud Speech is online. PocketSphinx is the Android version of CMUSphinx, which is more flexible than Sphinx. The online approaches have much longer latency than the offline since they also include data uploading latency. The average CPU usage of Sphinx can be much larger than that of PocketSphinx, but the average CPU usage of the Google Cloud Speech is much less than that of Google Speech Recognition. In contrast, the memory usage of online approaches is less than offline. As far as we are concerned, the CPU and memory usage are not related to online or offline, but rather to specific algorithms and models. Offline approaches do not need network communication.

For driving behavior detection, the latency difference between online and offline approaches is less than that of speech recognition, as the driving data size is much smaller than the audio data. In addition, we do not observe a clear distinction between the average CPU usage and memory usage between online and offline.

For object classification, as video is the largest among all the data sets, the latency difference between online and offline is also the largest. The same happens on network bandwidth difference. There seems to be no strong correlation between CPU usage, memory usage and online or offline status. Our third insight is based on the above observations.

Insight 3: The CPU and memory usage are not related with online or offline status. They are more related with specific algorithms and models.

D. Performance Comparison of Moving & Stationary Scenarios

Compared with the edge-based approach, one of the most important disadvantages for the cloud-based approach is that the data needs to be uploaded to the cloud. Therefore, the performance of the cloud-based approach can be greatly affected when the network communication is not good enough. In order to evaluate the impact, we conducted two experiments, i.e., Google Cloud Speech and online object classification, when the vehicles was running at different speeds. The latency and accuracy were collected . The results of Google Cloud Speech are shown in Figure 13. As we can see, when the vehicle's speed increases, the latency of speech recognition increases as well. On the other hand, the accuracy is not affected. Compared with the stationary scenario, the latency of Google Cloud Speech increases by 44.6% when the driving speed is 70 miles per hour.

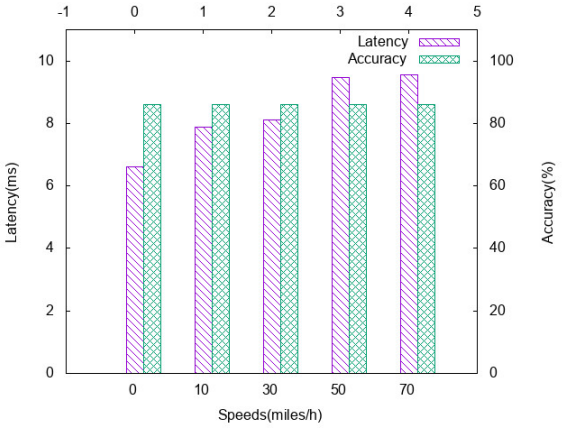

Fig. 13. The Latency and Accuracy of Google Cloud Speech under different driving speeds

The results of online object classification are shown in Figure 14. Similar to Google Cloud Speech, the accuracy of object classification is also not affected by driving speed. However, latency is greatly affected by speed. When the driving speed is at 60 miles per hour, the latency is almost 18 times greater than in the stationary scenario. Applications that have low latency requirements will face huge challenges in moving scenarios. In addition, from Figure 13 and Figure 14, we can see that the latency of smaller data seems less affected by the moving condition. Therefore, for large data sets such as video, it can be more effective to use the edge-based approach or divide the data into smaller sets before uploading to the cloud. We present the fourth insight below:

Insight 4: It can be a huge challenge for cloud-based approaches in moving scenarios. Edge-based approaches may be more desirable under moving conditions.

VII. Discussions and Limitations

From our system design and evaluation, we can see that the edge-based approach is more efficient than the pure cloud-based approach for detecting dangerous events in ridesharing services. However, there are still many open problems.

The first problem is the system overhead. To date, we have launched three activities at three detection stages of *SafeShareRide*. For each activity, we set up threads for various tasks, such as data collection, data upload, and model inference. A synchronous method was used to schedule the back-end threads of these activities. We did not take the system overhead into account when designing the system. The *SafeShareRide* system needs more optimization to guarantee availability.

TABLE VI
THE PERFORMANCE COMPARISON BETWEEN ONLINE AND OFFLINE

Metric	Latency(ms)	Bandwidth(KB/s)	Average CPU Usage (one core)	Memory(MB)
PocketSphinx (offline)	613.04	0	5.40%	106.00
Google Speech Recognition (offline)	4.22	0	2.80%	62.00
Sphinx	1856.94	0	90.9%	16.25
Google Cloud Speech (online)	40.28	23	0.69%	2.15
Driving Behavior Detection (offline)	81.16	0	12.30%	164.60
Driving Behavior Detection (online)	248.26	12	107.00%	42.68
Object Classification (offline)	76.64	0	14.00%	197.80
Object Classification (online)	11325.90	2274	116.00%	60.40

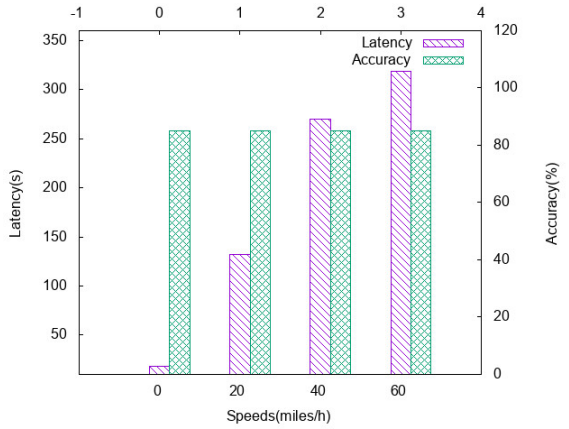

Fig. 14. The Latency and Accuracy of Object Classification under different driving speeds

The second problem is the application of different detection models. Currently, these models are implemented using off-the-shelf open source packages. Their resource utilization can be further improved with more customization and optimization. In addition, these models are trained with data sets of limited sizes and sources. In practice, the performance of the models will vary with different training and test data.

The third problem is the adaptive scheduling task between edge and cloud computing. According to our experiments, both edge and cloud computing have shown advantages under different conditions. Therefore, it can be more effective to have a collaborative edge-cloud model. For a specific task, whether it should be run in the cloud or at the edge can be dynamically determined by factors such as the task type, available network bandwidth, and battery status.

Last but not the least, data privacy remains a big issue. *SafeShareRide* analyzes audio, driving data, and video to detect attacks. However, these data may contain sensitive personal information. It is important to provide detection services while complying with privacy regulations.

In the future, we plan to develop an asynchronous framework to optimize the back-end threads scheduling in the *SafeShareRide* system. We will also collect more data to train the models at each detection stage and improve the detection performance with more customization. In addition, we will design an edge-cloud collaboration scheme based on task type and computing environment. Finally, privacy preserving data sharing mechanisms can leveraged to better protect user privacy.

VIII. RELATED WORK

We present the related work in the following two categories: 1) work that is relevant to driving safety and 2) work on edge computing enabled applications.

A. Driving Safety

Many recent studies have focused on driving behavior detection. [39] developed an android application to detect dangerous driving behavior using dual cameras on smartphones. They applied computer vision and machine learning algorithms to detect whether the driver is tired or distracted. [40] proposed a method to detect driver frustration using the video and audio data collected through an in-vehicle navigation system. SVM is applied to discriminate between classes. This work is more focused on the detection of the abnormal emotions of the drivers, not on dangerous events in ridesharing services. [41] predicted unsafe driving behaviors based on facial-expression analysis. They developed a driver-safety framework to capture vehicle dynamics and the driver's facial expressions. A virtual driving safety simulator is used to collect a large set of accidents. They applied many classifiers, including Bayesian nets, decision trees, and support vector machine (SVM), to predict the probability of unsafe driving behavior. However, their work only uses video data to predict unsafe driving behavior,s not other types of data, such as in-vehicle audio streams and driving data. [42] analyzed driver behaviors using large scale machine learning algorithms. They collected data from 78 participants and obtained 3.5 billion video frames. Their data streams include Inertial Measurement Unit(IMU), GPS, and Controller Area Network(CAN) messages. The video data is integrated with contextual information such as vehicle state, driver characteristics, and mental models. This work is more

focused on data management and improvement of the algorithms.

B. Edge Computing Enabled Applications

Edge computing has enjoyed great popularity in a wide range of applications, due to its advantage of conducting computation near the data source. Before this work, we proposed the framework of *SafeShareRide* and set up three demos to illustrate the availability of the detection system [43]. In this paper we redesign the entire system, implement it as an Android app, and evaluate its performance in a more systematic way. Edge computing has been widely used in video analysis. [44] proposed a video analytic software stack, namely, Rocket. A video pipeline optimizer is used to optimize the allocation of resources based on the data and the properties of deep neural network models. The allocation message is sent to the resource manager to allocate hardware resources. [45] applied edge computing to enable latency aware video analysis. They adopted an edge first design, and optimized the selection of uploading tasks in order to minimize the response time. They also designed task placement schemes to achieve inter-edge collaboration. [46] proposed a new computing framework called Firework to facilitate distributed data processing and sharing for Internet of Things applications. Firework is implemented using virtual shared data view and service composition. Firework can significantly reduce latency and bandwidth cost, compared with cloud-based approaches. However, this work did not take moving scenarios into account. When a vehicle is moving, the performance of data sharing can be very different.

IX. Conclusion

SafeShareRide is an edge-based approach for attack detection in ride sharing services. It uses three stages to provide detections with high accuracy and low overhead. The three stages include speech recognition, driving behavior detection, and video capture and analysis. The trigger mechanism is used to improve detection accuracy and reduce bandwidth requirement for video analysis. We designed and implemented the *SafeShareRide* system as an android application. We compared the performance of *SafeShareRide* with other edge-based and cloud-based approaches, in terms of the CPU and memory usage in online and offline scenarios and the latency and accuracy in moving and stationary scenarios. We drew several insights from these experiments. *SafeShareRide* has shown better performance than cloud-based approach in most scenarios. It is an effective and efficient edge-based attack detection framework for ride sharing services.

Acknowledgment

The authors are very grateful to the reviewers and our shepherd Aakanksha Chowdhery for their constructive comments and suggestions. This work is supported in part by National Science Foundation (NSF) grant CNS-1741635.

References

[1] Safe rides, safer cities, 2018.
[2] Trip safety, our commitment to riders, 2018.
[3] Safety and confidence behind the wheel, our commitment to drivers, 2018.
[4] Rider and driver safety technology, 2017.
[5] Didi chuxing upgrades rider and driver safety framework, 2016.
[6] Didi upgrades driver training system with compulsory safety courses, 2016.
[7] Suk Yu Hui and Kai Hau Yeung. Challenges in the migration to 4g mobile systems. *IEEE Communications magazine*, 41(12):54–59, 2003.
[8] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.
[9] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.
[10] Qingyang Zhang, Yifan Wang, Xingzhou Zhang, Liangkai Liu, Xiaopei Wu, Weisong Shi, and Hong Zhong. Openvdap: An open vehicular data analytics platform for cavs. In *Distributed Computing Systems (ICDCS), 2018 IEEE 38th International Conference on*. IEEE, 2018.
[11] Data never sleeps 5.0, 2018.
[12] Kyungmin Lee, Jason Flinn, and Brian D Noble. Gremlin: scheduling interactions in vehicular computing. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 4. ACM, 2017.
[13] Bozhao Qi, Lei Kang, and Suman Banerjee. A vehicle-based edge computing platform for transit and human mobility analytics. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 1. ACM, 2017.
[14] Introduction to TensorFlow Lite, 2018.
[15] Core ml: Integrate machine learning models into your app, 2018.
[16] Open source toolkits for speech recognition, February 2017.
[17] Christian Gaida, Patrick Lange, Rico Petrick, Patrick Proba, Ahmed Malatawy, and David Suendermann-Oeft. Comparing open-source speech recognition toolkits.
[18] CMUSphinx: open source speech recognition toolkit, 2017.
[19] Sean R Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.
[20] Luigi Gerosa, Giuseppe Valenzise, Marco Tagliasacchi, Fabio Antonacci, and Augusto Sarti. Scream and gunshot detection in noisy environments. In *Signal Processing Conference, 2007 15th European*, pages 1216–1220. IEEE, 2007.
[21] Chunmei Ma, Xili Dai, Jinqi Zhu, Nianbo Liu, Huazhi Sun, and Ming Liu. Drivingsense: Dangerous driving behavior identification based on smartphone autocalibration. *Mobile Information Systems*, 2017, 2017.
[22] Xingzhou Zhang, Yifan Wang, and Weisong Shi. pcamp: Performance comparison of machine learning packages on theedges. 2018.
[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
[24] Zhongyang Chen, Jiadi Yu, Yanmin Zhu, Yingying Chen, and Minglu Li. Abnormal driving behaviors detection and identification using smartphone sensors. In *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on*, pages 524–532. IEEE, 2015.
[25] Massimo Piccardi. Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on*, volume 4, pages 3099–3104. IEEE, 2004.
[26] Andrews Sobral and Antoine Vacavant. A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos. *Computer Vision and Image Understanding*, 122:4–21, 2014.

[27] Yaser Sheikh, Omar Javed, and Takeo Kanade. Background subtraction for freely moving cameras. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1219–1225. IEEE, 2009.

[28] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[30] Trepn Profiler6.2: Qualcomm innovation center, inc., 2016.

[31] Javier Ramirez, Juan Manuel Górriz, and José Carlos Segura. Voice activity detection. fundamentals and speech recognition system robustness. In *Robust speech recognition and understanding*. InTech, 2007.

[32] Hassan Satori, Hussein Hiyassat, Mostafa Harti, Nouredine Chenfour, et al. Investigation arabic speech recognition using cmu sphinx system. *Int. Arab J. Inf. Technol.*, 6(2):186–190, 2009.

[33] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[34] Simple RTMP Server, 2017.

[35] Ravi Bhoraskar, Nagamanoj Vankadhara, Bhaskaran Raman, and Purushottam Kulkarni. Wolverine: Traffic and road condition estimation using smartphone sensors. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, pages 1–6. IEEE, 2012.

[36] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[37] OpenCV: Open source computer vision library, 2018.

[38] Mozilla Common Voice Datasets, 2018.

[39] Chuang-Wen You, Martha Montes-de Oca, Thomas J Bao, Nicholas D Lane, Hong Lu, Giuseppe Cardone, Lorenzo Torresani, and Andrew T Campbell. Carsafe: a driver safety app that detects dangerous driving behavior using dual-cameras on smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 671–672. ACM, 2012.

[40] Irman Abdić, Lex Fridman, Daniel McDuff, Erik Marchi, Bryan Reimer, and Björn Schuller. Driver frustration detection from audio and video in the wild. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1354–1360. AAAI Press, 2016.

[41] Maria Jabon, Jeremy Bailenson, Emmanuel Pontikakis, Leila Takayama, and Clifford Nass. Facial expression analysis for predicting unsafe driving behavior. *IEEE Pervasive Computing*, 10(4):84–95, 2011.

[42] Lex Fridman, Daniel E. Brown, Michael Glazer, William Angell, Spencer Dodd, Benedikt Jenik, Jack Terwilliger, Julia Kindelsberger, Li Ding, Sean Seaman, Hillary Abraham, Alea Mehler, Andrew Sipperley, Anthony Pettinato, Bobbie Seppelt, Linda Angell, Bruce Mehler, and Bryan Reimer. MIT autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation. *CoRR*, abs/1711.06976, 2017.

[43] Liangkai Liu, Xingzhou Zhang, Qiao Mu, and Weisong Shi. Safeshareride: Edge-based attack detection in ridesharing services. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, 2018. USENIX Association.

[44] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *Computer*, 50(10):58–67, 2017.

[45] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 15. ACM, 2017.

[46] Quan Zhang, Qingyang Zhang, Weisong Shi, and Hong Zhong. Firework: Data processing and sharing for hybrid cloud-edge analytics. *IEEE Transactions on Parallel and Distributed Systems*, 2018.