# EdgeSharing: Edge Assisted Real-time Localization and Object Sharing in Urban Streets

Luyang Liu
*Google Research*
luyangliu@google.com

Marco Gruteser
*WINLAB, Rutgers University*
gruteser@winlab.rutgers.edu

***Abstract*—Collaborative object localization and sharing at smart intersections promises to improve situational awareness of traffic participants in key areas where hazards exist due to visual obstructions. By sharing a moving object's location between different camera-equipped devices, it effectively extends the vision of traffic participants beyond their field of view. However, accurately sharing objects between moving clients is extremely challenging due to the high accuracy requirements for localizing both the client position and positions of its detected objects. Therefore, we introduce EdgeSharing, a localization and object sharing system leveraging the resources of edge cloud platforms. EdgeSharing holds a real-time 3D feature map of its coverage region to provide accurate localization and object sharing service to the client devices passing through this region. We further propose several optimization techniques to increase the localization accuracy, reduce the bandwidth consumption and decrease the offloading latency of the system. The result shows that the system is able to achieve a mean vehicle localization error of 0.28-1.27 meters, an object sharing accuracy of 82.3%-91.4%, and a 54.7% object awareness increment in urban streets and intersections. In addition, the proposed optimization techniques reduce bandwidth consumption by 70.12% and end-to-end latency by 40.09%.**

## I. Introduction

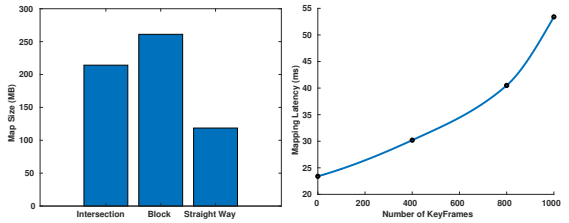
Urban streets and intersections are notorious traffic trouble spots. According to the U.S. Department of Transportation, 51 percent of all injury crashes and 28 percent of all fatal crashes in the United States occur at intersections [1]. Many accidents, which occur at intersections involve visual occlusions of cars or vulnerable road users. This is a challenge that instrumenting individual vehicles with sensors, as in current automated driving and advanced driver assistance system, cannot fully solve since they can suffer from similar occlusions.

Ubiquitous sensors and computational resources on the road raise the possibility of smart intersections that address such occlusions through object sharing. Object sharing systems promise to extend traffic participants' awareness beyond their field of views by sharing the moving objects detected from different camera perspectives and positions. For example, a leading vehicle can share its detected front vehicle or pedestrian locations to a following vehicle whose field of view has been blocked by the leading vehicle. Meanwhile, the following vehicle can also share the traffic participants detected in the blind spot of the leading vehicle to extend its awareness. Furthermore, a deployed camera at an intersection can localize each object in the intersection and provides this information to all clients in the nearby region.

Accurately sharing objects between moving clients is extremely challenging due to the high accuracy and low latency requirement for localizing both the client position and positions of its detected objects. Compared to GPS and other inertial sensing methods that are widely recognized to be less accurate in dense city scenarios, visual odometry solutions (e.g. SLAM) are more feasible in such situations where rich visual features exist. These solutions typically determine the position and orientation of a client device by analyzing the associated vision inputs (e.g. camera image, Lidar depth map, etc.) with a map constructed by 3D features. However, these solutions require large amounts of computation and storage resources on the end devices to store the large feature map and run computational intensive tasks on captured frames. In a profiling task of a popular open-sourced SLAM algorithm ORB-SLAM, we find that the 3D feature map of a single intersection, block, or straight way in a normal city requires more than 100MB of memory to store (Figure 1(a)). This makes it almost impossible for mobile devices to store a city or state-level 3D feature map locally. We also find that the mapping latency of ORB-SLAM increases with the number of keyframes in the map, as shown in Figure 1(b). A feature map with around 800 keyframes (similar to the number of keyframes in an intersection) requires around 40ms to track a frame on an Intel I7 5500U CPU, which makes it extremely hard to run at high frequency on current mobile devices. Additionally, running sophisticated object detection on mobile devices is also extremely challenging [2].

Today's visual odometry method on mobile devices either run in small closed spaces (e.g. a room) or offload image features up to the cloud for assistance. Most of existing AR/VR platforms, including Apple ARKit [3], Google ARCore [4], Microsoft Hololens [5], and Oculus Go [6], are able to achieve real-time SLAM in small spaces with mostly stationary scenes. Mapping in large and open spaces are even more challenging due to the large variations in the environment. Existing outdoor mapping technologies adopted by self-driving cars requires the vehicle to equipped with high computational resources (e.g. GPU or FPGA) and also store a large precomputed 3D feature maps onboard to achieve accurate visual based localization. Other applications such as Google Map's AR navigation app [7] offloads image feature matching process to the cloud. For object sharing, AVR [8] allows vehicles

Most of the work was done when Luyang Liu was a PhD student in WINLAB at Rutgers University.




(a) Storage requirement for three scenarios.

(b) Mapping latency increases with the number of keyframes in the map.

Fig. 1. Challenges of running object sharing system on mobile nodes.

to share point clouds between each another through V2V communication. However, it still requires each vehicle to have large computation resources and comprehensive 3D feature map on board. While cloud offload is a feasible way to overcome these limitations of the device's resources, it requires a long transmission latency to upload the visual information to the remote cloud. It is therefore highly desirable to offload localization and object sharing platform on in a way that reduces latency incurred by frame transmission so that it can support driving and vehicle control applications.

In this paper, we introduce EdgeSharing, a first collaborative localization and object sharing system leveraging the resources of an edge cloud platform and the visual inputs from participating mobile clients (e.g., vehicles and pedestrians). In EdgeSharing, the edge cloud constructs a 3D feature map of its coverage region from images and depth readings from a dedicated data collection vehicle or crowdsourced from participating clients. This 3D feature map is then used to provide accurate localization services to client devices passing through this region. Leveraging the computation power on the edge cloud, EdgeSharing also detects object locations on the images offloaded by participating clients. These locations are stored in a sharing database and can be shared with other clients in the same region. With EdgeSharing installed on the edge cloud, nearby vehicles are able to learn extra object (e.g., traffic participants) locations from the edge cloud, which are outside the vehicles field of view. This improves their situational awareness and safety.

To support accurate localization and object sharing while reducing latency and bandwidth consumption, we propose several optimization techniques. In particular, a *Context-Aware Feature Selection* uses the edge's computational resources to filter out feature points on potential moving objects in the offloaded images to increase SLAM localization accuracy. We also introduce a *Collaborative Local Tracking* mechanism to significantly reduce the bandwidth consumption of frame transmission by only offloading selected keyframes to the edge cloud, while using a lightweight local tracking method to keep track of the location of the client and its detected objects on the end device. In addition, we design a *Parallel Streaming and Processing* method to enable parallel video streaming and cloud processing, which largely reduces the end-to-end latency of EdgeSharing.

EdgeSharing is able to achieve high-quality and real-time localization and accurate object sharing with only small amounts of cloud computation and bandwidth resources. Our evaluation result demonstrates that the system is able to achieve a mean vehicle localization error of 0.2813-1.2717 meters, an object sharing accuracy of 82.3%-91.44%, and a 54.68% object awareness increment in urban streets and intersections. In addition, the proposed optimization techniques are able to reduce 70.12% of bandwidth consumption and reduce 40.09% of the end-to-end latency.

The contributions of this work can be summarized as follows:

- Designing the first real-time collaborative localization and object sharing system EdgeSharing, leveraging the support of the edge cloud.
- Proposing a practical solution to localize and share objects detected by dynamic moving devices.
- Improving the localization accuracy with context-aware feature selection mechanism.
- Reducing offloading bandwidth with a collaborative local tracking method.
- Decreasing the end-to-end latency using a parallel streaming and processing pipeline.
- Implementing and evaluating the EdgeSharing system based on commodity hardware and showing that the proposed system can achieve high-quality and real-time localization and accurate object sharing with only small amounts of cloud computation and bandwidth resources.

## II. EdgeSharing Design

EdgeSharing provides high-quality real-time object sharing services to travelers in dense city traffic scenarios leveraging large computational resources at the edge cloud. First, this platform allows mobile clients, such as vehicles and pedestrians to offload computations of the visual localization task onto the edge cloud while maintaining low latency and high accuracy. Second, it collects and merges sensor data from these mobile clients and other infrastructure sensors near the edge that offer different perspectives onto the intersection. Merging accuracy benefits from sharing rich data streams so that perspectives can be accurately aligned based on a large set of feature points. Merging information from different perspectives allows vehicles to see into blind spots and other areas of the intersection that are occluded by objects. More complete information about traffic participant positions and trajectories allows improving efficiency and safety in advanced driver assistance as well as automated driving systems. The whole system consists of two parties: clients and an edge server, as illustrated in Figure 2.

**Clients.** Clients can be broadly considered as any mobile nodes with visual sensors that have connectivity to the edge cloud server. In EdgeSharing, these mobile clients can be split into two categories: producer and consumer, with some devices acting as both. Producers are mobile clients like advanced self-driving cars, street cameras or flying drones that continuously offload captured RGB frames and depth readings to the server to help the server localize position of objects in the 3D environment. Consumers are mobile clients such as

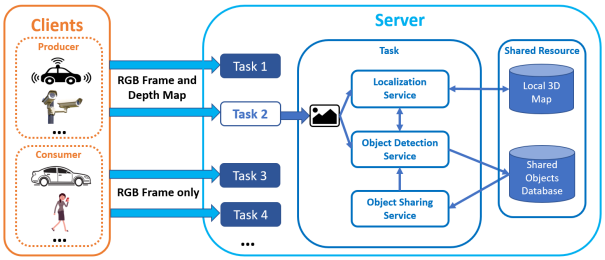



Fig. 2. System Design

smartphones, smartglasses, and commercial vehicles that keep receiving the information of surrounding objects within the local region by offloading the video frame captured by its equipped camera.

**Edge Server.** An edge server is a regional edge cloud or data center that gathers sensor readings of all participating mobile clients, detects objects from different perspectives, and provides such information back to mobile clients for safety services. As illustrated in Figure 2, the edge server consists of three key services: *Device Localization*, *Object Detection*, and *Object Sharing*. In order to benefit from the objects detected by different mobile clients, EdgeSharing needs to estimate accurate 3D pose information (location and orientation) of each participating client in the world coordinate system. While GPS and inertial sensor data provide low-cost location and orientation data, it is widely recognized to be noisy and inaccurate in urban canyons. The *Device Localization* procedure of EdgeSharing estimates the 3D transformation matrix with visual odometry techniques leveraging the offloaded frames from the client device. EdgeSharing redesigns a popular ORB-SLAM algorithm to work in this edge offloading scenario – the edge server collects the offloaded frames from mobile clients to generate the 3D map and uses it to help localize these devices. In the *Object Detection* service, the system first leverages a state-of-the-art object detection algorithm to detect locations of a new object in the image frame. EdgeSharing then uses the depth information provided by the producer client to estimate the 3D localization of each such object in its camera coordinate system, and then projects the position to the world coordinate system with the perspective projection from the *Device Localization* service. All detected object locations are stored in an object database on the edge server. The last component is *Object Sharing* service which shares the stored object locations in the database with nearby clients. EdgeSharing projects the position of each object in the database back to the client's coordinate system and returns this information back to the client. We further detail this system in the following subsections.

### A. Device Localization Service

To enable accurate object sharing in urban streets with dense traffic, EdgeSharing requires estimating the position and orientation of each participating client in the coverage region. This allows the system to calculate the relative transformation between two different objects and share the information with clients.

EdgeSharing leverages a popular SLAM framework (i.e. ORB-SLAM) as its device localization solution. We redesign the ORB-SLAM solution to generate a high-quality 3D feature map of the coverage region of an edge cloud server (e.g. an intersection) using offloaded images from participating clients and use this map to localize the participating mobile clients in this local region on-the-fly.

To build the feature map for *Device Localization*, we use a dedicated data collection vehicles to drive through the region from different directions for several times and offload captured frames to the edge server. The server uses the ORB-SLAM algorithm to generate 3D feature map either offline or online. There has been several similar research that constructs feature map using crowdsourced frames from travelers in the street [9]. Once the map is constructed, the edge server is able to provide *Device Localization* service to the mobile devices by matching the ORB feature points from their offloaded images to features in the 3D feature map. The system further optimizes the pose of the device with RANSAC and motion model constraints. The final output of the *Device Localization* service is a world to camera transformation matrix ($T_{cw}$) that is able to transform a point's 3D location between the world coordinate system and the camera coordinate system. This matrix is also refer to the combination of a rotation matrix ($R_{cw}$) and a translation matrix ($t_{cw}$) between the two coordinate system.

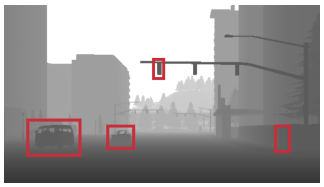
### B. Object Detection and Object Sharing Service

After localizing the 3D position and orientation of each client, EdgeSharing adopts an *Object Detection* service to detect objects in the field-of-view of those producer clients from their offloaded images and project these objects to the global coordinate system. Then the system uses an *Object Sharing* service to share locations of those objects to all consumer clients in the coverage of the edge server. This effectively extends the vision of traffic participants beyond their field of view. We first introduce *Object Detection* service, which contains two key components: Object Bounding Box Detection and Object 3D Localization.

**Object Bounding Box Detection.** EdgeSharing leverages state-of-the-art CNN based Object Detection models to identify object locations on each frame's pixel coordinate system. Specifically, we use a ResNet-50 based Faster RCNN Object Detection model to detect objects on each offloaded video frame. This model takes the raw RGB frame as the input and outputs the object types and their 2D bounding boxes in the frame's pixel coordinate system. The model is trained with Microsoft Coco dataset, which contains 91 different object types, including person, car, motorcycle, bicycle, bus, truck, traffic light, and different street signs that frequently appear in urban streets and useful for a variety of safety services.

**Object 3D Localization.** In order to share the detected object to other clients, EdgeSharing needs to project the location of each object from the producer's perspective to other consumer's perspective. To enable this, the Object 3D Localization process works in the following procedures: (1). The system estimates the depth of each object with respect

(a) Object detection on the RGB frame.



(b) Estimate depth of each detected object using the depth map of the frame.

Fig. 3. Object 3D Localization.

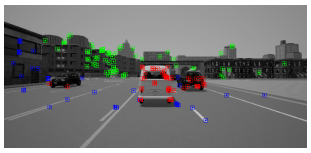


Fig. 4. Context Aware Feature Selection

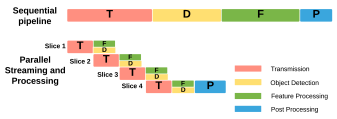



Fig. 5. Parallel Streaming and Processing

to the camera's coordinate system. This step can be achieved using depth sensors from the vehicle (e.g. Lidar, Radar, RGBD camera or stereo camera), or based on other monocular based depth estimation techniques [10], [11]. (2). The system then uses perspective projection to transform the location of all detected objects from the 2D pixel coordinate frame to the 3D world coordinate frame. Equation 1 shows the projection equation of projecting a point from the pixel coordinate system $(u, v)$ to the camera's coordinate system $(x_c, y_c, z_c)$, and then to the world coordinate system $(x_w, y_w, z_w)$. As shown in Figure 3, EdgeSharing uses the center point of each bounding boxes as the location of this object $(u, v)$ on the frame's pixel coordinate system, and uses the median depth value inside the bounding box as its depth towards the camera's coordinate system $(z_c)$. With the position information of the object and the intrinsic matrix $(K)$ of the camera, the system is able to transform each object to the camera's coordinate system. After that, EdgeSharing further calculates the position of each object in the world coordinate system leveraging the client's Extrinsic Matrix $(T_{cw})$ calculated by the *Device Localization* service. (3). All detected locations of objects are immediately stored in a Shared Object Database, a storage engine on the edge cloud that stores the real-time 3D object locations in its coverage area. The Shared Object Database gathers the location of each object from images offloaded by nearby clients, while providing this information for object sharing. To maintain a timely object sharing, each collected object will be expired if no position update is provided by the client after 50 ms from the time the object has been detected.

$$\begin{aligned} \lambda \begin{bmatrix} u & v & 1 \end{bmatrix}^T &= K \begin{bmatrix} x_c & y_c & z_c & 1 \end{bmatrix}^T \\ &= K T_{cw} \begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix}^T \end{aligned} \tag{1}$$

**Object Sharing.** Finally, EdgeSharing shares objects inside the Shared Object Database to the client to provide them more information beyond their line-of-sight. In particular, the system transforms all shared objects within its coverage region from the world coordinate system to the client camera's coordinate system based on the transformation matrix $(T_{cw})$ calculated from the *Device Localization* service. The server combines the shared object locations and send them back to consumer clients.

### C. System Optimization

In addition, we propose several optimization techniques to improve the performance of EdgeSharing. First, we design a *Context Aware Feature Selection* method to select only stable features on each frame to estimate the pose of the device in the *Device Localization* service (Section III). This method takes advantage of the powerful computation resource on the edge server to detect movable objects and uses the detected object bounding boxes to do the feature selection. Second, EdgeSharing adopts a *Collaborative Local Tracking* mechanism to reduce offloading bandwidth while maintaining a high tracking accuracy (Section IV). Last, we implement a *Parallel Streaming and Processing* pipeline to decrease the end-to-end latency of the whole offloading tasks by efficiently pipelining streaming and image processing processes in parallel (Section V).

## III. Context Aware Feature Selection

EdgeSharing applies a *Context Aware Feature Selection* mechanism to increase the accuracy of our device localization service in dense traffic scenarios. While SLAM based visual odometry method has been adopted as the key inside-out tracking method for mobile devices, it is still widely recognized to be vulnerable to moving objects in the scene [12]. This method relies on matching feature points between continuous captured frames and uses them with motion model constraints to derive instant changes over time. The key principle underlies this mechanism is that most of the matched feature points are not moving between frames in the world coordinate system. However, this is very hard to guarantee in dense traffic scenarios, where large amounts of moving objects (e.g. vehicles, pedestrians, etc.) exist. To overcome this challenge, we propose a *Context Aware Feature Selection* method to filter out potential moving features in the scene and only use stable feature matches to estimate the location changes. The intuition of this approach is that feature points located inside the detected object bounding boxes of movable objects are more likely to move than other background regions. By taking advantage of the *Object Detection* service, EdgeSharing directly filtered out the feature points that lie in the bounding boxes of potential moving objects. In particular, we consider the following objects as potential moving objects: car, bus, truck, motorcycle, bicycle, and person.

Figure 4 shows an illustration of our feature selection method when processing one of the offloaded frames. The green markers are the feature matches between the frame and feature points in the 3D map. The blue markers represent the matches with previous frames and red markers are the discarded matches due to they lie in the bounding boxes of detected potential moving objects. In the *Device Localization*

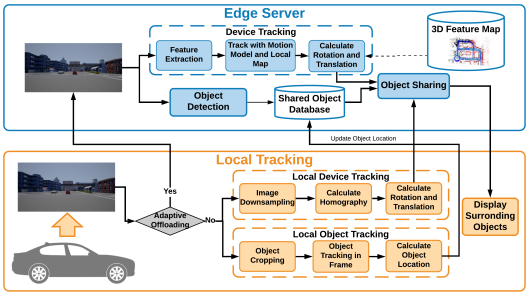



Fig. 6. Collaborative Local Tracking

service, the system only matches green and blue feature points with the 3D feature points and uses them to derive the location of the device. We further show how this mechanism can increase the localization accuracy in dense city traffic scenarios in Section VII.

## IV. Collaborative Local Tracking

EdgeSharing requires participants to keep offloading video frames to the server through the wireless link, which require significant bandwidth. To eliminate this large overhead, a natural way is to reduce the offloading frequency. However, this results in a low device localization accuracy due to the low update rate. To overcome this trade-off between the localization accuracy and bandwidth consumption, we propose a *Collaborative Local Tracking* to adaptively schedule frame offloading and apply local tracking on device to mitigate the localization quality degradation caused by the low offloading frequency. As shown in Figure 6, the offloaded frames still follow the regular procedures described in Section II, while the local frames go through a local tracking process to estimate the location changes of the client as well as the position of its detected objects. This *Collaborative Local Tracking* contains several key components: *Local Device Tracking*, *Local Object Tracking*, and *Adaptive Offloading*.

**Local Device Tracking.** As shown in Figure 6, the *Local Device Tracking* module has three components: (1) Image downsampling, (2) Homography calculation, and (3) Transformation calculation. In the first step, the system downsamples the raw frame to a smaller resolution (from 800x600 to 400*300) to reduce the time complexity of following feature extraction and matching. In the second step, the system calculates the homograpgy matrix between the current frame and the last frame, which contains the perspective transformation between two frames. In particular, the system first extracts ORB feature points from the current frame, and find feature matches between current frame with the last offloaded frame. The system then runs a RANSAC with projective motion model to get a group of reliable feature matches between two frames from the raw feature pairs. Finally, the system uses an SVD method to estimate the homography matrix between two frames. As shown in Equation 2, Homography matrix is a $3 \times 3$ matrix that is an image to image mapping matrix that is able to transform each pixel's homogeneous coordinate of the last frame ($[u, v, 1]^T$) to the corresponding coordinate on the current frame ($[u', v', 1]^T$).

$$\lambda \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = H \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (2)$$

With the homography matrix between two frames, the system extracts the rotation and translation matrix between two images, and uses them to build the transformation matrix $T_{c'c}$ that can transform a 3D point in the camera's homogeneous coordinate system of the last frame ($[x, y, z, 1]$) to the corresponding points in the current camera's homogeneous coordinate system ($[x', y', z', 1]$). After that, the system calculates the transformation matrix from the world coordinate system ($T_{c'w}$) to the camera's coordinate system using Equation 3.

$$T_{c'w} = T_{c'c} T_{cw} \quad (3)$$

With this *Local Device Tracking* method, EdgeSharing is able to localize the position of the client device in real-time without the necessity of offloading every frame to the edge server. For a local processed frame, the mobile client only needs to send the updated location of the device to the edge server, and the server can use this information to share objects from the database to this client.

**Local Object Tracking.** In addition to *Local Device Tracking*, EdgeSharing also enables object tracking when the frame is kept on the device for local processing. The *Local Object Tracking* has three main procedures. First, the system reuses the feature point matches between the current frame and the last frame to shift the bounding box of each object to the corresponding location in the current frame. Then the system uses the same method introduced in Section II to find the position of the object in the camera's coordinate frame using its new bounding box location and depth readings. After that, the location of the object can be projected to the world coordinate system using the transformation matrix $T_{c'w}$ calculated in the *Local Device Tracking* module. Finally, the client device transmits the device transformation matrix and all object locations to the edge server to update the share object database. This *Local Object Tracking* technique can help the producer client update the position of already detected object without continuously uploading frames to the edge server.

**Adaptive Offloading.** However, it is impossible to keep every frames process locally, since continuous frame by frame tracking on client devices leads to increasing drift overtime. The client device needs to periodically offload frames to the edge server to correct the localization drift using the 3D feature map. In addition, the edge server also generate fresh objects and their locations using its *Object Detection* service.

Therefore, we design an *Adaptive Offloading* mechanism to determine whether the client device should offload the current frame to the edge server or keep it for local tracking. The key intuition behinds this method is that we want to offload the frames that have large changes compare to the last offloaded frame. In EdgeSharing, *Adaptive Offloading* uses the number of feature matches between the last offloaded frame and the last frame to determine whether the last frame still have considerable amounts of feature matches with the last

offloaded frame. If the number of feature matches drops below the a threshold, Edgesharing can no longer maintain a high tracking accuracy on the client side. Therefore, the system should offload this frame to the edge server to process. On the other hand, if there are still decent amounts of matches with the last offloaded frame, the system processes the frame locally. We empirically pick 40 pairs as the threshold and report the result in Section VII.

## V. Parallel Streaming and Processing

EdgeSharing also strives to reduce the end-to-end latency of the transmission and processing pipeline. The whole processing pipeline consists of frame streaming and several cloud processing procedures that usually have an end-to-end latency of more than 50ms. Such long latency may significantly reduce the accuracy of localization and object sharing, since the location of objects may already change after the system get the new detection result. To provide users with an optimal experience, EdgeSharing adopts a *Parallel Streaming and Processing* pipeline to paralyze the processing pipeline of tasks on different resources of the edge cloud platform. This method results in a large decrease of end-to-end latency spend on each offloading task.

As shown in Figure 5, the baseline of the offloading pipeline consists of four key procedures: (1) Frame transmission from the client device to the edge server, (2) Object detection on the offloaded frame, (3) Feature processing on the offloaded frame (i.e. feature extraction and matching), and (4) Post processing, including camera pose estimation, object location estimation and object sharing. We observed that both the transmission and image processing on the edge server take considerable time and run sequentially, it ends up with a long end-to-end latency for the entire system. In EdgeSharing, the client device splits each offloaded frame into four slices and transmit them one by one. Once each slice arrives at the edge server, the system can immediately start feature extraction and object detection tasks on it without waiting for the whole frame to arrive. In addition, we also paralyze the procedures of object detection and feature processing by offloading the object detection to run on the onboard GPU. After all object detection and feature processing task finished, the system starts to the post processing stage and finish the rest of the task. *Parallel Streaming and Process* is able to achieve almost half latency reduction compared to the baseline approach.

## VI. Implementation

Our implementation is entirely based on commodity hardware and contains more than 5000 lines of code.

**Hardware Setup.** We emulate the whole system using a group of commodity hardware. On the client side, We use a mobile development board Nvidia Jetson TX2 as the client processing device. Jetson TX2 contains a 256-core NVIDIA Pascal GPU and a Dual-Core NVIDIA Denver 2 64-Bit CPU on board, which has similar computation power compared to today's high-end mobile phones, while much worse than the typical computation device of self-driving cars, such as the NVIDIA DRIVE series. This board also support up to 802.11ac wireless connection, which also satisfies the requirement of wireless video streaming.

On the edge cloud side, We use a workstation PC equipped with an Intel i7-6850K CPU and an Nvidia Titan XP GPU as our edge server to process all offloaded tasks. This PC also connects to a TP-Link AC1900 router through a 1Gbps Ethernet cable. This provides the wireless communication channel between the server and client devices. Both devices run Ubuntu 16.04 OS.

**Software Implementation.** The whole system is built upon several open sourced projects and libraries, including ORB-SLAM2 [12], OpenCV [13], Nvidia JetPack [14], Nvidia Multimedia API [15], Nvidia TensorRT [16], and the Nvidia Video Codec SDK [17]. We describe the implementation of the edge server and the mobile client as follow.

The edge server-side implementation contains three key components: *Frame Decoding*, *Object Detection* and *Frame Tracking*, which are designed to run in three different threads to avoid blocking each other. In *Frame Decoding* thread, the edge server keeps decoding frame slides transmitted from the client device and feeds them to the *Object Detection* thread and the *Frame Tracking* thread immediately after completion. The *Object Detection* thread is implemented using the Nvidia TensorRT, which is a high-performance deep learning inference optimizer designed for Nvidia GPUs. To push the limit of inference latency on the server side PC, we use the INT8 calibration tool [18] in TensorRT to optimize the object detection model, and achieves 3-4 times latency improvement on the same setup. The *Frame Tracking* thread is developed based on the ORB-SLAM2 algorithm [12]. We modify the original project to fit our requirement of parallel processing and feature selection. In particular, we create a Frame object for each frame slide and use it to extract key points and ORB descriptors on each slide. Then the feature matching is performed by matching each feature point on a slide with features on the entire last frame. To implement the *Context-aware Feature Selection* method, we perform a filter on the match feature pairs in the post processing stage, leveraging the object bounding boxes detected in the *Object Detection* thread.

We implement the client side functions on the Nvidia Jetson TX2 with its JetPack SDK and OpenCV. The implementation follows the design flow in Figure 2 and Figure 6. We use OpenCV to achieve the image downsampling, cropping, and various transformation matrix calculation. To implement the Parallel Streaming and Inference module, we enable the slice mode for the video encoder and use the setSliceLength() function with a proper length to let the encoder split a frame into four slices. Each slide is immediately transmitted out to the edge server to process.

## VII. Evaluation

In this section, we evaluate the performance of the EdgeSharing system in terms of its device localization accuracy,

(a) Dataset 1: 4-way Stop Sign. (b) Dataset 2: City Intersection. (c) Dataset 3: LA Intersection.

Fig. 7. Experimental datasets.

object sharing latency, bandwidth consumption, and end-to-end latency. The results demonstrate that our system is able to achieve both the high accuracy and the low latency requirement for device localization and object sharing in urban streets. The system is able to achieve a mean vehicle localization error of 0.28-1.27 meters, an object sharing accuracy of 82.3%-91.4%, and a 54.68% object awareness increment in urban streets and intersections. In addition, the proposed optimization techniques are able to reduce 70.12% of bandwidth consumption and reduce 40.09% of the end-to-end latency.

### A. Experiment Setup

We use the setup and implementation described in Section VI to conduct experiments. Two different object sharing tasks are designed to evaluate the performance of our system: a vehicle to vehicle object sharing task and an infrastructure to vehicle object sharing task. In V2V object sharing scenario, the EdgeSharing server collects object position from producer vehicles with both cameras and depth sensors and provides object sharing service to all consumer vehicles with only RGB cameras. In the infrastructure to vehicle sharing scenario, the EdgeSharing gets object position from a street camera, and shares detected objects to all vehicles with RGB cameras.

For repeatable experiments, we use three collected datasets as described in the section VII-B to conduct our experiment. During the experiment, we simulate the whole offloading and sharing process following the workflow described in Section II.

### B. Dataset Description

We collected three different datasets to evaluate the performance of our system. To first evaluate the system under perfect data inputs, we collect two datasets using an open-source simulator for autonomous driving called CARLA [19]. CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. We carefully choose one 4-way stop sign intersection and another city intersection from CARLA's map 3 and map 5 to collect dataset for our experiment. Figure 7(a) and Figure 7(b) show the top view of two intersections. During the data collection, we spawn 400 vehicles equipped with front view cameras in random locations on the map. In the 4-way stop sign intersection, we aim to evaluate V2V object sharing, therefore, each vehicle is equipped an additional depth camera to record the corresponding depth for each pixel on the RGB image. In map 5, we put a street camera and another depth camera at the capturing position of Figure 7(b) to evaluate I2V sharing scenario. The street camera keeps capturing 120-degree field of view and 1280x720 resolution frames at 30 fps, while vehicles are capturing frames 90 fov 800x600 resolution frames at 30 fps. All images are recorded with synchronized time-stamps for our evaluation. We also log the position and orientation data of each participant as groundtruth for the following evaluation.

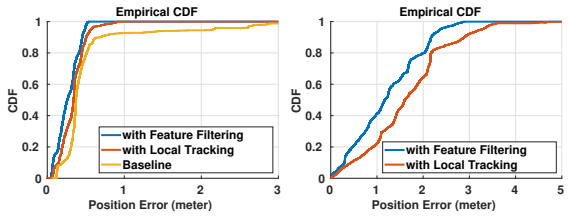
| Dataset | Approaches | Position (meter) | Orientation (degree) |
|---|---|---|---|
| 4-way Stop Sign | Baseline | 0.65 | 2.01 |
| | + Feature Selection | 0.32 | 0.57 |
| | + Local Tracking | 0.39 | 0.82 |
| City Intersection | Baseline | 0.58 | 1.88 |
| | + Feature Selection | 0.28 | 0.52 |
| | + Local Tracking | 0.37 | 0.76 |
| LA Intersection | Baseline | Fail | N/A |
| | + Feature Selection | 1.27 | N/A |
| | + Local Tracking | 1.68 | N/A |

TABLE I
MEAN POSITION AND ORIENTATION ERROR OF THREE DIFFERENT APPROACHES ON THREE DATASETS.

To further evaluate the performance of EdgeSharing in real traffic scenarios, we collect another 120-hour dataset in an intersection in Los Angeles, CA. In this dataset, we use GoPros mounted at the bottom center under the windshield to record the full driver's front view videos with 1280×720 resolution at 30 fps. The 30Hz GPS readings are also recorded using the embedded GPS sensor in GoPros. During the data collection, ten different drivers were driving together as a fleet in an urban street back and forth for 30 times. Figure 7(c) show an frame of the video captured by the third vehicle of the fleet.

### C. Accuracy of Device Localization

EdgeSharing is able to achieve high accuracy of device localization under many different urban traffic scenarios. We first measure the device localization accuracy of EdgeSharing in three approaches: the baseline solution (Baseline), our solution with the *Context-aware Feature Selection* method (+ Feature Selection), and our solution with both the *Context-aware Feature Selection* and the *Collaborative Local Tracking* method (+ Local Tracking). The baseline approach follows the standard pipeline we introduced in Section II. We evaluate the detection accuracy of EdgeSharing with two key metrics: the position error and the orientation error of the device, as shown in Table I. For the 4-way stop sign intersection and the city intersection dataset, we create a dedicated map generation vehicle in the simulator and let it drive through the intersection from different directions and lines for several times. The captured 30fps RGB frames and depth maps are used to generate the map. For the LA intersection dataset, we use two videos from the leading vehicle to construct the feature map. The map is then used to help the other vehicles localize themselves in the same intersection. As shown in Table I, the proposed *Context-aware Feature Selection* is able to increase the localization accuracy for the first two datasets. In the 4-way stop sign dataset, EdgeSharing reduces the mean position error from 0.65 meters to 0.32 meters, and reduces the mean orientation error from 2.01 degrees to 0.57 degrees. Similarly, EdgeSharing also reduces the mean position error and mean orientation error to 0.28 meters and




(a) Position error of the city intersection dataset. (b) Position error in the LA dataset.

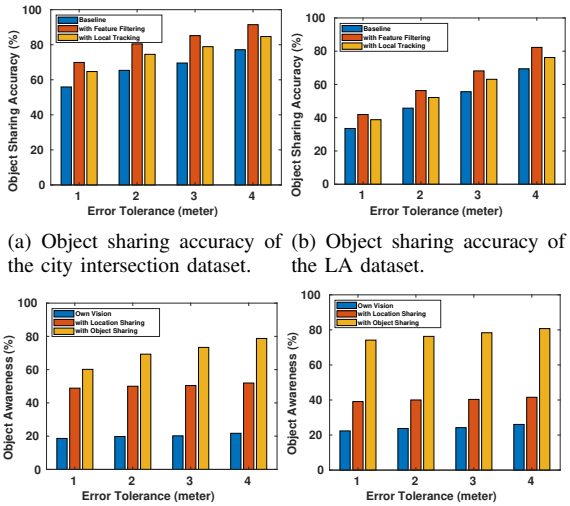
Fig. 8. CDF of device localization of EdgeSharing.

0.52 degrees correspondingly. For the LA dataset, we only evaluate the performance of position error since we did not collect orientation groundtruth. For the baseline approach, we find that EdgeSharing fails to build a meaningful feature map due to the interference of moving objects and the vehicle's own dash. With the *Context-aware Feature Selection* method, EdgeSharing is able to achieve a pretty good performance of around 1.27 meters. In addition, we show that the proposed *Collaborative Local Tracking* method does not significantly reduce the localization accuracy, as listed in Table I.

We further show the cumulative distribution function (CDF) of localization error for the city intersection dataset in Figure 8(a). The 95 percentile of the position error distribution is 0.49 meter for the approach with *Context-aware Feature Selection* and 2.2 meters for the baseline approach. Similarly, Figure 8(b) shows the CDF of position error in the LA dataset. The result demonstrates that our *Context-aware Feature Selection* technique is able to reduce the localization error and keep most of the tracking error in a very low value range.

### D. Accuracy of Object Sharing

Our system is able to achieve high localization accuracy for object sharing in urban streets, and provides higher object awareness to individual vehicles to enlarge their vision. We first estimate object localization accuracy on the first and third dataset to understand what percentage of detected objects is correct among all detected objects. To calculate the accuracy, we calculate the 3D location of the object in the world coordinate system use the object 3D localization method described in section II-B, and check whether we can find a vehicle from the collected groundtruth data in an given error tolerance range. As shown in Figure 9(a), we can observe the approach with *Context Aware Feature Selection* is able to achieve the highest sharing accuracy compared to other methods for the same error tolerance. And for the same approach, the accuracy increases as the error tolerance increases. The sharing accuracy of the *Context Aware Feature Selection* approach with an error tolerance of 4 meters is able to achieve 91.44%. For the LA dataset, we only calculate the sharing accuracy of our data collection vehicles in the scene. The result of *Context Aware Feature Selection* approach and *Collaborative Local Tracking* approach are depicted in Figure 9(b), which shows our system is able to achieve 82.3% sharing accuracy with 4-meter error tolerance in the dataset collected in real scenarios with imperfect sensors. Note that the wrong detection can be considered as the combination effect of device localization




(a) Object sharing accuracy of the city intersection dataset. (b) Object sharing accuracy of the LA dataset.

(c) Object awareness of the city intersection dataset. (d) Object awareness of the 4-way stop sign dataset.

Fig. 9. Accuracy of Object Sharing.

error, depth sensor error, object detection algorithm error, and the error caused by vehicle shape.

Second, we want to understand how many more objects can one vehicle be aware of with the support of EdgeSharing. To achieve this, we define *object awareness* as the percentage of objects that one vehicle is aware of in the region of the intersection. We compute the *object awareness* of each vehicle in the intersection at each timestamp with three approaches: own vision, location sharing only, and location and object sharing in three different error tolerance. In the own vision approach, each vehicle only knows the objects within its field of view. With location sharing, the producers in the street are able to store their position in the shared object database for sharing. With object sharing, these producers further store the locations of the detected object to the database. In the experiment of the 4-way stop sign dataset, we randomly assign 25 vehicles as producer and calculate the *object awareness* for the rest of vehicles in the region of the intersection. For the city intersection dataset, we only use the street camera as the producer and calculate the *object awareness* for all vehicles in the region of the intersection. In Figure 9(c) and Figure 9(d), we can find that *object awareness* is significantly increasing with the support of location sharing and object sharing from the producer clients.

### E. Bandwidth Consumption

EdgeSharing uses the *Collaborate Local Tracking* method to significantly reduce the bandwidth consumption while maintaining a low localization error. To demonstrate this, we tune the offloading threshold (feature matching pair number) used in *Collaborate Local Tracking* to illustrate the relationship between localization error and the bandwidth consumption of our system. Figure 10 shows this relationship of the city intersection dataset. Compared to the fully offloaded scenarios, EdgeSharing is able to reduce the bandwidth consumption, while only has a slight increment on the localization error. For example, if we choose to use 40 matching pairs as the

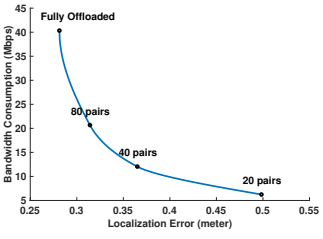



Fig. 10. Localization error vs Bandwidth consumption for the city intersection dataset.

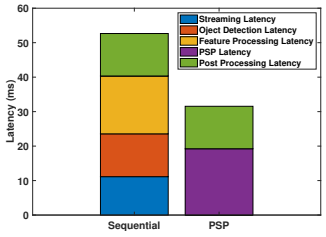



Fig. 11. Latency comparison between the sequential approach with the PSP approach.

offloading threshold, we can reduce the bandwidth consumption of the urban intersection dataset from 40.34 Mbps to only 12.05 Mbps, while only increasing the localization error from 0.2813 to 0.365 meters. Similarly, we also observe the same pattern for the LA dataset we collected in real cases.

### *F. Latency*

EdgeSharing includes a *Parallel Streaming and Processing* (PSP) pipeline to reduce the end-to-end latency of the system. We conduct a real-time experiment on the city intersection dataset using the setup introduced in Section VI, and compare the sequential approach and the PSP approach. As shown in Figure 11, we divide the offloading latency into streaming, object detection, feature processing, and post processing latency for the sequential approach, and use a PSP latency for the third method, because the processes run in parallel. The streaming latency contains time spending on encoding (1.6ms on Jetson TX2), transmission, and decoding tasks (less than 1ms on edge server). In the sequential approach, the mean end-to-end latency to finish the entire pipeline of one frame is 52.66ms, while our solution requires only 31.55ms, which makes it possible for the system to deliver 30fps object sharing experience to the participated clients.

## VIII. Related Work

**Collaborative Sensing in Connected Vehicles.** Connected vehicles allows cars to share information and sensor reading to other devices both inside as well as outside the vehicle. Traditional vehicular communication systems (e.g. DSRC) has been used to solve plenty of traffic issues by sharing safety messages to other nodes on the road [20]–[24]. Many recent works further explore to use millimeter wave to provide much higher bandwidth for the vehicular communication [25]–[27]. With such great resources, there have been other work that focuses on sharing the camera perceptions through V2V networks not limited for autonomous vehicles but also for other Advanced Driving Assistance Systems (ADAS). The see-through system [28] and AVR system [8] share visions between vehicles through wireless connections to extend their field of view. Compared to direct sharing methods that require vehicles to have large computation resources on-board, edge cloud-based object sharing system like EdgeSharing has the benefits of easier system upgrading and maintenance, as well as the better computation resources for more sophisticated algorithms.

**Vision Task Offloading.** Offloading computation-intensive tasks to cloud or edge cloud infrastructures is a feasible way to enable continuous vision analytics on power and computation constraint devices. Chen et al. [29] evaluate the performance of seven edge computing applications in terms of latency. DeepDecision [30] designs a framework to decide whether to offload the object detection task to the edge cloud or do local inference based on the network conditions. Lavea [31] offloads computation between clients and nearby edge nodes to provide low-latency video analytics. VideoStorm [32] and Chameleon [33] achieve higher accuracy video analytics with the same amount of computational resources on the cloud by adapting the video configurations. Several other works [34]–[36] also demonstrate the benefit of using edge cloud to accelerate AR systems. These works have demonstrated the benefit of using edge offloading in many different applications, which also supports the idea of EdgeSharing.

**Device Localization.** Accurate device localization is a key component of EdgeSharing system. Except from GPS signals, Some other techniques such as inertial sensor [37], wireless signal [38] and visual sensors [12] on the device to improve the localization accuracy. Among these techniques, visual odometry techniques have shown its great performance in tracking devices in the environment where rich visual features exist. Several popular visual SLAM algorithms, including ORB-SLAM2 [12] and LSD-SLAM [39], have shown their superior performance in mapping and localization in small space such as an indoor environment with very few moving objects. These techniques have also been adopted in commercial AR or VR platforms, such as ARKit [3], ARCore [4], Hololens [5] and Oculus Go [6]. In this work, we use the existing ORB-SLAM2 as our localization solution.

**Adaptive Video Streaming.** Several 360-degree video streaming works [40]–[42] also adopt the idea of RoI encoding to reduce the latency and bandwidth consumption of the streaming process. Adaptive video streaming techniques have also been adopted by mobile gaming [43], [44] and virtual reality system [45], [46] to achieve high-quality experience on mobile thin clients.

## IX. Conclusion

In this paper, we introduced EdgeSharing, a first collaborative localization and object sharing system leveraging the resources of an edge cloud platform and the visual inputs from participating mobile clients (e.g., vehicles and pedestrians). In EdgeSharing, the edge cloud uses a 3D feature map of its coverage region to provide accurate localization services to the client devices passing through this region. Additionally, EdgeSharing also leverages the computation power on the edge cloud to detect object locations on the images offloaded by participating clients, localizes them in 3D space, and shares them with other clients in the same region. With EdgeSharing, nearby vehicles can learn extra object (e.g., traffic participant) locations from the edge cloud, which are outside the vehicle's field of view, which improves their situational awareness and safety.

## REFERENCES


[1] Analysis of Fatal Motor Vehicle Traffic Crashes and Fatalities at Intersections, 1997 to 2004. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/810682/.

[2] TensorFlow Lite Performance. https://www.tensorflow.org/lite/models.

[3] Apple ARKit. https://developer.apple.com/arkit/.

[4] Google ARCore. https://developers.google.com/ar/.

[5] Microsoft HoloLens. https://www.microsoft.com/en-us/hololens/.

[6] Oculus Go. https://www.oculus.com/go/.

[7] Google Map's AR App. https://www.theverge.com/2019/2/10/18219325/google-maps-augmented-reality-ar-feature-app-prototype-test.

[8] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. Avr: Augmented vehicular reality. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 81–95. ACM, 2018.

[9] Fawad Ahmad, Hang Qiu, Xiaochen Liu, Fan Bai, and Ramesh Govindan. Quicksketch: Building 3d representations in unknown environments using crowdsourcing. In *2018 21st International Conference on Information Fusion (Fusion)*, pages 2314–2321. IEEE, 2018.

[10] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

[11] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[12] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[13] OpenCV. https://opencv.org/.

[14] Nvidia JetPack. https://developer.nvidia.com/embedded/jetpack/.

[15] Nvidia Multimedia API. https://developer.nvidia.com/embedded/downloads/.

[16] Nvidia TensorRT. https://developer.nvidia.com/tensorrt/.

[17] Nvidia Video Codec. https://developer.nvidia.com/nvidia-video-codec-sdk.

[18] Fast INT8 Inference with TensorRT 3. https://devblogs.nvidia.com/int8-inference-autonomous-vehicles-tensorrt/.

[19] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[20] Qing Xu, Tony Mak, Jeff Ko, and Raja Sengupta. Vehicle-to-vehicle safety messaging in dsrc. In *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, pages 19–28. ACM, 2004.

[21] Fan Bai and Hariharan Krishnan. Reliability analysis of dsrc wireless communication for vehicle safety applications. In *2006 IEEE intelligent transportation systems conference*, pages 355–362. IEEE, 2006.

[22] Ning Lu, Nan Cheng, Ning Zhang, Xuemin Shen, and Jon W Mark. Connected vehicles: Solutions and challenges. *IEEE internet of things journal*, 1(4):289–299, 2014.

[23] Khadige Abboud, Hassan Aboubakr Omar, and Weihua Zhuang. Interworking of dsrc and cellular network technologies for v2x communications: A survey. *IEEE transactions on vehicular technology*, 65(12):9457–9470, 2016.

[24] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. Carmap: Fast 3d feature map updates for automobiles. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 1063–1081, 2020.

[25] Junil Choi, Vutha Va, Nuria Gonzalez-Prelcic, Robert Daniels, Chandra R Bhat, and Robert W Heath. Millimeter-wave vehicular communication to support massive automotive sensing. *IEEE Communications Magazine*, 54(12):160–167, 2016.

[26] Vutha Va, Takayuki Shimizu, Gaurav Bansal, Robert W Heath Jr, et al. Millimeter wave vehicular communications: A survey. *Foundations and Trends® in Networking*, 10(1):1–113, 2016.

[27] Marco Giordani, Andrea Zanella, and Michele Zorzi. Millimeter wave communication in vehicular networks: Challenges and opportunities. In *2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–6. IEEE, 2017.

[28] Pedro Gomes, Fausto Vieira, and Michel Ferreira. The see-through system: From implementation to test-drive. In *2012 IEEE vehicular networking conference (VNC)*, pages 40–47. IEEE, 2012.

[29] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, et al. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 14. ACM, 2017.

[30] Xukan Ran, Haoliang Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *INFOCOM. IEEE*, 2018.

[31] 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.

[32] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, pages 377–392. USENIX Association, 2017.

[33] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 ACM SIGCOMM Conference*. ACM, 2018.

[34] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *MobiCom*. ACM, 2019.

[35] Xukan Ran, Carter Slocum, Maria Gorlatova, and Jiasi Chen. Sharear: Communication-efficient multi-user mobile augmented reality. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 109–116, 2019.

[36] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V Krishnamurthy, and Amit K Roy-Chowdhury. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 96–109, 2019.

[37] Cheng Bo, Xiang-Yang Li, Taeho Jung, Xufei Mao, Yue Tao, and Lan Yao. Smartloc: Push the limit of the inertial sensor based metropolitan localization using smartphone. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 195–198. ACM, 2013.

[38] Jin Wang, Nicholas Tan, Jun Luo, and Sinno Jialin Pan. Woloc: Wifi-only outdoor localization using crowdsensed hotspot labels. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[39] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[40] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360-degree streaming for smartphones. pages 482–494, 2018.

[41] Xiufeng Xie and Xinyu Zhang. Poi360: Panoramic mobile video telephony over lte cellular networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 336–349. ACM, 2017.

[42] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6. ACM, 2016.

[43] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 151–165. ACM, 2015.

[44] Eduardo Cuervo, Alec Wolman, Landon P Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu, and Madanlal Musuvathi. Kahawai: High-quality mobile gaming using gpu offload. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 121–135. ACM, 2015.

[45] Eduardo Cuervo, Krishna Chintalapudi, and Manikanta Kotaru. Creating the perfect illusion: What will it take to create life-like virtual reality headsets? 2018.

[46] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 68–80. ACM, 2018.