CARS: Collaborative Augmented Reality for Socialization

Wenxiao Zhang
Hong Kong University of Science and Technology
Hong Kong SAR, China
wzhangal@cse.ust.hk

Bo Han
AT&T Labs Research
Bedminster, New Jersey
bohan@research.att.com

Pan Hui
University of Helsinki
Hong Kong University of Science and Technology
panhui@cse.ust.hk

Vijay Gopalakrishnan
AT&T Labs Research
Bedminster, New Jersey
gvijay@research.att.com

Eric Zavesky
AT&T Labs Research
Bedminster, New Jersey
ezavesky@research.att.com

Feng Qian
Indiana University
Bloomington, Indiana
fengqian@indiana.edu

ABSTRACT

As Augmented Reality (AR) ties closely to the physical world, its users looking at overlapped scenes are likely to be in the vicinity of each other, which naturally enables the collaboration and interaction among them. In this paper, we propose CARS (Collaborative Augmented Reality for Socialization), a framework that leverages the social nature of human beings to improve the user-perceived Quality of Experience (QoE) for AR, especially the end-to-end latency. CARS takes advantage of the unique feature of AR to support intelligent sharing of information between nearby users when it is feasible. It brings various benefits at the user, application and system levels, *e.g.,* reduction of end-to-end latency and reuse of networking and computation resources. We demonstrate the efficacy of CARS through a preliminary evaluation based on a prototype implementation.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing design and evaluation methods**; • **Networks** → *Cloud computing*; • **Information systems** → *Image search*;

KEYWORDS

Augmented reality, collaborative augmented reality, cloud offloading, device to device communication

ACM Reference Format:
Wenxiao Zhang, Bo Han, Pan Hui, Vijay Gopalakrishnan, Eric Zavesky, and Feng Qian. 2018. CARS: Collaborative Augmented Reality for Socialization. In *HotMobile '18: 19th International Workshop on Mobile Computing Systems & Applications, February 12–13, 2018, Tempe , AZ, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3177102.3177107

HotMobile '18, February 12–13, 2018, Tempe , AZ, USA

ACM ISBN 978-1-4503-5630-5/18/02…$15.00
https://doi.org/10.1145/3177102.3177107

1 INTRODUCTION

Augmented Reality (AR) enhances the physical world by creating virtual annotations to augment one's perception of reality. AR systems can *recognize* objects inside the camera view of a mobile device and *augment* them with annotations [5, 11]. AR has found applications in various areas, such as training [13], healthcare [20], and communication [22]. However, most AR applications operate independently in a standalone way. Thus, the cooperation and interaction among mobile users running the same application are largely neglected.

With the recent advances of communication technologies (*e.g.,* 5G and 802.11ac/ad) and AR devices (*e.g.,* Microsoft HoloLens [19]), AR applications will become prevalent and be widely adopted by consumers and businesses. For example, Apple has released the ARKit [3] in iOS 11 which allows developers to create AR experiences using iPad and iPhone. The increasing popularity of AR offers tremendous opportunities for collaborations among its users. The key reason is that, by its design AR ties tightly with the real world, a unique feature that other applications such as virtual reality do not have. As a result, its users viewing overlapped scenes are in the close proximity and can communicate locally to share their common interests.

In this paper, we advocate to *leverage the social nature of human beings* for enabling the coordination and collaboration of sharing the results of computation intensive AR tasks, high quality AR annotations, and real-time annotation modifications by users. To this end, we propose CARS which, to the best of our knowledge, is the first collaborative AR framework for cloud-based AR. CARS supports instantaneous socialization (*e.g.,* real-time user interactions) and those experiences spread over time via local caches.

CARS brings various benefits at different levels. At the *user* level, it satisfies the general need of human beings for collaborations and interactions and creates an immersive user experience for AR. At the *application* level, CARS allows the exchange of invaluable information, for example, regarding the surrounding environment during a fire hazard for first responders with a single-minded focus. At the *system* level, it improves the efficiency of AR by reducing end-to-end latency, reusing computation resources in the cloud, saving mobile data usage, *etc.*

We illustrate the high level idea and benefits of CARS using an example in Figure 1. Suppose Alice and Bob are visiting a gallery and are appreciating two side-by-side paintings, Vincent and Mona

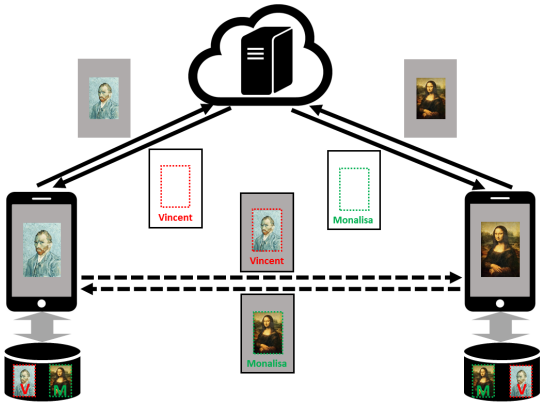


Figure 1: An illustrative example of CARS. Users can collaboratively perform object recognition: they can exchange results that each has received from the cloud.

Lisa, separately. They get the results of an AR application (*i.e.,* recognition result and annotation content) from the cloud. Now they change their positions, Alice moving to Mona Lisa and Bob to Vincent. Instead of running cloud-based recognition again which may waste mobile data usage and computation resources in the cloud, they can *collaboratively exchange the results* from previous runs via Device-to-Device (D2D) communications. We will discuss other mechanisms beyond D2D in § 5.

CARS is not limited to indoor environments and *does not require localization.* It is also helpful for outdoor scenarios, such as city sightseeing. Note that localization alone is not enough for AR. Even if we know which painting a user is looking at via localization, it cannot tell us the position of that painting within a camera view for augmentation.

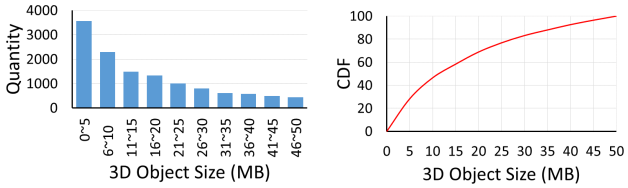
Although the above idea sounds straightforward, there are several practical issues in order to realize CARS. The key challenge is that CARS should maintain, or better improve, the user-perceived QoE (especially the end-to-end latency), when processing locally the shared information from others. Another hurdle is the support of interactions among users, which requires the synchronization of annotations and their changes made by users. To address these challenges, we carefully design the architecture and key components of CARS to make them *lightweight yet efficient and suitable* for mobile devices (§ 3). Our design makes it feasible to collaborate among users by *performing object recognition locally* based on the results from the cloud. We build a prototype of CARS which demonstrates it can reduce object-recognition latency by up to 40%, compared to cloud-based AR (§ 4).

2 BACKGROUND

In this section, we introduce how mobile AR works, by summarizing its pipeline, which was shown in Figure 1 of our previous work [30], and cloud-offloading feature of AR.

2.1 Pipeline of Mobile AR

A typical pipeline of mobile AR systems starts with *Object Detection* that identifies the Regions of Interest (ROIs) for target objects in



(a) Bar Graph **(b) CDF**

Figure 2: Size distribution of typical 3D objects for annotation (Source: Unity Asset Store).

a camera frame. For each ROI, *Feature Extraction* is the next step which extracts its feature points. The third step is *Object Recognition* that determines the original image for the target ROI stored in a database of to-be-recognized objects.

We utilize the classic image retrieval technology for object recognition. To compress raw feature points, we first build an offline probabilistic model (*e.g.,* Gaussian Mixture Model, GMM [25]) based on the feature points of all images in the database. Using this model, we encode the feature descriptors of an image into a compact representation (*e.g.,* Fisher Vector, FV [23]). We then store all compact feature representations of the images using a hash function (*e.g.,* Locality Sensitive Hashing, LSH [7]) for faster retrieval. Upon receiving an object recognition request, we process the request frame using the same procedure as described above, to get its compact feature representation. We then check its nearest neighbors in the hash table for the best matching result.

Template Matching validates object recognition result and calculates the pose of the target. *Object Tracking* takes the pose as its input with the goal of avoiding object recognition for every frame. Finally, *Annotation Rendering* renders the virtual content determined by the recognition result to augment the target object. Annotation content is usually a 3D object for achieving a better user experience. We retrieve the size of around 12,000 3D objects from the Asset Store of Unity [28] and plot the distribution and its CDF in Figure 2. The size of a 3D object ranges from a few hundreds of kilobytes to tens of megabytes. The large size of these 3D annotations is one of the motivations to enable collaborative sharing of them among users.

2.2 Cloud-Based Mobile AR

The performance of the current generation of mobile devices is still restricted by their computation power and battery life. AR systems can potentially offload some tasks to the cloud, in order to reduce the computation overhead on mobile devices. There are two common offloading scenarios. AR systems can offload tasks starting from object detection by sending camera frames to the cloud [11], or they can run object detection and feature extraction locally on a mobile device and upload the extracted feature points to the cloud for object recognition [12].

CARS inherits the benefits from cloud offloading of AR. First, image retrieval algorithms are usually computation intensive. Offloading them to the cloud can significantly reduce the end-to-end latency [5]. Second, the size of digital annotations may be large, as shown in Figure 2. It is challenging to store them for a dataset with a reasonable size (*e.g.,* 1,000 images) locally on mobile devices.

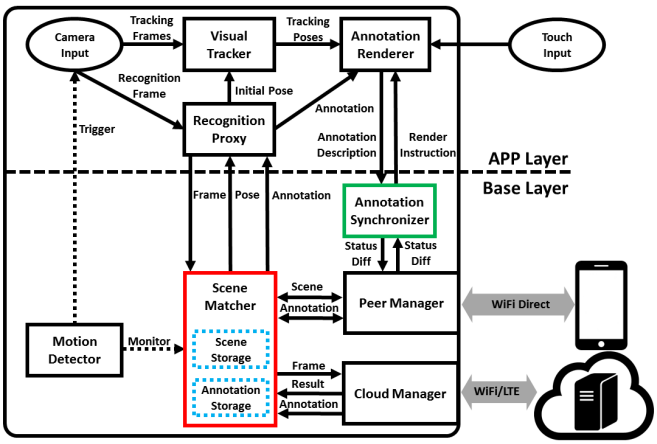


Figure 3: System architecture of CARS framework (dotted lines are for control messages). Its two key components are Scene Matcher and Annotation Synchronizer.

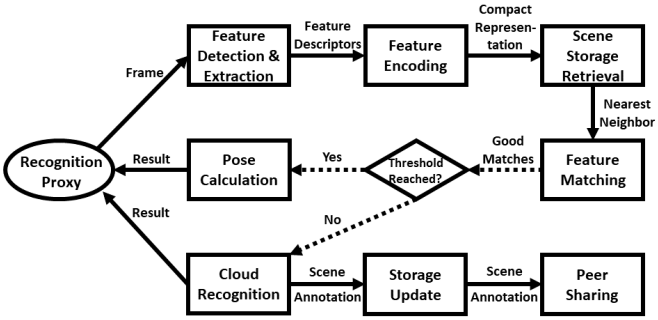


Figure 4: Workflow of Scene Matcher (We use dotted lines for control messages).

3 CARS SYSTEM DESIGN

In this section, we present an overview of CARS and describe its two key building blocks: the APP and Base layers.

3.1 CARS Overview

Using the pipeline in § 2.1 directly in CARS is not beneficial, as our previous work [30] has demonstrated that it takes several seconds for image retrieval on mobile devices, much longer than cloud-based AR. We show the system architecture of CARS in Figure 3. The major difference between the APP layer and the traditional AR applications (as elaborated in § 2.1) is that it does not handle object recognition itself, which is offloaded to the Base layer. The Base layer hides the details of object recognition and annotation download from the APP layer through a *simplified and lightweight* object-recognition engine.

The main benefit of decoupling the APP and Base layers is that they can evolve independently of each other. For example, we can make the Base layer lightweight and feasible for collaboratively exchanging annotations based on performing locally object recognition. It also makes the integration of traditional AR applications into CARS easy. They can send recognition requests to Recognition Proxy without caring about the operations happening underneath.

3.2 The APP Layer

The APP layer retrieves camera frames for object recognition and tracks recognized objects. It has three components: Recognition Proxy, Visual Tracker and Annotation Renderer.

Recognition Proxy is the bridge between the APP and Base layers for object recognition. Its input is a camera frame and the response contains the pose of the target object in that frame. It offloads the recognition to the cloud for improved latency performance and for supporting a large database of images. If the same object has been processed by the cloud for other nearby users and the result is available in the local cache, the Base layer returns it directly to Recognition Proxy without involving the cloud, as we will explain in § 3.3.

Visual Tracker tracks the pose of a recognized target when camera view changes. It is initialized by Recognition Proxy. After that Visual Tracker takes continuous camera frames as input and calculates the object pose frame-by-frame. Object pose contains three dimensions of translation and three dimensions of rotation (*i.e.,* the so-called Six Degrees of Freedom, 6DoF). We utilize feature points to calculate the object pose in each frame, and optical flow tracking [9] to track the feature points in the frame sequence.

Annotation Renderer augments the recognized objects using the virtual content which will be either downloaded directly from the cloud or fetched from others. It calculates a 6DoF pose of an annotation for rendering. We use a 3D graphics engine to render the annotation and to align it precisely with the physical target object. Annotation Renderer also reacts to users' touch input and instructions from Annotation Synchronizer of the Base layer for changes made by others, and modifies annotations accordingly.

3.3 The Base Layer

The Base layer handles object recognition and annotation download and synchronization for the APP layer. It has five components: Motion Detector, Scene Matcher, Annotation Synchronizer, Peer Manager and Cloud Manager.

Motion Detector indicates that a user intends to recognize the object inside the current camera frame. It triggers object recognition by avoiding specific user commands such as screen touch or voice input, which provides a seamless user experience. Object recognition will be initiated when a mobile device is in a relatively steady and vertical status, which filters out blurry images and camera frames with irrelevant objects (*e.g.,* when the camera is facing the ceiling). In order to ensure the accuracy of object recognition, Motion Detector continues monitoring the user's hand movement. It ignores the result if there is a significant movement during the recognition which may cause a mismatch between the request frame and the recognition result.

Scene Matcher is the core of the Base layer. We show its workflow in Figure 4. Upon receiving a recognition request from the APP layer, Scene Matcher first tries to find a local match by computing the feature descriptors and their compact representation of the request frame. It then calculates the distance between this compact feature representation and those in Scene Storage. Using the pair of feature-point sets from the request frame and the local

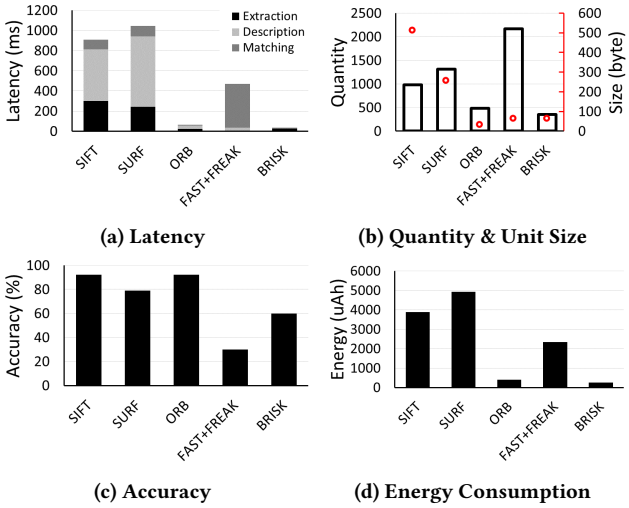


Figure 5: Performance comparison of various algorithms using `OpenCV4Android` library on a Xiaomi MI5 phone.

match, Scene Matcher determines the homography between them to find their geometric transformation and adjusts the corresponding recognition result from the local match accordingly.

Scene Matcher offloads object recognition to the cloud when it cannot find a local match, following the procedure described in § 2. When the recognition result is returned from the cloud, besides used by the APP layer, Scene Matcher also inserts it with the request frame into Scene Storage. Similarly, the request frames and results from other devices are cached in Scene Storage. To speed up the local search, Scene Matcher also extracts the feature points from the request frame, generates the compact feature representation, and stores them into Scene Storage. As a result, each scene is represented as a {*frame image, recognition result, feature points, compact feature representation*} tuple.

A key design challenge of Scene Matcher is the selection of a *lightweight yet efficient and suitable* object recognition algorithm for mobile devices. SIFT [18] and SURF [4] have been used extensively in cloud-based AR (*e.g.,* Overlay [11] uses SURF and VisualPrint [12] uses SIFT). Instead of using them blindly for CARS, we evaluate the performance of 5 representative algorithms, SIFT, SURF, ORB [26], FAST+FREAK [1] and BRISK [16]. We use the following metrics, latency, feature quantity (*i.e.,* the number of generated feature descriptors), the size of a single feature descriptor, recognition accuracy and mobile device energy consumption.

We plot the experimental results with a dataset of 100 images (*i.e.,* movie posters) in Figure 5. The results for latency and feature quantity are averaged over the 100 images. We use only 10 images to measure energy consumption, as the results for them already show similar patterns as those of latency in Figure 5a (as expected). Among these algorithms, ORB achieves the best tradeoff between these metrics, higher than 90% accuracy with low latency and energy consumption and small memory space for storage, which is consistent with the result in Rublee *et al.* [26]. Thus, CARS uses ORB in Scene Matcher. A limitation of ORB is that it is less robust to image-scale changes [26], compared to other schemes such as SIFT and SURF. Thus, we need to resize the camera frame to a scale similar with the original image.

An important design decision we have made is the utilization of a *probabilistic model* to significantly speed up object recognition on mobile devices. Object recognition is relatively slow when comparing raw feature descriptors directly, as the numbers of feature descriptors may be fairly different for various images. There are several probabilistic models available, such as Gaussian Mixture Model [25] and Bernoulli Mixture Model (BMM) [14]. CARS uses BMM as GMM does not fit with binary descriptors such as ORB. Moreover, FV with BMM built upon ORB features is about two orders of magnitude faster than FV with GMM on SIFT features [2].

Scene Matcher contains Annotation Storage to cache annotations along with the scenes. When Scene Matcher finds a local match, Annotation Storage provides the corresponding content without requesting it from the cloud, which reduces the end-to-end latency and mobile data usage.

Annotation Synchronizer enables instantaneous interactions among users via synchronizing the annotation status so that one can see in real-time the modifications made by others. It shares with others the description of annotations (*e.g.,* a 3D object's pose, texture, lightness, *etc.*) generated by Annotation Renderer. Since the objects in the camera view are recognized by Scene Matcher, Peer Manager can learn from it their identities and communicate with others to see if they are looking at the same objects.

Recognition and tracking of common real-world objects guarantee that the coordinate systems of annotations for different users are synchronized (*i.e.,* the coordinate system of the physical world is the reference here). Modifications from a user change only the coordinates of annotations within his/her virtual world, which are shared by Annotation Synchronizer with others. We presently support moving, scaling, and rotating actions from users, and this design is capable of extending to much more complex operations. Annotation Synchronizer can resolve the conflicts among users by utilizing the strategies proposed by Gautier *et al.* [6].

Peer Manager utilizes WiFi Direct or Bluetooth Low Energy for peer discovery and D2D communications. It periodically scans for nearby devices running the same AR application. It maintains a list of peering devices with whom it can collaborate with. A challenge here is how to determine what recognition results and annotations to share and store in the local cache with a small size, which has been investigated extensively in the literature. For example, Psephos [10] is a fully distributed caching policy for content sharing in mobile opportunistic networks. To control the size of the cache, CARS can also optimize the sharing and fetching policy via preference-aware schemes such as PrefCast [17], as we will discuss in § 5. We leave the integration of existing approaches into CARS as our future work.

Although D2D communication is helpful in this paradigm to share locally AR results, its performance depends on the network conditions between mobile devices. As collaborating users have overlapped scenes of common ROIs in their camera views, they should be close to each other which usually leads to a higher bandwidth of WiFi Direct than that when communicating with the cloud. However, if future technologies (*e.g.,* LTE-Advanced and 5G) can offer higher throughput than WiFi Direct, we can also leverage the edge cloud, for CARS (as to be discussed in § 5). In order to

Step	Latency (ms)
ORB Feature Extraction	40.40 ± 2.31
Fisher Encoding with BMM	78.58 ± 6.37
Scene Retrieval	1.750 ± 0.55
ORB Feature Matching	9.738 ± 1.64
Pose Calculation	30.45 ± 4.73

Table 1: Breakdown of latency for different AR tasks on mobile devices. The value after ± is the standard deviation (for 20 experimental runs).

improve the reliability of D2D communication, especially when many users are using CARS in a crowded environment, we can limit the collaboration between only users with good wireless channel quality (*e.g.,* with signal strength higher than a threshold).

Cloud Manager communicates with the cloud via WiFi or LTE for submitting the recognition request and receiving the result and annotation, as described in § 2. Each request contains a compressed file of the camera frame. The result has the identity and 6DoF pose of the target object.

4 PRELIMINARY EVALUATION

In this section, we present the experimental results based on our proof-of-concept implementation of CARS.

4.1 Implementation

We build a prototype of CARS using the `OpenCV4Android` [21] library on Xiaomi MI5 smartphones (2.15 GHz Snapdragon 820 processor). We use ORB [26] for feature extraction and Fisher Vector [23] with BMM [14] to encode feature descriptors. The phone downloads the BMM model for the entire database from the cloud. Note that the size of BMM model does not depend on the number of images in the database and is only 131.58 KB for our setup. Motion Detector uses gyroscope and accelerometer on the phone to analyze hand motion of users. Annotation Renderer employs a XML file for describing the status of annotations.

We simplify the setup and configuration of the object-recognition pipeline to make it feasible on mobile devices, as our previous work [30] demonstrated when using the same pipeline it takes much longer time than cloud-based object recognition. First, we reduce the number of ORB feature points from ∼500 (in cloud-based AR) to 200. This reduction has almost no impact on the recognition accuracy, given the small size of the local cache (with only 100 images). Second, due to the small search space in the local cache, we do not need to use the Local Sensitive Hashing (LSH) of FV. Third, we skip object detection and thus our mobile image retrieval supports the recognition of only one object.

4.2 Experimental Results

Our preliminary performance evaluation focuses on the end-to-end latency, as we believe it should be the first-class citizen for AR. The latency is the processing time from generating the recognition request to displaying the annotation. It has two parts, object recognition and annotation download.

We set up a VM (8 vCPUs @ 2.5GHz and 32GB memory) in a public cloud. The mobile device communicates with the cloud via WiFi, with a Round Trip Time (RTT) of ∼36 ms. With the WiFi interface connected to both an access point (in Station mode) and a peer device (in WiFi Direct mode), the bandwidth is ∼13 Mbps between the phone and the cloud, and ∼32 Mbps between the phones. When the phone connects to the cloud via cellular networks, the bandwidth of WiFi Direct increases to ∼60 Mbps, as it does not need to share the airtime with the Station mode. The RTT of WiFi Direct between the phones is only ∼3.5 ms. Using the pipeline described in § 2.1, the latency for cloud-based recognition is ∼266 ms. Downloading a 5 MB (the most common size shown in Figure 2a) annotation from the server takes ∼3084 ms.

The local object recognition has five steps, and we list their completion time in Table 1. The recognition latency depends on the nearest neighbor search in Scene Matcher. We define the recognition accuracy as the ratio between the number of successful recognition requests over the total number of requests. CARS can achieve ∼90% accuracy when searching up to 5 nearest neighbors (*i.e.,* repeating feature matching at most 5 times). Thus, the local recognition latency is at most ∼200 ms. When considering only the nearest neighbor (*i.e.,* executing feature matching only once), the recognition latency is ∼161 ms with close to 60% accuracy.

Downloading the same 5 MB annotation via WiFi Direct takes only ∼1266 ms, much faster than downloading from the cloud. If the phone communicates with the cloud via cellular networks, this download time would be even shorter (due to the higher throughput). Note that since annotations are usually shared before recognition happens, users may not perceive this download latency. In a nutshell, the reduction of end-to-end latency mainly benefits from the fact that CARS can *perform lightweight object recognition on mobile devices by leveraging the results of cloud-based AR.*

5 DISCUSSION AND FUTURE WORK

In this section, we discuss several unsolved issues and directions of future work.

Preemptive Caching. Other than D2D communication, CARS can also benefit from cloud-initiated push to enable collaborative AR. For example, if CARS knows there are three more paintings on the left side of Mona Lisa and Bob is moving toward that direction, it can request, *in advance*, the cloud to push existing recognition results from others for these paintings if available. In this case, the collaboration does not happen directly over D2D, but utilizes the cloud as a *remote cache* for sharing. Note that this approach requires localization/mobility prediction and may consume extra mobile data, especially for outdoor scenarios. AR applications can utilize the edge cloud, *e.g.,* Cloudlets [27], to further improve the QoE of the preemptive caching.

Sharing Policy. Regarding the sharing policy of CARS, users can express their personal interests via preferences (*i.e.,* utility for an object) [17], which reduces the chance of exchanging objects that a user may not be curious about. To further minimize the communication overhead on D2D, CARS users will not forward recognition results and annotations from others and should avoid always-on-sharing for reducing energy consumption. Moreover, we can assign a time-to-live (as a geographical bound) for the scene a user plans to share with others, which decreases the probability that the scene will not be consumed by them.

Evaluation and Deployment. The performance evaluation in this paper is preliminary. For example, we use only two smartphones in the experiments, measure mainly the latency reduction, and the target objects are all movie posters. As shown in CoMon [15], opportunistic cooperation among nearby users can reduce energy consumption of mobile sensing. As our future work, we plan to conduct large-scale real-world experiments by deploying CARS in places such as museums and galleries, and evaluate its performance thoroughly (*e.g.,* by measuring energy consumption on mobile devices).

6 RELATED WORK

We divide existing work into two categories: cloud-based augmented reality and mobile image recognition.

Cloud-Based AR can be either non-continuous or continuous. For the former, users cannot move their mobile devices before getting the results from the cloud. For example, it takes around 500 ms for Overlay [11] to finish object recognition and rendering, during which a user needs to pause the camera at the interested object. VisualPrint [12] optimizes the uplink bandwidth requirements by uploading fingerprints extracted from the feature points of a camera frame to the cloud for recognition. In contrast to Overlay and VisualPrint, Glimpse [5] is a continuous face and road-sign recognition system. It uses an active cache of camera frames to hide the cloud-offloading latency by tracking objects in real time.

Image Recognition is a key component of augmented reality. CrowdSearch [29] combines image search and real-time human validation via crowdsourcing systems, such as the Amazon Mechanical Turk. ThirdEye [24] tracks the browsing behaviors of customers in retail stores by identifying the item a user is gazing at through an online image search service. Gabriel [8] assists users with their cognitive abilities using Google Glass, which leverages mobile edge computing for reducing the end-to-end latency of image recognition. Different from the above work, CARS explores the collaborative nature of mobile users for cloud-based AR systems to improve the quality of user experience.

7 CONCLUSION

In this paper, we propose CARS, a collaborative AR framework for socialization. CARS leverages the unique feature of AR that its users who share common scenes are usually close to each other. By enabling the social nature of human beings, CARS supports the interaction and coordination among users of cloud-based AR applications. Given that the size of local cache is usually small, we simplify the pipeline of object recognition to make it feasible on mobile devices. We build a proof-of-concept for CARS on smartphones and demonstrate it can reduce the object-recognition latency by up to 40%, compared to purely cloud-based AR.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful comments. The research of Pan Hui was supported in part by the General Research Fund from the Research Grants Council of Hong Kong under Grant 26211515 and Grant 16214817. The research of Feng Qian was supported in part by a Google Faculty Award.

REFERENCES

[1] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. 2012. Freak: Fast Retina Keypoint. In *Proceedings of CVPR*.
[2] Giuseppe Amato, Fabrizio Falchi, and Lucia Vadicamo. 2016. Aggregating binary local descriptors for image retrieval. *Multimedia Tools and Applications* (2016), 1–31.
[3] Apple Inc. 2017. Introducing ARKit: Augmented Reality for iOS. https://developer.apple.com/arkit/. (2017). [accessed on 20-October-2017].
[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. SURF: Speeded Up Robust Features. *Proceedings of ECCV* (2006).
[5] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. In *Proceedings of SenSys*.
[6] Laurent Gautier, Christophe Diot, and Jim Kurose. 1999. End-to-end transmission control mechanisms for multiparty interactive applications on the Internet. In *Proceedings of INFOCOM*.
[7] Aristides Gionis, Piotr Indyk, Rajeev Motwani, and others. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB*, Vol. 99. 518–529.
[8] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards Wearable Cognitive Assistance. In *Proceedings of MobiSys*.
[9] Berthold KP Horn and Brian G Schunck. 1981. Determining Optical Flow. *Artificial intelligence* 17, 1-3 (1981), 185–203.
[10] Stratis Ioannidis, Laurent Massoulié, and Augustin Chaintreau. 2010. Distributed Caching over Heterogeneous Mobile Networks . In *Proceedings of SIGMETRICS*.
[11] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. 2015. Overlay: Practical Mobile Augmented Reality. In *Proceedings of MobiSys*.
[12] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. 2016. Low Bandwidth Offload for Mobile AR. In *Proceedings of CoNEXT*.
[13] Eric Johnson. 2015. Boeing Says Augmented Reality Can Make Workers Better, Faster. http://www.recode.net/2015/6/8/11563374/boeing-says-augmented-reality-can-make-workers-better-faster. (2015). [accessed on 20-October-2017].
[14] Alfons Juan and Enrique Vidal. 2004. Bernoulli mixture models for binary images. In *Proceedings of ICPR*.
[15] Youngki Lee, Younghyun Ju, Chulhong Min, Seungwoo Kang, Inseok Hwang, and Junehwa Song. 2011. CoMon: Cooperative Ambience Monitoring Platform with Continuity and Benefit Awareness. In *Proceedings of MobiSys*.
[16] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. 2011. BRISK: Binary robust invariant scalable keypoints. In *Proceedings of ICCV*.
[17] Kate Ching-Ju Lin, Chun-Wei Chen, and Cheng-Fu Chou. 2012. Preference-aware content dissemination in opportunistic mobile social networks. In *Proceedings of INFOCOM*.
[18] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *Journal of Computer Vision* 60, 2 (2004), 91–110.
[19] Microsoft. 2017. HoloLens. https://www.microsoft.com/microsoft-hololens/. (2017). [accessed on 20-October-2017].
[20] Ben Nelson. 2017. VR/AR Challenge finalist WayPoint RX take the guess work out of filling prescriptions. https://developer.att.com/blog/vr-ar-challenge-waypoint-rx. (2017). [accessed on 20-October-2017].
[21] OpenCV. 2017. OpenCV4Android. http://opencv.org/platforms/android/. (2017). [accessed on 20-October-2017].
[22] Sergio Orts-Escolano, Christoph Rhemann, and others. 2016. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST)*.
[23] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier. 2010. Large-scale image retrieval with compressed Fisher vectors. In *Proceedings of CVPR*.
[24] Swati Rallapalli, Aishwarya Ganesan, Krishna Chintalapudi, Venkat N Padmanabhan, and Lili Qiu. 2014. Enabling Physical Analytics in Retail Stores Using Smart Glasses. In *Proceedings of MobiCom*.
[25] Douglas A Reynolds, Thomas F Quatieri, and Robert B Dunn. 2000. Speaker Verification Using Adapted Gaussian Mixture Models. *Digital Signal Processing* 10, 1-3 (2000), 19–41.
[26] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *Proceedings ICCV*.
[27] Mahadev Satyanarayanan, Paramvir Bahl, RamÃşn Caceres, and Nigel Davies. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (2009), 14–23.
[28] Unity. 2017. Asset Store. https://www.assetstore.unity3d.com/. (2017). [accessed on 20-October-2017].
[29] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. 2010. CrowdSearch: Exploiting Crowds for Accurate Real-Time Image Search on Mobile Phones. In *Proceedings of MobiSys*.
[30] Wenxiao Zhang, Bo Han, and Pan Hui. 2017. On the Networking Challenges of Mobile Augmented Reality . In *Proceedings of VR/AR Network*.