On the Networking Challenges of Mobile Augmented Reality

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

Bo Han
AT&T Labs Research
1 AT&T Way
Bedminster, New Jersey 07921
bohan@research.att.com

Pan Hui
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong SAR, China
panhui@cse.ust.hk

ABSTRACT

In this paper, we conduct a reality check for Augmented Reality (AR) on mobile devices. We dissect and measure the cloud-offloading feature for computation-intensive visual tasks of two popular commercial AR systems. Our key finding is that their cloud-based recognition is still not mature and not optimized for latency, data usage and energy consumption. In order to identify the opportunities for further improving the Quality of Experience (QoE) for mobile AR, we break down the end-to-end latency of the pipeline for typical cloud-based mobile AR and pinpoint the dominating components in the critical path.

CCS CONCEPTS

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

KEYWORDS

Augmented reality, cloud offloading, reality check, networking challenges, end-to-end latency

ACM Reference format:
Wenxiao Zhang, Bo Han, and Pan Hui. 2017. On the Networking Challenges of Mobile Augmented Reality. In *Proceedings of VR/AR Network '17, Los Angeles, CA, USA, August 25, 2017,* 6 pages.
https://doi.org/10.1145/3097895.3097900

1 INTRODUCTION

Augmented Reality (AR) has recently drawn tremendous attention from both the industry and the research community, due to the advances on AR devices and platforms (*e.g.,* Google Glass[1], Google Tango platform[2], and Microsoft HoloLens[3]). A typical AR system recognizes nearby objects and augments them with content (either real or virtual). When running on mobile devices, such as smartphones, an AR application takes the camera feed as the input and locates objects of interest in the current view using computer vision technologies [1, 7]. The key challenge of enabling AR on mobile devices, that needs no prior knowledge of a user's environment, is the heavy computation demands of recognizing surrounding objects for augmentation. Another issue is the ability of recognizing a large number of objects, which usually requires the storage of large datasets and may not be feasible on mobile devices.

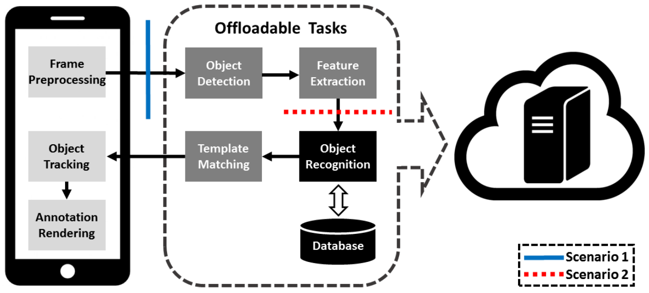


Figure 1: Pipeline of typical mobile AR systems. Tasks are colored according to their computation demands (*i.e.,* the darker, the heavier). The blue solid line and red dotted line mark two cloud offloading scenarios (*i.e.,* sending camera frames or their feature points to the cloud).

Cloud offloading is a promising technology to fill the gap between the insufficient mobile processing capability and the high computation demands of AR applications. A cloud-based AR system offloads certain visual tasks (*e.g.,* feature extraction and object recognition) to the cloud [4]. While it addresses the computation issue, it introduces extra latency which may render the Quality of Experience (QoE) unsatisfactory. For a 30 frame-per-second (FPS) camera feeding rate, the frame interval is around 33 ms. Thus, ideally the cloud should return the recognition results within that interval, especially for continuous recognition [1] which does not require a user to pause the camera at an object for seconds. Otherwise, the results may be stale.

In this paper, we attempt to answer the question *whether it is feasible to enable cloud-based mobile AR with acceptable QoE?* We investigate carefully and systematically the cloud recognition procedure of a few existing commercial AR systems and measure their performance (Section 3). The metrics we are interested in are end-to-end latency, data usage and energy consumption on mobile devices. Our key finding is that they are still in the very early stage. For example, the end-to-end latency of two systems under consideration

[1]https://www.google.com/glass/
[2]https://get.google.com/tango/
[3]https://www.microsoft.com/microsoft-hololens/

VR/AR Network '17, August 25, 2017, Los Angeles, CA, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5055-6/17/08…$15.00
https://doi.org/10.1145/3097895.3097900

is longer than 1 second. To better understand the performance bottleneck, we then break down the latency of the typical AR pipeline via our prototype implementation and pinpoint the dominating visual tasks (Section 4). The latency lower bound is around 250ms, which means there is a room for further improvements. We also discuss the opportunities to optimize the performance and QoE of mobile AR systems by thoroughly exploring the design space (Section 5). For example, one possible improvement is to leverage technologies such as GPU to further push the limit of AR latency.

2 BACKGROUND

In this section, we first analyze the pipeline of typical mobile AR systems and existing cloud offloading scenarios. We then present several use cases of mobile AR, such as entertainment and training.

2.1 Mobile AR Pipeline

We draw the pipeline of typical mobile AR systems in Figure 1. The first step is *object detection*, which recognizes targets in the view of a phone's camera by checking the existence of targets and finding the regions of interest (ROI) corresponding to the target. For each ROI, the next step is to apply *feature extraction* to extract feature points from it. The final step is to utilize *object recognition* to identify the original image stored in a database corresponding to target.

With the original image and the current frame, *template matching* verifies the result of object recognition as well as calculates the poses of targets, which are used as the initial input to *object tracking*. The goal of a tracker is to avoid object recognition for every frame. After the target is recognized, the identified original image determines the content associated with the target. Finally, *annotation rendering* will render the associated content to augment the recognized target. In Figure 1, we color different tasks according to their computation demands (the darker, the heavier), in terms of the time taken to complete a task on either a mobile device or a cloud server. For example, object recognition may take much more time than feature extraction (as we will show in Section 4).

Computation power and battery life are still restricting the performance of the current generation of mobile devices. To avoid heavy computation on them, AR systems can offload a set of the tasks mentioned above to the cloud. We use the blue and red lines to mark two existing common cloud offloading scenarios, by separating the tasks that will be executed on mobile devices and cloud servers. Scenario 1 (blue solid line) sends camera frames to the cloud for tasks starting from object detection [7]; whereas Scenario 2 (red dotted line) performs object detection and feature extraction on mobile devices and uploads the feature points to the cloud for object recognition [8]. Moreover, mobile devices can first run all the tasks locally and try to recognize the target with a small database. If object recognition fails, they can leverage the above two schemes to rescue and offload part of the tasks to the cloud with a large database.

2.2 Use Cases

In this section, we explain the use cases of mobile AR, including entertainment, training, driving, communication and healthcare.

Entertainment. The most popular use case of AR may be entertainment. There are numerous games leveraging AR technology. The most famous one is Pokemon Go[4], which is a location-based augmented reality game released in July 2016. Using the GPS on mobile devices, players can locate, train, capture, and battle the so called Pokemon (*i.e.,* virtual creatures), who appears on the screen as if it was in the same real-world location as the player. The player may throw a Poke Ball to capture that Pokemon.

Training. AR has also been widely used in education and training. The assembly tasks of aircrafts are known to be complicated, where mistakes from erroneously installing in the wrong order can be costly. To reduce costs and boost efficiencies, Boeing has been actively exploring augmented reality to train workers with complex and changing manufacturing requirements [9]. According to the study from Boeing, compared to the traditional training, the AR-trained workers are more willing to accept quickly the instructions.

Driving. Another emerging use case of AR is related to the safety of driving, for example, enhancing the dashboard and windshield of vehicles. Toyota is developing a system to display readings from speed and steering-angle sensors on the windshield in real time [18]. It employs an interior camera to track the view of a driver and a front-mounted camera to recognize the lane markings. As vehicles move, the system displays an image showing where the boundaries of the vehicle are within a lane. This real-time positioning information can greatly improve driving safety.

Communication. AR enables efficient collaborative communication among users from geo-separated sites. Holoportation [11] can capture, compress and transmit in real time high-quality 3D models of people to anywhere in the world. The models are then reconstructed and displayed by HoloLens. In order to make the system truly mobile, the recently enhanced Holoportation significantly reduces its bandwidth requirements by 97% (to 30–50 Mbps), and at the same time maintains good quality.

Healthcare. Healthcare is probably one of the most practical AR applications, as AR can significantly improve the quality of treatment that healthcare providers offer to their patients. On average pharmacists may fill up to 150 prescriptions per day and pharmaceutical mistakes can happen due to unfamiliarity with dosage, drug names, pill appearance, *etc.* AR apps can guide pharmacists to fill prescriptions quickly without any errors [10]. Moreover, by showing nearby defibrillator, AR can save the life of a person under heart attack.

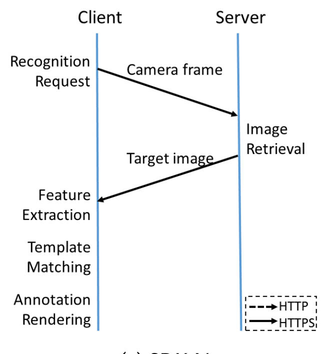
3 STATE-OF-THE-ART CLOUD AR

In this section, we first review the cloud offloading initiatives for AR in the research community. We then investigate a few commercial AR Software Development Kits (SDKs) that provide cloud-offloading features.

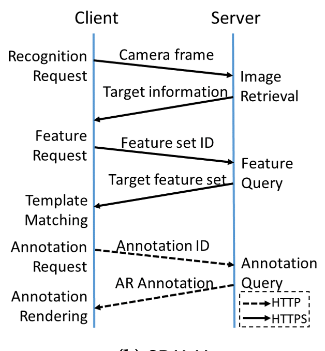
3.1 Research Initiatives

We can divide existing research projects into two categories: non-continuous and continuous AR. For the non-continuous case, mobile devices cannot move before the results are returned by the cloud.

[4]http://www.pokemongo.com/



(a) SDK-V



(b) SDK-H

Figure 2: Cloud recognition procedures of SDK-V **and** SDK-H**. Only one HTTPS request is involved in** SDK-V**, but two HTTPS requests and one HTTP request are involved in** SDK-H**.**

Non-Continuous Case. Overlay [7] is an AR system for indoor labeling and annotation. Its input consists of both camera images and sensor data which is leveraged to narrow down the object recognition search space by learning the geometric relation of objects. The cloud returns simple annotation text back to mobile devices after recognizing the uploaded camera frame. Built on top of Overlay, VisualPrint [8] reduces the bandwidth requirements by uploading fingerprints of extracted feature points instead of camera frames. VisualPrint carefully selects these fingerprints in order to increase the chance of yielding a unique match on the cloud. However, the impact on the end-to-end latency is not reported, when mobile devices run the feature extraction and fingerprint generation tasks.

Continuous Case. As we mentioned above, it is challenging to perform continuous object recognition, because the end-to-end latency may be longer than the camera-frame intervals. Glimpse [1] is a continuous object recognition system for faces and road signs. It hides the offloading latency by leveraging an active cache of camera frames to track objects on mobile devices upon receiving the recognition results from the cloud. In order to reduce bandwidth usage, Glimpse selects trigger frames and sends only them to servers for recognition, instead of uploading all camera frames.

3.2 Commercial SDKs

In this section, we investigate two popular commercial AR SDKs, SDK-V and SDK-H, by analyzing their cloud offloading strategies.

We study the cloud offloading features of these SDKs using the following approaches. Since we have no access to the source code of SDK-V and SDK-H, we set up a man-in-the-middle proxy, Fiddler[5], to capture and decrypt the messages exchanged between our mobile device and the cloud. We use the official Android demo App from SDK-V and the official Unity demo App from SDK-H (also deployed on the Android platform).

By analyzing the traces captured on our proxy, we plot the cloud recognition procedure of SDK-V in Figure 2a, which finishes within a single HTTPS session. The request contains a frame in the view of the camera, which is compressed as a JPEG file in gray-scale with a resolution of 640×360 (the resolution of the camera feed is 1920×1080). The response includes the recognized image saved in the cloud database. It is also compressed as a JPEG file in gray-scale with a width of 320, while the height varies between different images. After receiving the recognized image, the Android app applies feature extraction and template matching to identify the pose of the target within the current frame. When we conducted this study, SDK-V did not provide the cloud storage for annotation content and thus we packed the content into the Android application.

Similarly, we plot the cloud recognition of SDK-H in Figure 2b, which employs two HTTPS sessions and one HTTP session. The request of the first HTTPS session contains a camera frame, which is compressed as a JPEG file in color with a resolution of 640×360 (the same as SDK-V). Its response includes the URL of the original image, the key to the corresponding feature-point set in a server database, and the name of an annotation content file. The second HTTPS session requests the feature-point set. Compared to SDK-V, the feature extraction step is skipped on the client. SDK-H applies template matching to find the pose of the target in the view of current frame. Finally the last HTTP session requests the annotation content to be rendered on mobile devices.

We also investigated a third SDK, SDK-W, which does not provide a free-trial license for developers. Thus, we obtain the information presented below from its API documentation. The cloud recognition of SDK-W is similar to SDK-V, which finishes within one HTTPS session. The request body contains a camera frame, and the response includes the original image of the recognized target in the view of the camera.

3.3 Measurement Study

In this section, we evaluate the performance of SDK-V and SDK-H, by measuring the data usage, latency and energy consumption. As Overlay [7], VisualPrint [8] and Glimpse [1] are not open sourced yet, we cannot evaluate their performance at this moment.

The experimental procedure is as follows. We collect 100 movie posters online, upload them to the cloud servers of both SDK-V and SDK-H, and bind the image database with the corresponding apps. We then generate 20 image recognition requests for each of the apps from a Galaxy Note 4 phone via its WiFi interface. They all can be recognized correctly by both apps. The end-to-end latency is defined as the time between the camera frame acquisition and the rendering of results. We capture packet traces on the proxy to measure the data usage for both uplink and downlink. We use the Monsoon power monitor[6] to measure the energy consumption during the recognition procedure on mobile devices.

We show the experimental results in Table 1. SDK-V requires less bandwidth than SDK-H due to the following reasons. First, SDK-V uses images in gray-scale to reduce the uplink traffic. Second, the feature-point set returned by SDK-H is not optimized for network transfer and has a much larger size than the source image. The average size of these 20 original images is around 73.56 KB.

The end-to-end latency of SDK-V is lower than SDK-H, but they are both much higher than the 33 ms frame interval for 30 FPS. The cloud recognition of SDK-H is slower than SDK-V due to mainly two

[5]https://www.telerik.com/fiddler

[6]https://www.msoon.com/LabEquipment/PowerMonitor/

SDK	Uplink Traffic		Downlink Traffic		Latency (s)	Energy Usage
	Content	Size (KB)	Content	Size (KB)		(mAh)
SDK-V	Image	16.28 ± 1.99	Image	33.45 ± 8.24	1.410 ± 0.14	0.454 ± 0.04
SDK-H	Image	26.59 ± 2.59	Features	159.9 ± 54.9	19.32 ± 5.28	7.911 ± 2.11

Table 1: Experimental results of the performance evaluation for SDK-V and SDK-H's cloud recognition service. For SDK-H, the results include only the two HTTPS sessions to make the comparison fair. The value after ± is the standard deviation.

reasons. First, the recognition request of SDK-H is usually delayed by a few seconds after the request is triggered in the application. This delay increases almost linearly with the number of objects already recognized by SDK-H. Our hypothesis is that it may try to avoid the cloud-based recognition by comparing the image to be recognized with the existing results locally using a *sequential one-by-one matching*. However, we cannot confirm it due to the lack of access to the source code of SDK-H.

Second, the Round Trip Time (RTT) from our client to the SDK-V cloud server is lower than that of SDK-H. We run the `ping` experiments in Hong Kong to measure the RTT. The servers of SDK-V and SDK-H are located in Singapore and Shanghai, respectively (measured via IP geo-location lookup). Although the SDK-V server is further away from Hong Kong, the RTT is lower than SDK-H, 48 vs. 162 ms. This low RTT of SDK-V leads to a smaller completion time of the HTTPS session (~200 vs. 800–1000 ms). The result demonstrates the importance of deploying computation resources at locations with low RTT to end-users. Note that SDK-H has one more HTTPS session than SDK-V, which also adds extra latency. The high end-to-end latency of SDK-H is also a major contributor of its higher energy consumption than SDK-V (7.91 vs. 0.45 mAh).

Takeaway: *This measurement study demonstrates that the commercial cloud-based AR SDKs are still in their early stage with limited optimization of end-to-end latency and data usage (at least for their free version).*

4 END-TO-END LATENCY BREAKDOWN

In this section, we measure the time taken for each task listed in Figure 1, using our prototype implementation of a cloud-based mobile AR system.

4.1 Prototype Implementation

We implement most of the functions with OpenCV[7]. The client is an Android App running on a Xiaomi MI5 phone. We develop the server-side program in C++ and run it on a Google Cloud VM instance (8 vCPUs @ 2.5GHz and 32GB memory). The VM is located in Taiwan, not far away from where we run the experiments (in Hong Kong). Before presenting the measurement results, we first describe the implementation details of each task, although we have briefly introduced their functionality in Section 2.1.

Frame Preprocessing shrinks the size of a camera frame, by downscaling it from the 1920×1080 resolution to 480×270, then converting it from YUV color space to gray-scale, and finally encoding it into a JPEG file.

[7]http://opencv.org/

Object Detection identifies the ROI of possible image targets. We apply a sliding window that iterates through the image, and computes the variance in pixel gray-scale values for each window. We further extract rectangular areas with high variances from the image which are supposed to contain foreground textures.

Feature Extraction is supposed to extract features from the ROI using ORB [14]. Since we currently do not have the object detection algorithm implemented on the client, we extract features from the whole camera frames to make the comparison fair.

Object Recognition is an image retrieval pipeline that recognizes images. We build offline a Bernoulli Mixture Model (BMM) based on the features of all dataset images (the same as the ones described in Section 3.3). Using the BMM model, we first encode the feature descriptors of an image into a single Fisher Vector (FV). We then store all FVs of the images using a Locality Sensitive Hashing (LSH) for faster retrieval. To recognize an object, we find the nearest neighbor in the hash table of the extracted features from the previous stage.

Template Matching verifies the result of object recognition and calculates a 6DOF (Six Degrees of Freedom) pose of the target. With the feature descriptors from both the camera frame and the recognized image, we use a brute-force match to find the corresponding feature pairs, and calculate the homography of the target within the camera view, which stands for a 2D position transition.

Object Tracking tracks the feature points of an object for every frame. We use optical flow tracking [6] to keep the process lightweight and real-time.

Annotation Rendering updates the 6DOF pose of an object and calculates a 3D pose of the annotation, based on which we render a 3D content.

4.2 Experimental Results

We show the experimental results in Table 2, which are averaged over 100 runs. As image preprocessing, object tracking and annotation rendering should be performed on mobile devices, we do not report their results for the cloud. The tasks of the image retrieval pipeline, (*i.e.,* object detection, feature extraction, object recognition, and template matching), are offloadable. We show the results for both mobile devices and the cloud.

Among the four offloadable tasks, object recognition is the heaviest one (58% of the image retrieval pipeline on the server), which requires large storage of the image dataset and intensive computation. According to a study [16], it takes on average longer than 2 seconds to finish object recognition on a mobile CPU (Snapdragon 800). Considering the computation power and battery lifetime of mobile devices, object recognition should be done on the cloud (at least for the foreseeing future).

Stage	Mobile (ms)	Cloud (ms)
Image Preprocessing	34.70 ± 10.9	N/A
Object Detection	N/A	11.49 ± 0.37
Feature Extraction	131.0 ± 43.8	47.41 ± 1.17
Object Recognition	>2000	92.37 ± 19.5
Template Matching	143.1 ± 31.7	9.223 ± 4.08
Object Tracking	15.11 ± 6.28	N/A
Annotation Rendering	19.28 ± 0.47	N/A

Table 2: Breakdown of latency for different AR tasks. The value after ± is the standard deviation.

Object detection locates the ROI so that feature extraction and image recognition are executed in a smaller area which reduces the computation overhead. Object detection takes around 11.49ms on the server side which is not negligible. Although we currently have not implemented it on the phone yet, we expect it may take much more time. Feature extraction with the ORB [14] feature detector and descriptor takes 131ms on a high-end smartphone, which is almost three times as much as for the cloud.

Template matching on the cloud saves a lot of time compared to on mobile devices, as we can see from Table 2 (9.22 vs. 143.1ms). When it is done on the cloud, the server needs to transfer only simple pose data of the target after calculation, instead of the original image. However, as the calculated pose on the server is the one before the recognition request, this pose may be outdated due to the offloading latency. Another issue of transferring only object pose information is that tracking failure on the client becomes unrecoverable (without the original image). As described in Section 3.2, both SDK-V and SDK-H choose to perform template matching on the client for better robustness.

We also measure the network latency under various conditions. The results range from tens of milliseconds to a few hundred milliseconds. The lowest latency we can achieve on a local LTE testbed is around 14ms. After adding the network delay, the end-to-end latency of our cloud-based mobile AR system is at least 244ms.

To visualize the dominating factors of the end-to-end latency, we plot a pie chart in Figure 3 which shows the percentage of each task. We assume the network latency is 50 ms. Among these tasks, we offload object detection, feature extraction, object recognition and template matching to the cloud. This figure clearly demonstrates that even if we offload feature extraction and object recognition to the cloud, they still dominate the end-to-end latency.

Takeaway: *The latency of cloud mobile AR is still much higher than the 33 ms normal frame interval for 30 FPS. It is mainly dominated by the object recognition. Although it is challenging, there is plenty of room to optimize the performance and quality of experience for mobile AR.*

5 DISCUSSION

In this section, we discuss the opportunities to optimize the performance and QoE of mobile AR.

5.1 What to Offload?

As mentioned in Section 2.1, we can offload the tasks starting from either object detection (*i.e.,* sending images) or object recognition

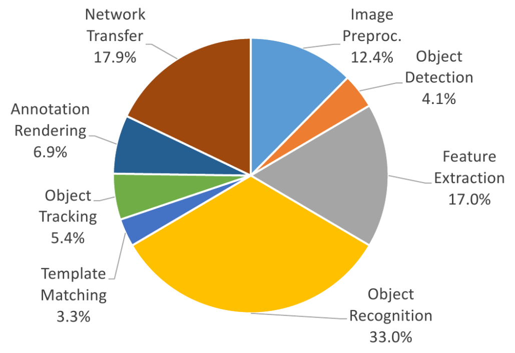


Figure 3: Percentage of each task in the end-to-end latency (assuming a 50-ms network transfer latency). Object detection, feature extraction, object recognition and template matching are on the cloud. Image preprocessing, object tracking and annotation rendering are on the mobile device.

(*i.e.,* sending feature points) to the cloud. There are pros and cons of each approach.

Two options exist when sending images to the cloud: video streams (*e.g.,* H.264) or individual images. Besides the intra-frame compression that is utilized by images, video streams also leverage inter-frame compression. Thus, they can provide better frame quality than individual images under the same data usage and improve the recognition accuracy. However, sending video streams requires buffering frames on mobile devices for encoding, which will add extra latency. There are also two options when sending feature points to the cloud, either the raw feature points or their fingerprints. The issue with raw feature points is that their size may be much larger than the image itself [8]. The challenge of sending fingerprints (as in VisualPrint [8]) is that generating these fingerprints from feature points is also computationally intensive, which may consume more energy and probably add extra latency.

Based on the above discussion, there is clearly a trade-off between latency, data usage, energy consumption and recognition accuracy. For example, sending images may lead to a higher data usage with a lower latency; whereas sending fingerprints of features may save data usage but increase the end-to-end latency.

5.2 Where to Offload?

There are multiple places where we can offload the object recognition task to, the remote cloud, the edge cloud [2, 4], and nearby devices. The first two have more computation power. The main difference between them is the network latency they introduce. Since the edge cloud is geographically closer to mobile devices, users would experience a lower overall latency. However, it may require architectural changes to the cellular access and core networks. As proposed in Serendipity [15], it is also possible to offload computation tasks to nearby devices. The challenge here is to guarantee the latency requirements for AR systems.

5.3 What Protocol to Use?

For cloud-based object recognition, network transfer influences both the end-to-end latency and system reliability. SDK-V and

SDK-H use HTTPS over TCP/IP to maintain a reliable and secure connection. The problem with a TCP-based scheme is the latency overhead introduced by its handshake mechanism. We can also employ UDP-based protocols, such as RTP (Real-time Transport Protocol) and QUIC (Quick UDP Internet Connections). They aim at providing lower transmission latency, which is still an important factor affecting the performance of mobile AR systems.

5.4 Push the Limit of AR Latency

From the above discussion, object recognition should be offloaded to the cloud. However, executing these tasks on commercial servers is still computationally intensive. As we can see from Table 2, the time taken by these tasks is more than 150ms. This server processing time is the dominating part of the overall latency incurred with cloud-based AR. While these computer vision tasks are slow with CPU, utilizing GPU could significantly speed up these visual tasks. We can implement the entire image recognition pipeline on GPU to achieve a much lower execution latency.

6 RELATED WORK

In this section, we briefly review existing work on mobile augmented reality and image search.

Augmented Reality. There is a plethora of work on enabling AR on mobile devices. Gammeter *et al.* [3] introduce a landmark recognition system which utilizes mobile clients for object tracking and servers for object recognition. The client-side tracking leverages sensing data to reset the visual tracking when necessary. Nestor [5] is a mobile system to recognize and estimate 6DOF pose of planar shapes. It applies recursive tracking to achieve interactive frame rate. Wagner *et al.* [17] enhances several feature extraction algorithms and makes them suitable for mobile platforms. Augmented Vehicular Reality (AVR) [12] is a system to improve the visibility of autonomous vehicles, by sharing visual information among neighboring vehicles and thus broadening the horizon of autonomous driving cars.

Image Recognition has been a key component of mobile augmented reality. CrowdSearch [19] is a crowdsourcing based system that combines cloud-based image search and real-time human validation via the Amazon Mechanical Turk. ThirdEye [13] is a smart glasses based system to track the physical browsing behaviors of customers in retail stores. When a user is gazing at an item, ThirdEye triggers an image search to an online service to identify the item. Gabriel [4] is another system that leverages Google Glass to assist users with their cognitive abilities. To reduce the end-to-end latency, it employs mobile edge computing for image recognition.

The above work focuses on enhancing or optimizing the performance of specific functions for mobile AR. Our work improves the understanding of commercial AR SDKs and the latency related to each component in a work-flow of typical cloud-based AR systems.

7 CONCLUSION

In this paper, we take a bird's-eye view of cloud-based mobile AR systems, by analyzing their typical pipeline, investigating the cloud-offloading features of several commercial SDKs, and measuring the time consumed by each component in the pipeline on both mobile devices and servers. We also offer practical suggestions and discuss potential opportunities to improve the QoE, especially the end-to-end latency, for mobile AR systems. We are currently building such a system by exploring the trade-offs between latency, data usage, energy consumption and recognition accuracy.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful comments. We thank Jennifer Yates, Vijay Gopalakrishnan, Bilal Anwer and Julius Mueller for discussions and suggestions. This work was supported in part by the General Research Fund from the Research Grants Council of Hong Kong under Grant 26211515 and in part by the Innovation and Technology Fund from the Hong Kong Innovation and Technology Commission under Grant ITS/369/14FP.

REFERENCES

[1] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. In *Proceedings of SenSys*, 2015.
[2] J. Cho, K. Sundaresan, R. Mahindra, J. Van der Merwe, and S. Rangarajan. ACACIA: Context-aware Edge Computing for Continuous Interactive Applications over Mobile Networks. In *Proceedings of CoNEXT*, 2016.
[3] S. Gammeter, A. Gassmann, L. Bossard, T. Quack, and L. Van Gool. Server-side object recognition and client-side object tracking for mobile augmented reality. In *Proceedings of IEEE Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2010.
[4] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards Wearable Cognitive Assistance. In *Proceedings of MobiSys*, 2014.
[5] N. Hagbi, O. Bergig, J. El-Sana, and M. Billinghurst. Shape recognition and pose estimation for mobile augmented reality. In *Proceedings of IEEE International Symposium Mixed and Augmented Reality (ISMAR)*, 2009.
[6] B. K. Horn and B. G. Schunck. Determining Optical Flow. *Artificial intelligence*, 17(1-3):185–203, 1981.
[7] P. Jain, J. Manweiler, and R. Roy Choudhury. Overlay: Practical Mobile Augmented Reality. In *Proceedings of MobiSys*, 2015.
[8] P. Jain, J. Manweiler, and R. Roy Choudhury. Low Bandwidth Offload for Mobile AR. In *Proceedings of CoNEXT*, 2016.
[9] E. Johnson. 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. [Online; accessed 31-May-2017].
[10] B. Nelson. 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. [Online; accessed 31-May-2017].
[11] S. Orts-Escolano, C. Rhemann, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. L. Davidson, S. Khamis, M. Dou, et al. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST)*, 2016.
[12] H. Qiu, F. Ahmad, R. Govindan, M. Gruteser, F. Bai, and G. Kar. Augmented Vehicular Reality: Enabling Extended Vision for Future Vehicles. In *Proceedings of HotMobile*, 2017.
[13] S. Rallapalli, A. Ganesan, K. Chintalapudi, V. N. Padmanabhan, and L. Qiu. Enabling Physical Analytics in Retail Stores Using Smart Glasses. In *Proceedings of MobiCom*, 2014.
[14] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2011.
[15] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura. Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices. In *Proceedings of MobiHoc*, 2012.
[16] M. Shoaib, S. Venkataramani, X.-S. Hua, J. Liu, and J. Li. Exploiting on-device image classification for energy efficiency in ambient-aware systems. In *Mobile Cloud Visual Media Computing*, pages 167–199. Springer, 2015.
[17] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):355–368, 2010.
[18] M. Watanabe. Head-up display apparatus for vehicle, Apr. 22 2013. US Patent App. 14/774,564.
[19] T. Yan, V. Kumar, and D. Ganesan. CrowdSearch: Exploiting Crowds for Accurate Real-Time Image Search on Mobile Phones. In *Proceedings of MobiSys*, 2010.